Σχεδίαση και Ανάλυση Αλγορίθμων

Μιχάλης Πλεξουσάκης

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
Τμήμα Μαθηματικών και Εφαρμοσμένων Μαθηματικών

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
Τμήμα Μαθηματικών και Εφαρμοσμένων Μαθηματικών

Σχεδίαση και Ανάλυση Αλγορίθμων

Μιχάλης Πλεξουσάκης

Πρόλογος

Οι σημειώσεις αυτές γράφτηκαν για τις ανάγκες του μαθήματος ΜΕΜ291 Σχεδίαση και Ανάλυση Αλγορίθμων, το χειμερινό εξάμηνο 2023-24.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Ένας *αλγόριθμος* είναι ένα σύνολο βημάτων για την εκτέλεση μιας εργασίας που περιγράφεται με αρκετή ακρίβεια ώστε ένας υπολογιστής να μπορεί να το εκτελέσει. Οι αλγόριθμοι επιλύουν υπολογιστικά προβλήματα. Απαιτούμε δύο πράγματα από έναν αλγόριθμο: για οποιαδήποτε δεδομένα ενός προβλήματος, θα πρέπει πάντα να παράγει μια σωστή λύση του, και θα πρέπει να χρησιμοποιεί αποτελεσματικά τους υπολογιστικούς πόρους. Σε αυτές τις σημειώσεις θα επικεντρωθούμε σε έναν μόνο πόρο: τον χρόνο. Πώς κρίνουμε τον χρόνο που απαιτεί ένας αλγόριθμος; Σε αντίθεση με την ορθότητα του αλγορίθμου, η οποία δεν εξαρτάται από τον συγκεκριμένο υπολογιστή στον οποίο εκτελείται ο αλγόριθμος, ο πραγματικός χρόνος εκτέλεσης ενός αλγορίθμου εξαρτάται από διάφορους παράγοντες που δεν σχετίζονται με τον ίδιο τον αλγόριθμο: την ταχύτητα του υπολογιστή, τη γλώσσα προγραμματισμού στην οποία υλοποιείται ο αλγόριθμος, τον μεταγλωττιστή ή τον διερμηνέα που μεταφράζει το πρόγραμμα σε κώδικα που εκτελείται στον υπολογιστή, την ικανότητα του προγραμματιστή που γράφει το πρόγραμμα και άλλες δραστηριότητες που λαμβάνουν χώρα στον υπολογιστή ταυτόχρονα με το πρόγραμμα που εκτελείται. Και όλα αυτά προϋποθέτουν ότι ο αλγόριθμος εκτελείται σε έναν μόνο υπολογιστή με όλα τα δεδομένα του στη μνήμη. Αξιολογούμε την ταχύτητα ενός αλγορίθμου με ένα συνδυασμός δύο ιδεών. Πρώτον, προσδιορίζουμε πόσο χρόνο χρειάζεται ο αλγόριθμος σε συνάρτηση με το μέγεθος της εισόδου του. Για παράδειγμα, στο πρόβλημα της αναζήτησης ενός στοιχείου σε μια δεδομένη λίστα, το μέγεθος της εισόδου θα ήταν ο αριθμός των στοιχείων της λίστας. Δεύτερον, εστιάζουμε στο πόσο γρήγορα αυξάνεται η συνάρτηση που χαρακτηρίζει τον χρόνο εκτέλεσης με το μέγεθος της εισόδου, δηλαδή στο ρυθμό αύξησης του χρόνου εκτέλεσης.

Οι αλγόριθμοί βρίσκονται στο επίκεντρο όλων όσων συμβαίνουν μέσα σε έναν υπολογιστή, αλλά είναι εξίσου μια *τεχνολογία* με όλα τα όσα άλλα βρίσκονται σε έναν υπολογιστή. Για παράδειγμα, θα δούμε αργότερα αλγόριθμους για την ταξινόμηση σε αύξουσα σειρά μια λίστας με n στοιχεία. Κάποιοι από αυτούς τους αλγόριθμους έχουν χρόνους εκτέλεσης n^2 ή $n \lg n$, με $\lg n = \log_2 n$. Ας υποθέσουμε ότι θέλουμε να ταξινομήσουμε 10^7 αριθμούς[1] στον ταχύ υπολογιστή Α ο οποίος εκτελεί έναν αλγόριθμο ταξινόμησης με χρόνο εκτέλεσης ανάλογο του n^2, και σε έναν λιγότερο ταχύ υπολογιστή Β ο οποίος εκτελεί έναν αλγόριθμο ταξινόμησης με χρόνο εκτέλεσης ανάλογο του $n \lg n$. Ας υποθέσουμε ότι ο υπολογιστής Α εκτελεί ένα δισεκατομμύριο εντολές ανά δευτερόλεπτο και ο υπολογιστής Β εκτελεί μόνο ένα εκατομμύριο εντολές ανά δευτερόλεπτο, οπότε ο υπολογιστής Α είναι 1000 φορές ταχύτερος από τον υπολογιστή Β. Ας υποθέσουμε τέλος, ότι ο αλγόριθμος που εκτελείται στον υπολογιστή Α απαιτεί $2n^2$ εντολές για την ταξινόμηση μια λίστας με n στοιχεία ενώ ο αλγόριθμος

[1] Αν και 10 εκατομμύρια αριθμοί μπορεί να φαίνονται πολλοί, αν οι αριθμοί είναι ακέραιοι που καταλαμβάνουν τέσσερα bytes, τότε η είσοδος στον αλγόριθμο ταξινόμησης είναι περίπου 40 megabytes, ελάχιστος χώρος στη μνήμη ακόμη και ενός φθηνού φορητού υπολογιστή ή κινητού τηλεφώνου.

του υπολογιστή Β χρειάζεται $100n \lg n$ εντολές. Αυτό σημαίνει ότι ο υπολογιστής Α θα χρειαστεί

$$\frac{2(10^7)^2 \text{ εντολές}}{10^9 \text{ εντολές/sec}} = 200\,000\,sec$$

κατι περισσότερο από 55 ώρες, ενώ ο υπολογιστής Β θα χρειαστεί

$$\frac{100 \cdot 10^7 \lg 10^7 \text{ εντολές}}{10^6 \text{ εντολές/sec}} \approx 23\,254\,sec$$

το οποίο είναι περίπου 6.5 ώρες! Επομένως, ο υπολογιστής Β, αν και 1000 φορές πιο αργός από τον υπολογιστή Α ταξινομεί τη λίστα των αριθμών 9 φορές πιο γρήγορα! Προφανώς, όσο το μέγεθος του προβλήματος μεγαλώνει τόσο πιο δραματικό γίνεται το πλεονέκτημα του αλγόριθμου με χρόνο εκτέλεσης ανάλογο του $n \lg n$. Ακόμα και με τις εντυπωσιακές εξελίξεις που βλέπουμε συνεχώς στους υπολογιστές, η συνολική απόδοση του συστήματος εξαρτάται από την επιλογή αποδοτικών αλγορίθμων όσο και από την γρήγορων υπολογιστών ή αποδοτικών λειτουργικών συστημάτων. Ακριβώς όπως γίνονται ραγδαίες πρόοδοι σε άλλες τεχνολογίες υπολογιστών, γίνονται και στους αλγορίθμους. Παραπέμπουμε τον αναγνώστη στο βιβλίο του MacCormick[4] το οποίο παραθέτει διάφορους αλγόριθμους που επηρεάζουν τη καθημερινή μας ζωή.

1.1 Ασυμπτωτική ανάλυση αλγορίθμων

Ο χρόνος που χρειάζεται ένας αλγόριθμος, εκφρασμένος ως συνάρτηση του μεγέθους της εισόδου, ονομάζεται *χρονική πολυπλοκότητα* του αλγορίθμου. Η οριακή συμπεριφορά της πολυπλοκότητας καθώς αυξάνεται το μέγεθος ονομάζεται *ασυμπτωτική χρονική πολυπλοκότητα*. Η ασυμπτωτική πολυπλοκότητα ενός αλγορίθμου είναι αυτή που τελικά καθορίζει το μέγεθος των προβλημάτων που μπορούν να επιλυθούν από τον αλγόριθμο. Ας υποθέσουμε ότι η ασυμπτωτική χρονική πολυπλοκότητα των αλγορίθμων A_1 έως A_5 είναι όπως φαίνεται στον Πίνακα 1.1. Αν υποθέσουμε ότι η μονάδα

Αλγόριθμος	Πολυπλοκότητα
A_1	n
A_2	$n \lg n$
A_3	n^2
A_4	n^3
A_5	2^n

Πίνακας 1.1: Χρονική πολυπλοκότητα των αλγορίθμων A_1 έως A_5.

του χρόνου ισούται με ένα microsecond, ο αλγόριθμος A_1 μπορεί σε ένα δευτερόλεπτο να επιλύσει ένα πρόβλημα με μέγεθος εισόδου 10^6, ενώ για τον αλγόριθμο A_5 το μέγεθος αυτό είναι μόνο 19. Στον Πίνακα 1.2 φαίνεται το μέγεθος των προβλημάτων που μπορούν να επιλυθούν σε ένα δευτερόλεπτο, ένα λεπτό, και μια ώρα, από τους πέντε αυτούς αλγόριθμους. Ας υποθέσουμε ότι η επόμενη γενιά υπολογιστών είναι δέκα φορές ταχύτερη από την τρέχουσα γενιά. Ο Πίνακας 1.3 δείχνει την αύξηση του μεγέθους του προβλήματος που μπορούμε να λύσουμε λόγω αυτής της αύξησης της ταχύτητας. Σημειώστε ότι με τον αλγόριθμο A_5 μια δεκαπλάσια αύξηση της ταχύτητας αυξάνει μόνο κατά τρία το μέγεθος του προβλήματος που μπορεί να επιλυθεί, ενώ με τον αλγόριθμο A_3 το μέγεθος υπερτριπλασιάζεται. Αν αντί για αύξηση της ταχύτητας, εξετάσουμε την επίδραση της χρήσης ενός πιο αποδοτικού αλγορίθμου, βλέπουμε από τον Πίνακα 1.3, χρησιμοποιώντας το ένα λεπτό ως βάση σύγκρισης, ότι αντικαθιστώντας τον αλγόριθμο A_4 με τον A_3 μπορούμε να λύσουμε ένα πρόβλημα 19 φορές μεγαλύτερο. Αντικαθιστώντας τον A_4 με τον A_2 μπορούμε να λύσουμε ένα πρόβλημα 300 φορές μεγαλύτερο. Τα αποτελέσματα αυτά είναι πολύ πιο εντυπωσιακά από τη διπλάσια βελτίωση που επιτυγχάνεται με δεκαπλάσια αύξηση της ταχύτητας. Αν ως βάση σύγκρισης χρησιμοποιηθεί

		μέγεθος προβλήματος		
Αλγόριθμος	Πολυπλοκότητα	1 sec	1 min	1 hour
A_1	n	10^6	6×10^7	3.6×10^9
A_2	$n \lg n$	6.2×10^4	2.8×10^6	1.3×10^8
A_3	n^2	1000	7745	60000
A_4	n^3	100	391	1532
A_5	2^n	19	25	31

Πίνακας 1.2: Μέγεθος προβλημάτων που επιλύονται από τους αλγόριθμους A_1 έως A_5 σε ένα δευτερόλεπτο, ένα λεπτό και μία ώρα.

Αλγόριθμος	Πολυπλοκότητα	μέγεθος (τώρα)	μέγεθος (νέα γενιά)
A_1	n	σ_1	$10\sigma_1$
A_2	$n \lg n$	σ_2	$\sim 10\,\sigma_2$
A_3	n^2	σ_3	$3.16\,\sigma_3$
A_4	n^3	σ_4	$2.1\,\sigma_4$
A_5	2^n	σ_5	$\sigma_5 + 3.3$

Πίνακας 1.3: Μέγεθος προβλημάτων που επιλύονται από τους αλγόριθμους A_1 έως A_5 μετά από τον δεκαπλασιασμό της ταχύτητας του υπολογιστή.

μια ώρα, οι διαφορές είναι ακόμη πιο σημαντικές. Καταλήγουμε στο συμπέρασμα ότι η ασυμπτωτική πολυπλοκότητα ενός αλγορίθμου είναι ένα σημαντικό μέτρο της ποιότητας ή της αποτελεσματικότητας ενός αλγορίθμου, η οποίο υπόσχεται να γίνει ακόμη πιο σημαντική με τις μελλοντικές αυξήσεις της υπολογιστικής ταχύτητας.

1.2 Ο αλγόριθμος της ενθετικής ταξινόμησης

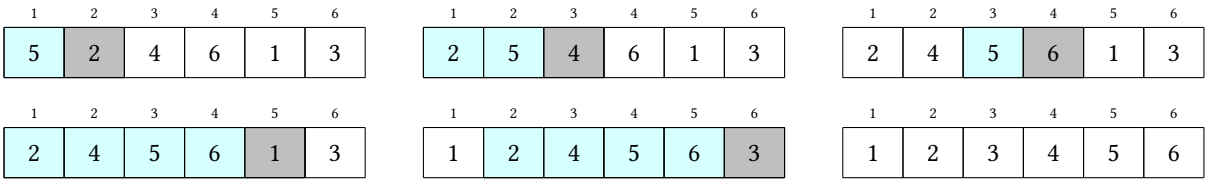
Ξεκινούμε τη μελέτη με τον αλγόριθμο της *ενθετικής ταξινόμησης* (insertion sort), ο οποίος είναι αρκετά ταχύς όταν πρόκειται για την ταξινόμηση μικρού πλήθους στοιχείων. Η ιδέα σε αυτόν τον αλγόριθμο είναι παρόμοια με αυτήν που χρησιμοποιείται όταν ταξινομούμε ένα σύνολο από τραπουλόχαρτα: για να βρούμε τη σωστή θέση ενός φύλλου τα συγκρίνουμε με καθένα από τα φύλλα που έχουμε ήδη στο χέρι μας, από δεξιά προς τ' αριστερά. Ο ψευδοκώδικας που παρουσιάζεται παρακάτω υπό τη μορφή της διαδικασίας Insertion-Sort ταξινομεί τα στοιχεία του πίνακα $A[1\,..\,n]$. Τα στοιχεία του πίνακα ταξινομούνται *επι τόπου*, δηλαδή αναδιατάσσονται εντός του πίνακα A. Όταν ολοκληρωθεί η εκτέλεση της Insertion-Sort τα στοιχεία του πίνακα A είναι ταξινομημένα κατ' αύξουσα σειρά. Οι δείκτες σ' έναν πίνακα είναι διαδοχικοί αριθμοί που μπορούν να ξεκινούν οπουδήποτε[2], αλλά συνήθως τους ξεκινάμε από το 1. Δεδομένου του ονόματος ενός πίνακα και ενός δείκτη στον πίνακα, τους συνδυάζουμε με αγκύλες για να υποδείξουμε ένα συγκεκριμένο στοιχείο του πίνακα. Για παράδειγμα, συμβολίζουμε το i-οστό στοιχείο ενός πίνακα A με $A[i]$.

[2] Αν προγραμματίζετε σε Python, C ή C++, είστε συνηθισμένοι σε πίνακες που ξεκινούν από το 0. Το να ξεκινούν οι πίνακες από το 0 είναι ωραίο για τους υπολογιστές, αλλά είναι συχνά πιο διαισθητικό να ξεκινάμε από το 1.

INSERTION-SORT(A, n)

for $j = 2$ **to** n	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$		
$i = j - 1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
$i = i - 1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
$A[i+1] = key$	c_8	$n-1$

Παρακάτω βλέπουμε τα βήματα της διαδικασίας INSERT-SORT για την ταξινόμηση των στοιχείων του πίνακα $A = [5, 2, 4, 6, 1, 3]$. Οι δείκτες των στοιχείων του πίνακα εμφανίζονται πάνω από τα στοιχεία του πίνακα. Σε γκρί φόντο εμφανίζεται το στοιχείο *key* ενώ σε κυανό φόντο είναι τα στοιχεία του πίνακα που συγκρίνονται με αυτό σε κάθε βήμα.



Στην αρχή κάθε επανάληψης του εξωτερικού βρόχου **for** με δείκτη j, ο υποπίνακας που αποτελείται από τα στοιχεία $A[1\,..\,j-1]$ είναι ήδη ταξινομημένος. Στην πραγματικότητα, τα στοιχεία $A[1\,..\,j-1]$ είναι τα στοιχεία που βρίσκονταν αρχικά στις θέσεις 1 έως $j-1$ αλλά τώρα είναι σε ταξινομημένη σειρά. Δηλώνουμε αυτές τις ιδιότητες του υποπίνακα $A[1\,..\,j-1]$ ως *αναλλοίωτη συνθήκη βρόχου* (loop invariant):

> Στην αρχή κάθε επανάληψης του βρόχου **for** με δείκτη j ο υποπίνακας $A[1\,..\,j-1]$ αποτελείται από στοιχεία που βρίσκονταν αρχικά στις θέσεις 1 έως $j-1$ αλλά τώρα είναι σε ταξινομημένη σειρά.

Χρησιμοποιούμε την αναλλοίωτη συνθήκη βρόχου για τον έλεγχο της ορθότητας του αλγορίθμου δείχνοντας ότι:

- ισχύει πριν την πρώτη επανάληψη του βρόχου **for**
- αν ισχύει πριν μία επανάληψη του βρόχου τότε παραμένει σε ισχύ και πριν την επόμενη επανάληψη
- ισχύει κατα τον τερματισμό των επαναλήψεων του βρόχου

Πριν την πρώτη επανάληψη, όταν $j = 2$, ο υποπίνακας $A[1\,..\,j-1]$ αποτελείται από το μοναδικό στοιχείο $A[1]$, και είναι, επομένως, τετριμμένα ταξινομημένος. Δείχνουμε τώρα ότι κάθε επανάληψη διατηρεί την αναλλοίωτη συνθήκη βρόχο. Η επανάληψη **for** με δείκτη j μετακινεί τα στοιχεία $A[j-1]$, $A[j-2], \ldots$, μια θέση προς τα δεξιά μέχρι να βρεθεί η κατάλληλη θέση για το στοιχείο $A[j]$. Τέλος, όταν η επανάληψη **for** τερματίζει έχουμε $j = n+1$, το οποίο σημαίνει ότι ο υποπίνακας $A[1\,..\,n]$ είναι ταξινομημένος. Επομένως ο πίνακας είναι ταξινομημένος.

1.3 Ανάλυση της ενθετικής ταξινόμησης

Στον αλγόριθμο της ενθετικής ταξινόμησης έχουμε σημειώσει το κόστος κάθε εντολής και πόσες φορές αυτή εκτελείται. Για $j = 2, 3, \ldots, n$, συμβολίζουμε με t_j τον αριθμό των φορών που εκτελείται ο έλεγχος της επανάληψης **while** για την συγκεκριμένη τιμή της μεταβλητής j. Θυμόμαστε, βέβαια, ότι η εντολή ελέγχου μιας ανακύκλωσης εκτελείται μια φορά *επιπλέον* από τις εντολές στο σώμα της ανακύκλωσης, και ότι τα σχόλια είναι μη εκτελέσιμες εντολές και δεν συμβάλλουν στον χρόνο εκτέλεσης ενός αλγόριθμου. Συμβολίζουμε με $T(n)$ τον χρόνο εκτέλεσης του αλγόριθμου της ενθετικής ταξινόμησης με είσοδο έναν πίνακα με n στοιχεία. Εύκολα βλέπουμε ότι

$$\begin{aligned} T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\ + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1). \end{aligned}$$

Η παραπάνω σχέση δείχνει ότι ο χρόνος εκτέλεσης του αλγόριθμου εξαρτάται από τα στοιχεία που ταξινομούνται. Για τον αλγόριθμο της ενθετικής ταξινόμησης η καλύτερη περίπτωση είναι αυτή ενός ήδη ταξινομημένου πίνακα. Σε αυτή την περίπτωση έχουμε ότι $t_j = 1$ για $j = 2, 3, \ldots, n$, και επομένως ο χρόνος εκτέλεσης είναι

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8),$$

μια *γραμμική* συνάρτηση του n. Αν όμως ο πίνακας εισόδου είναι ταξινομημένος με φθίνουσα σειρά, τότε $t_j = j$ για $j = 2, 3, \ldots, n$, και επομένως ο χειρότερος χρόνος εκτέλεσης είναι[3]

$$\begin{aligned} T(n) = \frac{c_5 + c_6 + c_7}{2} n^2 + \left(c_1 + c_2 + c_4 + c_8 + \frac{c_5 - c_6 - c_7}{2} \right) n \\ - (c_2 + c_4 + c_5 + c_8), \end{aligned}$$

μια *τετραγωνική* συνάρτηση του n.

Επικεντρωνόμαστε στην εύρεση του μεγαλύτερου χρόνου εκτέλεσης για οποιαδήποτε είσοδο μεγέθους n για τους εξής λόγους: πρώτα, ο χρόνος εκτέλεσης χειρότερης περίπτωσης ενός αλγορίθμου είναι ένα άνω φράγμα του χρόνου εκτέλεσης για οποιαδήποτε είσοδο. Τέλος, η "μέση περίπτωση" είναι συνήθως τόσο κακή όσο και η χειρότερη περίπτωση. Ας υποθέσουμε ότι επιλέγουμε τυχαία n αριθμούς και εφαρμόζουμε την ενθετική ταξινόμηση. Πόσο χρόνο χρειάζεται για να να καθορίσει πού στον υποπίνακα $A[1\,..\,j-1]$ να εισαγάγει το στοιχείο $A[j]$; Κατά μέσο όρο τα τα μισά στοιχεία θα είναι μικρότερα του $A[j]$ και τα υπόλοιπα μεγαλύτερα. Επομένως $t_j = j/2$. Αν υπολογίσουμε εκ νέου τον χρόνο εκτέλεσης θα διαπιστώσουμε ότι είναι πάλι μια τετραγωνική συνάρτηση του μεγέθους εισόδου, όπως ακριβώς και ο χρόνος εκτέλεσης της χειρότερης περίπτωσης.

1.4 Ασυμπτωτική ανάλυση

Για τον αλγόριθμο της ενθετικής ταξινόμησης δείξαμε ότι ο χρόνος εκτέλεσης χειρότερης περίπτωσης είναι της μορφής $T(n) = an^2 + bn + c$, για κάποιες σταθερές a, b, c, οι οποίες εξαρτώνται από τα κόστη c_i των χρόνων εκτέλεσης των εντολών του αλγόριθμου. Μας ενδιαφέρει κυρίως ο ρυθμός μεταβολής της συνάρτησης χρονικής πολυπλοκότητας, δηλαδή, ο κυρίαρχος όρος an^2. Θα αγνοούμε, συνήθως, και τον συντελεστή του κυρίαρχου όρου μια και αυτός δεν είναι σημαντικός για τον

[3] Χρησιμοποιήσαμε εδώ το γεγονός ότι

$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

ρυθμό μεταβολής της συνάρτησης χρονικής πολυπλοκότητας. Επομένως, για τον χρόνο εκτέλεσης χειρότερης περίπτωσης του αλγόριθμου της ενθετικής ταξινόμησης θα γράφουμε $T(n) = \Theta(n^2)$ ή θα λέμε ότι είναι $\Theta(n^2)$.

Ορισμός. Για τη συνάρτηση $g(n)$ ορίζουμε $\Theta(g(n))$ ως το σύνολο των συναρτήσεων $f(n)$ για τις οποίες υπάρχουν θετικές σταθερές c_1, c_2 και δείκτης $n_0 \geq 1$ έτσι ώστε

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \quad \text{για κάθε } n \geq n_0. \tag{1.1}$$

Παράδειγμα. Δείχνουμε ότι για τη συνάρτηση $f(n) = n^2/10 - 3n$ ισχύει $f(n) = \Theta(n^2)$. Πράγματι, αρκεί να βρούμε θετικές σταθερές c_1, c_2 και $n_0 \geq 1$ τέτοιες ώστε

$$c_1 n^2 \leq \frac{n^2}{10} - 3n \leq c_2 n^2,$$

για κάθε $n \geq n_0$. Διαρώντας όλους τους όρους με n^2 παίρνουμε τη σχέση

$$c_1 \leq \frac{1}{10} - \frac{3}{n} \leq c_2.$$

Η δεξιά ανισότητα ισχύει για κάθε $n \geq 1$ αν επιλέξουμε $c_2 \geq 1/10$. Η αριστερή ανισότητα μπορεί να διασφαλιστεί για οποιαδήποτε τιμή $n \geq 31$ αν πάρουμε $c_1 \leq 1/310$. Επομένως, επιλέγοντας $c_1 = 1/301$, $c_2 = 1/10$ και $n_0 = 31$ επιβεβαιώνουμε ότι $f(n) = \Theta(n^2)$. Γενικότερα, αν $f(n) = an^2 + bn + c$, όπου a, b, c είναι σταθερές με $a > 0$ τότε και πάλι $f(n) = \Theta(n^2)$, δείτε την Άσκηση 3.

Παρατήρηση. Είναι προφανές ότι οποιαδήποτε σταθερή συνάρτηση είναι $\Theta(1)$ (ακριβέστερα $\Theta(n^0)$, για να καταστήσουμε σαφές ποιά μεταβλητή τείνει στο άπειρο).

Ο συμβολισμός Θ δηλώνει ότι μια συνάρτηση είναι ασυμπτωτικά φραγμένη από επάνω και από κάτω. Όταν έχουμε μόνο ένα ασυμπτωτικό άνω φράγμα, χρησιμοποιούμε τον συμβολισμό O.

Ορισμός. Για τη συνάρτηση $g(n)$ ορίζουμε $O(g(n))$ ως το σύνολο των συναρτήσεων $f(n)$ για τις οποίες υπάρχουν θετικές σταθερές c και n_0 τέτοιες ώστε

$$0 \leq f(n) \leq c\, g(n), \quad \text{για κάθε } n \geq n_0. \tag{1.2}$$

Είναι προφανές ότι αν $f(n) = \Theta(g(n))$ τότε έχουμε $f(n) = O(g(n))$. Για παράδειγμα, από την Άσκηση 3 έχουμε ότι οποιαδήποτε τετραγωνική συνάρτηση an^2+bn+c, όπου a, b, c είναι σταθερές με $a > 0$, είναι $O(n^2)$. Ισχύει, επίσης, ότι η γραμμική συνάρτηση $an+b$, με $a > 0$ είναι $O(n^2)$. Πράγματι, αν $n_0 = \max(1, -|b|/a)$ τότε, για $n \geq n_0$ έχουμε ότι $an + b \geq 0$ και βέβαια, $an + b \leq (a + |b|)n^2$. Επομένως, ισχύει η (1.2) με $c = a + |b|$.

Όταν έχουμε μόνο ένα ασυμπτωτικό κάτω φράγμα, χρησιμοποιούμε τον συμβολισμό Ω.

Ορισμός. Για τη συνάρτηση $g(n)$ ορίζουμε $\Omega(g(n))$ ως το σύνολο των συναρτήσεων $f(n)$ για τις οποίες υπάρχουν θετικές σταθερές c και n_0 τέτοιες ώστε

$$0 \leq c\, g(n) \leq f(n), \quad \text{για κάθε } n \geq n_0. \tag{1.3}$$

Παρατήρηση. Αφού ο συμβολισμός O ορίζει ένα άνω φράγμα, όταν φράσσουμε μέσω αυτού τον χρόνο εκτέλεσης χειρότερης περίπτωσης ενός αλγόριθμου, εξασφαλίζουμε ένα φράγμα του χρόνου εκτέλεσης του αλγόριθμου για οποιαδήποτε είσοδο. Επομένως το φράγμα $O(n^2)$ στον χρόνο εκτέλεσης χειρότερης περίπτωσης της ενθετικής ταξινόμησης ισχύει και για τον χρόνο εκτέλεσης με οποιαδήποτε είσοδο. Όμοια, αφού ο συμβολισμός Ω ορίζει ένα κάτω φράγμα, όταν φράσσουμε μέσω αυτού τον χρόνο εκτέλεσης καλύτερης περίπτωσης ενός αλγορίθμου, έπεται ότι φράσσουμε και τον χρόνο εκτέλεσης του αλγορίθμου για οποιαδήποτε είσοδο. Επομένως, το γεγονός ότι ο χρόνος καλύτερης περίπτωσης της ενθετικής ταξινόμησης είναι $\Omega(n)$ συνεπάγεται ότι ο χρόνος εκτέλεσης της ενθετικής ταξινόμησης είναι $\Omega(n)$.

Με βάση τους ορισμούς του ασυμπτωτικού συμβολισμού που δώσαμε παραπάνω, μπορούμε εύκολα να αποδείξουμε το ακόλουθο θεώρημα:

Θεώρημα. Για οποιοδήποτε ζεύγος συναρτήσεων $f(n)$ και $g(n)$, ισχύει ότι $f(n) = \Theta(g(n))$ όταν και μόνο όταν $f(n) = O(g(n))$ και $f(n) = \Omega(g(n))$.

Επίσης, πολλές από τις ιδιότητες που ισχύουν για τις σχέσεις μεταξύ πραγματικών αριθμών ισχύουν και για τις ασυμπτωτικές συγκρίσσεις. Έτσι, εύκολα φαίνεται ότι αν $f(n), g(n)$ και $h(n)$ είναι ασυμπτωτικά θετικές, τότε, για παράδειγμα, $f(n) = O(f(n))$ και ισχύει η μεταβατική σχέση $f(n) = \Theta(g(n))$ και $g(n) = \Theta(h(n))$ τότε $f(n) = \Theta(h(n))$.

1.5 Συμβολισμός o και ω

Ορισμός. Λέμε ότι $f(n) = o(g(n))$ αν για οποιαδήποτε θετική σταθερά c υπάρχει σταθερά n_0 τέτοια ώστε

$$0 \leq f(n) < cg(n),$$

για κάθε $n \geq n_0$.

Για παράδειγμα, $2n = o(n^2)$ γιατί, αν c είναι μια οποιαδήποτε θετική σταθερά και πάρουμε $n_0 = \lceil 2/c \rceil + 1$, τότε για $n \geq n_0$ έχουμε

$$n \geq n_0 > \frac{2}{c} \implies cn > 2 \implies cn^2 > 2n.$$

Προφανώς, $2n \geq 0$ για κάθε $n \geq 1$.

Στον ορισμό του συμβολισμού o η συνάρτηση $f(n)$ γίνεται αμελητέα σε σχέση με την $g(n)$, καθώς το n τείνει στο άπειρο, δηλαδή,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0,$$

σχέση η οποία θα μπορούσε να χρησιμοποιηθεί εναλλακτικά του ορισμού που δώσαμε.

Κατ΄αναλογία, ο συμβολισμός ω συνδέεται με τον συμβολισμό Ω δηλώνοντας ένα κάτω φράγμα δεν είναι ασυμπτωτικά αυστηρό: θα λέμε ότι $f(c) = \omega(g(n))$ αν για οποιαδήποτε θετική σταθερά c υπάρχει σταθερά n_0 τέτοια ώστε

$$0 \leq cg(n) < f(n),$$

για κάθε $n \geq n_0$. Από τον ορισμό του συμβολισμού ω εύκολα βλέπουμε ότι

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty,$$

και βέβαια ότι $f(n) = \omega(g(n))$ αν και μόνο αν $g(n) = o(f(n))$, σχέσεις τις οποίες θα μπορούσαμε να χρησιμοποιήσουμε εναλλακτικά του ορισμού που δώσαμε.

Είναι εύκολο να δούμε, με χρήση του ορισμού του συμβολισμού ω, ότι $n^2/100 = \omega(n)$. Πράγματι, αν c είναι οποιαδήποτε θετική σταθερά και $n_0 = \lceil 100c \rceil + 1$, τότε για $n \geq n_0$ έχουμε $cn < n^2/100$. Προφανώς $cn \geq 0$ για κάθε $n \geq 1$.

1.6 Μαθηματικός συμβολισμός και συνήθεις συναρτήσεις

Κατώφλι και ανώφλι. Αν $x \in \mathbb{R}$ θα συμβολίζουμε με $\lfloor x \rfloor$ τον μεγαλύτερο ακέραιο που είναι μικρότερος ή ίσος του x, δηλαδή,

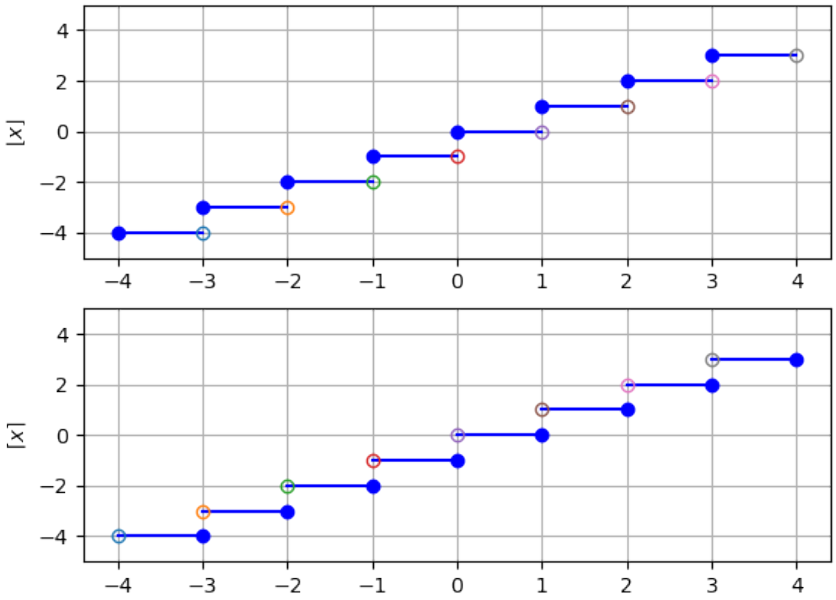
$$\lfloor x \rfloor = \max\{n \in \mathbb{Z} : n \leq x\}.$$

Θα συμβολίζουμε με $\lceil x \rceil$ τον μικρότερο ακέραιο που είναι μεγαλύτερος ή ίσος του x, δηλαδή,

$$\lceil x \rceil = \min\{n \in \mathbb{Z} : n \geq x\}.$$

Από τον ορισμό τον συναρτήσεων κατώφλι (floor) και ανώφλι (ceiling) προκύπτει η χρήσιμη σχέση

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1,$$



Σχήμα 1.1: Οι συναρτήσεις κατώφλι και ανώφλι.

και ότι

$$\lceil x \rceil - \lfloor x \rfloor = \begin{cases} 0 & \text{αν } x \in \mathbb{Z}, \\ 1 & \text{αν } x \notin \mathbb{Z}. \end{cases}$$

Τέλος, είναι εύκολο να δει κανείς ότι τόσο η συνάρτηση κατώφλι όσο και η συνάρτηση ανώφλι είναι μονότονα αύξουσες συναρτήσεις.

Εκθετικές συναρτήσεις. Θυμόμαστε ότι για κάθε n και $a > 1$, η συνάρτηση a^n είναι μονότονα αύξουσα ως προς n. Θυμόμαστε επίσης ότι αν b είναι οποιαδήποτε πραγματική σταθερά τότε

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0, \tag{1.4}$$

σχέση από την οποία έπεται ότι $n^b = o(a^n)$. Επομένως, οποιαδήποτε εκθετική συνάρτηση με βάση $a > 1$ αυξάνεται ταχύτερα από οποιαδήποτε πολυωνυμική συνάρτηση.

Λογάριθμοι. Θα γράφουμε $\ln n$ για τον φυσικό λογάριθμο του n και $\lg n = \log_2 n$ για τον δυαδικό λογάριθμο. Αν για μια συνάρτηση $f(n)$ ισχύει $f(n) = O(\lg^k n)$, για κάποια σταθερά k, θα λέμε ότι η $f(n)$ είναι *πολυλογαριθμικά φραγμένη*. Από τη σχέση (1.4), θέτοντας όπου n και a τα $\lg n$ και 2^a, αντίστοιχα, προκύπτει ότι

$$\lim_{n\to\infty} \frac{\lg^b n}{n^a} = 0,$$

δηλαδή, $\lg^b n = o(n^a)$, για οποιαδήποτε σταθερά $a > 0$. Συνεπώς, οποιαδήποτε θετική, πολυωνυμική συνάρτηση αυξάνεται ταχύτερα από οποιαδήποτε πολυλογαριθμική.

1.7 Ασκήσεις

1. Δείξτε τη λειτουργία της ενθετικής ταξινόμησης για τον πίνακα $A = [31, 41, 59, 26, 41, 58]$.

2. Τροποποιήστε τον αλγόριθμο της ενθετικής ταξινόμησης έτσι ώστε να ταξινομεί τους αριθμούς σε φθίνουσα αντί αύξουσα σειρά.

3. Δείξτε ότι αν $f(n) = an^2 + bn + c$, όπου a, b, c είναι σταθερές με $a > 0$, τότε $f(n) = \Theta(n^2)$, με $c_1 = a/4$, $c_2 = 7a/4$ και $n_0 = \max\left\{|b|/a, \sqrt{|c|/a}\right\}$.

4. Έστω $f(n), g(n)$ ασυμπτωτικά μη αρνητικές συναρτήσεις. Δείξτε ότι

$$\max(f(n), g(n)) = \Theta(f(n) + g(n)).$$

5. Δείξτε ότι για οποιεσδήποτε πραγματικές σταθερές a και b, όπου $b > 0$, ισχύει $(n+a)^b = \Theta(n^b)$.

6. Δείξτε με ένα παράδειγμα ότι μεταξύ δύο ασυμπτωτικά θετικών συναρτήσεων $f(n)$ και $g(n)$ είναι δυνατόν να μην ισχύει η σχέση $f(n) = O(g(n))$ ούτε η σχέση $g(n) = O(f(n))$.

7. Δείξτε ότι $n! = o(n^n)$ και $n! = \omega(2^n)$.

8. Έστω n ακέραιος και x πραγματικός αριθμός. Δείξτε ότι

   (α′) $\lfloor x \rfloor < n$ αν και μόνο αν $x < n$.

   (β′) $n \leq \lfloor x \rfloor$ αν και μόνο αν $n \leq x$.

   (γ′) $\lceil x \rceil \leq n$ αν και μόνο αν $x \leq n$.

   (δ′) $n < \lceil x \rceil$ αν και μόνο αν $n < x$.

   (ε′) $\lfloor x \rfloor = n$ αν και μόνο αν $x - 1 < n \leq x$, και αν και μόνο αν $n \leq x < n + 1$.

   (ς′) $\lceil x \rceil = n$ αν και μόνο αν $x \leq n < x + 1$, και αν και μόνο αν $n - 1 < x \leq n$.

9. Δείξτε ότι $\lfloor -x \rfloor = -\lceil x \rceil$.

10. Θεωρήστε τον αλγόριθμο του Ευκλείδη για τον υπολογισμό του μέγιστου κοινού διαιρέτη δύο θετικών ακέρων m και n:

    Βήμα 1. Έστω r το υπόλοιπο της διαίρεσης του m με το n. (Έχουμε, βέβαια, $0 \leq r < n$.)

    Βήμα 2. Αν $r = 0$ τότε n είναι ο μέγιστος κοινός διαιρέτης και ο αλγόριθμος τερματίζεται.

    Βήμα 3. Θέτουμε $m \leftarrow n$, $n \leftarrow r$ και επιστρέφουμε στο **Βήμα 1**.

    Δείξτε ότι ο παραπάνω αλγόριθμος τερματίζει σε πεπερασμένο αριθμό βημάτων και η έξοδός του είναι όντως ο μέγιστος κοινός διαιρέτης του m και του n.

11. Ο αλγόριθμος της *επιλεκτικής ταξινόμησης* (selection sort) ταξινομεί τα στοιχεία του πίνακα $A[1\,..\,n]$ ως εξής: αρχικά βρίσκουμε το μικρότερο στοιχείο του A και το εναλλάσσουμε με το στοιχείο στη θέση $A[1]$. Στη συνέχεια βρίσκουμε το αμέσως μεγαλύτερο και το εναλλάσσουμε με το στοιχείο στη θέση $A[2]$. Η διαδικασία αυτή επαναλαμβάνεται για τα πρώτα $n-1$ στοιχεία του πίνακα. Γράψτε ψευδοκώδικα για αυτόν τον αλγόριθμο ταξινόμησης. Ποιά αναλλοίωτη συνθήκη διέπει τον αλγόριθμο; Προσδιορίστε τον χρόνο εκτέλεσης καλύτερης και χειρότερης περίπτωσης.

ΚΕΦΑΛΑΙΟ 2

Η τεχνική σχεδίασης διαίρει-και-κυρίευε. Λύση αναδρομικών σχέσεων

Υπάρχουν πολλές τεχνικές σχεδίασης αλγορίθμων. Η ενθετική ταξινόμηση, για παράδειγμα, ακολουθεί την *αυξητική* προσέγγιση: έχοντας ταξινομήσει τα στοιχεία $A[1\,..\,j-1]$ τοποθετούμε το στοιχείο $A[j]$ στη σωστή του θέση έτσι ώστε να πάρουμε έναν ταξινομημένο υποπίνακα $A[1\,..\,j]$.

Μια εναλλακτική προσέγγιση σχεδίασης είναι με χρήση της τεχνικής "διαίρει-και-κυρίευε" ή "διαίρει-και-βασίλευε". Ένας αλγόριθμος αυτής της κατηγορίας διαιρεί το πρόβλημα σε διάφορα υποπροβλήματα, παρόμοια με το αρχικό πρόβλημα αλλά μικρότερου μεγέθους, επιλύει αυτά τα προβλήματα αναδρομικά και στη συνέχεια συνδυάζει τις διάφορες λύσεις ώστε να συνθέσουν μια λύση του αρχικού προβλήματος. Έτσι, το μοντέλο "διαίρει-και-βασίλευε" περιλαμβάνει τρία βήματα σε κάθε επίπεδο αναδρομής:

Διαιρούμε το πρόβλημα σε διάφορα υποπροβλήματα

"**Κυριεύουμε**" τα υποπροβλήματα, επιλύοντας τα αναδρομικά, εκτός και αν είναι αρκετά μικρού μεγέθους έτσι ώστε να επιλύονται απευθείας.

Συνδυάζουμε τις λύσεις των υποπροβλημάτων ώστε να συνθέσουμε μια λύση του αρχικού προβλήματος.

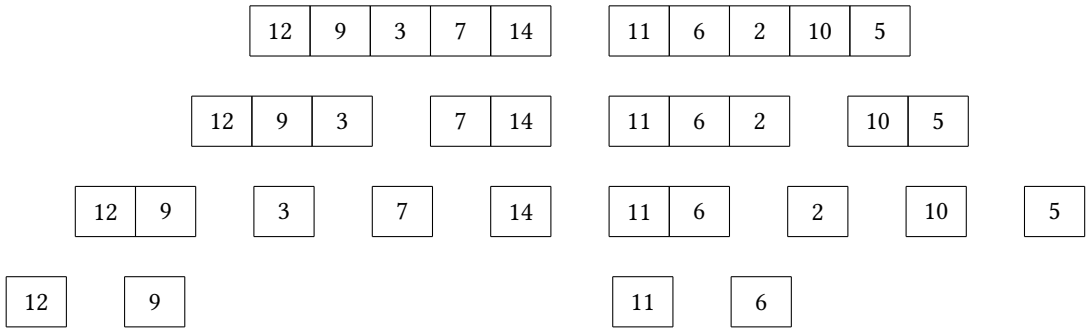
Ένας αλγόριθμος ταξινόμησης που ακολουθεί το μοντέλο "διαίρει-και-βασίλευε" είναι ο αλγόριθμος της *συγχωνευτικής ταξινόμησης* (merge sort), η λειτουργία του οποίου, για την ταξινόμηση μιας ακολουθίας n στοιχείων, έχει ως εξής:

- Διαιρούμε την ακολουθία n στοιχείων σε δύο υπακολουθίες $n/2$ στοιχείων η κάθε μία.
- Ταξινομούμε τις δύο υπακολουθίες αναδρομικά, χρησιμοποιώντας τον αλγόριθμο της συγχωνευτικής ταξινόμησης
- Συνδυάζουμε τις δυο ταξινομημένες υπακολουθίες ώστε να σχηματίσουμε την τελική ταξινομημένη ακολουθία.

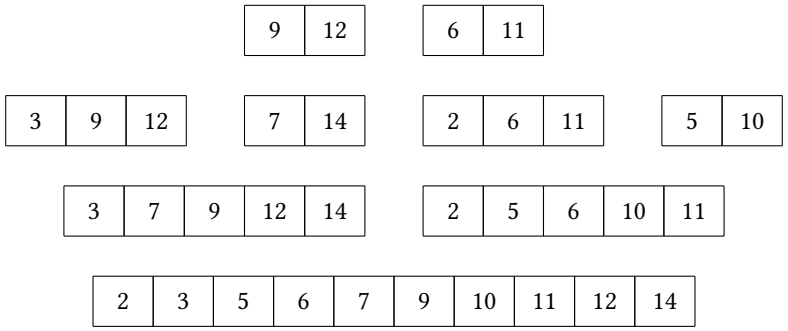
Η αναδρομή εξαντλείται όταν η ακολουθία προς ταξινόμηση έχει μήκος ένα, οπότε είναι ήδη τετριμμένα ταξινομημένη. Ας υποθέσουμε ότι θέλουμε να ταξινομήσουμε τα στοιχεία του υποπίνακα $A[p\,..\,r]$, για κάποιους δείκτες $1 \leq p \leq r \leq n$. Εφόσον $p < r$ (διαφορετικά, ο υποπίνακας είτε δεν περιέχει κανένα στοιχεία είτε περιέχει ακριβώς ένα, και επομένως είναι τετριμμένα ταξινομημένος) επιλέγουμε έναν δείκτη q μεταξύ των p και r και ταξινομούμε αναδρομικά τους υποπίνακες $A[p\,..\,q]$ και $A[q+1\,..\,r]$. Στη συνέχεια, συγχωνεύουμε τους δύο υποπίνακες $A[p\,..\,q]$ και $A[q+1\,..\,r]$ σε έναν ταξινομημένο υποπίνακα $A[p\,..\,r]$. Ας υποθέσουμε ότι η συγχώνευση των δύο υποπινάκων εκτελείται από τη διαδικασία MERGE(A, p, q, r), την οποία θα περιγράψουμε αργότερα. Τότε η διαδικασία της συγχωνευτικής ταξινόμησης είναι

MERGE-SORT(A, p, r)
  if $p < r$
    $q = \lfloor (p + r)/2 \rfloor$
    MERGE-SORT(A, p, q)
    MERGE-SORT$(A, q + 1, r)$
    MERGE(A, p, q, r)

Για να ταξινομήσουμε τον πίνακα $A[1\,..\,b]$ καλούμε τη διαδικασία MERGE-SORT$(A, 1, n)$. Για παράδειγμα, για την ταξινόμηση των στοιχείων του πίνακα $A = [12, 9, 3, 7, 14, 11, 6, 2, 10, 5]$ καλούμε τη διαδικασία MERGE-SORT$(A, 1, 10)$. Στο πρώτο βήμα υπολογίζουμε $q = \lfloor (1 + 10)/2 \rfloor = 5$ και ταξινομούμε αναδρομικά τους υποπίνακες $[1\,..\,5]$ και $A[6\,..\,10]$ με τις κλήσεις MERGE-SORT$(A, 1, 5)$ και MERGE-SORT$(A, 6, 10)$ οι οποίες με τη σειρά τους οδηγούν σε περισσότερες αναδρομικές κλήσεις της MERGE-SORT, μέχρι το μέγεθος των υποπινάκων να γίνει μικρότερο του δύο.



Στη συνέχεια, καλείται αναδρομικά η διαδικασία MERGE για να συγχωνεύσει τους υποπίνακες, όπως φαίνεται παρακάτω:



Μένει τώρα να δούμε πως γίνεται η διαδικασία της ταξινόμησης δύο ήδη ταξινομημένων υποπινάκων $A[p\,..\,q]$ και $A[q + 1\,..\,r]$ σε έναν ταξινομημένο υποπίνακα $A[p\,..\,r]$. Η ιδέα είναι να τοποθετήσουμε διαδοχικά στον πίνακα $A[p\,..\,r]$ το μικρότερο από τα στοιχεία των δύο υποπινάκων στις θέσεις p και $q + 1$, ακολουθούμενο από τα μικρότερο από τα στοιχεία στις θέσεις $p + 1$ και $q + 2$, μέχρι να τελειώσουν τα στοιχεία κάποιου υποπίνακα. Μετά απά τοποθετούμε διαδοχικά τα στοιχεία που μένουν από τον δεύτερο υποπίνακα. Παρατηρούμε ότι το βασικό βήμα αυτής της διαδικασίας είναι η σύγκριση δύο στοιχείων, και αυτή εκτελείται το πολύ $n = r - p + 1$ φορές. Συνεπώς η συγχώνευση απαιτεί χρόνο $\Theta(n)$.

Ο ψευδοκώδικας που ακολουθεί υλοποιεί αυτή τη διαδικασία συγχώνευσης με ένα επιπλέον τέχνασμα με το οποίο μπορούμε να αποφύγουμε την υποχρέωση να ελέγχουμε σε κάθε βήμα αν τα στοιχεία κάποιου από τους υποπίνακες που συγχωνεύουμε έχουν εξαντληθεί. Τοποθετούμε στο τέλος κάθε ενός από τους υποπίνακες που συγχωνεύουμε ένα στοιχείο–φρουρό, με τιμή μεγαλύτερη

από κάθε άλλο στοιχείο του πίνακα, και σταματάμε τη διαδικασία συγχώνευσης όταν έχουμε τοποθετήσει στον πίνακα $A[p\,..\,r]$ ακριβώς $r-p+1$ στοιχεία. Θα χρησιμοποιήσουμε το σύμβολο ∞ για να συμβολίσουμε αυτό το στοιχείο–φρουρό.

```
MERGE(A, p, q, r)
    n_1 = q − p + 1
    n_2 = r − q
    // Initialize arrays L[1 .. n_1 + 1] and R[1 .. n_2 + 1]
    for i = 1 to n_1
        L[i] = A[p + i − 1]
    for i = 1 to n_2
        R[i] = A[q + i]
    L[n_1 + 1] = ∞
    R[n_2 + 1] = ∞
    i = 1
    j = 1
    for k = p to r
        if L[i] ≤ R[j]
            A[k] = L[i]
            i = i + 1
        else
            A[k] = R[j]
            j = j + 1
```

Παρατηρούμε ότι οι δύο πρώτες επαναλήψεις **for** στη διαδικασία της συγχώνευσης απαιτούν συνολικό χρόνο $\Theta(n_1+n_2)=\Theta(n)$, ενώ ο τελευταίος βρόχος επαναλαμβάνεται n φορές, κάθε μια από τις οποίες απαιτεί σταθερό χρόνο. Για τον προσδιορισμό του χρόνο εκτέλεσης του αλγόριθμου της συγχωνευτικής ταξινόμησης θα υποθέσουμε αρχικά ότι ο αριθμός n των στοιχείων που ταξινομούνται είναι κάποια δύναμη του δύο, έτσι ώστε κάθε βήμα της διαίρεσης[1] να δίνει υποπίνακες μεγέθους ακριβώς $n/2$. Έστω $T(n)$ ο χρόνος εκτέλεσης του αλγόριθμου της συγχωνευτικής ταξινόμησης για ένα πρόβλημα μεγέθους n. Τότε

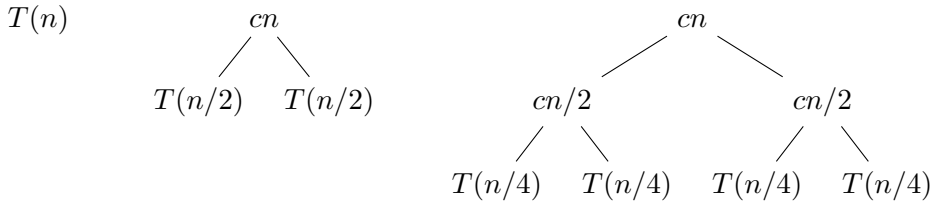
- Η διαδικασία της διαίρεσης παίρνει σταθερό χρόνο, είναι απά ο υπολογισμός του δείκτη q.
- Κάθε αναδρομική κλίση της διαδικασίας MERGE-SORT απαιτεί χρόνο ίσο με $T(n/2)$
- Η συγχώνευση των στοιχείων των δύο ταξινομημένων υποπινάκων απαιτεί χρόνο $\Theta(n)$

Συνεπώς, για τον χρόνο εκτέλεσης της συγχωνευτικής ταξινόμησης έχουμε

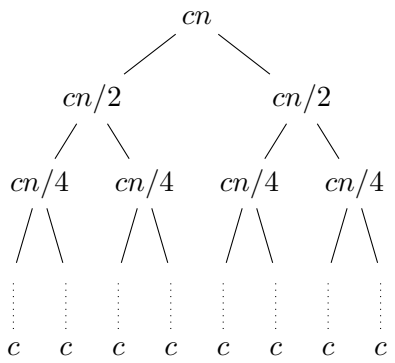
$$T(n)=\begin{cases} c & \text{αν } n=1,\\ 2T(n/2)+cn & \text{αν } n>1, \end{cases} \tag{2.1}$$

με τη σταθερά c να αντιπροσωπεύει τον χρόνο που απαιτείται για την επίλυση ενός προβλήματος μεγέθους ένα, καθώς και τον χρόνο της διαίρεσης και συγχώνευσης σε ένα ταξινομημένο πίνακα, ανά στοιχείο του πίνακα. Στο σχήμα που ακολουθεί, ο χρόνος εκτέλεσης $T(n)$ έχει αναπτυχθεί σε ένα ισοδύναμο δένδρο που αντιπροσωπεύει την αναδρομική σχέση (2.1) με ρίζα τον όρο cn και με τα υποδένδρα να αντιπροσωπεύουν τους δύο μικρότερους χρόνους $T(n/2)$, στο αμέσως επόμενο επίπεδο της αναδρομής. Το δένδρο αυτό με τη σειρά του έχει αναπτυχθεί σε ισοδύναμο υποδένδρα με τον όρο $cn/2$ να αντικαθιστά τους χρόνους $T(n/2)$.

[1] Το βήμα της διάιρεσης κατά την ταξινόμηση των στοιχείων $A[p\,..\,r]$ υπολογίζει μια θέση q η οποία διαμερίζει τον $A[p\,..\,r]$ σε δύο υποπίνακες $A[p\,..\,q]$ και $A[q+1\,..\,r]$ που περιέχουν $\lceil n/2\rceil$ και $\lfloor n/2\rfloor$ στοιχεία, αντίστοιχα.



Συνεχίζουμε αναπτύσσοντας τον κάθε κόμβο του δένδρου, μέχρις ότου να καταλήξουμε σε προβλήματα μεγέθους ένα, το καθένα από τα οποία έχει κόστος c.



Στη συνέχεια αθροίζουμε τα κόστη σε κάθε επίπεδο του δένδρου. Το υψηλότερο επίπεδο έχει συνολικό κόστος cn, το αμέσως επόμενο $cn/2+cn/2 = cn$, το αμέσως επόμενο $cn/4+cn/4+cn/4+cn/4 = cn$, κ.ο.κ. Εν γένει, το επίπεδο που απέχει i βαθμίδες από την κορυφή έχει 2^i κόμβους, καθένας από τους οποίους συνεισφέρει στο κόστος μια ποσότητα ίση με $cn/2^i$. Επομένως, αυτό το επίπεδο έχει συνολικό κόστος $2^i cn/2^i = cn$. Στο χαμηλότερο επίπεδο υπάρχουν n κόμβοι, καθένας από τους οποίους συνεισφέρει κόστος c, οπότε το συνολικό κόστος του επιπέδου αυτού είναι πάλι cn.

Δείχνουμε τώρα ότι ο αριθμός των επιπέδων σε αυτό το δέντρο είναι ακριβώς $\lg n + 1$. Όταν $n = 1$, το δένδρο έχει ένα επίπεδο και $\lg n + 1 = 1$. Δεχόμαστε τώρα ότι το πλήθος των επιπέδων σε ένα δένδρο με $n = 2^i$ καταληκτικούς κόμβους είναι $\lg 2^i + 1 = i + 1$. Τώρα, ένα δένδρο με 2^{i+1} καταληκτικούς κόμβους έχει ένα επίπεδο παραπάνω, και επομένως το ολικό πλήθος των επιπέδων είναι $(i + 1) + 1 = i + 2 = \lg 2^{i+1} + 1$, γεγονός που ολοκληρώνει την επαγωγική απόδειξη.

Δεδομένου ότι υπάρχουν $\lg n+1$ επίπεδα και καθένα από αυτά κοστίζει cn, το ολικό κόστος είναι

$$cn(\lg n + 1) = cn \lg n + cn.$$

Αν αγνοήσουμε τον όρο κατώτερης τάξης και τη σταθερά c, έχουμε ότι $T(n) = \Theta(n \lg n)$.

2.1 Μέθοδοι επίλυσης αναδρομικών σχέσεων

Εξετάζουμε τρεις μεθόδους επίλυσης αναδρομικών σχέσεων. Στη *μέθοδο της αντικατάστασης* εικάζουμε κάποιο φράγμα και αποδεικνύουμε την ορθότητα της εικασίας μας με τη μέθοδο της επαγωγής. Στη *μέθοδο του δένδρου αναδρομής* απεικονίζουμε την αναδρομική σχέση με τη μορφή ενός δένδρου όπου οι κόμβοι αντιπροσωπεύουν τα κόστη που υπεισέρχονται σε κάθε επίπεδο. Τέλος, η *κεντρική μέθοδος* παρέχει φράγματα για αναδρομικές σχέσεις της μορφής

$$T(n) = aT(n/b) + f(n), \tag{2.2}$$

όπου $a \geq 1$, $b > 1$, και $f(n)$ μια δεδομένη συνάρτηση.

2.1.1 Η μέθοδος της αντικατάστασης

Η μέθοδος της αντικατάστασης συνίσταται στην διατύπωση μιας εικασίας για τη λύση μιας αναδρομικής σχέσης και απόδειξή της με επαγωγή. Για παράδειγμα, για την αναδρομική σχέση

$$T(n) = 2T\left(\lfloor n/2 \rfloor\right) + n, \tag{2.3}$$

η οποία είναι παρόμοια με την αναδρομική σχέση για το χρόνο εκτέλεσης της συγχωνευτικής ταξινόμησης, εικάζουμε ότι $T(n) = O(n \lg n)$. Πρέπει λοιπόν να αποδείξουμε ότι υπάρχει σταθερά $c > 0$ και $n_0 \geq 1$ τέτοια ώστε $T(n) \leq cn \lg n$, για $n \geq n_0$. Πράγματι, αν το φράγμα αυτό ισχύει για τον δείκτη $\lfloor n/2 \rfloor$, αντικαθιστώντας στην αναδρομική σχέση (2.3) έχουμε

$$\begin{aligned} T(n) &\leq 2c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + n \\ &\leq cn \lg(n/2) + n \\ &= cn \lg n - cn + n \\ &\leq cn \lg n, \end{aligned}$$

υπό την προϋπόθεση ότι $c \geq 1$. Μια και η επαγωγική απόδειξη απαιτεί να δείξουμε ότι η λύση ισχύει και για τυχόν συνοριακές συνθήκες, παρατηρούμε ότι το φράγμα που αποδείξαμε συνεπάγεται $T(1) \leq 0$, σχέση η οποία μπορεί να είναι ασυμβίβαστη με όποια συνοριακή συνθήκη έχουμε. Σε αυτή την περίπτωση μπορούμε να θεωρήσουμε εναρκτήρια περίπτωση κάποιο $n_0 \geq 2$ και να επιλέξουμε κατάλληλα τη σταθερά c έτσι ώστε η συνοριακή συνθήκη να ικανοποιείται. Για παράδειγμα, η αναδρομική σχέση (2.3) συνοδεύεται από την συνοριακή συνθήκη $T(1) = 1$, τότε $T(2) = 4$ και $T(3) = 5$, οπότε αρκεί να επιλέξουμε τη σταθερά c αρκετά μεγάλη έτσι ώστε να ικανοποιούνται οι σχέσεις $T(2) \leq 2c \lg 2$ και $T(3) \leq 3c \lg 3$. Προφανώς, οποιαδήποτε $c \geq 2$ αρκεί για να ικανοποιούνται και οι δύο σχέσεις.

Παρατήρηση. Επειδή δεν υπάρχει γενικός τρόπος να εικάσει κανείς τη λύση μιας αναδρομικής σχέσης, είναι εύλογο, ότι αν αυτή μοιάζει με κάποια αναδρομική σχέση την οποία έχουμε ήδη λύσει, να υποθέσουμε μια παρόμοια λύση και για αυτήν, όπως κάναμε στο προηγούμενο παράδειγμα. Ας θεωρήσουμε, για παράδειγμα την αναδρομική σχέση

$$T(n) = T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + 1. \tag{2.4}$$

Αυτή μοιάζει με την ακριβή αναδρομική σχέση για τη συγχωνευτική ταξινόμηση, αλλά έχουμε αντικαταστήσει το κόστος της συγχώνευσης από τη μονάδα. Μια λογική εικασία για τη λύση αυτής της αναδρομικής σχέσης θα ήταν η $T(n) = O(n)$. Υποθέτοντας ότι $T(n) \leq cn$ για κάποια σταθερά c και αντικαθιστώντας στην αναδρομική σχέση (2.4) παίρνουμε

$$T(n) \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 = cn + 1,$$

απ' όπου όμως *δεν* συνεπάγεται ότι $T(n) \leq cn$ για καμία τιμή της c. Αν και σε ποιοτικό επίπεδο η εικασία μας είναι σωστή, απαιτείται να διορθώσουμε με έναν όρο κατώτερης τάξης. Αν λοιπόν υποθέσουμε ότι $T(n) \leq cn - b$, όπου $b \geq 0$ μια σταθερά, έχουμε

$$\begin{aligned} T(n) &\leq (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1 \\ &= cn - 2b + 1 \\ &\leq cn - b, \end{aligned}$$

υπό την προϋπόθεση ότι $b \geq 1$.

2.1.2 Αλλαγή μεταβλητών

Μερικές φορές, στοιχειώδεις αλγεβρικοί χειρισμοί μπορούν να ανάγουν την αναδρομική σχέση σε κάποια την οποία έχουμε ήδη λύσει. Για παράδειγμα, αν θεωρήσουμε τη σχέση

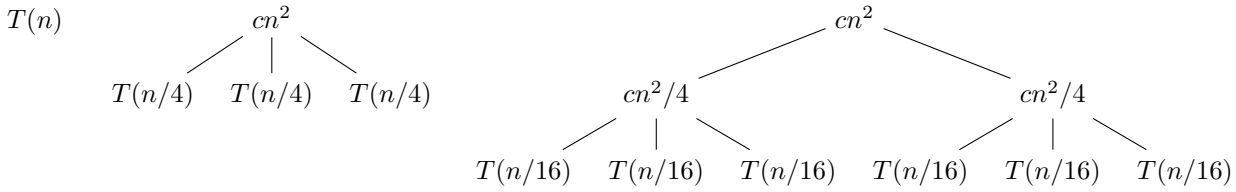
$$T(n) = 2T\left(\lfloor \sqrt{n} \rfloor\right) + \lg n,$$

με την αλλαγή μεταβλητής $m = \lg n$ έχουμε $T(2^m) = 2T(2^{m/2}) + m$. Θέτοντας $S(m) = T(2^m)$ παίρνουμε τη νέα αναδρομική σχέση $S(m) = 2S(m/2) + m$, που μοιάζει πολύ με αναδρομική σχέση που έχουμε ήδη λύσει. Αν λοιπόν $S(m) = O(m \lg m)$, τότε

$$T(n) = T(2^m) = S(m) = O(m \lg n) = O(\lg n \lg \lg n).$$

2.1.3 Η μέθοδος του δένδρου αναδρομής

Ένας καλός τρόπος να σχηματίσει κανείς μια εικασία για τη λύση μιας αναδρομικής σχέσης, είναι να σχεδιάσει το λεγόμενο *δένδρο αναδρομής*, όπου κάθε κόμβος αντιπροσωπεύει το κόστος ενός μόνο υποπροβλήματος από το σύνολο των αναδρομικών κλήσεων της συνάρτησης. Ας θεωρήσουμε, για παράδειγμα, την αναδρομική σχέση $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$. Μια και η συνάρτηση κατώφλι είναι επουδιώδης όσον αφορά την επίλυση αναδρομικών σχέσεων, κατασκευάζουμε ένα αναδρομικό δένδρο για την αναδρομική σχέση $T(n) = 3T(n/4) + cn^2$. Όπως και στη κατασκευή του αναδρομικού δένδρου για τη συγχωνευτική ταξινόμηση αναπτύσσουμε τον όρο $T(n)$ σε ισοδύναμο δένδρο που αντιπροσωπεύει την αναδρομική σχέση. Στο πρώτο βήμα το δένδρο έχει ρίζα τον όρο cn^2 και υποδένδρα να αντιπροσωπεύουν τους μικρότερους όρους $T(n/4)$. Στο επόμενο βήμα, κάθε ένας από τους όρους $T(n/4)$ αντικαθιστάται από ένα δέδνρο με ρίζα $c(n/4)^2$ και υποδένδρα να αντιπροσωπεύουν τους όρους $T(n/16)$.



Αφού το μέγεθος των υποπροβλημάτων ελαττώνεται καθώς απομακρυνόμαστε από τη ρίζα, θα πρέπει τελικά να καταλήξουμε σε κάποια συνοριακή συνθήκη. Το μέγεθος του υποπροβλήματος σε κάποιο κόμβο ο οποίος βρίσκεται σε βάθος i είναι $n/4^i$. Αυτό το μέγεθος γίνεται ίσο με τη μονάδα όταν $i = \log_4 n$, συνεπώς το δένδρο αναδρομής έχει συνολικά $\log_4 n + 1$ επίπεδα, τα $0, 1, \ldots, \log_4 n$.
Το πλήθος των κόμβων σε βάθος i είναι 3^i και το κόστος κάθε κόμβου είναι $c(n/4^i)^2$, συνεπώς το συνολικό κόστος των κόμβων στο επίπεδο i είναι ίσο με $3^i c\,(n/4^i)^2 = (3/16)^i cn^2$, για $i = 0, 1, \ldots, \log_4 n - 1$. Το τελευταίο επίπεδο έχει $3^{\log_4 n} = n^{\log_4 3}$ κόμβους και κάθε ένας από αυτούς έχει κόστος $T(1)$. Τότε, το κόστος ολόκληρου του δένδρου είναι

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) \\ &< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) \\ &= \frac{16}{13} cn^2 + \Theta\left(n^{\log_4 3}\right). \end{aligned}$$

Επειδή $\log_4 3 < 2$ καταλήγουμε στην εικασία $T(n) = O(n^2)$. Χρησιμοποιούμε τώρα τη μέθοδο της αντικατάστασης για να επαληθεύσουμε ότι η εικασία μας είναι σωστή. Αρκεί να δείξουμε ότι $T(n) \le dn^2$, για κάποια σταθερά $d > 0$. Πράγματι,

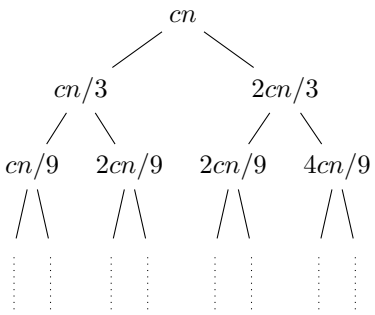
$$\begin{aligned} T(n) &\le 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\le 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\le 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16} dn^2 + cn^2 \\ &\le dn^2, \end{aligned}$$

υπό την προϋπόθεση ότι $d \ge (16/13)c$.

Για ένα ακόμα παράδειγμα, χρησιμοποιούμε τη μέθοδο του δένδρου αναδρομής για να σχηματίσουμε μια εικασία για τη λύση της αναδρομικής σχέσης

$$T(n) = T(n/3) + T(2n/3) + cn,$$

όπου $c > 0$ κάποια σταθερά και πάλι έχουμε παραλείψει τις συναρτήσεις ανώφλι και κατώφλι στον ορισμό της $T(n)$. Το δένδρο αναδρομής που αντιστοιχεί σε αυτή τη σχέση είναι



Το κόστος σε κάθε επίπεδο του δένδρου είναι cn. Για να υπολογίσουμε τον αριθμό των επιπέδων στο δένδρο αναδρομής σκεπτόμαστε ως εξής: η μακρύτερη διαδρομή από τη ρίζα μέχρι τους καταληκτικούς κόμβους είναι

$$n \to (2/3)n \to (2/3)^2 n \to \cdots \to 1.$$

Αφού $(2/3)^k n = 1$ όταν $k = \log_{3/2} n$, ο αριθμός των επιπέδων στο δένδρο αναδρομής είναι ακριβώς $\log_{3/2} n + 1$. Αν και το δένδρο αναδρομής δεν είναι ένα πλήρες δυαδικό δένδρο ύψους $\log_{3/2} n$, δηλαδή, έχει λιγότερους από $2^{\log_{3/2} n} = n^{log_{3/2} 2}$ κόμβους, είναι λογικό να περιμένουμε ότι το συνολικό κόστος θα είναι περίπου $cn \log_{3/2} n$, δηλαδή $O(n \lg n)$. Χρησιμοποιούμε τώρα τη μέθοδο της αντικατάστασης για να επαληθεύσουμε ότι η λύση της αναδρομικής σχέσης έχει άνω φράγμα $O(n \lg n)$, δηλαδή ότι $T(n) \leq dn \lg n$, όπου d κατάλληλη θετική σταθερά. Πράγματι,

$$\begin{aligned} T(n) &\leq T(n/3) + T(2n/3) + cn \\ &\leq d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn \\ &= dn \lg n - dn(\lg 3 - 2/3) + cn \\ &\leq dn \lg n, \end{aligned}$$

υπό την προϋπόθεση ότι $d \geq c/(\lg 3 - 2/3)$.

2.2 Η κεντρική μέθοδος

Η κεντρική μέθοδος είναι ένα συνταγολόγιο για τη λύση αναδρομικών σχέσεων της μορφής

$$T(n) = aT(n/b) + f(n), \tag{2.5}$$

όπου $a \geq 1$ και $b > 1$ σταθερές και $f(n)$ μια ασυμπτωτικά θετική συνάρτηση. Η αναδρομική σχέση (2.5) περιγράφει τον χρόνο εκτέλεσης ενός αλγόριθμου ο οποίος διαιρεί το δεδομένο πρόβλημα μεγέθους n σε a υποπροβλήματα, το καθένα μεγέθους n/b. Τα a το πλήθος υποπροβλήματα επιλύονται σναδρομικά, το καθένα σε χρόνο $T(n/b)$. Το κόστος της διαίρεσης του προβλήματος και του συνδυασμού των λύσεων των υποπροβλημάτων δίνεται από τη συνάρτηση $f(n)$. Όπως και πριν, όποτε γράφουμε αναδρομικές σχέσεις τύπου διαίρει-και-βασίλευε, κατά κανόνα παραλείπουμε τις συναρτήσεις ανωφλίου και κατωφλίου. Η κεντρική μέθοδος βασίζεται στο παρακάτω θεώρημα:

Θεώρημα (Κεντρικό θεώρημα). Έστω $a \geq 1$ και $b > 1$ σταθερές, $f(n)$ μια συνάρτηση και $T(n)$ μια συνάρτηση που δίνεται από την αναδρομική σχέση

$$T(n) = aT(n/b) + f(n).$$

Τότε το $T(n)$ μπορεί να φραγεί ασυμπτωτικά ως εξής:

1. Αν $f(n) = O(n^{\log_b a - \epsilon})$ για κάποια σταθερά $\epsilon > 0$, τότε ισχύει ότι $T(n) = \Theta(n^{\log_b a})$.
2. Αν $f(n) = \Theta(n^{\log_b a})$, τότε $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. Αν $f(n) = \Omega(n^{\log_b a + \epsilon})$ για κάποια σταθερά $\epsilon > 0$, και αν $af(n/b) \leq cf(n)$ για κάποια σταθερά $c < 1$ και για όλα τα n αρκετά μεγάλα, τότε $f(n) = \Theta(f(n))$.

Παρατηρήσεις. Στο κεντρικό θεώρημα συγκρίνουμε τη συνάρτηση $f(n)$ με τη συνάρτηση $n^{\log_b a}$. Αν μεγαλύτερη είναι η συνάρτηση $n^{\log_b a}$ (περίπτωση 1), τότε η λύση είναι η $T(n) = \Theta(n^{\log_b a})$. Αν μεγαλύτερη είναι η $f(n)$ (περίπτωση 3), τότε η λύση είναι η $T(n) = \Theta(f(n))$. Τέλος, αν οι δύο συναρτήσεις είναι ισομεγέθεις (περίπτωση 2), τότε η λύση είναι $T(n) = \Theta(n^{\log_b a} \lg n)$.

Παραδείγματα.

- Για την αναδρομική σχέση $T(n) = 9T(n/3) + n$ έχουμε $a = 9$, $b = 3$ και $f(n) = n$. Επομένως, $n^{\log_b a} = n^{\log_3 9} = n^2$. Μπορούμε λοιπόν να εφαρμόσουμε την περίπτωση 1 του κεντρικού θεωρήματος έτσι ώστε $T(n) = \Theta(n^2)$.
- Για την αναδρομική σχέση $T(n) = T(2n/3) + 1$ έχουμε $a = 1, b = 3/2$ και $f(n) = 1$. Τώρα $n^{\log_b a} = 1$ επόμένως από την περίπτωση 2 του κεντρικού θεωρήματος έχουμε $T(n) = \Theta(\lg n)$.
- Για την αναδρομική σχέση $T(n) = 3T(n/4) + n \lg n$ έχουμε $a = 3$, $b = 4$ και $f(n) = n \lg n$. Επομένως, $n^{\log_b a} = n^{\log_4 3} \approx n^{0.793}$. Αφού $f(n) = \Omega(n^{\log_4 3 + \epsilon}$, όπου $\epsilon \approx 0.2$, είμαστε την περίπτωση 3 του κεντρικού θεωρήματος. Η σχέση $3f(n/4) \leq cf(n)$ ισχύει με $c = 3/4$, επομένως η λύση της αναδρομικής σχέσης είναι $T(n) = \Theta(n \lg n)$.
- Για την αναδρομική σχέση $T(n) = 2T(n/2) + n \lg n$ έχουμε $a = b = 2$ και $f(n) = n \lg n$. Τότε $n^{\log_b a} = n$. Αν και η $f(n)$ είναι ασυμπτωτικά μεγαλύτερη από την n, δεν είναι *πολυωνυμικά μεγαλύτερη* και συνεπώς δεν μπορούμε να εφαρμόσουμε το κεντρικό θεώρημα.

2.3 Ασκήσεις

1. Δείξτε ότι η λύση της αναδρομικής σχέσης $T(n) = T(\lceil n/2 \rceil) + 1$ είναι $O(\lg n)$.
2. Δείξτε ότι η λύση της αναδρομικής σχέσης $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ είναι $O(n \lg n)$.
3. Χρησιμοποιήστε τη μέθοδο της αλλαγής μεταβλητής για να λύσετε την αναδρομική σχέση $T(n) = 2T(\sqrt{n}) + 1$.
4. Προσδιορίστε ένα άνω φράγμα για τη λύση της αναδρομικής σχέσης $T(n) = 3T(\lceil n/2 \rceil) + n$, χρησιμοποιώντας το αντίστοιχο δένδρο αναδρομής. Επαληθεύστε την απάντησή σας με τη μέθοδο της αντικατάστασης.
5. Κατασκευάστε το δένδρο αναδρομής για την αναδρομική σχέση $T(n) = 4T(\lceil n/2 \rceil) + cn$, όπου c μια σταθερά. Επαληθεύστε με τη μέθοδο της αντικατάστασης.
6. Χρησιμοποιήστε το κεντρικό θεώρημα, όπου είναι δυνατόν, για να προσδιορίσετε αυστηρά ασυμπτωτικά φράγματα για τις παρακάτω αναδρομικές σχέσεις

   (α′) $T(n) = 4T(n/2) + n$
   (β′) $T(n) = 4T(n/2) + n^2$
   (γ′) $T(n) = 4T(n/2) + n^3$
   (δ′) $T(n) = 4T(n/2) + n^2 \lg n$

7. Χρησιμοποιήστε το κεντρικό θεώρημα, όπου είναι δυνατόν, για να προσδιορίσετε αυστηρά ασυμπτωτικά φράγματα για τις παρακάτω αναδρομικές σχέσεις

   (α′) $T(n) = 2T(n/2) + n^3$
   (β′) $T(n) = T(9n/10) + n$
   (γ′) $T(n) = 7T(n/3) + n^2$
   (δ′) $T(n) = 2T(n/4) + \sqrt{n}$
   (ε′) $T(n) = 3T(n/2) + n \lg n$
   (ς′) $T(n) = 2T(n/2) + n/\lg n$

ΚΕΦΑΛΑΙΟ 3

Η ταχυταξινόμηση

Η *ταχυταξινόμηση* (*quicksort*) είναι ένας αλγόριθμος ταξινόμησης με αναμενόμενο χρόνο εκτέλεσης $\Theta(n \lg n)$, χρόνο εκτέλεσης χειρότερης περίπτωσης $O(n^2)$, με το επιπλέον πλεονέκτημα να είναι επιτόπιος, δηλαδή, τα στοιχεία προς ταξινόμηση να αναδιατάσσονται εντός της συστοιχίας εισόδου. Όπως και ο αλγόριθμος της συγχωνευτικής ταξινόμησης, η ταχυταξινόμηση βασίζεται στην τεχνική σχεδίασης διαίρει-και-κυρίευε. Για την ταξινόμηση της ακολουθίας στοιχείων $A[p\,..\,r]$ ακολουθούμε τα εξης βήματα:

- Διαμερίζουμε τη συστοιχία $A[p\,..\,r]$ σε δύο μικρότερες υποσυστοιχίες $A[p\,..\,q-1]$ και $A[q+1\,..\,r]$ έτσι ώστε κάθε στοιχείο της $A[p\,..\,r]$ να είναι μικρότερο ή ίσο του $A[q]$ το οποίο με τη σειρά του να είναι μικρότερο ή ίσο κάθε στοιχείου της $A[q+1\,..\,r]$.
- Ταξινομούμε τις υποσυστοιχίες $A[p\,..\,q-1]$ και $A[q+1\,..\,r]$ με αναδρομικές κλήσεις της διαδικασίας της ταξινόμησης

Αφού οι υποσυστοιχίες ταξινομούνται επιτόπου, δεν χρειάζεται καμία επιπλέον εργασία κατά τον συνδυασμό τους. Η συστοιχία $A[p\,..\,r]$ είναι ήδη ταξινομημένη. Η υλοποίησή της φαίνεται παρακάτω:

```
QUICKSORT(A, p, r)
    if p < r
        q = PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
```

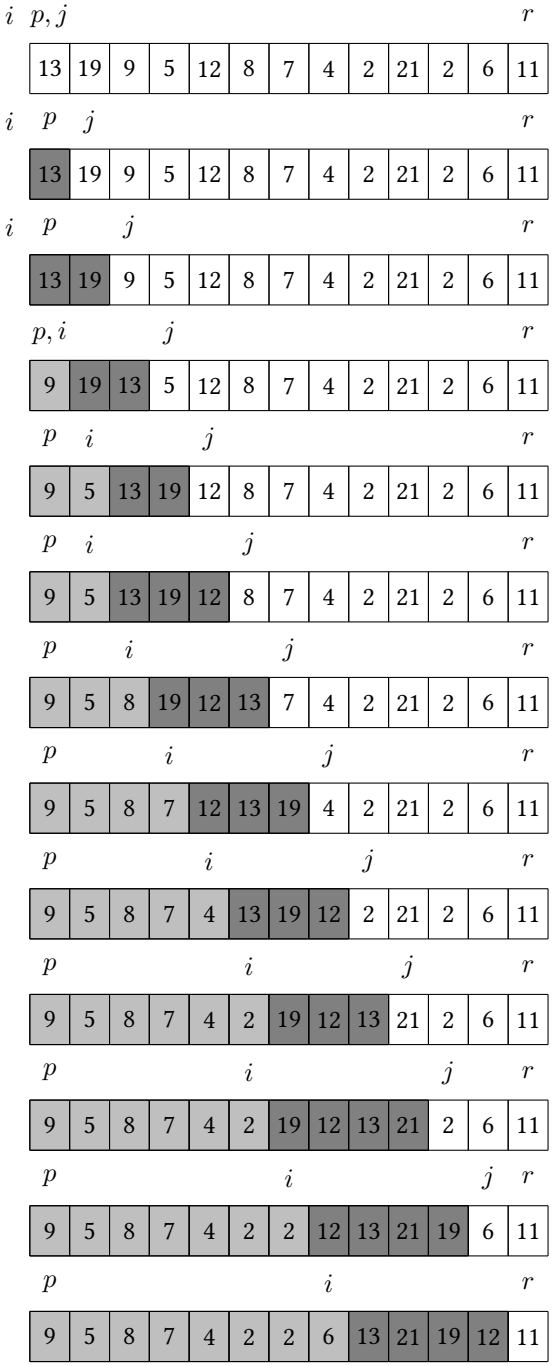
Η αρχική κλήση για την ταξινόμηση της συστοιχίας $A[1\,..\,n]$ είναι QUICKSORT$(A, 1, n)$. Το βασικό συστατικό της αλγόριθμου της ταχυταξινόμησης είναι η διαδικασία της διαμέρισης της συστοιχίας $A[p\,..\,r]$:

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4       if A[j] ≤ x
5            i = i + 1
6            exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

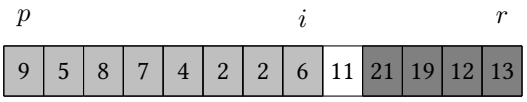
Η διαδικασία PARTITION επιλέγει το στοιχείο $x = A[r]$ ως *οδηγό* (*pivot*), δηλαδή, ως το στοιχείο γύρω από το οποίο θα διαμεριστεί η υποσυστοιχία $A[p\,..\,r]$. Καθώς εκτελείται η διαδικασία PARTITION η συστοιχία διαμερίζεται σε τέσσερις (πιθανόν κενές) περιοχές οι οποίες ικανοποιούν τις παρακάτω συνθήκες και θα μπορούσαν να διατυπωθούν ως μια αναλλοίωτη συνθήκη βρόχου (οι απαιτήσεις 1, 2 και 4, μόνο):

1. Αν $p \leq k \leq i$, τότε $A[k] \leq x$.
2. Αν $i+1 \leq k \leq j-1$, τότε $A[k] > x$.
3. Τα στοιχεία στις θέσεις $j \leq k \leq r-1$ δεν έχουν εξεταστεί ακόμα
4. Αν $k = r$ τότε $A[k] = x$.

Παρακάτω φαίνεται σχηματικά η λειτουργία PARTITION για μια συστοιχία με στοιχεία 13, 19, 9, 5, 12, 8, 7, 4, 2, 21, 2, 6, 11. Τα ελαφρά σκιασμένα στοιχεία είναι αυτά που ανήκουν στην πρώτη περιοχή, δηλαδή, έχουν τιμές μικρότερες από αυτή του οδηγού, ενώ τα έντονα σκιασμένα στοιχεία είναι αυτά που ακήνουν στη δεύτερη περιοχή.



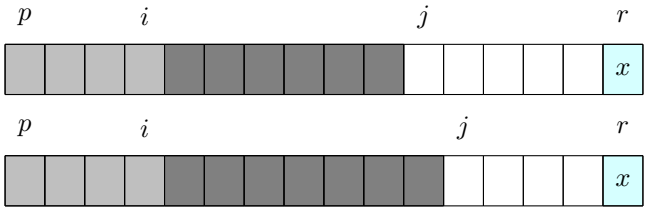
Η τελευταία πράξη της PARTITION είναι εναλλαγή των στοιχείων $A[i+1]$ και $A[r]$ και η επιστροφή του δείκτη $i+1$ ως τον δείκτη του οδηγού:



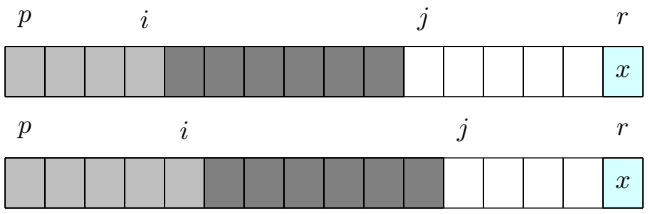
Δείχνουμε τώρα την ορθότητα του αλγόριθμου της διαμέρισης χρησιμοποιώντας την αναλλοίωτη συνθήκη. Η διαδικασία PARTITION τηρεί τέσσερεις περιοχές στην υποσυστοιχία $A[p\,..\,r]$, όπως φαίνεται σχηματικά παρακάτω: Τα στοιχεία στην περιοχή $A[p\,..\,i]$ είναι όλα μικρότερα ή ίσα του x, τα στοιχεία της περιοχής $A[i+1\,..\,j-1]$ είναι μεγαλύτερα του x, και $A[r]=x$. Τα στοιχεία της περιοχής $A[j\,..\,r-1]$ μπορεί να έχουν οποιαδήποτε τιμή.

Πριν από την πρώτη επανάληψη του βρόχου, έχουμε $i=p-1$ και $j=p$, επομένως δεν υπάρχει αριθμός μεταξύ των p και i, αλλά ούτε και μεταξύ των $i+1$ και $j-1$. Επομένως οι δύο πρώτες απαιτήσεις της αναλλοίωτης συνθήκης ικανοποιούνται κατά τετριμμένο τρόπο. Η τρίτη απαίτηση εκπληρώνεται από την ανάθεση στη γραμμή 1 της διαδικασίας QUICKSORT.

Δείχνουμε τώρα ότι η αναλλοίωτη συνθήκη διατηρείται σε κάθε επανάληψη. Όταν $A[j]>x$ η μόνη ενέργεια που πραγματοποιείται είναι η αύξηση του j κατά ένα.



Επομένως η απαίτηση 2 και πάλι ικανοποιείται, ενώ τα άλλα στοιχεία της συστοιχίας παραμένουν αμετάβλητα. Αν τώρα $A[j]\leq x$ τότε αυξάνεται το i κατά ένα, εναλλάσσουν θέσεις τα στοιχεία $A[i]$ και $A[j]$, και στη συνέχεια αυξάνεται το j κατά ένα.



Λόγω αυτής της εναλλαγής έχουμε $A[i]\leq x$, και επομένως η απαίτηση 1 ικανοποιείται. Επίσης έχουμε $A[j-1]>x$, αφού το στοιχείο που μετακινήθηκε στη θέση $A[j-1]$ είναι μεγαλύτερο του x. Αυτό δείχνει ότι η απαίτηση 2 ικανοποιείται επίσης.

Τέλος, μετά τον τερματισμό των επαναλήψεων, έχουμε $j=r$ και κάθε στοιχεία της συστοιχίας ανήκει σε μια από τις τρεις περιοχές που περιγράφονται στην αναλλοίωτη συνθήκη. Σημειώνουμε τέλος, ότι οι δύο τελευταίες γραμμές στη διαδικασίας PARTITION μετακινούν το στοιχείο–οδηγό εναλλάσοντάς το με το πρώτο στοιχείο, το οποίο είναι μεγαλύτερο του x.

Παρατήρηση. Ο αλγόριθμος της ταχυταξινόμησης, όπως διατυπώθηκε, εμπεριέχει δύο αναδρομικές κλήσεις στον εαυτό του. Η δεύτερη αναδρομική κλήση μπορεί στην πραγματικότητα να αποφευχθεί μέσω μιας επαναληπτικής δομής ελέγχου, όπως φαίνεται παρακάτω:

QUICKSORT2(A, p, r)
  while $p < r$
    $q =$ PARTITION(A, p, r)
    QUICKSORT2$(A, p, q-1)$
    $p = q + 1$

Όπως βλέπουμε, τόσο η QUICKSORT όσο και η QUICKSORT2 βασίζονται στην ίδια διαδικασία διαμέρισης και στη συνέχεια καλούν αναδρομικά τον εαυτό τους με ορίσματα $A, p, q-1$. Στη συνέχεια, η QUICKSORT καλεί αναδρομικά τον εαυτό με ορίσματα $A, q+1, r$ ενώ η QUICKSORT2 θέτει $p = q+1$ και εκτελεί ξανά μια επανάληψη του βρόχου **while** . Η επανάληψη αυτή εκτελεί ακριβώς τις ίδιες πράξεις όπως η αναδρομική κλήση με ορίσματα $A, q+1, r$.

3.0.1 Επίδοση της ταχυταξινόμησης

Είναι προφανές ότι ο χρόνος εκτέλεσης της ταχυταξινόμησης εξαρτάται από το κατά πόσο η διαμέριση της συστοιχίας είναι ισομερής, και αυτό με τη σειρά του εξαρτάται από την επιλογή του οδηγού. Θα δείξουμε ότι η χειρότερη περίπτωση προκύπτει όταν η διαδικασία PARTITION οδηγεί σε δύο υποπροβλήματα μεγέθους $n-1$ και 0, αντίστοιχα. Ας υποθέσουμε ότι αυτή η διαμέριση προκύπτει σε κάθε αναδρομική κλήση. Γνωρίζουμε ('Ασκηση 2) ότι το κόστος της διαμέρισης είναι $\Theta(n)$. Δεδομένου του ότι το κόστος μιας κλήσης της διαδικασίας ταχυταξινόμησης για ένα πρόβλημα μηδενικού μεγέθους είναι $\Theta(1)$, έχουμε την αναδρομική σχέση για τον χρόνο εκτέλεσης

$$T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n).$$

Μπορούμε να δείξουμε εύκολα ότι η λύση αυτής της αναδρομικής σχέσης είναι $T(n) = \Theta(n^2)$ ('Ασκηση 3). Αυτό σημαίνει ότι για τη δεδομένη ανισομερή διαμέριση, η ταχυταξινομήση δεν υπερτερεί ούτε καν της ενθετικής ταξινόμησης. Ακόμα, ο χρόνος $\Theta(n^2)$ αφορά και στην περίπτωση που η συστοιχία εισόδου είναι ήδη ταξινομημένη, περίπτωση για την οποία η ενθετική ταξινόμηση εκτελείται σε χρόνο $O(n)$.

Στον πλέον ισόρροπο διαχωρισμό, η διαδικασία PARTITION παράγει δύο υποπροβλήματα μεγέθους $\lfloor n/2 \rfloor$ και $\lceil n/2 \rceil - 1$, αντίστοιχα, μεγέθη μικρότερα από $n/2$. Σε αυτή την περίπτωση η αναδρομική σχέση για τον χρόνο εκτέλεσης είναι

$$T(n) \leq 2T(n/2) + \Theta(n).$$

Από την περίπτωση 2 του κεντρικού θεωρήματος παίρνουμε τη λύση $T(n) = O(n \lg n)$, δηλαδή έναν ασυμπτωτικά ταχύτερο αλγόριθμο. Μπορεί να αποδειχθεί ότι ο χρόνος εκτέλεσης της μέσης περίπτωσης προσεγγίζει πολύ περισσότερο την καλύτερη περίπτωση παρά τη χειρότερη. Ειδικότερα, οποιοσδήποτε διαχωρισμός με *σταθερή* αναλογία δίνει χρόνο εκτέλεσης $O(n \lg n)$. Αν, για παράδειγμα, υποθέσουμε ότι ο αλγόριθμος διαμέρισης δίνει πάντοτε υποπροβλήματα με αναλογία μεγεθών 9 προς 1, τότε ο χρόνος εκτέλεσης της ταχυταξινόμησης περιγράφεται από την αναδρομική σχέση

$$T(n) \leq T(9n/10) + T(n/10) + cn,$$

με τη σταθερά c αυτή που εμπεριέχεται στον όρο $\Theta(n)$. Στο δένδρο αναδρομής για αυτή τη σχέση βλέπουμε ότι κάθε επίπεδο του δένδρου έχει κόστος cn και η αναδρομή τερματίζεται σε βάθος $\log_{10/9} n = \Theta(\lg n)$. Επομένως το συνολικό κόστος είναι $O(n \lg n)$.

Δείχνουμε τέλος, ότι ο χρόνος εκτέλεσης της χειρότερης περίπτωσης του αλγόριθμου της ταχυταξινόμησης είναι $O(n^2)$. Αν $T(n)$ είναι ο χρόνος εκτέλεσης της χειρότερης περίπτωσης τότε θα πρέπει

$$T(n) \leq \max_{0 \leq q \leq n-1} (T(q) + T(n-q-1)) + \Theta(n), \tag{3.1}$$

με την παράμετρο q να μεταβάλλεται μεταξύ 0 και $n-1$ διότι η διαδικασίας PARTITION παράγει δύο υποπροβλήματα συνολικού μεγέθους $n-1$. Εικάζουμε ότι ισχύει $T(n) \leq cn^2$ για κάποια σταθερά c. Αντικαθιστώντας στην αναδρομική σχέση (3.1) έχουμε

$$\begin{aligned} T(n) &\leq \max_{0 \leq q \leq n-1} \left(cq^2 + c(n-q-1)^2\right) + \Theta(n) \\ &\leq c \max_{0 \leq q \leq n-1} \left(q^2 + (n-q-1)^2\right) + \Theta(n). \end{aligned}$$

Η ποσότητα $q^2 + (n-q-1)^2$ λαμβάνει τη μέγιστη τιμή της (στο διάστημα $0 \leq q \leq n-1$ όταν $q = n-1$ και αυτή είναι ίση με $(n-1)^2$. Τότε,

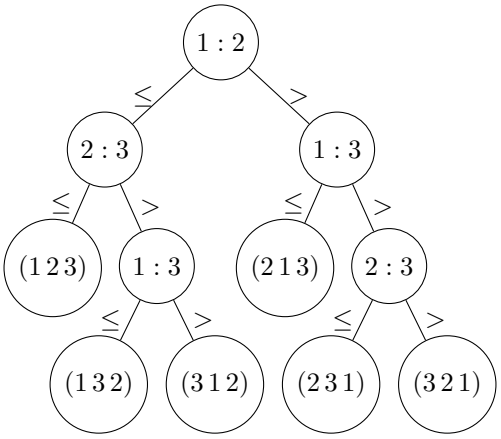
$$\begin{aligned} T(n) &\leq cn^2 - c(2n-1) + \Theta(n) \\ &\leq cn^2, \end{aligned}$$

επιλέγοντας τη σταθερά c αρκετά μεγάλη έτσι ώστε να επικρατεί του όρου $\Theta(n)$. Συνεπώς, $T(n) = O(n^2)$. Έχουμε ήδη δει ότι στην περίπτωση μιας ειδικής ανισομερούς διαμέρισης η ταχυταξινόμηση απαιτεί χρόνο $\Omega(n^2)$. Επομένως ο χρόνος εκτέλεσης χειρότερης περίπτωσης της ταχυταξινόμησης είναι $\Theta(n^2)$.

3.1 Κάτω φράγματα αλγόριθμων ταξινόμησης

Όλοι οι αλγόριθμοι ταξινόμησης που μελετήσαμε έχουν ως κοινό χαρακτηριστικό ότι η ταξινομημένη διάταξη που παράγουν βασίζεται μόνο σε συγκρίσεις των στοιχείων εισόδου. Αυτού του είδους οι αλγόριθμοι ταξινόμησης ονομάζονται αλγόριθμοι *συγκριτικής ταξινόμησης* (*comparison sort algorithms*).

Οι αλγόριθμοι συγκριτικής ταξινόμησης μπορούν να αναπαρασταθούν σε αφηρημένο επίπεδο μέσω *δένδρων αποφάσεων*. Ένα δένδρο αποφάσεων είναι ένα πλήρες δυαδικό δένδρο το οποίο αντιπροσωπεύει τις συγκρίσεις που πραγματοποιούνται μεταξύ στοιχείων από κάποιο αλγόριθμο ταξινόμησης. Αν υποθέσουμε ότι όλα τα στοιχεία εισόδου είναι διαφορετικά μεταξύ τους, συγκρίσεις της μορφής $a_i = a_j$ είναι περιττές και μπορούμε να υποθέσουμε ότι δεν εκτελούνται. Οι συγκρίσεις όμως $a_i \leq a_j$, $a_i \geq a_j$, $a_i < a_j$ και $a_j < a_i$ είναι όλες ισοδύναμες μεταξύ τους αφού παρέχουν την ίδια ακριβώς πληροφορία σχετικά με τη διάταξη των στοιχείων a_i και a_j. Μπορούμε επομένως να υποθέσουμε ότι όλες οι συγκρίσεις είναι της μορφής $a_i \leq a_j$. Το σχήμα που ακολουθεί απεικονίζει το δένδρο αποφάσεων για τον αλγόριθμο της ενθετικής ταξινόμησης για την ακολουθία στοιχείων $a_1 = 6$, $a_2 = 8$ και $a_3 = 5$. Κόμβοι με ετικέτες της μορφής $i : j$ υποδεικνύουν μια σύγκριση μεταξύ των a_i και a_j. Οι καταληκτικοί κόμβοι αναφέρουν τη διάταξη των στοιχείων εισόδου. Είναι προφανές ότι αφού υπάρχουν $3! = 6$ δυνατές μεταθέσεις των στοιχείων εισόδου, το δένδρο αποφάσεων θα πρέπει να έχει τουλάχιστον 6 καταληκτικούς κόμβους.



Γενικότερα, κάθε ορθός αλγόριθμος συγκριτικής ταξινόμησης θα πρέπει να έχει τη δυνατότητα να παράγει οποιαδήποτε μετάθεση από τις $n!$ μεταθέσεις των n στοιχείων εισόδου. Κάθε μια από αυτές τις μεταθέσεις θα πρέπει να εμφανίζεται σε έναν από τους καταληκτικούς κόμβους του δένδρου αποφάσεων και κάθε τέτοιος κόμβος θα πρέπει να είναι προσβάσιμος από τη ρίζα μέσω κάποιας διαδρομής.

Το ύψος του δένδρου αποφάσεων είναι το πλήθος των συγκρίσεων στη χειρότερη περίπτωση για έναν αλγόριθμο συγκριτικής ταξινόμησης. Επομένως ένα κάτω φράγμα για τα ύψη των δένδρων αποφάσεων στα οποία οποιαδήποτε μετάθεση εμφανίζεται σε κάποιον καταληκτικό κόμβο αντιπροσωπεύει ένα κάτω φράγμα για τον χρόνο εκτέλεσης οποιουδήποτε αλγόριθμου συγκριτικής ταξινόμησης. Ένα τέτοιο φράγμα αποδεικνύεται στο παρακάτω θεώρημα.

Θεώρημα. Οποιοσδήποτε αλγόριθμος συγκριτικής ταξινόμησης απαιτεί $\Omega(n \lg n)$ συγκρίσεις στη χειρότερη περίπτωση.

Απόδειξη. Έστω ένα δένδρο αποφάσεων με ύψος h και με l καταληκτικούς κόμβους. Πρέπει $l \leq n!$, αφού κάθε μια από τις $n!$ μεταθέσεις της εισόδου εμφανίζεται σε κάποιον καταληκτικό κόμβο. Γνωρίζουμε ότι ένα δυαδικό δένδρο με ύψος h έχει το πολύ 2^h καταληκτικούς κόμβους, συνεπώς

$$n! \leq l \leq 2^h.$$

Παίρνοντας λογάριθμους έχουμε ότι $h \geq \lg(n!)$. Δείχνουμε τώρα ότι $\lg(n!) = \Omega(n \lg n)$. Πράγματι, από τον τύπο του Stirling έχουμε ότι

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n},$$

όπου

$$\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}.$$

Παίρνοντας λογάριθμους έχουμε

$$\begin{aligned}\lg(n!) &= n\lg\left(\frac{n}{e}\right) + \frac{1}{2}\lg n + \frac{1}{2}\lg(2\pi) + (\lg e)\alpha_n \\ &\geq n\lg\left(\frac{n}{e}\right) \\ &\geq cn\lg n,\end{aligned}$$

για κάποια θετική σταθερά c. □

Ως πόρισμα του παραπάνω θεωρήματος προκύπτει το γεγονός ότι τόσο η ταξινόμηση σωρού όσο και η συγχωνευτική ταξινόμηση είναι ασυμπτωτικά βέλτιστοι αλγόριθμοι συγκριτικής ταξινόμησης.

3.2 Ασκήσεις

1. Περιγράψτε τη λειτουργία της διαδικασίας PARTITION$(A, 1, 8)$ για τη συστοιχία A με στοιχεία 2, 8, 7, 1, 3, 5, 6, 4.

2. Δικαιολογήστε σχηματικά γιατί ο χρόνος εκτέλεσης της διαδικασίας PARTITION(A, p, r) είναι $\Theta(n)$, με $n = r - p + 1$.

3. Δείξτε, με τη μέθοδο της αντικατάστασης ότι η αναδρομική σχέση $T(n) = T(n-1) + \Theta(n)$ έχει λύση $T(n) = \Theta(n^2)$. *Υπόδειξη.* Από τον ορισμό του συμβολισμού Θ, υπάρχουν σταθερές c_1, c_2 τέτοιες ώστε όρος $\Theta(n)$ να βρίσκεται μεταξύ των $c_1 n$ και $c_2 n$. Κάντε την επαγωγική υπόθεση $c_1 k^2 \leq T(k) \leq c_2 k^2$ για όλα τα $k < n$.

4. Ποιός είναι ο χρόνος εκτέλεσης της ταχυταξινόμησης όταν όλα τα στοιχεία της συστοιχίας έχουν την ίδια τιμή;

5. Ο πρωτότυπος αλγόριθμος της QUICKSORT (C.A.R. Hoare, 1959) χρησιμοποιούσε μια διαφορετική τεχνική διαμέρισης η οποία περιγράφεται στον παρακάτω ψευδοκώδικα

```
HOARE(A, p, r)
 1  x = A[p]
 2  i = p − 1
 3  j = r + 1
 4  while TRUE
 5      repeat
 6          j = j − 1
 7      until A[j] ≤ x
 8      repeat
 9          i = i + 1
10      until A[i] ≥ x
11      if i < j
12          exchange A[i] ↔ A[j]
13      else
14          return j
```

Παρατηρήστε ότι τώρα ο οδηγός, ο οποίος αρχικά βρίσκεται στη θέση $A[p]$, τοποθετείται πάντοτε σε μια από τις δύο περιοχές $A[p\,..\,j]$ και $A[j+1\,..\,r]$. Δεδομένου του ότι $p \leq j < r$, ο διαχωρισμός αυτός είναι πάντοτε μη τετριμμένος.

(α′) Περιγράψτε τη λειτουργία της διαδικασίας Hoare για τη συστοιχία με στοιχεία 13, 19, 9, 5, 12, 8, 7, 4, 2, 6, 21 καταγράφοντας τις τιμές της συστοιχίας και των βοηθητικών μεταβλητών μετά από κάθε επανάληψη του βρόχου στη γραμμή 4.

(β′) Ξαναγράψτε τη διαδικασία Quicksort με βάση τη διαμέριση Hoare.

ΚΕΦΑΛΑΙΟ 4

Δυναμικός προγραμματισμός

Ο *δυναμικός προγραμματισμός* (*dynamic programming*) είναι μια μέθοδος επίλυσης προβλημάτων μέσω του συνδυασμού λύσεων υποπροβλημάτων, όπως η μέθοδος διαίρει-και-κυρίευε. Σε αντίθεση όμως με αυτή, ο δυναμικός προγραμματισμός είναι εφαρμόσιμος όταν τα διάφορα υποπροβλήματα δεν είναι ανεξάρτητα μεταξύ τους, δηλαδή όταν τα ίδια έχουν κοινά υποπροβλήματα. Ένας αλγόριθμος δυναμικού προγραμματισμού επιλύει το κάθε (κοινό) υποπρόβλημα μόνο μια φορά και αποθηκεύει τη λύση σε έναν πίνακα, αποφεύγοντας τον εκ νέου υπολογισμό της λύσης κάθε φορά που απαντά το συγκεκριμένο υποπρόβλημα. Η λέξη "προγραμματισμός" αναφέρεται σε αυτήν ακριβώς τη μέθοδο τήρησης στοιχείων.

Ο δυναμικός προγραμματισμός εφαρμόζεται συχνά σε προβλήματα βελτιστοποίησης. Τέτοια προβλήματα μπορεί να έχουν πολλές λύσεις και το ζητούμενο είναι να βρεθεί *μία* βέλτιστη λύση. Η ανάπτυξη ενός αλγόριθμου δυναμικού προγραμματισμού ακολουθεί τέσσερα βήματα:

1. Χαρακτηρίζουμε τη δομή μιας βέλτιστης λύσης.
2. Ορίζουμε αναδρομικά την τιμή μιας βέλτισης λύσης.
3. Υπολογίζουμε την τιμή μιας βέλτισης λύσης εργαζόμενοι από τα "μερικά προς τα γενικά".
4. Κατασκευάζουμε μια βέλτιστη λύση από τα δεδομένα μας.

Παρουσιάζουμε την προσέγγιση του δυναμικού προγραμματισμού στην επίλυση προβλημάτων, με ένα απλό παράδειγμα. Το Σχήμα 4.1 αναπαριστά ένα χάρτη δρόμων που συνδέουν σπίτια και χώρους στάθμευσης στο κέντρο της πόλης. Οι κόμβοι αντιστοιχούν σε διασταυρώσεις, με τον αριθμό που σημειώνεται σε κάθε κόμβο να αντιστοιχεί στην καθυστέρηση (σε λεπτά) που υπόκειται κάποιος σε αυτήν.
Θα θέλαμε να ελαχιστοποιήσουμε τη συνολική καθυστέρηση που μπορεί να υποστεί οποιοσδήποτε εργαζόμενος στις διασταυρώσεις, ενώ οδηγεί από το σπίτι του στο κέντρο της πόλης. Απλοποιούμε κατ' αρχάς τον χάρτη αφαιρώντας τους δρόμους και διατηρώντας μόνο τις διασταυρώσεις, οι οποίες τώρα αναπαριστάνονται ως κουτιά, όπως στο Σχήμα 4.2. Έτσι, κάθε ένας που μετακινείται από το σπίτι του προς το κέντρο της πόλης θα πρέπει κινηθεί από αριστερά προς τα δεξιά σε γειτονικά μόνο κουτιά.
Η πιο αφελής προσέγγιση για την επίλυση του προβλήματος θα ήταν η απαρίθμηση και των 150 διαδρομών μέσω του διαγράμματος, επιλέγοντας τη διαδρομή που δίνει τη μικρότερη καθυστέρηση. Ο δυναμικός προγραμματισμός μειώνει τον αριθμό των υπολογισμών μετακινούμενος συστηματικά από τη μια πλευρά στην άλλη, κατασκευάζοντας την καλύτερη λύση καθώς προχωράει.

Ας υποθέσουμε ότι κινούμαστε προς τα πίσω, από τα δεξιά προς τα αριστερά στο Σχήμα 4.2. Εάν βρισκόμαστε σε κάποια διασταύρωση χωρίς να έχουμε άλλες διασταυρώσεις να περάσουμε, δεν έχουμε να πάρουμε καμία απόφαση και απλά αναλαμβάνουμε την καθυστέρηση που αντιστοιχεί σε αυτή τη διασταύρωση. Η τελευταία στήλη του διαγράμματος στο Σχήμα 4.2 συνοψίζει τις καθυστερήσεις όταν δεν υπάρχουν άλλες διασταυρώσεις να περάσουμε. Η πρώτη μας απόφαση, καθώς κινούμαστε προς τα πίσω, λαμβάνεται όταν απομένει μια διασταύρωση. Αν, για παράδειγμα, βρισκόμαστε στη διασταύρωση που αντιστοιχεί στο σκιασμένο κουτί στο Σχήμα 4.2, έχουμε καθυστέρηση τριών λεπτών σε αυτήν και καθυστέρηση είτε οκτώ είτε δύο λεπτών στην τελευταία διασταύρωση, ανάλογα με το αν θα κινηθούμε προς τα πάνω ή προς τα κάτω. Επομένως, η μικρότερη

δυνατή καθυστέρηση ή η *βέλτιστη λύση* σε αυτή τη διασταύρωση είναι $3 + 2 = 5$ λεπτά. Ομοίως, μπορούμε να εξετάσουμε κάθε διασταύρωση σε αυτή τη στήλη με τη σειρά και να υπολογίσουμε τη μικρότερη συνολική καθυστέρηση ως αποτέλεσμα της παραμονής σε κάθε διασταύρωση. Οι βέλτιστες λύσεις φαίνονται από τους αριθμούς σε έντονη γραφή στο Σχήμα 4.3. Τα βέλη υποδεικνύουν τη βέλτιστη απόφαση, προς τα πάνω ή προς τα κάτω, σε κάθε διασταύρωση όταν απομένει μια διασταύρωση να διανυθεί.

Σημειώνουμε ότι οι αριθμοί με έντονη γραφή στο Σχήμα 4.3 συνοψίζουν πλήρως, για σκοπούς λήψης αποφάσεων, τις συνολικές καθυστερήσεις των δύο τελευταίων στηλών. Παρόλο που οι αρχικοί αριθμοί των δύο τελευταίων στηλών χρησιμοποιήθηκαν για τον προσδιορισμό των αριθμών με έντονη γραφή, όποτε λαμβάνουμε αποφάσεις στα αριστερά αυτών των στηλών χρειάζεται να γνωρίζουμε μόνο τους αριθμούς με έντονη γραφή. Σε μια διασταύρωση, ας πούμε την πρώτη από πάνω, όταν απομένει μια διασταύρωση, γνωρίζουμε ότι η (βέλτιστη) εναπομένουσα καθυστέρηση μας, συμπεριλαμβανομένης της καθυστέρησης σε αυτή τη διασταύρωση, είναι πέντε λεπτά. Οι αριθμοί με έντονη γραφή συνοψίζουν όλες τις καθυστερήσεις από αυτό το σημείο και μετά. Για τη λήψη αποφάσεων στα αριστερά των αριθμών με έντονη γραφή, η τελευταία στήλη μπορεί να αγνοηθεί.

Με αυτό υπ΄όψιν μας, μετακινούμαστε κατά μια στήλη προς τ' αριστερά και υπολογίζουμε τις βέλτιστες λύσεις σε κάθε διασταύρωση, όταν απομένουν δύο διασταυρώσεις να διανυθούν. Οι βέλτιστες λύσεις, όταν απομένουν δύο διασταυρώσεις να διανυθούν φαίνονται στο Σχήμα 4.4 (αριστερά). Στη συνέχεια μετακινούμαστε κατά μια στήλη προς τ' αριστερά και επαναλαμβάνουμε τους ίδιους υπολογισμούς. Οι βέλτιστες λύσεις φαίνονται στο Σχήμα 4.4 (δεξιά).

Στο σχήμα Σχήμα 4.5 παρουσιάζεται η βέλτιστη λύση του προβλήματος. Η μικρότερη δυνατή καθυστέρηση μέσω του δικτύου είναι 18 λεπτά. Για να ακολουθήσει κάποιος τη διαδρομή με το μικρότερο κόστος, θα πρέπει να ξεκινήσει από τη δεύτερη διασταύρωση της πρώτης στήλης και να ακολουθήσει τη διαδρομή που υποδεικνύουν τα βέλη.

Ωστόσο, οι εργαζόμενοι μάλλον δεν είναι ελεύθεροι να επιλέξουν αυθαίρετα τη διασταύρωση από την οποία επιθυμούν να ξεκινήσουν. Μπορούμε να υποθέσουμε ότι τα σπίτια τους γειτνιάζουν με μία μόνο από τις διασταυρώσεις στην πρώτη στήλη, και επομένως το σημείο εκκίνησης κάθε εργαζόμενου είναι σταθερό. Αυτή η υπόθεση δεν προκαλεί καμία δυσκολία, αφού έχουμε, στην πραγματικότητα, καθορίσει τις διαδρομές με την ελάχιστη καθυστέρηση από τους χώρους στάθμευσης στο κέντρο της πόλης προς τα σπίτια όλων των εργαζομένων. Αυτό προϋποθέτει ότι οι εργαζόμενοι δεν ενδιαφέρονται σε ποιο χώρο στάθμευσης στο κέντρο της πόλης θα παρκάρουν. Αντί να επιλύσουμε το πρόβλημα της ελάχιστης καθυστέρησης για έναν μόνο συγκεκριμένο εργαζόμενο, έχουμε ενσωματώσει το πρόβλημα του συγκεκριμένου εργαζόμενου στο γενικότερο πρόβλημα της εύρεσης των διαδρομών με την ελάχιστη καθυστέρηση από όλα τα σπίτια προς την ομάδα χώρων στάθμευσης στο κέντρο της πόλης. Σημειώνουμε τέλος, ότι καμία βέλτιστη διαδρομή δεν χρησιμοποίησε τρεις από τις διασταυρώσεις της τελευταίας στήλης. Επιλύσαμε το πρόβλημα της εύρεσης των διαδρομών από κάθε σπίτι προς την ομάδα χώρων στάθμευσης και όχι προς κάθε συγκεκριμένο χώρο στάθμευσης.

Στην ορολογία του δυναμικού προγραμματισμού, κάθε σημείο όπου λαμβάνονται αποφάσεις ονομάζεται συνήθως στάδιο της διαδικασίας λήψης αποφάσεων. Σε κάθε στάδιο, αρκεί να γνωρίζουμε σε ποια διασταύρωση βρισκόμαστε για να μπορούμε να λάβουμε τις επόμενες αποφάσεις. Οι επόμενες αποφάσεις μας δεν εξαρτώνται από το πώς φτάσαμε στη συγκεκριμένη διασταύρωση. Οι πληροφορίες που συνοψίζουν τη γνώση που απαιτείται για το πρόβλημα προκειμένου να ληφθούν οι τρέχουσες αποφάσεις, όπως η διασταύρωση στην οποία βρισκόμαστε σε ένα συγκεκριμένο στάδιο, ονομάζονται κατάσταση της διαδικασίας λήψης αποφάσεων. Με βάση αυτές τις έννοιες, η λύση μας στο πρόβλημα της ελάχιστης καθυστέρησης περιλαμβάνει την ακόλουθη διαισθητική ιδέα, που συνήθως αναφέρεται ως *αρχή της βέλτιστης λειτουργίας*, ή *αρχή της βέλτιστης υποδομής*.

> Κάθε βέλτιστη πολιτική έχει την ιδιότητα ότι, ανεξάρτητα από την τρέχουσα κατάσταση και απόφαση, οι υπολοιπόμενες αποφάσεις πρέπει να αποτελούν βέλτιστη πολιτική σε σχέση με την κατάσταση που προκύπτει από την τρέχουσα απόφαση.

Για να συγκεκριμενοποιήσουμε αυτή την αρχή, αριθμούμε τις διασταυρώσεις s_{ij}, $i, j = 1, \ldots, 6$, από κάτω προς τα πάνω και από δεξιά προς τ' αριστερά, και συμβολίζουμε με t_{ij} την καθυστέρηση στη διασταύρωση s_{ij}. Ορίζουμε v_{ij} να είναι η ελάχιστη καθυστέρηση δεδομένου ότι βρισκόμαστε στη διασταύρωση s_{ij}, οπότε απομένουν j διασταυρώσεις να διανυθούν. Για παράδειγμα, για το σκιασμένο κουτί στο Σχήμα 4.2 έχουμε $i = 6, j = 2$, και, βέβαια, $t_{ij} = 3$. Δεδομένου του ότι η διασταύρωση s_{ij} συνδέεται είτε με την $s_{i,j-1}$ ή την

$s_{i+1,j-1}$, αν j είναι περιττός, ή την $s_{i-1,j-1}$ αν j είναι άρτιος, μπορούνε να γράψουμε μια αναδρομική σχέση για τον υπολογισμό της ελάχιστης καθυστέρησης

$$v_{ij} = \begin{cases} \min\{t_{ij} + v_{i,j-1}, t_{ij} + v_{i+1,j-1}\} & \text{αν } j \text{ περιττός}, \\ \min\{t_{ij} + v_{i,j-1}, t_{ij} + v_{i-1,j-1}\} & \text{αν } j \text{ άρτιος}, \end{cases} \tag{4.1}$$

με τους προφανείς περιορισμούς για την πρώτη και την τελευταία διασταύρωση. Προφανώς,

$$v_{i,1} = t_{i,1}, \qquad i = 1, \ldots, 6, \tag{4.2}$$

μια και η βέλτιστη διαδρομή για τις διασταυρώσεις στην πρώτη από δεξιά στήλη απαιτεί μόνο τη διάνυση αυτής. Από τις εξισώσεις (4.1) και (4.2) είναι δυνατόν να υπολογίσουμε τα v_{ij} για $1 \leq i, j \leq 6$. Η συγκεκριμένη μέθοδος υπολογισμού ονομάζεται *επαγωγή προς τα πίσω*, μια και ξεκινάει από τα δεξιά και κινείται προς τα πίσω.

4.0.1 Πολλαπλασιασμός αλληλουχίας πινάκων

Μας δίνονται n πίνακες $A_1, A_2, \ldots, A_n$ και μας ζητείται να υπολογίσουμε το γινόμενο $A_1 A_2 \cdots A_n$. Από τη στιγμή που θα ομαδοποιήσουμε παρενθετικά τους όρους του γινομένου προσδιορίζουμε μονοσήμαντα τη σειρά πολλαπλασιαμών μεταξύ τους. Δεδομένου ότι ο πολλαπλασιασμός πινάκων είναι προσεταιριστική πράξη όλες οι παρενθετικές ομαδοποιήσεις δίνουν το ίδιο γινόμενο. Θα λέμε ότι ένα γινόμενο πινάκων είναι γραμμένο σε *πλήρως παρενθετική μορφή* αν είναι είτε ένας και μόνο πίνακας είτε γινόμενο πινάκων τα οποία είναι εκφρασμένα σε πλήρως παρενθετική μορφή. Για παράδειγμα το γινόμενο $A_1 A_2 A_3 A_4$ μπορεί να γραφεί σε πλήρως παρενθετική μορφή με πέντε διαφορετικούς τρόπους:

$$\begin{gathered} (A_1(A_2(A_3 A_4))), \\ (A_1((A_2 A_3)A_4)), \\ ((A_1 A_2))(A_3 A_4)), \\ ((A_1(A_2 A_3))A_4), \\ (((A_1 A_2)A_3)A_4). \end{gathered}$$

Αν θεωρήσουμε ως κόστος υπολογισμού ενός γινομένου το πλήθος των βαθμωτών πολλαπλασιασμών που εκτελούνται, είναι εύκολο να δούμε ότι διαφορετικές ομαδοποιήσεις έχουν διαφορετικά κόστη[1]. Αν, για παράδειγμα, θεωρήσουμε τους πίνακες A_1, A_2, A_3 με διαστάσεις 10×100, 100×5, 5×30, αντίστοιχα, τότε η παρενθετική ομαδοποίηση $((A_1 A_2)A_3)$ έχει κόστος 7500, ενώ η παρενθετική ομαδοποίηση $(A_1(A_2 A_3))$ έχει κόστος 75000, είναι, δηλαδή, 10 φορές πιο ακριβή.

Θα ορίσουμε το πρόβλημα του πολλαπλασιασμού αλληλουχίας πινάκων ως εξής:

> για μια δεδομένη αλληλουχία πινάκων $A_1, A_2, \ldots, A_n$, όπου για κάθε $i = 1, 2, \ldots, n$, ο πίνακας A_i έχει διαστάσεις $p_{i-1} \times p_i$, εκφράστε το γινόμενο $A_1 A_2 \cdots A_n$ σε πλήρως παρενθετική μορφή έτσι ώστε να ελαχιστοποιείται το πλήθος των βαθμωτών πολλαπλασιασμών.

Ας βεβαιωθούμε πρώτα ότι ο εξαντλητικός έλεγχος όλων των παρενθετικών ομαδοποιήσεων δεν οδηγεί σε έναν αποτελεσματικό αλγόριθμο. Έστω $P(n)$ το πλήθος των εναλλακτικών ομαδοποιήσεων μιας αλληλουχίας n πινάκων. Προφανώς $P(1) = 1$. Για $n \geq 2$ ένα γινόμενο πινάκων εκφρασμένο σε πλήρως παρενθετική μορφή είναι το γινόμενο δύο πινάκων εκφρασμένων σε πλήρως παρενθετική μορφή με τη διαχωριστική γραμμή να βρίσκεται μεταξύ του k-οστού και $(k+1)$-οστού πίνακα, όπου το k μπορεί να πάρει οποιαδήποτε από τις τιμές $k = 1, 2, \ldots, n-1$. Συνεπώς,

$$P(n) = \begin{cases} 1 & \text{αν } n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{αν } n \geq 2. \end{cases} \tag{4.3}$$

Μπορεί να δειχθεί (Άσκηση 1) ότι η λύση της αναδρομικής σχέσης (4.3) είναι $\Omega(2^n)$ και συνεπώς η μέθοδος της εξαντλητικής αναζήτησης δεν αποτελεί ικανοποιητική στρατηγική για τον προσδιορισμό της βέλτιστης λύσης, δηλαδή, της βέλτιστης παρενθετικής ομαδοποίησης μιας αλληλουχίας πινάκων.

[1] Αν ο A είναι ένας $p \times q$ πίνακας και ο B είναι ένας $q \times r$ πίνακας, τότε το γινόμενο AB είναι ένας $p \times r$ πίνακας και ο υπολογισμός του απαιτεί pqr βαθμωτούς υπολογισμούς.

Όπως και στο προηγούμενο παράδειγμα, το πρώτο βήμα είναι η εύρεση της βέλτιστης υποδομής και η κατασκευή μιας βέλτιστης λύσης από βέλτιστες λύσεις υποπροβλημάτων. Για ευκολία, αν $i \leq j$, θα γράφουμε $A_{i..j}$ για να δηλώσουμε τον πίνακα που προκύπτει από τον πολλαπλασιασμό $A_i A_{i+1} \cdots A_j$. Αν $i < j$, οποιαδήποτε ομαδοποίηση του γινομένου $A_i A_{i+1} \cdots A_j$ θα πρέπει να το διαχωρίζει ανάμεσα στον πίνακα A_k και A_{k+1}, για κάποιο $i \leq k < j$. Αυτό σημαίνει ότι για κάποια τιμή του k υπολογίζουμε πρώτα τους πίνακες $A_{i..k}$ και $A_{k+1..j}$ και στη συνέχεια τους πολλαπλασιάζουμε για να πάρουμε το γινόμενο $A_{i..j}$. Αυτό σημαίνει ότι το κόστος της ομαδοποίησης ισούται με το κόστος υπολογισμού του πίνακα $A_{i..k}$ συν το κόστος υπολογισμού του πίνακα $A_{k+1..j}$, συν το κόστος υπολογισμού του γινομένου τους.

Ας υποθέσουμε τώρα ότι μια βέλτιστη ομαδοποίηση του γινομένου $A_i A_{i+1} \ldots A_j$ το διαχωρίζει ανάμεσα στους πίνακες A_k και A_{k+1}. Τότε αυτή η ομαδοποίηση αποτελεί βέλτιστη ομαδοποίηση τόσο για το γινόμενο $A_i A_{i+1} \ldots A_k$ όσο και για το $A_{k+1} A_{k+2} \ldots A_j$. Σε διαφορετική περίπτωση, χρησιμοποιώντας τις βέλτιστες ομαδοποιήσεις τους θα είχαμε μια ομαδοποίηση του γινομένου $A_i A_{i+1} \ldots A_j$ με μικρότερο κόστος, το οποίο δεν μπορεί να συμβεί. Συνοψίζουμε αυτό το αποτέλεσμα ως εξής:

> **Δομή της βέλτιστης ομαδοποίησης**: μπορούμε να κατασκευάσουμε μια βέλτιστη ομαδοποίηση για το γινόμενο $A_i A_{i+1} \ldots A_j$ συνδυάζοντας μια βέλτιστη ομαδοποίηση του γινομένου $A_i A_{i+1} \ldots A_k$ και μια βέλτιστη ομαδοποίηση του $A_{k+1} A_{k+2} \ldots A_j$.

Έστω $m[i, j]$ το ελάχιστο πλήθος βαθμωτών πολλαπλασιασμών που απαιτούνται για τον υπολογισμό του γινομένου $A_{i..j}$ για $1 \leq i \leq j \leq n$. Χρησιμοποιώντας τη δομή της βέλτιστης ομαδοποίησης έχουμε τη σχέση $m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$, με τον τελευταίο όρο να δηλώνει τον αριθμό των βαθμωτών υπολογισμών για τον υπολογισμό του γινομένου του $A_{i..k}$ με τον $A_{k+1..j}$. Η σχέση αυτή προϋποθέτει ότι γνωρίζουμε την τιμή του k. Επειδή υπάρχουν μόνο $j - i$ πιθανές επιλογές του k και η βέλτιστη ομαδοποίηση πρέπει να αντιστοιχεί σε κάποια από αυτές, τις εξετάζουμε όλες. Έτσι,

$$m[i, j] = \begin{cases} 0 & \text{αν } i = j, \\ \min\limits_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j\} & \text{αν } i < j. \end{cases} \tag{4.4}$$

Παρατηρούμε ότι ο αριθμός των υποπροβλημάτων που πρέπει να λύσουμε είναι σχετικά μικρός: ένα πρόβλημα για κάθε ζεύγος i και j το οποίο ικανοποιεί τη σχέση $1 \leq i \leq j \leq n$, δηλαδή συνολικά $n(n + 1)/2 = \Theta(n^2)$ υποπροβλήματα. Ένας αναδρομικός αλγόριθμος πιθανόν να συναντά το ίδιο υποπρόβλημα πολλές φορές σε διαφορετικούς κλάδους του δένδρου αναδρομής, ιδιότητα η οποία αποτελεί το δεύτερο κριτήριο εφαρμοσιμότητας του δυναμικού προγραμματισμού.

Μπορούμε τώρα εύκολα να κατασκευάσουμε έναν αλγόριθμο για τον υπολογισμό της λύσης της αναδρομικής σχέσης (4.4). Η διαδικασία ΟΡΤΜULT που φαίνεται παρακάτω, δέχεται ως είσοδο την ακολουθία $p = (p_0, p_1, \ldots, p_n)$, με τις διαστάσεις των πινάκων A_i: ο A_i έχει διαστάσεις $p_{i-1} \times p_i$, για $i = 1, \ldots, n$. Η διαδικασία χρησιμοποιεί τον $n \times n$ πίνακα m για την αποθήκευση των ποσοτήτων $m[i, j]$ και τον πίνακα s, διάστασης $n \times n$, για την αποθήκευση της τιμής του δείκτη k για τον οποίο επιτυγχάνεται το ελάχιστο κόστος κατά τον υπολογισμό του $m[i, j]$.

Εξετάζουμε πρώτα ποια στοιχεία του πίνακα m χρησιμοποιούνται κατά τον υπολογισμό του $m[i, j]$ για ένα ζεύγος i και j το οποίο ικανοποιεί τη σχέση $1 \leq i \leq j \leq n$. Η ποσότητα $m[i, j]$ είναι το πλήθος των

βαθμωτών πολλαπλασιασμών που απαιτούνται για τον υπολογισμό του γινομένου $A_{i..j}$. Εξαρτάται, δηλαδή, από τα γινόμενα $A_{i..k}$ και $A_{k+1..j}$, για $k = i, i + 1, \ldots, j - 1$. Συνεπώς, τα στοιχεία $m[i, k]$ και $m[k + 1, j]$, για $k = i, i + 1, \ldots, j - 1$, πρέπει να υπολογιστούν πρώτα. Αυτά αντιστοιχούν στα σκιασμένα κουτάκια στο Σχήμα 4.6. Γίνεται φανερό ότι τα στοιχεία του άνω τριγώνου του πίνακα m μπορούν να υπολογιστούν κατά διαγωνίους, ξεκινώντας από την κύρια διαγώνιο, όπου $m[i, i] = 0$, $i = 1, \ldots, n$. Στον ψευδοκώδικα που ακολουθεί, παρατηρούμε ότι η ποσότητα $m[i, j]$ υπολογίζεται για δείκτες i και j τέτοιους ώστε $j - i = l - 1$, με τον δείκτη l στο διάστημα $[2, n]$. Επομένως οι διαφορές των δεικτών i και j είναι στο διάστημα $[1, n - 1]$, αντιστοιχούν δηλαδή στα στοιχεία $m[i, i + l]$, $l = 1, \ldots, n - 1$. Αυτά είναι ακριβώς τα στοιχεία του πίνακα m στην l-οστή διαγώνιο πάνω από την κύρια διαγώνιο του πίνακα.

```
OptMult(n, p, m, s)
    for i = 1 to n
        m[i, i] = 0
    for l = 2 to n
        for i = 1 to n − l + 1
            j = i + l − 1
            m[i, j] = ∞
            for k = i to j − 1
                q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
                if q < m[i, j]
                    m[i, j] = q
                    s[i, j] = k
```

Η πλήρως παρανθετική μορφή μπορεί να βρεθεί με τη βοήθεια του πίνακα s και το γεγονός ότι για κάθε $i < j$ αν $k = s[i, j]$ τότε η πλήρως παρενθετική μορφή του γινομένου $A_{i..j}$ προκύπτει από τις πλήρως παρενθετικές μορφές των γινομένων $A_{i..k}$ και $A_{k+1..j}$. Προφανώς, ότι $i = j$ το γινόμενο έχει μόνο ένα όρο, τον πίνακα A_i, και αυτή είναι η πλήρως παρενθετική μορφή του.

```
OptParens(s, i, j)
    if i = j
        print "A i"
    else
        print "("
        OptParens(s, i, s[i, j])
        OptParens(s, s[i, j] + 1, j)
        print ")"
```

Παράδειγμα. Ας υποθέσουμε ότι θέλουμε να υπολογίσουμε με τον οικονομικότερο δυνατό τρόπο το γινόμενο $A_1A_2A_3A_4A_5A_6$, όπου οι πίνακες A_i έχουν διαστάσεις, 30×35, 35×15, 15×5, 5×10, 10×10, και 20×25, αντίστοιχα. Εκτελώντας τη διαδικασία OptMult παίρνουμε τους πίνακες m και s στο Σχήμα 4.7. Και οι δύο πίνακες έχουν στραφεί έτσι ώστε οι διαγώνιοι τους να είναι οριζόντιοι. Εδώ φαίνεται ότι $m[1, 6] = 15125$. Στοιχεία με ίδια χρώματα δηλώνουν ότι αυτά συνδυάστηκαν κατά τον υπολογισμό του $m[2, 5]$

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1p_2p_5 = 0 + 2500 + 10500 = 13000, \\ m[2,3] + m[4,5] + p_1p_3p_5 = 2625 + 1000 + 3500 = 7125, \\ m[2,4] + m[5,5] + p_1p_4p_5 = 4375 + 0 + 7000 = 11375. \end{cases}$$

Η βέλτιστη, πλήρως παρενθετική μορφή του γινομένου $A_1A_2A_3A_4A_5A_6$ είναι η $((A_1(A_2A_3))((A_4A_5)A_6))$.

Μπορούμε τώρα να συνοψίσουμε τα χαρακτηριστικά που είχαν τόσο το πρόβλημα του πολλαπλασιασμού αλληλουχίας πινάκων όσο και το πρόβλημα της εύρεσης μιας βέλτιστης διαδρομής. Όποτε ένα πρόβλημα παρουσιάζει αυτά τα χαρακτηριστικά τότε η μέθοδος του δυναμικού προγραμματισμού μπορεί να είναι εφαρμόσιμη.

- Το πρώτο βήμα στην επίλυση ενός προβλήματος βελτιστοποίησης με δυναμικό προγραμματισμό είναι ο χαρακτηρισμός της δομής μιας βέλτιστης λύσης. Λέμε ότι ένα πρόβλημα παρουσιάζει *βέλτιστη υποδομή* αν μια βέλτιστη λύση του προβλήματος περιέχει στο εσωτερικό της βέλτιστες λύσεις υποπροβλημάτων. Για παράδειγμα, στο πρόβλημα του πολλαπλασιασμού μιας αλληλουχίας πινάκων η βέλτιστη παρενθετική μορφή του γινομένου $A_iA_{i+1} \cdots A_j$ η οποία το χωρίζει στα $A_iA_{i+1} \cdots A_k$ και $A_kA_{k+1} \cdots A_j$ περιέχει βέλτιστες λύσεις για τα προβλήματα παρενθετικής μορφής των δύο αυτών γινομένων.

- Η βέλτιστη υποδομή διαφέρει στα προβλήματα εφαρμογής της μεθόδου του δυναμικού προγραμματισμού με δύο τρόπους: πόσα υποπροβλήματα χρησιμοποιεί μια βέλτιστη λύση, και πόσες επιλογές υπάρχουν για να καθορίσουμε ποια υποπροβλήματα θα χρησιμοποιηθούν σε μια βέλτιστη λύση. Για παράδειγμα, για το πρόβλημα του πολλαπλασιασμού της αλληλουχίας $A_iA_{i+1} \cdots A_j$ είχαμε δύο υποπροβλήματα και $j - i$ δυνατές επιλογές υποπροβλημάτων.

- Ο χρόνος εκτέλεσης ενός αλγόριθμου δυναμικού προγραμματισμού εξαρτάται από το γινόμενο του αριθμού των υποπροβλημάτων και των αριθμό των επιλογών που εξετάζουμε σε κάθε υποπρόβλημα. Για το πρόβλημα του πολλαπλασιασμού μιας αλληλουχίας πινάκων τα υποπροβλήματα είναι $\Theta(n^2)$ με $n-1$ επιλογής στη χειρότερη περίπτωση, συνολικό χρόνο δηλαδή, $O(n^3)$.

- Στη μέθοδο του δυναμικού προγραμματισμού λύνουμε με βέλτιστο τρόπο τα υποπροβλήματα και συνδυάζουμε τη λύση τους σε μια βέλτιστη λύση του αρχικού προβλήματος. Η βέλτιστη λύση του αρχικού προβλήματος προϋποθέτει την επιλογή μεταξύ των υποπροβλημάτων που θα χρησιμοποιηθούν στην επίλυσή του και ένα επιπλέον κόστος που συνδέεται άμεσα με την επιλογή αυτή. Στο πρόβλημα του πολλαπλασιασμού αλληλουχίας πινάκων αυτό το κόστος είναι το κόστος $p_{i-1}p_kp_j$.

- Τα υποπροβλήματα που επιλέγουμε είναι *ανεξάρτητα.* με την έννοια ότι η λύση του ενός δεν επηρεάζει τη λύση του άλλου. Στο πρόβλημα του πολλαπλασιασμού μιας αλληλουχίας πινάκων τα υποπροβλήματα του πολλαπλασιασμού των αλληλουχιών $A_iA_{i+1}\cdots A_k$ και $A_{k+1}A_{k+2}\cdots A_j$ είναι *ανεξάρτητα* γιατί δεν υπάρχει πίνακας ο οποίος να εμφανίζεται και στις δύο αλληλουχίες.

- Ο αριθμός των υποπροβλημάτων πρέπει να είναι μικρός έτσι ώστε ένας αναδρομικός αλγόριθμος να ανακαλεί τη λύση κάποιου υποπροβλήματος αντί να δημιουργεί συνέχεια καινούργια υποπροβλήματα προς επίλυση. Όταν ένας αναδρομικός αλγόριθμος αναφέρεται σε κάποιο υποπρόβλημα συνέχεια λέμε ότι έχει *επικαλυπτόμενα υποπροβλήματα.* Η μέθοδος του δυναμικού προγραμματισμού εκμεταλλεύεται την ύπαρξη επικαλυπτόμενων υποπροβλημάτων αποθηκεύοντας τη λύση τους και ανατρέχοντας σε αυτές αν χρειαστούν εκ νέου. Για το πρόβλημα του πολλαπλασιασμού αλληλουχίας πινάκων, μια αναδρομική υλοποίηση θα μπορούσε να είναι η

```
RecMult(p, i, j)
    if i == j
        return 0
    m[i, j] = ∞
    for k = i to j − 1
        q = RecMult(p, i, k) + RecMult(p, k + 1, j) + p_{i−1} p_k p_j
        if q < m[i, j]
            m[i, j] = q
    return m[i, j]
```

Το Σχήμα 4.8 παρουσιάζει το δένδρο αναδρομής για την κλήση της διαδικασίας RecMult$(p, 1, 4)$. Ο αριθμός των υποπροβλημάτων που υπολογίζει συνεχώς η διαδικασία RecMult φαίνεται να είναι μεγάλος. Στην πραγματικότητα, ο χρόνος που απαιτείται για τον υπολογισμό της ποσότητας $m[1, n]$ με χρήση της διαδικασίας RecMult είναι εκθετικά μεγάλος. Πράγματι, έστω $T(n)$ αυτός ο χρόνος. Τότε,

$$T(n) \geq \begin{cases} 1 & \text{αν } n = 1, \\ 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) & \text{αν } n \geq 2. \end{cases}$$

Για κάθε $i = 1, \ldots, n-1$, ο όρος $T(i)$εμφανίζεται μια φορά εξαιτίας του όρου $T(k)$ και μια φορά εξαιτίας του όρου $T(n-k)$. Επομένως,

$$T(n) \geq 2\sum_{i=1}^{n-1} T(i) + n. \tag{4.5}$$

Εύκολα μπορούμε να δείξουμε ότι $T(n) = \Omega(2^n)$ χρησιμοποιώντας τη μέθοδο της αντικατάστασης. Για $n = 1$ έχουμε $T(1) = 1 = 2^{1-1}$. Για $n \geq 2$ έχουμε

$$T(n) \geq 2\sum_{i=1}^{n-1} 2^{i-1} + n = 2(2^{n-1} - 1) + n = 2^n - 2 + n \geq 2^{n-1},$$

το οποίο ολοκληρώνει την απόδειξη.

- Για την ανακατασκευή της βέλτιστης λύσης, είναι πιθανό να χρειαστεί η αποθήκευση της επιλογής που έγινε σε κάθε υποπρόβλημα, όπως, για παράδειγμα, στο πρόβλημα του πολλαπλασιασμού αλληλουχίας πινάκων με τη χρήση του πίνακα s.

4.1 Ασκήσεις

1. Χρησιμοποιώντας τη μέθοδο της αντικατάστασης δείξτε ότι η λύση της αναδρομικής εξίσωσης (4.3) είναι $\Omega(2^n)$. *Υπόδειξη.* Δείξτε επαγωγικά ότι $P(n) \geq 2^n - 1$, για $n \geq 1$.

2. Δείξτε ότι μια πλήρης παρενθετική ομαδοποίηση μιας έκφρασης n στοιχείων περιέχει ακριβώς $n - 1$ ζεύγη παρενθέσεων.

3. Προσδιορίστε μια βέλτιστη ομαδοποίηση του γινομένου μιας αλληλουχίας πινάκων για την οποία η ακολουθία των διαστάσεων των πινάκων είναι 5, 10, 3, 12, 5, 50, 6.

4.2 Μέγιστη κοινή υπακολουθία

Το DNA, ένα είδος νουκλεϊνικού οξέος, ελέγχει τις λειτουργίες και τα κληρονομικά χαρακτηριστικά των οργανισμών. Η πληροφορία του DNA είναι κωδικοποιημένη στα *νουκλεοτίδια* τα οποία προέρχονται από τη σύνδεση τριών μορίων: ενός σακχάρου, ενός μορίου φωσφορικού οξέος και μιας οργανικής αζωτούχας βάσης. Οι αζωτούχες βάσεις που συναντώνται στο μόριο του DNA είναι η *αδενίνη* (A), η *γουανίνη* (G), η *θυμίνη* (T) και η *κυτοσίνη* (C). Τα νουκλεοτίδια ενώνονται δημιουργώντας μια μεγάλου μήκους αλυσίδα νουκλεϊνικών οξέων. Το βακτήριο *E. coli* περιέχει 4.6×10^6 ζεύγη βάσεων, ενώ το ανθρώπινο DNA συνίσταται από 3×10^9 ζεύγη βάσεων. Περισσότερο από 99% των βάσεων είναι τα ίδια σε όλους τους ανθρώπους. Η αλυσίδα του DNA αντιμετωπίζεται ως μια ακολουθία γραμμάτων από το σύνολο $\{A, C, G, T\}$, στην οποία αναζητούνται σχηματισμοί, συνήθως με 3 βάσεις, που επαναλαμβάνονται περισσότερο από δύο φορές ο ένας μετά τον άλλο. Τέτοιες εμφανίσεις φαίνεται να παίζουν σημαντικό ρόλο στην ανάπτυξη του ανοσοποιητικού συστήματος των κυττάρων, αλλά και στην εμφάνιση κάποιων ασθενειών. Ενδιαφέρον επίσης παρουσιάζει η αναζήτηση ομοιοτήτων μεταξύ των ακολουθιών δύο διαφορετικών οργανισμών που μπορεί να συνεπάγεται τη συσχέτιση των λειτουργιών τους ή της προέλευσης τους. Σε αυτή τη διαδικασία σύγκρισης αναζητούνται υπακολουθίες οι οποίες ταυτίζονται.

Αν $X = \{x_1, \ldots, x_n\}$ και $Z = \{z_1, \ldots, z_k\}$ είναι δύο ακολουθίες, θα λέμε ότι η Z είναι *υπακολουθία* της X αν υπάρχει μια αυστηρά αύξουσα ακολουθία δεικτών $\{i_1, i_2, \ldots, i_k\}$ τέτοια ώστε

$$x_{i_j} = z_j, \qquad j = 1, \ldots k.$$

Για παράδειγμα, η ακολουθία $\{B, C, B, D\}$ είναι υπακολουθία της $\{A, B, C, B, D, A, B\}$ με αντίστοιχη ακολουθία δεικτών την $\{2, 3, 5, 7\}$. Δεδομένων δύο ακολουθιών X και Y, το πρόβλημα της *μέγιστης κοινής υπακολουθίας* (ΜΚΥ) (*longest common subsequence*, LCS) συνίσταται στην εύρεση μιας κοινής υπακολουθίας με το μεγαλύτερο δυνατό μήκος.

Το πρόβλημα της μέγιστης κοινής υπακολουθίας μπορεί να λυθεί αφελώς, δηλαδή με απαρίθμηση όλων των κοινών υπακολουθιών και ελέγχου για το μεγαλύτερο μήκος. Αλλά αν η ακολουθία έχει

μήκος n, τότε το σύνολο των υπακολουθιών της έχει 2^n στοιχεία, και επομένως η αφελής προσέγγιση δεν είναι πρακτική για ακολουθίες με μεγάλα μήκη. Όμως, το πρόβλημα της ΜΚΥ έχει βέλτιστη υποδομή και επομένως τεχνικές δυναμικού προγραμματισμού είναι πιθανόν να μπορούν να εφαρμοστούν. Θα χρειαστούμε την έννοια του προθέματος μιας ακολουθίας: αν $X = \{x_1, x_2, \ldots, x_m\}$ είναι μια ακολουθία, τότε το i-στό *πρόθεμα* της είναι το $X_i = \{x_1, x_2, \ldots, x_i\}$, $i = 0, 1, \ldots, m$. Για παράδειγμα, αν $X = \{A, B, C, B, D, A, B\}$ τότε $X_3 = \{A, B, C, B\}$ και, βέβαια, X_0 είναι μια κενή ακολουθία.

Θεώρημα. Έστω $X = \{x_1, x_2, \ldots, x_m\}$ και $Y = \{y_1, y_2, \ldots, y_n\}$ δύο ακολουθίες και $Z = \{z_1, z_2, \ldots, z_k\}$ μια μέγιστη κοινή υπακολουθία των X και Y. Τότε:

1. Αν $x_m = y_n$ τότε $z_k = x_m = y_n$ και Z_{k-1} είναι μια ΜΚΥ των X_{m-1} και Y_{n-1}.
2. Αν $x_m \neq y_n$ και $z_k \neq x_m$ τότε Z είναι μια ΜΚΥ των X_{m-1} και Y.
3. Αν $x_m \neq y_n$ και $z_k \neq y_n$ τότε Z είναι μια ΜΚΥ των X και Y_{n-1}.

Απόδειξη. Αν $z_k \neq x_m$ τότε μπορούμε να προσαρτήσουμε τον όρο $x_m = y_n$ στην ακολουθία Z έτσι ώστε να έχουμε μια κοινή υπακολουθία των X και Y μήκους $k + 1$, που έρχεται σε αντίθεση με την υπόθεση ότι η Z είναι μια μέγιστη κοινή υπακολουθία. Συνεπώς, $z_k = x_m = y_n$. Δείχνουμε τώρα ότι Z_{k-1} είναι μέγιστη κοινή υπακολουθία των X_{m-1} και Y_{n-1}. Πράγματι, αν υπήρχε κοινή υπακολουθία W των X_{m-1} και Y_{n-1} με μήκος μεγαλύτερο του $k - 1$ τότε, η προσάρτηση του όρου $x_m = y_n$ σε αυτή θα έδινε μια υπακολουθία των X και Y με μήκος μεγαλύτερο του k, το οποίο δεν μπορεί να συμβαίνει.

Για τον δεύτερο ισχυρισμό του θεωρήματος, αν $z_k \neq x_m$ τότε Z είναι κοινή υπακολουθία των X_{m-1} και Y. Αν υπήρχε κοινή υπακολουθία W των X_{m-1} και Y με μήκος μεγαλύτερο του k, τότε αυτή θα ήταν κοινή υπακολουθία των $X_m = X$ και Y, το οποίο έρχεται σε αντίθεση με την υπόθεση ότι η Z είναι μέγιστη κοινή υπακολουθία των X και Y. Ο τρίτος ισχυρισμός του θεωρήματος αποδεικνύεται με ανάλογο τρόπο. □

Αυτό που προκύπτει από το παραπάνω θεώρημα είναι το γεγονός ότι μια μέγιστη κοινή υπακολουθία δύο ακολουθιών περιέχει μια μέγιστη κοινή υπακολουθία προθεμάτων των δύο ακολουθιών. Συνεπώς, το πρόβλημα της εύρεσης μιας μέγιστης κοινής υπακολουθίας έχει βέλτιστη υποδομή. Επιπλέον, υποδεικνύει και τον τρόπο με τον οποίο θα μπορούσε να ορίσει κάποιος μια αναδρομική διαδικασία για την εύρεσή της. Θα πρέπει να εξετάσουμε ένα ή δύο υποπροβλήματα:

- Αν $x_m = y_n$ τότε πρέπει να βρούμε μια ΜΚΥ των X_{m-1} και Y_{n-1}. Προσαρτώντας σε αυτήν τον όρο $x_m = y_n$ θα έχουμε μια ΜΚΥ των X και Y.
- Αν $x_m \neq y_n$ τότε έχουμε να λύσουμε δύο υποπροβλήματα: να βρούμε τη μέγιστη κοινή υπακολουθία των X_{m-1} και Y_n και τη μέγιστη κοινή υπακολουθία των X και Y_{n-1}. Όποια έχει το μεγαλύτερο μήκος είναι ΜΚΥ των X και Y.

Αφού αυτές οι δύο περιπτώσεις εξαντλούν όλες τις πιθανότητες, μια από τις βέλτιστες λύσεις των υποπροβλημάτων πρέπει να εμφανίζεται σε μια ΜΚΥ των X και Y.

Είναι επίσης σωστό ότι το πρόβλημα της μέγιστης κοινής υπακολουθίας έχει επικαλυπτώμενα υποπροβλήματα: για την εύρεση μιας ΜΚΥ των X και Y απαιτείται η εύρεση μιας ΜΚΥ των X και Y_{n-1} ή/και των X_{m-1} και Y. Και τα δύο αυτά προβλήματα έχουν ως κοινό υποπρόβλημα την εύρεση μιας ΜΚΥ για τις ακολουθίες X_{m-1} και Y_{n-1}.

Για τη λύση του προβλήματος της μέγιστης κοινής υπακολουθίας χρειαζόμαστε μια αναδρομική σχέση για την τιμή της βέλτιστης λύσης $c[i, j]$, δηλαδή του μήκους της μέγιστης υπακολουθίας των προθεμάτων X_i και Y_j, για $0 \leq i \leq m$ και $0 \leq j \leq n$. Με βάση τους ισχυρισμούς του θεωρήματος

έχουμε

$$c[i,j] = \begin{cases} 0 & \text{αν } i = 0 \text{ ή } j = 0, \\ c[i-1, j-1] + 1 & \text{αν } i, j > 0 \text{ και } x_i = y_j, \\ \max\{c[i, j-1], c[i-1, j]\} & \text{αν } i, j > 0 \text{ και } x_i \neq y_j. \end{cases} \tag{4.6}$$

Το πρόβλημα της μέγιστης κοινής υπακολουθίας βλέπουμε ότι έχει $\Theta(mn)$ διακριτά υποπροβλήματα επομένως είναι δυνατός ο υπολογισμός λύσεων από τα μικρότερα προς τα μεγαλύτερα υποπροβλήματα (bottom–up). Ο υπολογισμός των στοιχείων του πίνακα μπορεί να γίνει κατά γραμμές, ξεκινώντας από την πρώτη προς την τελευταία και από την πρώτη στήλη προς την τελευταία. Από την αναδρομική σχέση (4.6) παρατηρούμε ότι η τιμή $c[i,j]$ εξαρτάται από τις τιμές $c[i-1,j-1]$, η οποία είναι σε προηγούμενη γραμμή και προηγούμενη στήλη, και από τις τιμές $c[i,j-1]$ και $c[i-1,j]$ οι οποίες εμφανίζονται σε προηγούμενη στήλη ή προηγούμενη γραμμή, αντίστοιχα. Θα χρησιμοποιήσουμε ακόμα τον πίνακα b, διάστασης $m \times n$, για την κατασκευή της βέλτιστης λύσης. Σε αυτόν καταγράφεται το υποπρόβλημα που χρησιμοποιήθηκε για τον υπολογισμό του $c[i,j]$. Μπορούμε να χρησιμοποιήσουμε το σύμβολο $\nwarrow$ για να αναφερθούμε στο υποπρόβλημα $c[i-1,j-1]$, το σύμβολο $\uparrow$ για να αναφερθούμε στο υποπρόβλημα $c[i-1,j]$ και το σύμβολο $\leftarrow$ για να αναφερθούμε στο πρόβλημα $c[i,j-1]$ (σκεφτείτε τη σχετική θέση των υποπροβλημάτων). Ο χρόνος εκτέλεσης της διαδικασίας LengthLCS που παρουσιάζεται παρακάτω είναι $\Theta(mn)$ αφού κάθε στοιχείο του πίνακα c υπολογίζεται σε χρόνο $\Theta(1)$.

```
LengthLCS(X, Y, m, n)
    for i = 0 to m
        c[i, 0] = 0
    for j = 0 to n
        c[0, j] = 0
    for i = 1 to m
        for j = 1 to n
            if x_i == y_j
                c[i, j] = c[i − 1, j − 1] + 1
                b[i, j] = ' ↖'
            elseif c[i − 1, j] ≥ c[i, j − 1]
                c[i, j] = c[i − 1, j]
                b[i, j] = ' ↑'
            else
                c[i, j] = c[i, j − 1]
                b[i, j] = ' ←'
```

Έχοντας υπολογίσει τους πίνακες c και b, μια μέγιστη κοινή υπακολουθία των X και Y εκτυπώνεται από τη διαδικασία PrintLCS:

```
PrintLCS(b, X, Y, i, j)
    if i == 0 or j == 0
        return
    if b[i, j] == ' ↖'
        PrintLCS(b, X, i − 1, j − 1)
        print x_i
    elseif b[i, j] == ' ↑'
        PrintLCS(b, X, i − 1, j)
    else
        PrintLCS(b, X, i, j − 1)
```

Ο χρόνος εκτέλεσης της PRINTLCS(b, X, m, n) είναι $\Theta(m+n)$, αφού είτε το i είτε το j μειώνονται κατά ένα σε κάθε αναδρομική κλήση.

Παράδειγμα. Θεωρούμε τις ακολουθίες $X = \{A, B, C, B, D, A, B\}$ και $Y = \{B, D, C, A, B, A\}$. Οι πίνακες b και c που υπολογίζει η διαδικασία LENGTHLCS φαίνονται στο Σχήμα 4.9. Το μήκος μιας μέγιστης κοινής ακολουθίας των X και Y είναι η τιμή του στοιχείου $c[7, 6]$. Η ανακατασκευή των στοιχείων της ΜΚΥ γίνεται ακολουθώντας τα βέλη από το στοιχείο $[7, 6]$. Τα τετράγωνα με μπλέ φόντο και το σύμβολο "$\nwarrow$" αντιστοιχούν σε στοιχεία με $x_i = y_j$. Στην πραγματικότητα, ο πίνακας b δεν χρειάζεται: αν η τιμή $c[i, j]$ είναι γνωστή τότε μπορούμε σε χρόνο $O(1)$ να δούμε ποια από τις τιμές $c[i-1, j-1]$, $c[i-1, j]$ ή $c[i, j-1]$ χρησιμοποιήθηκε για τον υπολογισμό της.

4.3 Το πρόβλημα του σακιδίου

Το πρόβλημα του σακιδίου, ένα κλασικό πρόβλημα συνδυαστικής βελτιστοποίησης, εμφανίζεται σε διαδικασίες λήψης αποφάσεων σε πολλούς τομείς, από την επιλογή επενδύσεων και χαρτοφυλακίων μέχρι σε συστήματα παραγωγής και αποθεμάτων, τη δειγματοληψία και τα τηλεπικοινωνιακά συστήματα. Στην πιο απλή του μορφή το πρόβλημα του σακιδίου αφορά στην εύρεση μεταξύ n αντικειμένων με βάρη $w_1, w_2, \ldots, w_n$ και αξίες $v_1, v_2, \ldots v_n$, ένα υποσύνολο με άθροισμα βαρών το πολύ W και συνολική αξία όσο μεγαλύτερη γίνεται. Υποθέτουμε, φυσικά, ότι $w_i \leq W$, για κάθε $1 \leq i \leq n$.

Ορίζουμε $F(i, w)$ να είναι η λύση του προβλήματος του σακιδίου για τα αντικείμενα $\{1, \ldots, i\}$ με τη χωρητικότητα του σακιδίου να είναι w, $0 \leq w \leq W$. Θα θέσουμε $F(0, w) = 0$, $0 \leq w \leq W$ και $F(i, 0) = 0$, $0 \leq i \leq n$. Αναζητούμε την τιμή $F(n, W)$. Είναι φανερό ότι αν η βέλτιστη επιλογή των αντικειμένων δεν περιλαμβάνει το αντικείμενο i τότε $F(i, w) = F(i-1, w)$. Αντίθετα, αν το αντικείμενο i είναι στη βέλτιστη επιλογή τότε $F(i, w) = v_i + F(i-1, w-w_i)$, υπό την προϋπόθεση, βέβαια ότι $w_i \leq w$. Αυτή η παρατήρηση μας οδηγεί στην αναδρομική σχέση

$$F(i, w) = \max\{F(i-1, w), v_i + F(i-1, w-w_i)\},$$

για κάθε $1 \leq i \leq n$, $0 \leq w \leq W$, και στη διαδικασία KNAPSACK που φαίνεται παρακάτω. Ο πίνακας K που περιέχει τις τιμές $F(i, w)$ για $0 \leq i \leq n$ και $0 \leq w \leq W$. Είναι προφανές ότι η πολυπλοκότητα του αλγόριθμου είναι $O(nW)$.

```
KNAPSACK(n, v, w, W, K)
    for i = 0 to n
        K[i, 0] = 0
    for j = 0 to W
        K[0, j] = 0
    for i = 1 to n
        for j = 1 to W
            if w[i] > j
                K[i, j] = K[i − 1, j]
            else
                K[i, j] = max{K[i − 1, j], v[i] + K[i − 1, j − w[i]]}
```

Παράδειγμα. Για το πρόβλημα το σακιδίου με χωρητικότητα $W = 15$, βάρη αντικειμένων $\{2, 3, 5, 7, 1, 4, 1\}$ και αξίες $\{10, 5, 15, 7, 6, 18, 3\}$, η διαδικασία KNAPSACK παράγει τον πίνακα (δεν τυπώνεται η πρώτη

γραμμή και στήλη)

0	10	10	10	10	10	10	10	10	10	10	10	10	10	10
0	10	10	10	15	15	15	15	15	15	15	15	15	15	15
0	10	10	10	15	15	25	25	25	30	30	30	30	30	30
0	10	10	10	15	15	25	25	25	30	30	30	30	32	32
6	10	16	16	16	21	25	31	31	31	36	36	36	36	38
6	10	16	18	24	28	34	34	34	39	43	49	49	49	54
6	10	16	19	24	28	34	37	37	39	43	49	52	52	54

Μπορούμε να βρούμε ποια αντικείμενα συμπεριλαμβάνονται στο σακίδιο εξετάζοντας αν η τιμή $K[i.j]$ προέρχεται από την $K[i-1,j]$ ή από την $v[i]+K[i-1,j-w[i]]$. Το αντικείμενο i συμπεριλαμβάνεται στο σακίδιο μόνο στη δεύτερη περίπτωση. Η διαδικασία KnapsackItems παρακάτω εκτυπώνει τη λίστα των αντικειμένων του σακιδίου.

```
Knapsack(n, v, w, W, K)
    opt = K[n, W]
    j = W
    for i = n downto 1
        if opt == K[i − 1, j]
            continue
        else
            print 'i'
            opt = opt − v[i]
            j = j − w[i]
```

Για το παράδειγμα παραπάνω, τα αντικείμενα που περιλαμβάνονται στο σακίδιο είναι τα 6, 5, 3, 2 και 1.

4.4 Απόσταση διόρθωσης

Αν $s = s_1 \dots s_m$ και $t = t_1 \dots t_n$ είναι ακολουθίες χαρακτήρων, ορίζουμε την *απόσταση διόρθωσης* της s από την t ως τον ελάχιστο αριθμό πράξεων για τη μετατροπή της s στην t. Επιτρεπτές πράξεις είναι η *εισαγωγή*, η *διαγραφή* ή η *αντικατάσταση* ενός χαρακτήρα με έναν άλλο. Υποθέτουμε ότι το κόστος της αντικατάστασης c_R είναι μικρότερο από το άθροισμα του κόστους μιας διαγραφής c_D και μιας εισαγωγής c_I, διαφορετικά, η πράξη της αντικατάστασης δεν θα χρησιμοποιηθεί ποτέ. Αν $d(i,j)$, $1 \le i \le m$, $1 \le j \le n$, είναι η απόσταση διόρθωσης των ακολουθιών $s_1 \dots s_i$ και $t_1 \dots t_j$ τότε

$$d(i,j) = \begin{cases} d(i-1,j-1) & \text{αν } s_i = t_j \\ \min \begin{cases} d(i-1,j-1) + c_R \\ d(i,j-1) + c_I \\ d(i-1,j) + c_D \end{cases} & \text{αν } s_i \neq t_j, \end{cases} \tag{4.7}$$

για $1 \le i \le m$ και $1 \le j \le n$. Για τις οριακές περιπτώσεις, όπου η μία ή και οι δύο ακολουθίες χαρακτήρων είναι κενές έχουμε

$$d(i,0) = ic_D, \qquad 1 \le i \le m, \tag{4.8}$$

$$d(0,j) = jc_I, \qquad 1 \le j \le n. \tag{4.9}$$

Η αναδρομική σχέση (4.7) και οι οριακές συνθήκες (4.8) και (4.9) εύκολα μεταφράζονται σε μια υπολογιστική διαδικασία για τον υπολογισμό της απόστασης διόρθωσης. Η διαδικασία EditDistance

παρακάτω υπολογίζει έναν πίνακα D διάστασης $(m+1) \times (n+1)$ με $D[i,j]$ την απόσταση διόρθωσης των $s_1 \dots s_i$ και $t_1 \dots t_j$. Η απόσταση διόρθωσης της ακολουθίας s από την t είναι, φυσικά, $D[m,n]$.

```
EditDistance(m, s, n, t, D)
    for i = 1 to m
        D[i, 0] = i * c_D
    for j = 1 to n
        D[0, j] = j * c_I
    for i = 1 to m
        for j = 1 to n
            if s[i] == t[j]
                rcost = 0
            else
                rcost = c_R
            D[i, j] = min{d[i − 1, j] + c_D, d[i, j − 1] + c_I,
                          d[i − 1, j − 1] + rcost}
    return D[m,n]
```

Ο χρόνος εκτέλεσης της διαδικασίας EditDistance είναι $O(mn)$. Οι πράξεις που εκτελούνται για την μετατροπή της μιας ακολουθίας στην άλλη μπορούν να εκτυπωθούν ξεκινώντας από το στοιχείο $[m,n]$ και ακολουθώντας τα βήματα προς τα πίσω.

4.5 Ασκήσεις

1. Βρείτε μια μέγιστη κοινή υπακολουθία των $X = \{1,0,0,1,0,1,0,1\}$ και $Y = \{0,1,0,1,1,0,1,1,0\}$.

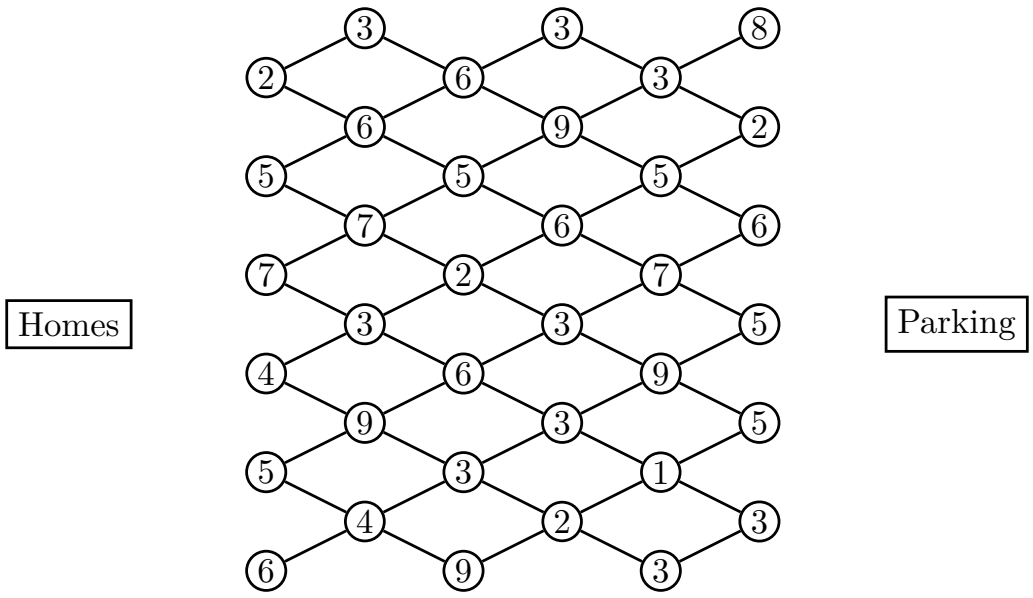
2. Δείξτε πως θα μπορούσαμε να ανακτήσουμε τα στοιχεία μιας ΜΚΥ, χρησιμοποιώντας μόνο τον πίνακα c και τις ακολουθίες X και Y, σε χρόνο $O(m+n)$.

3. Λύστε το πρόβλημα του σακιδίου για τα βάρη $\{3,4,2,6,7,3,5\}$ και αντίστοιχες αξίες $\{7,9,5,12,14,6,12\}$. Πάρτε τη χωρητικότητα του σακιδίου να είναι $W = 15$.

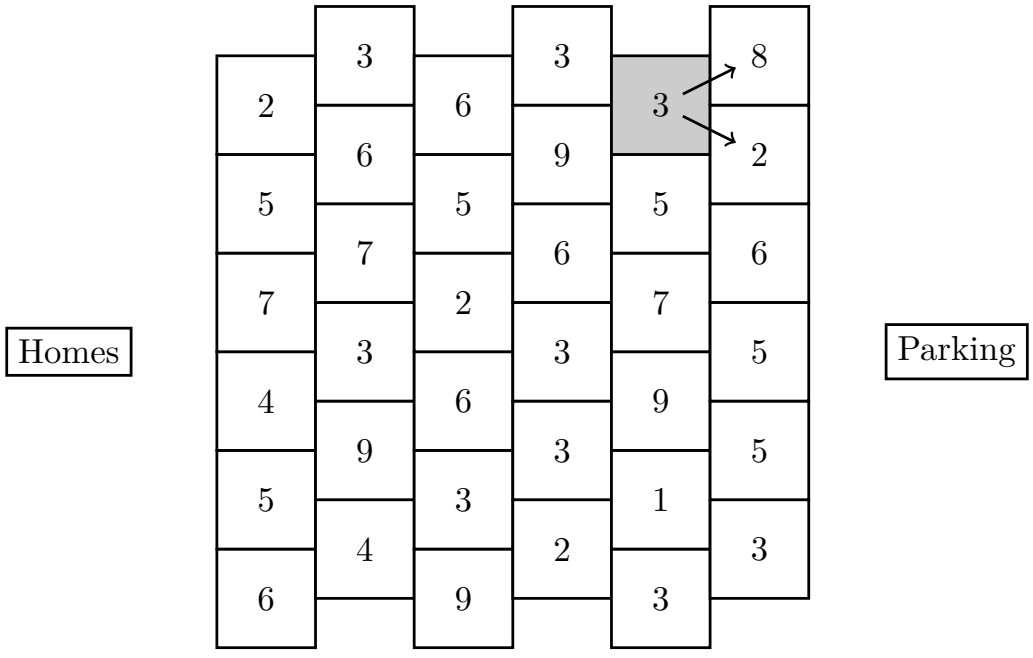
4. Βρείτε τις αποστάσεις διόρθωσης για τα ζεύγη oslo και snow, intention και execution, algorithm και altruistic. Καταγράψτε τις πράξεις που οδηγούν την μια ακολουθία χαρακτήρων στην άλλη.

5. Σχεδιάστε έναν αλγόριθμο δυναμικού προγραμματισμού για την εύρεση της μεγαλύτερης παλινδρομικής υπακολουθίας μιας ακολουθίας χαρακτήρων.

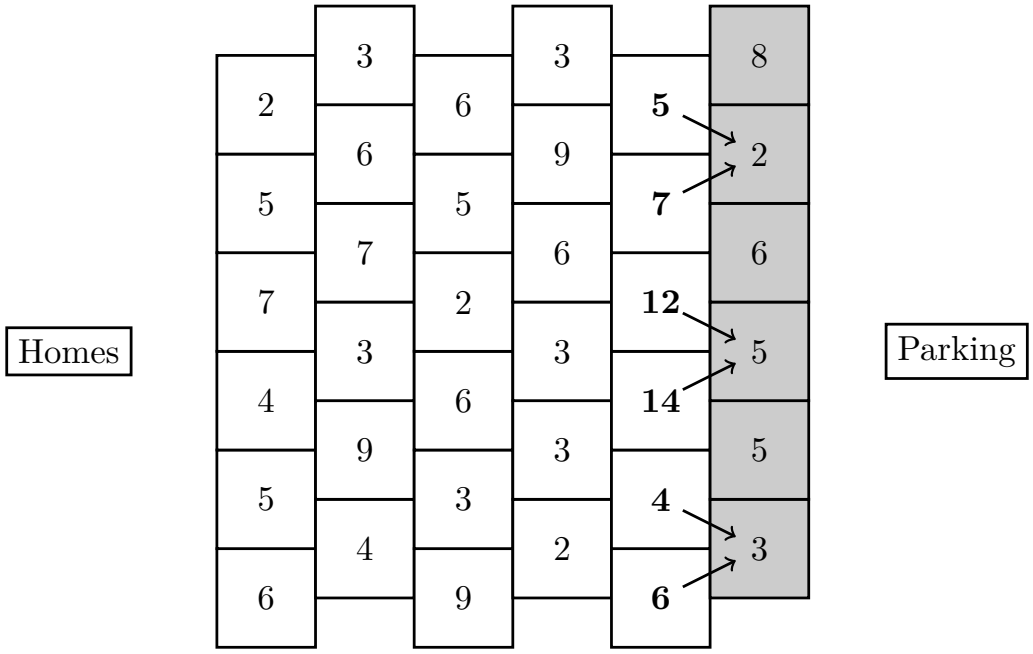
6. Σχεδιάστε έναν αλγόριθμο δυναμικού προγραμματισμού ο οποίος δεδομένου ενός $m \times n$ πίνακα A με στοιχεία από το σύνολο $\{0,1\}$ βρίσκει τον μεγαλύτερο τετραγωνικό υποπίνακα τα στοιχεία του οποίου είναι όλα μηδενικά.



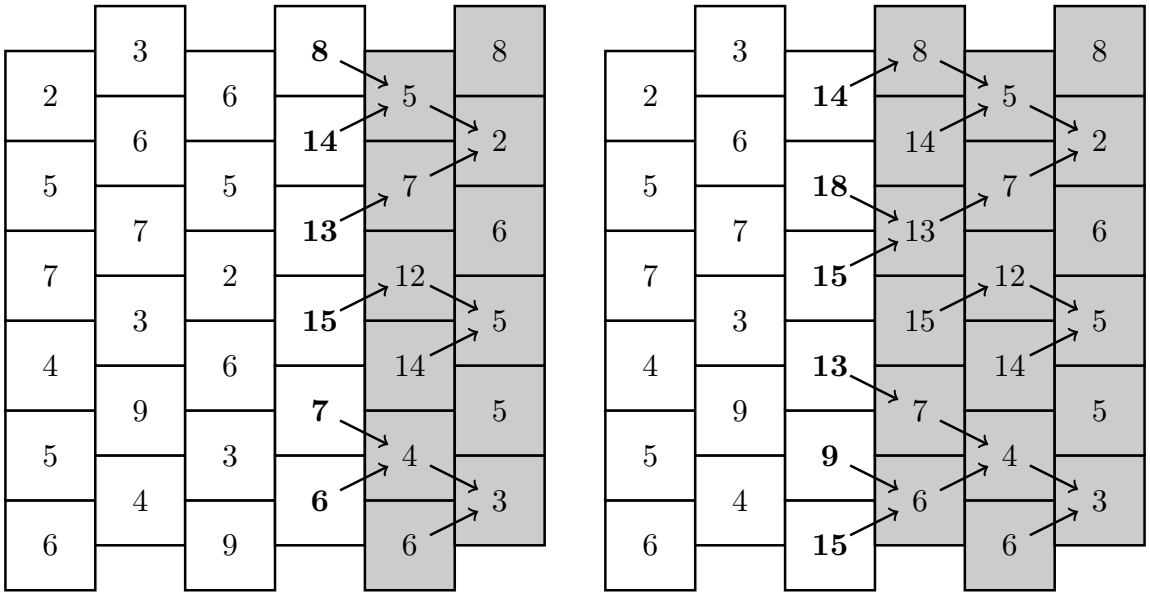
Σχήμα 4.1: Αναπαράση του δικτύου δρόμων και διασταυρώσεων.



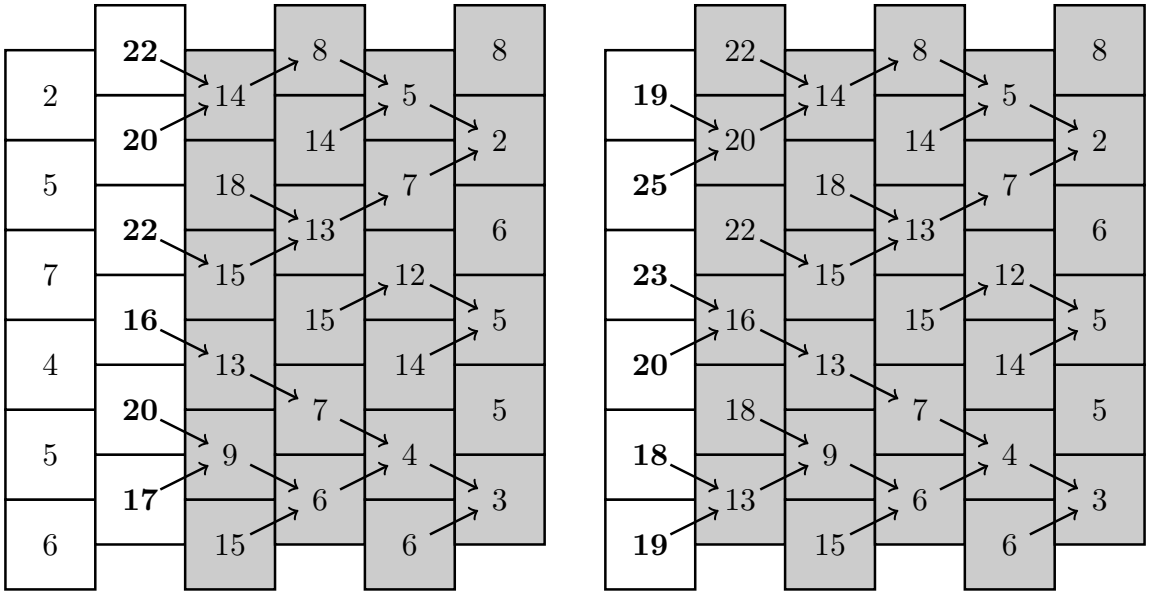
Σχήμα 4.2: Συμπαγής αναπαράση του δικτύου δρόμων και διασταυρώσεων.



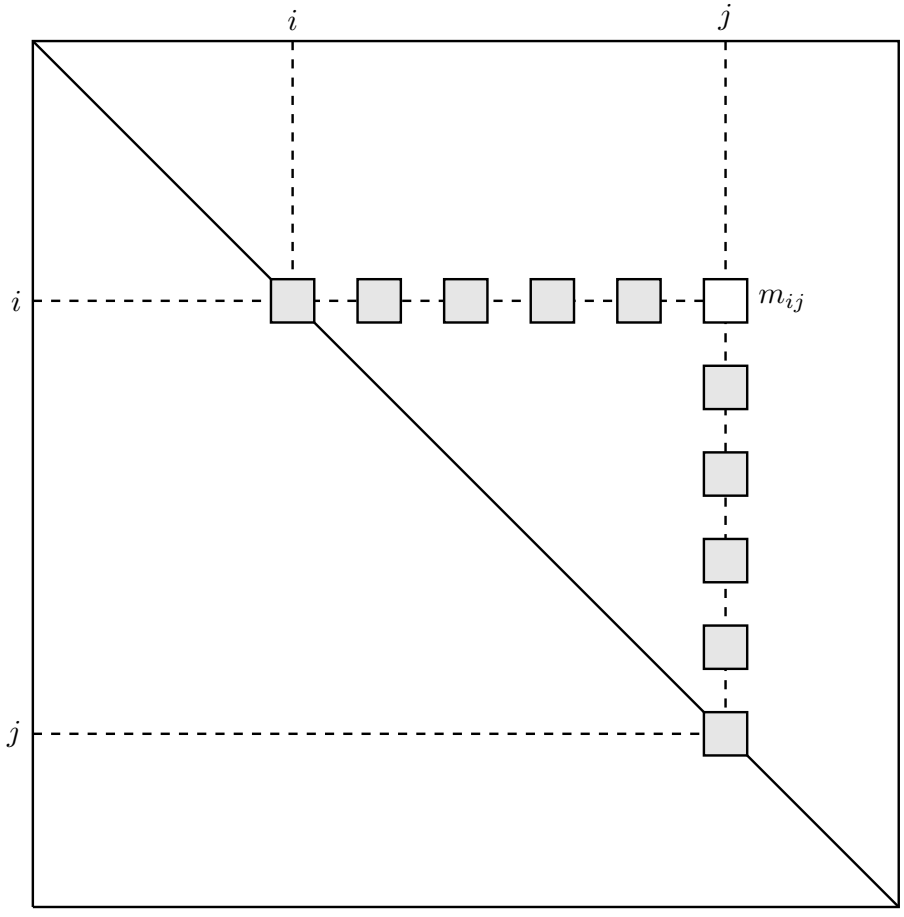
Σχήμα 4.3: Συνολικές καθυστερήσεις και διαδρομές όταν απομένει μια διασταύρωση να διανυθεί.



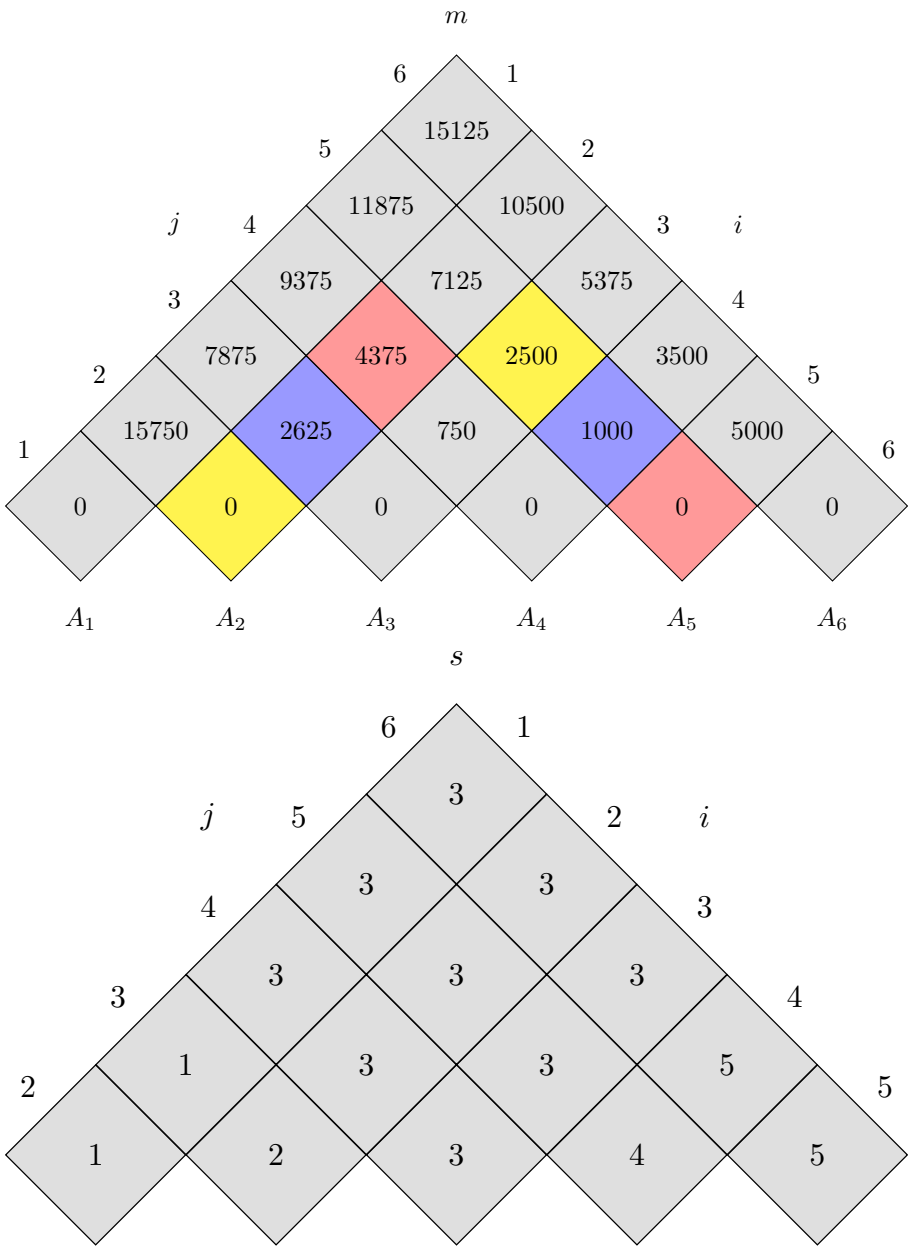
Σχήμα 4.4: Συνολικές καθυστερήσεις και διαδρομές όταν απομένουν δύο (αριστερά) ή τρεις διασταυρώσεις να διανυθούν.



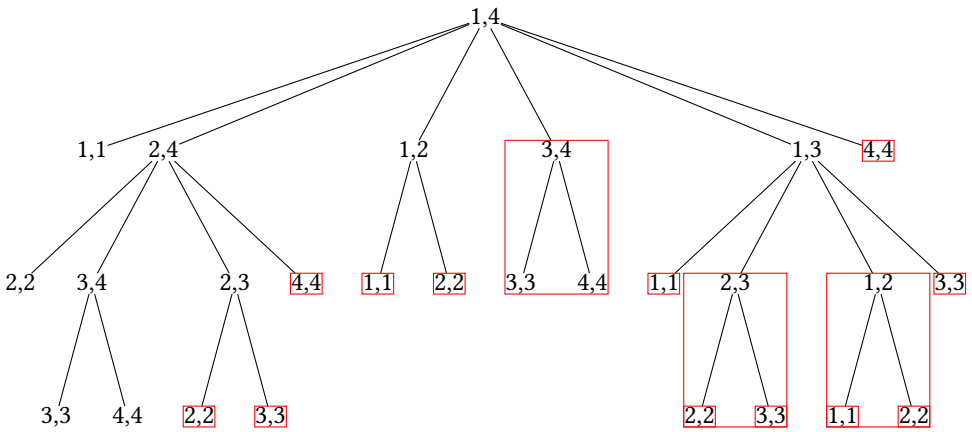
Σχήμα 4.5: Συνολικές καθυστερήσεις και διαδρομές όταν απομένουν τέσσερεις διαδρομές (αριστερά) και η βέλτιστη λύση (δεξιά).



Σχήμα 4.6: Η εξάρτηση του όρου $m[i, j]$ από τα στοιχεία $m[i, k]$ και $m[k+1, j]$ για $k = i, i+1, \ldots, j-1$, (σκιασμένα).



Σχήμα 4.7: Οι πίνακες m και s για τον υπολογισμό του ελάχιστου αριθμού πολλαπλασιασμών για τους πίνακες στο Παράδειγμα 1.



Σχήμα 4.8: Το δένδρο αναδρομής για τον υπολογισμό του RecMult$(p, 1, 4)$. Κόμβοι ή υποδένδρα σε κόκκινο πλαίσιο αντικαθιστούνται από αναφορά σε στοιχείο του πίνακα m στη διαδικασία RecMult().

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Σχήμα 4.9: Οι πίνακες b και c που παράγει η διαδικασία LengthLCS για τις ακολουθίες $X = \{A, B, C, B, D, A, B\}$ και $Y = \{B, D, C, A, B, A\}$.

ΚΕΦΑΛΑΙΟ 5

Άπληστοι αλγόριθμοι

Ένας *άπληστος αλγόριθμος* (*greedy algorithm*) ακολουθεί την ευρετική (heuristic) επίλυση προβλημάτων κάνοντας σε κάθε βήμα μια τοπικά βέλτιστη επιλογή, με την ελπίδα τελικά να βρεί μια ολική βέλτιστη λύση του προβλήματος, με την επιλογή αυτή να μην επιτρέπεται να αναθεωρηθεί αργότερα. Σε πολλά προβλήματα η άπληστη στρατηγική δουλεύει καλά. Σκεφτείτε το πρόβλημα του σχηματισμού του ποσού των 39 λεπτών του ευρώ με τον ελάχιστο αριθμό κερμάτων. Αν, σε κάθε βήμα, επιλέξουμε το κέρμα με τη μεγαλύτερη αξία η οποία δεν ξεπερνάει το τρέχον ποσό τότε οδηγούμαστε στη βέλτιστη λύση:

$$38 - 20 = 18, \quad 18 - 10 = 8, \quad 8 - 5 = 3, \quad 3 - 2 = 1, \quad 1 - 1 = 0.$$

Είναι όμως εύκολο να δει κανείς ότι η άπληστη στρατηγική δεν οδηγεί πάντοτε στη βέλτιστη λύση: για παράδειγμα, στο πρόβλημα της επιλογής του ελάχιστου αριθμού γραμματοσήμων με αξίες 1, 5, 10, 20, 35, 70, και 100 λεπτών του ευρώ, για ταχυδρομικά τέλη $1,40$ ευρώ η άπληστη στρατηγική θα έκανε την επιλογή $1 \times 100, 1 \times 35, 1 \times 5$. Αντίθετα, η βέλτιστη λύση είναι, προφανώς, η επιλογή δύο γραμματοσήμων των 70 λεπτών το καθένα.

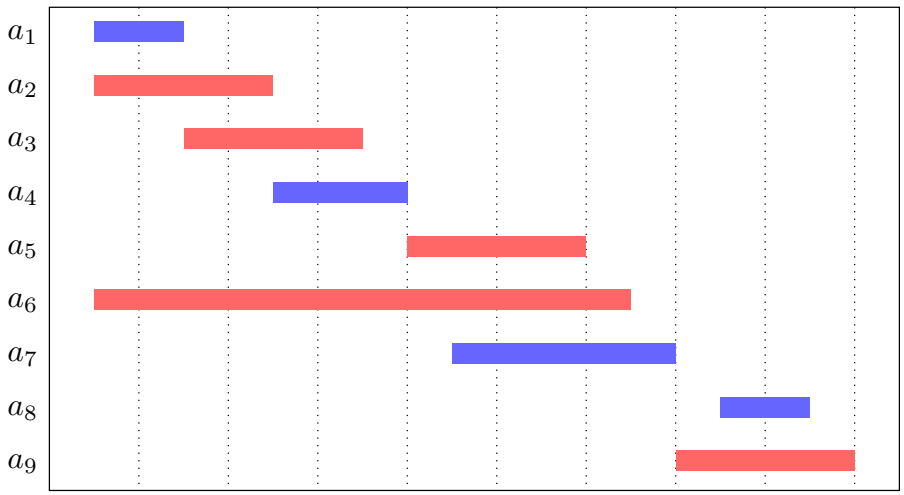
Είναι εύκολο να επινοήσει κανείς άπληστους αλγόριθμους για σχεδόν οποιοδήποτε πρόβλημα. Δυσκολότερο όμως είναι να βρει περιπτώσεις όπου λειτουργούν καλά, και, βέβαια, να αποδείξει ότι αποδίδουν.

Θεωρούμε ένα πρόβλημα χρονοπρογραμματισμού δραστηριοτήτων, όπου n δραστηριότητες $S = \{a_1, a_2, \ldots, a_n\}$ με χρόνους έναρξης s_i, χρόνους λήξης f_i, τέτοιοι ώστε $0 \leq s_i < f_i$, $i = 1, \ldots n$, ανταγωνίζονται για έναν πόρο. Θα λέμε ότι δύο δραστηριότητες a_i και a_j είναι *συμβατές* αν τα διαστήματα $[s_i, f_i)$ και $[s_j, f_j)$ δεν επικαλύπτονται. Θέλουμε να βρούμε ένα όσο το δυνατόν μεγαλύτερο υποσύνολο αμοιβαία συμβατών δραστηριοτήτων. Θα υποθέσουμε ότι οι δραστηριότητες είναι διατεταγμένες έτσι ώστε

$$f_1 \leq f_2 \leq \cdots \leq f_{n-1} \leq f_n. \tag{5.1}$$

Για παράδειγμα, αν το σύνολο δραστηριοτήτων είναι αυτό που παρουσιάζεται στο Σχήμα 5.1, τότε ένα υποσύνολο αμοιβαία συμβατών δραστηριοτήτων είναι το $\{1, 4, 8\}$ αλλά το βέλτιστο υποσύνολο αμοιβαία συμβατών δραστηριοτήτων είναι το $\{1, 4, 7, 8\}$.

Δείχνουμε κατ' αρχάς ότι το πρόβλημα του χρονοπρογραμματισμού δραστηριοτήτων έχει βέλτιστη υποδομή. Έστω S_{ij} το σύνολο των δραστηριοτήτων με χρόνους εκκίνησης μεγαλύτερους του s_i και χρόνους τερματισμού μικρότερους του s_j. Έστω A_{ij} ένα μέγιστο σύνολο αμοιβαία συμβατών δραστηριοτήτων του S_{ij} και $a_k \in A_{ij}$. Αν η δραστηριότητα a_k συμπεριλαμβάνεται στη βέλτιστη λύση



Σχήμα 5.1: Ένα σύνολο εννέα δραστηριοτήτων. Το $\{1, 4, 8\}$ είναι ένα υποσύνολο αμοιβαία συμβατών δραστηριοτήτων και το $\{1, 4, 7, 8\}$ είναι ένα βέλτιστο υποσύνολο αμοιβαία συμβατών δραστηριοτήτων.

τότε μένει να εξετάσουμε δύο υποπροβλήματα: να βρούμε το μέγιστο σύνολο των αμοιβαία συμβατών δραστηριοτήτων από το S_{ik} και το μέγιστο σύνολο των αμοιβαία συμβατών δραστηριοτήτων από το S_{kj}. Έστω $A_{ik} = A_{ij} \cap S_{ik}$, δηλαδή, το σύνολο των δραστηριοτήτων του A_{ij} με χρόνο τερματισμού μικρότερου του s_k, και $A_{kj} = A_{ij} \cap S_{kj}$, δηλαδή, το σύνολο των δραστηριοτήτων του A_{ij} με χρόνο εκκίνησης μεγαλύτερο του f_k. Συνεπώς, $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$, δηλαδή, $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$ είναι το πλήθος των αμοιβαία συμβατών δραστηριοτήτων του S_{ij}. Η σχέση αυτή μας επιτρέπει να αποδείξουμε ότι η βέλτιστη λύση A_{ij} περιέχει βέλτιστες λύσεις για τα υποπροβλήματα S_{ik} και S_{kj}: αν υπήρχε σύνολο A'_{kj} αμοιβαία συμβατών δραστηριοτήτων του S_{kj} με $|A'_{kj}| > |A_{kj}|$, τότε θα μπορούσαμε να χρησιμοποιήσουμε αυτό για την κατασκευή μιας βέλτιστης λύσης για το S_{ij}. Τότε όμως, $|A_{ik}| + |A'_{kj}| + 1 > |A_{ik}| + |A_{kj}| + 1 = |A_{ij}|$, γεγονός που έρχεται σε αντίθεση με την υπόθεση ότι A_{ij} είναι βέλτιστη λύση. Ακριβώς ανάλογο επιχείρημα ισχύει και για τη βέλτιστη λύση A_{ik}.

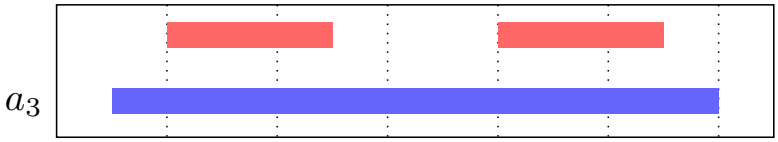
Αυτός ο χαρακτηρισμός της βέλτιστης λύσης μας επιτρέπει να χρησιμοποιήσουμε την τεχνική του δυναμικού προγραμματισμού για να λύσουμε το πρόβλημα του χρονοπρογραμματισμού δραστηριοτήτων. Αν $c[i, j]$ είναι το μέγεθος μιας βέλτιστης λύσης για το σύνολο S_{ij} τότε $c[i, j] = c[i, k]+c[k, j]+1$. Αφού η δραστηριότητα a_k δεν είναι εκ των προτέρων γνωστή πρέπει να τις εξετάσουμε όλες, έτσι ώστε

$$c[i,j] = \begin{cases} 0 & \text{αν } S_{ij} = \emptyset, \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{αν } S_{ij} \neq \emptyset. \end{cases} \tag{5.2}$$

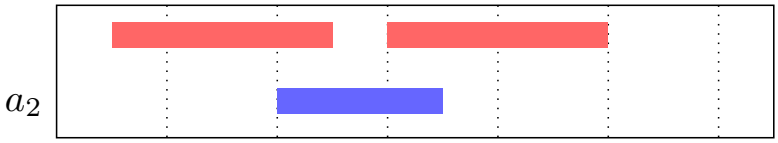
Τα στοιχεία του πίνακα c μπορούν να υπολογιστούν από κάτω προς τα πάνω, δηλαδή, από τις μικρότερες τιμές των i και j προς τις μεγαλύτερες, δηλαδή, από τα μικρότερα υποπροβλήματα προς τα μεγαλύτερα. Θα μπορούσε όμως να σκεφτούμε κατά πόσο είναι απαραίτητη η λύση *όλων* των υποπροβλημάτων προτού προστεθεί κάποια δραστηριότητα στο βέλτιστο σύνολο A_{ij}.

Δοκιμάζουμε στη συνέχεια κάποιους απλούς κανόνες για την επιλογή ενός συνόλου αμοιβαία συμβατών δραστηριοτήτων και εξετάζουμε κατά πόσο οδηγούν σε βέλτιστη λύση του προβλήματος μας.

Επιλέγουμε σε κάθε βήμα τη δραστηριότητα η οποία ξεκινάει νωρίτερα (αυτή με το μικρότερο s_i): Για το σύνολο δραστηριοτήτων στο Σχήμα 5.1 η πολιτική αυτή οδηγεί στο υποσύνολο δραστηριοτήτων $\{1, 3, 5, 9\}$, το οποίο αποτελεί βέλτιστη λύση για το πρόβλημα μας. Όμως, δεν οδηγεί πάντα σε βέλτιστη λύση, όπως φαίνεται στο Σχήμα 5.2. Η δραστηριότητα a_3 έχει τον μικρότερο χρόνο εκκίνησης αλλά η επιλογή της στο σύνολο των αμοιβαία συμβατών δραστηριοτήτων αποκλείει την επιλογή σε αυτό των άλλων δύο δραστηριοτήτων.



Σχήμα 5.2: Η επιλογή, σε κάθε βήμα, της δραστηριότητας με τον μικρότερο χρόνο εκκίνησης δεν οδηγεί πάντα σε βέλτιστη λύση.

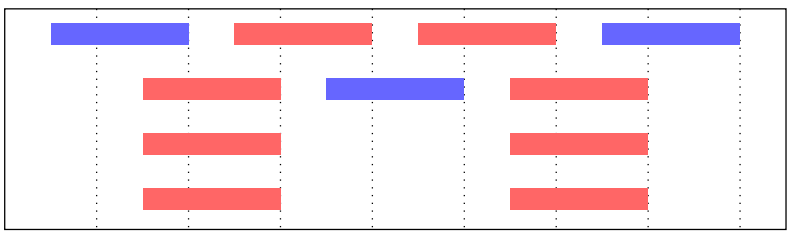


Σχήμα 5.3: Η επιλογή, σε κάθε βήμα, της δραστηριότητας με τη μικρότερη διάρκεια δεν οδηγεί πάντα σε βέλτιστη λύση.

Επιλέγουμε σε κάθε βήμα τη δραστηριότητα η οποία έχει τη μικρότερη διάρκεια (αυτή για την οποία η διαφορά $f_i - s_i$ είναι η μικρότερη δυνατή): Για το σύνολο δραστηριοτήτων στο Σχήμα 5.1 η πολιτική αυτή οδηγεί στο υποσύνολο $\{1, 4, 5, 8\}$, το οποίο αποτελεί βέλτιστη λύση. Αυτό όμως δεν συμβαίνει για κάθε σύνολο δραστηριοτήτων, όπως φαίνεται στο Σχήμα 5.3. Η επιλογή της διαδικασίας a_2 στη βέλτιστη λύση αποκλείει τις άλλες δύο δραστηριότητες οι οποίες αποτελούν βέλτιστη λύση για το συγκεκριμένο πρόβλημα χρονοπρογραμματισμού.

Επιλέγουμε σε κάθε βήμα τη δραστηριότητα η οποία έχει τον μικρότερο αριθμό μη συμβατών δραστηριοτήτων: Για το σύνολο δραστηριοτήτων στο Σχήμα 5.1 η πολιτική αυτή οδηγεί στο υποσύνολο $\{1, 3, 7, 8\}$, το οποίο αποτελεί βέλτιστη λύση. Όμως για το πρόβλημα χρονοπρογραμματισμού που φαίνεται στο Σχήμα 5.4 η συγκεκριμένη πολιτική επιλογής οδηγεί στις δραστηριότητες που σημειώνονται με μπλε χρώμα και οι οποίες δεν αποτελούν βέλτιστο σύνολο.

Επιλέγουμε σε κάθε βήμα τη δραστηριότητα η οποία ολοκληρώνεται νωρίτερα (αυτή με τον μικρότερο χρόνο f_i): χαρακτηρίζουμε τη συγκεκριμένη επιλογή ως *άπληστη* (*greedy*) με την λογική ότι ο χρόνος που απομένει για τις υπόλοιπες δραστηριότητες είναι ο μεγαλύτερος δυνατός. Στην περίπτωση που οι δραστηριότητες διατάσσονται σύμφωνα με τον χρόνο τερματισμού, όπως στην (5.1), τότε η άπληστη επιλογή στο πρώτο βήμα είναι η δραστηριότητα a_1. Η συγκεκριμένη επιλογή οδηγεί σε ένα και μόνο υποπρόβλημα: την εύρεση όλων των δραστηριοτήτων οι οποίες εκκινούν μετά το τέλος της a_1. Επειδή όμως $s_1 < f_1$ καμιά δραστηριότητα δεν μπορεί να έχει χρόνο τερματισμού μικρότερου του s_1. Αυτό σημαίνει ότι όλες οι συμβατές με την a_1 δραστηριότητες πρέπει να εκκινούν μετά το τέλος της a_1. Αν ορίσουμε $S_k = \{a_i \in S : s_i \geq f_k\}$, το σύνολο των δραστηριοτήτων που εκκινούν μετά τον τερματισμό της δραστηριότητας a_k τότε, η άπληστη επιλογή a_1 οδηγεί στο μοναδικό υποπρόβλημα S_1. Από τη βέλτιστη υποδομή του προβλήματος μας συμπεραίνουμε ότι αν η δραστηριότητα a_1 είναι στη βέλτιστη λύση τότε η βέλτιστη λύση του προβλήματος είναι αυτή η δραστηριότητα και οι δραστηριότητες της βέλτιστης λύσης του υποπροβλήματος S_1. Το θεώρημα



Σχήμα 5.4: Η επιλογή, σε κάθε βήμα, της δραστηριότητας με τον μικρότερο αριθμό μη συμβατών δραστηριοτήτων δεν οδηγεί πάντα σε βέλτιστη λύση.

που ακολουθεί δείχνει ότι η δραστηριότητα a_1 είναι όντως στο σύνολο των δραστηριοτήτων της βέλτιστης λύσης.

Θεώρημα. Έστω S_k ένα υποπρόβλημα του προβλήματος χρονοπρογραμματισμού δραστηριοτήτων και a_m μια δραστηριότητα του $S-_k$ με τον μικρότερο χρόνο τερματισμού. Τότε η δραστηριότητα a_m περιλαμβάνεται σε ένα υποσύνολο αμοιβαία συμβατών δραστηριοτήτων του S_k μέγιστου μεγέθους.

Απόδειξη. Έστω A_k ένα υποσύνολο αμοιβαία συμβατών δραστηριοτήτων του S_k μέγιστου μεγέθους και a_j μια δραστηριότητα του A_k με τον μικρότερο χρόνο τερματισμού. Αν $a_j = a_m$ τότε το θεώρημα έχει αποδειχθεί. Διαφορετικά, θεωρούμε το σύνολο $A'_k = (A_k \backslash \{a_j\}) \cup \{a_m\}$. Όλες οι δραστηριότητες του A'_k είναι συμβατές γιατί οι δραστηριότητες του A_k είναι συμβατές, ενώ η δραστηριότητα a_j είναι η δραστηριότητα με τον μικρότερο χρόνο τερματισμού στο A_k και $f_m \leq f_j$. Επειδή $|A'_k| = |A_k|$ συμπεραίνουμε ότι το σύνολο A'_k είναι ένα σύνολο αμοιβαία συμβατών δραστηριοτήτων του S_k μέγιστου μεγέθους το οποίο περιλαμβάνει τη δραστηριότητα a_m. □

Το παραπάνω θεώρημα επιβεβαιώνει ότι η άπληστη επιλογή, δηλαδή η επιλογή σε κάθε βήμα της δραστηριότητας με τον μικρότερο χρόνο τερματισμού οδηγεί σε βέλτιστη λύση του προβλήματος. Επιπλέον, αφού κάθε φορά επιλέγεται η δραστηριότητα με τον μικρότερο χρόνο τερματισμού, οι χρόνοι τερματισμού των δραστηριοτήτων της βέλτιστης λύσης αυξάνονται μονότονα. Η διαδικασία GreedySelection παρακάτω επιλέγει σε κάθε βήμα της τη διαδικασία με τον μικρότερο χρόνο τερματισμού υποθέτοντας ότι οι δραστηριότητες ικανοποιούν τη συνθήκη (5.1).

GreedySelection(n, s, f)
$A = \{a_1\}$
$k = 1$
for $m = 2$ **to** n
  if $s[m] \geq f[k]$
    $A = A \cup \{a_m\}$
    $k = m$
return A

Ο δείκτης k είναι ο δείκτης της πιο πρόσφατης δραστηριότητας που προστέθηκε στο A. Αφού οι δραστηριότητες διατάσσονται κατά αύξοντα χρόνο τερματισμού, ο χρόνος f_k είναι μέγιστος χρόνος τερματισμού των δραστηριοτήτων στο σύνολο A. Η επανάληψη **for** διατρέχει όλες τις δραστηριότητες και επιλέγει την δραστηριότητα του S_k με τον μικρότερο χρόνο τερματισμού. Αυτή προστίθεται στο σύνολο A εφόσον είναι συμβατή με όλες τις προηγούμενες διαδικασίες. Γι' αυτό αρκεί να ελεγχθεί ότι ο χρόνος έναρξης είναι μεγαλύτερος από τον χρόνο τερματισμού της πιο πρόσφατης προσθήκης στο σύνολο A.

5.0.1 Στοιχεία της άπληστης πολιτικής

Ένας άπληστος αλγόριθμος βρίσκει τη βέλτιστη λύση ενός προβλήματος κάνοντας μια σειρά επιλογών. Σε κάθε σημείο απόφασης, ο αλγόριθμος κάνει την επιλογή που φαίνεται καλύτερη εκείνη τη στιγμή. Αυτή η ευρετική (heuristic) στρατηγική δεν παράγει πάντα βέλτιστη λύση, αλλά όπως στο πρόβλημα επιλογής δραστηριότητας, μερικές φορές το κάνει. Τα βήματα τα οποία ακολουθούνται στο σχεδιασμό άπληστων αλγόριθμων, όπως είδαμε στο πρόβλημα χρονοπρογραμματισμού δραστηριοτήτων, είναι τα ακόλουθα:

1. Να βεβαιωθούμε ότι η επιλογή που γίνεται σε κάθε βήμα οδηγεί στη λύση ενός μόνο υποπροβλήματος.

2. Να δείξουμε ότι άπληστη επιλογή είναι ασφαλής, δηλαδή, ότι υπάρχει πάντοτε μια βέλτιστη λύση του αρχικού προβλήματος όταν γίνεται αυτή η επιλογή.

3. Να δείξουμε ότι το πρόβλημα έχει βέλτιστη υποδομή, έτσι ώστε, η άπληστη επιλογή και η βέλτιστη λύση του μοναδικού υποπροβλήματος συναδυαζόμενες να παρέχουν τη λύση του αρχικού προβλήματος.

Οι άπληστοι αλγόριθμοι, σε σύγκριση με τους αλγόριθμους δυναμικού προγραμματισμού, συνήθως αποτυγχάνουν να βρουν μια βέλτιστη λύση γιατί κάνουν επιλογές πολύ νωρίς, τις οποίες δεν μπορούν να αναιρέσουν. Είναι όμως απλοί, γρήγοροι, εύκολα υλοποιήσιμοι και πολλές φορές δίνουν καλές προσεγγίσεις της βέλτιστης λύσης. Ο αλγόριθμος για την εύρεση βέλτιστων κωδίκων Huffman που ακολουθεί είναι ένα ακόμα παράδειγμα άπληστου αλγόριθμου που οδηγεί σε βέλτιστη λύση.

5.0.2 Κώδικες Huffman

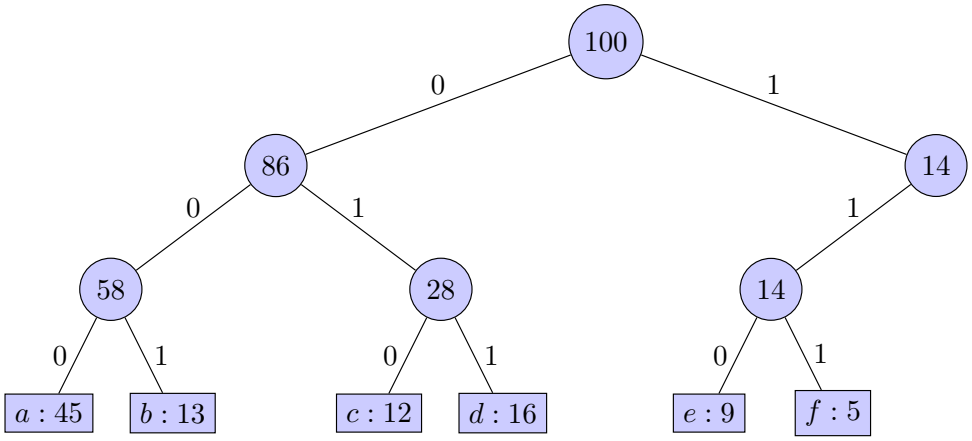
Η κωδικοποίηση Huffman είναι μια τεχνική συμπίεσης δεδομένων για τη μείωση του μεγέθους τους χωρίς την απώλεια πληροφορίας. Είναι ιδιαίτερα χρήσιμη για τη συμπίεση δεδομένων στα οποία υπάρχουν συχνά εμφανιζόμενοι χαρακτήρες. Αναπτύχθηκε από τον David Huffman το 1952. Ο άπληστος αλγόριθμος του Huffman χρησιμοποιεί τις συχνότητες εμφάνισης των χαρακτήρων για να κατασκευάσει έναν βέλτιστο τρόπο αναπαράστασης τους ως σειρά δυαδικών ψηφίων. Σε κάθε χαρακτήρα αντιστοιχούμε έναν δυαδικό κωδικό (μια ακολουθία δυαδικών ψηφίων) με τέτοιο τρόπο ώστε να ελαχιστοποιηθεί ο συνολικός αριθμών δυαδικών ψηφίων που απαιτούνται. Εύκολα βλέπουμε ότι αν επιμένουμε σε κωδικούς σταθερού μήκους, τότε για την κωδικοποίηση $n \geq 2$ χαρακτήρων απαιτούνται $\lfloor \lg n \rfloor + 1$ δυαδικά ψηφία[1]. Για παράδειγμα, αν $n = 6$ τότε απαιτούνται κώδικες με 3 δυαδικά ψηφία, π.χ., οι 000, 001, 010, 011, 100 και 101. Συνεπώς, αν μια ακολουθία 100 000 στοιχείων αποτελείται από ακριβώς 6 διαφορετικούς χαρακτήρες τότε η αναπαράστασή της απαιτεί 300 000 δυαδικά ψηφία. Μπορούμε όμως να επιτύχουμε μια οικονομικότερη αναπαράσταση χρησιμοποιώντας κωδικούς μεταβλητού μήκους, όπως αυτοί που φαίνονται στον Πίνακα 5.1. Θα θεωρήσουμε

	a	b	c	d	e	f
Συχνότητα ($\times 10^3$)	45	13	12	16	9	5
Σταθερό μήκος	000	001	010	011	100	101
Μεταβλητό μήκος	0	101	100	111	1101	1100

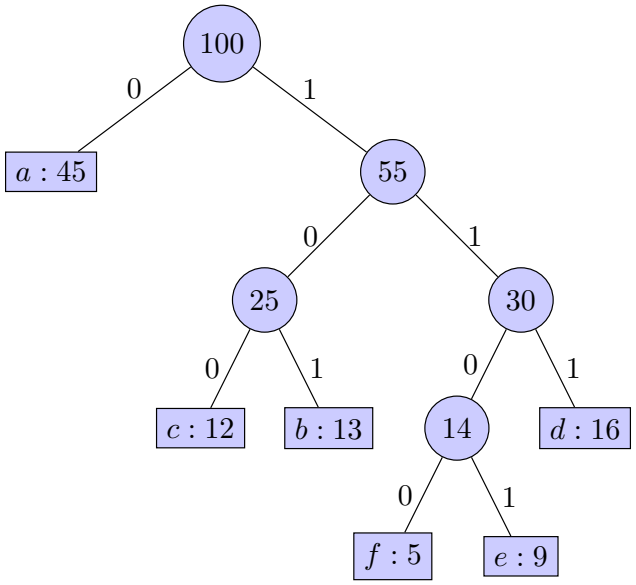
Πίνακας 5.1: Κωδικοί σταθερού και μεταβλητού μήκους για την αναπαράσταση έξι χαρακτήρων. Μια ακολουθία 100 000 στοιχείων όπου εμφανίζονται οι έξι αυτοί χαρακτήρες με τις συχνότητες που σημειώνονται, απαιτεί 224 000 δυαδικά ψηφία για την αναπαράστασή της με χρήση κωδίκων μεταβλητού μήκους.

μόνο *απροθηματικούς κώδικες* (*prefix-free codes*) δηλαδή κώδικες στους οποίους κανένας κωδικός δεν αποτελεί πρόθεμα άλλου κωδικού. Η κωδικοποίηση χαρακτήρων επιτυγχάνεται συνενώνοντας τους κωδικούς για κάθε χαρακτήρα. Έτσι, για τον κώδικα του Πίνακα 5.1 η ακολουθία χαρακτήρων `face` κωδικοποιείται ως 1100 0 100 1101. Η αποκωδικοποίηση είναι επίσης απλή: αφού κανένας κωδικός δεν εμφανίζεται ως πρόθεμα άλλου κωδικού, ο κωδικός στην αρχή μιας κωδικοποίησης είναι σαφώς καθορισμένος. Το ίδιο, προφανώς, ισχύει και για την υπόλοιπη ακολουθία κωδικοποιημένων χαρακτήρων. Παρ' όλα αυτά, απαιτείται μια βολική αναπαράσταση ενός απροθηματικού κώδικα η οποία να κάνει εύκολη την ταυτοποίηση κάθε κωδικού. Μια τέτοια αναπαράσταση παρέχεται από ένα δυαδικό δένδρο με τους καταληκτικούς κόμβους να αντιστοιχούν στους χαρακτήρες που κωδικοποιούνται. Ο κωδικός που αντιστοιχεί σε κάθε χαρακτήρα μπορεί να βρεθεί ξεκινώντας από τη ρίζα του δένδρου και σημειώνοντας το ψηφίο 0, αν μεταβούμε στον αριστερό θυγατρικό κόμβο, διαφορετικά το ψηφίο 1. Στο Σχήμα 5.5 εμφανίζεται το δυαδικό δένδρο που αντιστοιχεί στους κωδικούς σταθερού μήκους για την αναπαράσταση των έξι χαρακτήρων του Πίνακα 5.1. Στο Σχήμα 5.6 εμφανίζεται το δυαδικό δένδρο που αντιστοιχεί στους κωδικούς σταθερού μήκους για την αναπαράσταση των έξι χαρακτήρων του Πίνακα 5.1. Παρατηρήστε ότι, σε αντίθεση με το δυαδικό δένδρο που αντιστοιχεί

[1] Έχουμε $\lfloor \lg n \rfloor = k$ αν και μόνο αν $k \leq \lg n < k + 1$. Αυτό σημαίνει ότι $2^k \leq n < 2^{k+1}$.



Σχήμα 5.5: Το δυαδικό δένδρο που αντιστοιχεί στους κωδικούς σταθερού μήκους για την αναπαράσταση των έξι χαρακτήρων του Πίνακα 5.1.



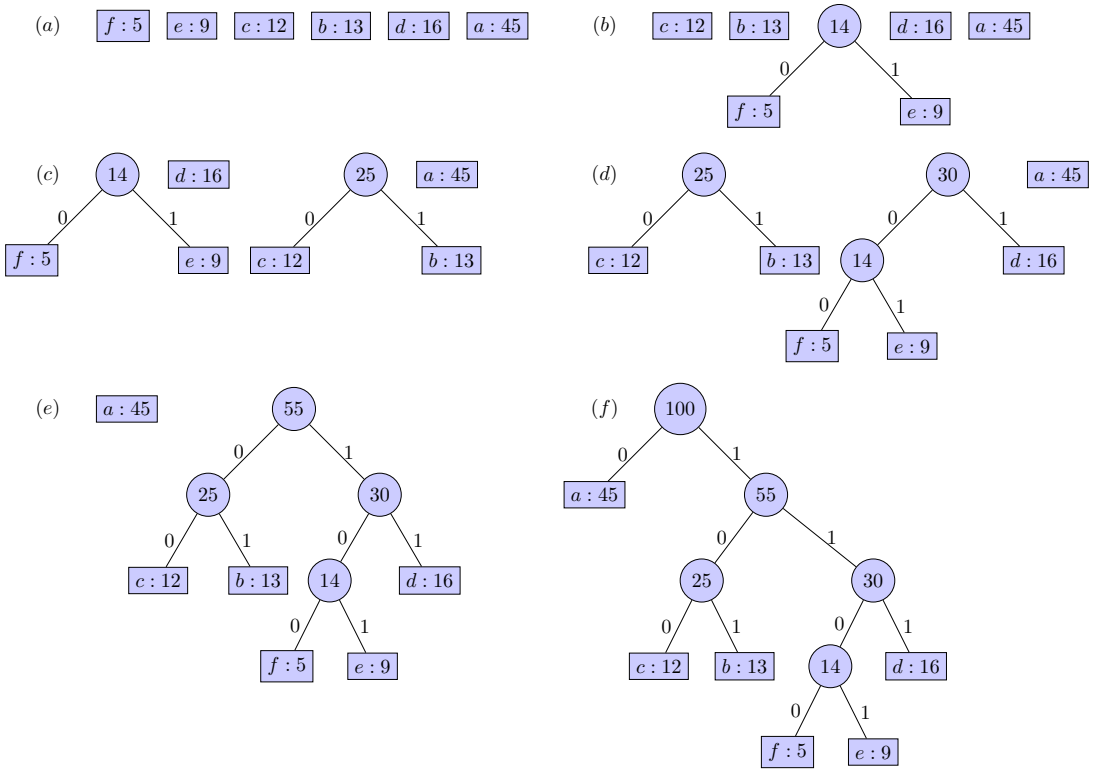
Σχήμα 5.6: Το δυαδικό δένδρο που αντιστοιχεί στους κωδικούς μεταβλητού μήκους για την αναπαράσταση των έξι χαρακτήρων του Πίνακα 5.1.

στους κωδικούς σταθερού μήκους, το δυαδικό δένδρο τώρα είναι *πλήρες*. Μπορεί να αποδειχθεί ότι βέλτιστοι κώδικες αντιπροσωπεύονται από πλήρη δυαδικά δένδρα. Αν τώρα C είναι το σύνολο των πιθανών χαρακτήρων προς κωδικοποίηση, το λεγόμενο *αλφάβητο*, τότε το δένδρο που αντιστοιχεί σε ένα βέλτιστο κώδικα έχει $|C|$ καταληκτικούς κόμβους. Αφού το δένδρο είναι πλήρες, ο αριθμός των εσωτερικών κόμβων, είναι ακριβώς $|C|-1$. Έστω c ένας χαρακτήρας από το αλφάβητο C. Θα συμβολίζουμε με $f(c)$ τη συχνότητα εμφάνισης του χαρακτήρα c και με $d_T(c)$ το βάθος[2] του καταληκτικού κόμβου που αντιστοιχεί στον χαρακτήρα c. Προφανώς, το $d_T(c)$ είναι επίσης το μήκος του κωδικού που αντιστοιχεί στον χαρακτήρα c. Συνεπώς, ο αριθμός των δυαδικών ψηφίων που απαιτούνται για την κωδικοποίηση του αλφάβητου είναι

$$B(T) = \sum_{c \in C} f(c) d_T(c), \tag{5.3}$$

ποσότητα την οποία θα ονομάζουμε *κόστος* του δένδρου T. Η διαδικασία κατασκευής ενός βέλτιστου απροθηματικού κώδικα ή *κώδικα Huffman* ξεκινάει από το σύνολο των $|C|$ καταληκτικών κόμβων και

[2] Το *βάθος* ενός κόμβου είναι ο αριθμός των ακμών από τη ρίζα του δένδρου μέχρι αυτόν τον κόμβο.



Σχήμα 5.7: Τα βήματα της κατασκευής του κώδικα Huffman για την αναπαράσταση των έξι χαρακτήρων του Πίνακα 5.1. Σε κάθε βήμα της κατασκευής τα περιεχόμενα της ουράς ελαχίστου είναι ταξινομημένα κατ' αύξουσα συχνότητα.

με μια σειρά από $|C| - 1$ συγχωνεύσεις κατασκευάζει το δένδρο που αντιστοιχεί στους κώδικες μεταβλητού μήκους. Με συγχώνευση εδώ εννοούμε την αντικατάσταση των δύο δένδρων που εκφύονται από τους κόμβους με τη μικρότερη συχνότητα με έναν κόμβο του οποίου η συχνότητα είναι το άθροισμα των συχνοτήτων των αντικειμένων που συγχωνεύτηκαν. Η κατασκευή του κώδικα Huffman για τους χαρακτήρες τον Πίνακα 5.1 φαίνεται στο Σχημα 5.7. Ο αλγόριθμος χρησιμοποιεί μαι ουρά ελαχίστου Q για να αποφασίζει ποιούς κόμβους θα συγχωνεύσει.

HuffmanCode(C)
  $n = |C|$
  $Q = C$
  for $i = 1$ **to** $n - 1$
    allocate a new node z
    $x =$ ExtractMin(Q)
    $y =$ ExtractMin(Q)
    Left(z) $= x$
    Right(z) $= y$
    $f(z) = f(x) + f(y)$
    Insert(Q, z)
  return ExtractMin(Q)

5.1 Ασκήσεις

1. Γράψτε έναν αλγόριθμο δυναμικού προγραμματισμού για τη λύση του προβλήματος του χρονοπρογραμματισμού δραστηριοτήτων ο οποίο βασίζεται στην αναδρομική σχέση (5.2). Υποθέστε ότι οι χρόνοι τερματισμού των δραστηριοτήτων είναι όπως στη σχέση (5.1).

2. Λύστε το πρόβλημα του χρονοπρογραμματισμού δραστηριοτήτων για τις δραστηριότητες με χρόνους έναρξης $\{1, 3, 0, 5, 3, 5, 6, 7, 8, 2, 12\}$ και χρόνους τερματισμού $\{4, 5, 6, 7, 9, 9, 10, 11, 12, 14, 16\}$.

3. Βρείτε έναν βέλτιστο κώδικα για την κωδικοποίηση των χαρακτήρων a-h με συχνότητες: a:1, b:1, c:2, d:3, e:5, f:8, g:13, και h:21.

4. Αποδείξτε ότι το κόστος του δένδρου $B(T)$ ενός πλήρους δυαδικού δένδρου που αντιστοιχεί σε κάποιο κώδικα ισούται με το άθροισμα πάνω σε κάθε μη καταληκτικό κόμβο των συχνοτήτων των θυγατρικών του κόμβων.

ΚΕΦΑΛΑΙΟ 6

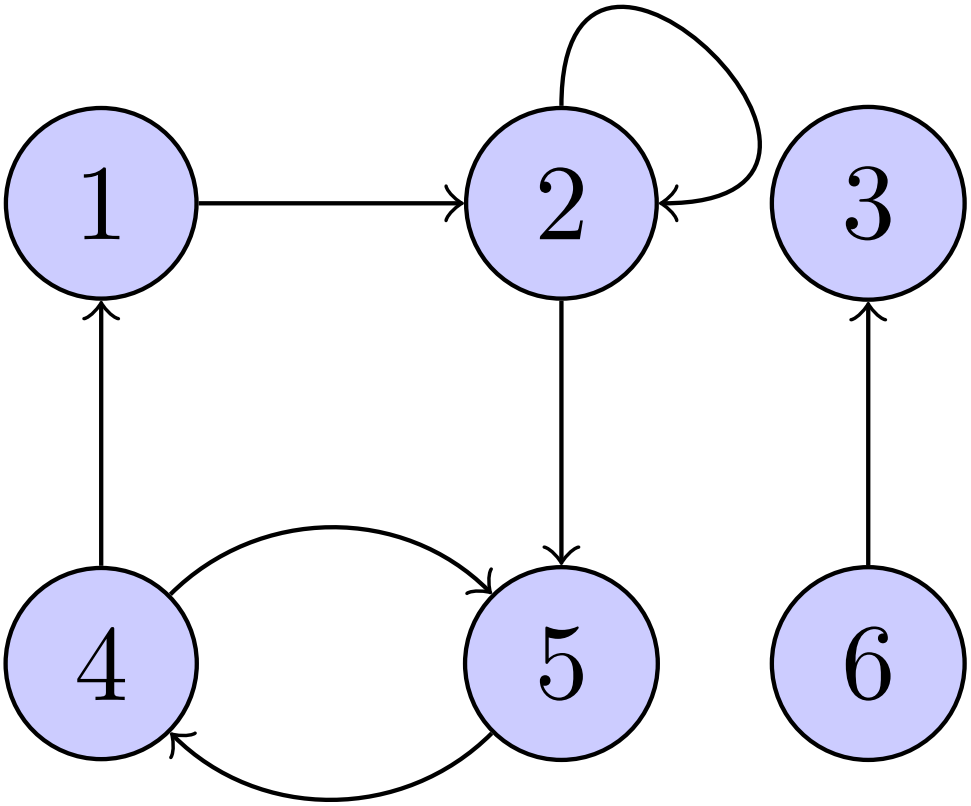
Γραφήματα

Ένα *κατευθυνόμενο γράφημα* G είναι ένα ζεύγος (V, E), όπου V είναι ένα πεπερασμένο σύνολο, οι λεγόμενοι *κόμβοι* του γράφου, και E μια διμελής σχέση επί του V. Υπενθυμίζουμε ότι μια *διμελής σχέση* επί των συνόλων A και B είναι ένα υποσύνολο του καρτεσιανού γινομένου $A \times B$. Μια διμελής σχέση επί του A είναι ένα υποσύνολο του $A \times A$. Αν R είναι μια διμελής σχέση επί των συνόλων A και B και $(a, b) \in R$, θα γράφουμε επίσης $aR\,b$. Το σύνολο E είναι το σύνολο των *ακμών* του γραφήματος. Στο Σχήμα 6.1 βλέπουμε την αναπαράσταση ενός κατευθυνόμενου γραφήματος με σύνολο κόμβων $\{1, 2, 3, 4, 5, 6\}$. Οι κόμβοι αναπαρίστανται ως κύκλοι και οι ακμές ως βέλη. Σε ένα τέτοιο γράφημα είναι δυνατόν να υπάρχουν *ιδιοβρόχοι* (*loops*), δηλαδή ακμές οι οποίες εκκινούν και καταλήγουν στον ίδιο κόμβο.

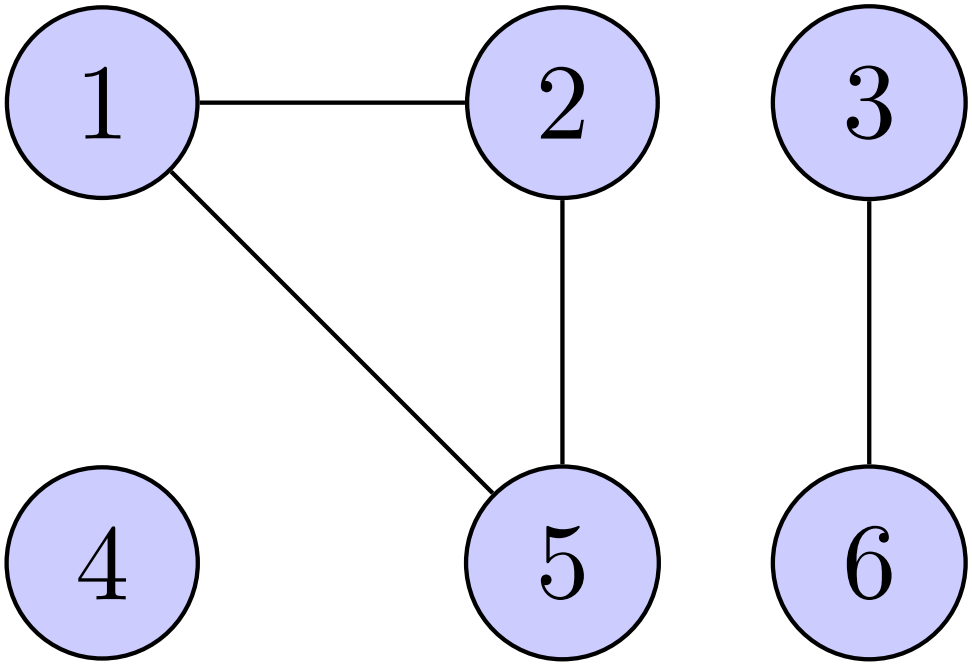
Σε ένα *μη κατευθυνόμενο γράφημα* $G = (V, E)$ το σύνολο των ακμών E αποτελείται από μη διατεταγμένα ζεύγη. Επομένως μια ακμή είναι ένα σύνολο $\{u, v\}$, όπου $u, v \in V$ και $u \neq v$. Θα εξακολουθούμε να γράφουμε (u, v) αντί $\{u, v\}$, και οι γραφές (u, v) και (u, v) θα δηλώνουν την ίδια ακμή. Σε ένα μη κατευθυνόμενο γράφημα δεν επιτρέπονται οι ιδιοβρόχοι, επομένως κάθε ακμή συνδέει δύο διαφορετικούς κόμβους. Στο Σχήμα 6.2 βλέπουμε την αναπαράσταση ενός μη κατευθυνόμενου γραφήματος με σύνολο κόμβων $\{1, 2, 3, 4, 5, 6\}$.

Αν $G = (V, E)$ είναι ένα κατευθυνόμενο γράφημα και $(u, v) \in E$ τότε λέμε ότι η ακμή *ξεκινά* από τον κόμβο u και *καταλήγει* στον κόμβο v. Αν (u, v) είναι ακμή οποιοδήποτε γραφήματος, λέμε ότι οι κόμβοι u και v είναι *γειτονικοί*. Η σχέση γειτνίασης είναι συμμετρική για μη κατευθυνόμενα γραφήματα, αλλά δεν είναι απαραίτητα τέτοια για κατευθυνόμενα γραφήματα. Ο *βαθμός* ενός κόμβου σε ένα μη κατευθυνόμενο γράφημα είναι το πλήθος των ακμών που καταλήγουν σε αυτόν. Ένας κόμβος με βαθμό μηδέν ονομάζεται *απομονωμένος*. Σε ένα κατευθυνόμενο γράφημα ο *βαθμός εξόδου* ενός κόμβου είναι το πλήθος των ακμών που εξέρχονται από αυτόν , ενώ ο *βαθμός εισόδου* είναι το πλήθος των ακμών που εισέρχονται σε αυτόν. Ο *βαθμός* ενός κόμβου σε ένα κατευθυνόμενο γράφημα είναι το άθροισμα του βαθμού εισόδου και του βαθμού εξόδου του. Για παράδειγμα, ο κόμβος 2 στο Σχήμα 6.1 έχει βαθμό εισόδου 2, βαθμό εξόδου επίσης 2 και συνολικό βαθμό 4.

Μια *διαδρομή μήκους* k από έναν κόμβο u μέχρι έναν κόμβο u' σε ένα γράφημα $G = (V, E)$ είναι μια ακολουθία κόμβων $\{v_0, v_1, \ldots, v_k\}$ τέτοια ώστε $v_0 = u, v_k = u'$ και $(v_{i-1}, v_i) \in E$ για $i = 1, 2, \ldots, k$. Το μήκος της διαδρομής είναι, βέβαια, το πλήθος των ακμών της. Εάν υπάρχει κάποια διαδρομή p από έναν κόμβο u μέχρι τον u', λέμε ότι ο u' είναι *προσπελάσιμος* από τον u μέσω της p, και γράφουμε, στην περίπτωση που το γράφημα είναι κατευθυνόμενο, $u \overset{p}{\rightsquigarrow} u'$. Μια διαδρομή ονομάζεται *απλή* αν όλοι οι κόμβοι της είναι διαφορετικοί μεταξύ τους. Για παράδειγμα, στο Σχήμα 6.1 η διαδρομή $\{1, 2, 5, 4\}$ είναι απλή διαδρομή μήκους 3. Μια *υποδιαδρομή* της διαδρομής $p = \{v_0, v_1, \ldots, v_k\}$ είναι



Σχήμα 6.1: Ένα κατευθυνόμενο γράφημα με σύνολο κόμβων $V = \{1, 2, 3, 4, 5, 6\}$ και σύνολο ακμών $E = \{(1, 2), (2, 2), (2, 5), (4, 5), (5, 4), (6, 3)\}$.



Σχήμα 6.2: Ένα μη κατευθυνόμενο γράφημα με σύνολο κόμβων $V = \{1, 2, 3, 4, 5, 6\}$ και σύνολο ακμών $E = \{(1, 2), (1, 5), (2, 5), (3, 6)\}$.

μια υπακολουθία διαδοχικών κόμβων της p. Δηλαδή, για οποιαδήποτε $0 \leq i \leq j \leq k$, η υπακολουθία κόμβων $\{v_i, v_{i+1}, \ldots, v_j\}$ αποτελεί υποδιαδρομή της p.

Σε ένα κατευθυνόμενο γράφημα, μια διαδρομή $\{v_0, v_1, \ldots, v_k\}$ αποτελεί *κύκλο* αν $v_0 = v_k$ και η διαδρομή περιλαμβάνει τουλάχιστον μια ακμή. Αν οι κόμβοι, $v_1, v_2, \ldots, v_k$ είναι διαφορετικοί μεταξύ τους τότε ο κύκλος είναι *απλός*. Ένας ιδιοβρόχος είναι ένας κύκλος με μήκος 1. Δύο διαδρομές $\{v_0, v_1, \ldots, v_{k-1}, v_0\}$ και $\{v'_0, v'_1, \ldots, v'_{k-1}, v'_0\}$ συνιστούν τον ίδιο κύκλο εάν υπάρχει ακέραιος j τέτοιος ώστε $v'_i = v_{(i+j) \bmod k}$ για $i = 0, 1, \ldots, k-1$. Στο Σχήμα 6.1 για παράδειγμα, η διαδρομή $\{1, 2, 5, 4, 1\}$ συνιστά τον ίδιο κύκλο με τον $\{5, 4, 1, 2, 5\}$. Ένα κατευθυνόμενο γράφημα χωρίς κύκλους ονομάζεται *απλό*. Σε ένα μη κατευθυνόμενο γράφημα μια διαδρομή $\{v_0, v_1, \ldots, v_k\}$ αποτελεί *απλό κύκλο* αν $k \geq 3$, $v_0 = v_k$, και οι κόμβοι $v_1, v_2, \ldots v_k$ είναι διαφορετικοί μεταξύ τους. Στο Σχήμα 6.2 η διαδρομή $\{1, 2, 5, 1\}$ αποτελεί από κύκλο. Ένα γράφημα το οποίο δεν περιέχει κανένα κύκλο ονομάζεται *άκυκλο*.

6.0.1 Συνδεδεμένα γραφήματα και συνδεδεμένες συνιστώσες

Ένα μη κατευθυνόμενο γράφημα ονομάζεται *συνδεδεμένο* αν όλοι οι κόμβοι του συνδέονται αν δύο μέσω κάποιας διαδρομής. Οι *συνδεδεμένες συνιστώσες* ενός γραφήματος είναι οι κλάσεις ισοδυναμίας των κόμβων ως προς τη σχέση "είναι προσπελάσιμος από τον". Είναι εύκολο να δούμε ότι αυτή η σχέση είναι σχέση ισοδυναμίας. Η σχέση είναι ανακλαστική γιατί οποιοσδήποτε κόμβος είναι προσπελάσιμος από τον εαυτό του (με μια διαδρομή μηδενικού μήκους). Η συμμετρία της σχέσης είναι επίσης προφανής. Τέλος, αν ο u είναι προσπελάσιμος από τον v τότε υπάρχει μια διαδρομή $\{v_0, v_1, \ldots, v_k\}$ με $v_0 = v$ kai $v_k = u$. Αν επίσης ο v είναι προσπελάσιμος από τον w υπάρχει μια διαδρομή $\{w_0, w_1, \ldots, w_l\}$ τέτοια ώστε $w_0 = w$ και $w_l = v$. Αν θεωρήσουμε τη διαδρομή $\{w_0, w_1, \ldots, w_l, v_0, v_1, \ldots v_k\}$ τότε αυτή είναι μια διαδρομή από τον w στον u, το οποίο δείχνει ότι ο u είναι προσπελάσιμος από τον w και ολοκληρώνει την απόδειξη ότι η σχέση "είναι προσπελάσιμος από τον" είναι σχέση ισοδυναμίας. Το γράφημα του Σχήματος 6.2 έχει τρεις συνδεδεμένες συνιστώσες: $\{1, 2, 5\}$, $\{3, 6\}$ και $\{4\}$. Ένα μη κατευθυνόμενο γράφημα είναι συνδεδεμένο εάν έχει μια και μόνο μία συνδεδεμένη συνιστώσα, δηλαδή εάν κάθε κόμβος είναι προσπελάσιμος από κάθε άλλο κόμβο.

Αν R είναι σχέση ισοδυναμίας επί του συνόλου A, τότε για $a \in A$, η *κλάση ισοδυναμίας* του a είναι το σύνολο $[a] = \{b \in A : aR\,b\}$. Θυμόμαστε ότι μια σχέση ισοδυναμίας επί ενός συνόλου A συμπίπτει με μια διαμέριση[1] του A, όπως δείχνει το επόμενο αποτέλεσμα:

Θεώρημα 1. Οι κλάσεις ισοδυναμίας μιας σχέσης ισοδυναμίας R επί ενός συνόλου A αποτελούν μια διαμέριση του A. Αντίστροφα, οποιαδήποτε διαμέριση του A καθορίζει μια σχέση ισοδυναμίας R επί του A, της οποίας οι κλάσεις ισοδυναμίας συμπίπτουν με τα σύνολα της διαμέρισης.

Απόδειξη. Έστω R μια σχέση ισοδυναμίας επί του συνόλου A και $a \in A$. Αφού η R είναι ανακλαστική έχουμε $aR\,a$, δηλαδή $a \in [a]$. Συνεπώς οι κλάσεις ισοδυναμίας είναι μη κενές και η ένωση τους είναι το σύνολο A. Μένει να δείξουμε ότι είναι ανά δύο ξένες μεταξύ τους. Πράγματι, αν $c \in [a]$ και $c \in [b]$ τότε $cR\,a$ και $cR\,b$. Από συμμετρία και μεταβατικότητα έχουμε $aR\,b$. Αν $x \in [a]$, τότε οι σχέσεις $xR\,a$ και $aR\,b$ δίνουν ότι $xR\,b$. Αυτό σημαίνει ότι $x \in [b]$. Επειδή το x ήταν αυθαίρετο, συμπεραίνουμε ότι $[a] \subseteq [b]$. Όμοια δείχνουμε ότι $[b] \subseteq [a]$ και συνεπώς $[a] = [b]$.

Για το δεύτερο μέρος της απόδειξης, έστω $\mathcal{A} = \{A_i\}$ μια διαμέριση του A. Ορίζουμε τη σχέση

$$R = \{(a, b) : \text{ υπάρχει } i \text{ τέτοιο ώστε } a \in A_i \text{ και } b \in A_i\}.$$

Η σχέση R είναι ανακλαστική γιατί αν $a \in A_i$ για κάποιο i τότε $aR\,a$. Αν τώρα $aR\,b$ τότε $a \in A_i$ και $b \in A_i$ για κάποιο i, δηλαδή, $bR\,a$ και η σχέση R είναι συμμετρική. Με τον ίδιο τρόπο, αν $aR\,b$ και $bR\,c$ τότε $a, b, c \in A_i$ για κάποιον δείκτη i, συνεπώς, $aR\,c$, δηλαδή, η R είναι μεταβατική. Για να δείξουμε ότι τα σύνολα της διαμέρισης συμπίπτουν με τις κλάσεις ισοδυναμίας της R επιχειρηματολογούμε ως εξής: έστω $a \in A_i$. Αν $x \in [a]$ τότε $xR\,a$. Συνεπώς $x \in A_i$. Αντίστροφα, αν $x \in A_i$ τότε μαζί με τη σχέση $a \in A_i$ έχουμε $xR\,a$ το οποίο δείχνει ότι $x \in [a]$. Άρα $[a] = A_i$. □

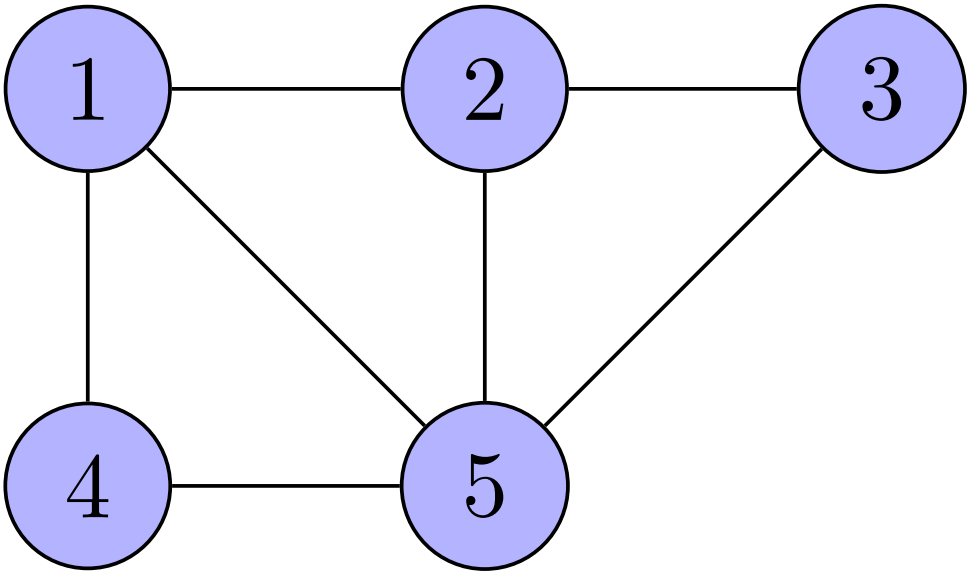
Ένα κατευθυνόμενο γράφημα είναι *ισχυρά συνδεδεμένο* αν για κάθε ζεύγος κόμβων, ο ένας κόμβος είναι προσπελάσιμος από τον άλλο.Οι *ισχυρά συνδεδεμένες συνιστώσες* ενός κατευθυνόμενου γραφήματος είναι οι κλάσεις ισοδυναμίας των κόμβων ως προς τη σχέση "είναι αμοιβαία προσπελάσιμοι". Ένα κατευθυνόμενο γράφημα είναι ισχυρά συνδεδεμένο αν έχει μόνο μια ισχυρά συνδεδεμένη συνιστώσα. Το γράφημα του Σχήματος 6.1 έχει τρείς ισχυρά συνδεδεμένες συνιστώσες: $\{1, 2, 4, 5\}$, $\{3\}$ και $\{6\}$.

6.0.2 Αναπαραστάσεις γραφημάτων

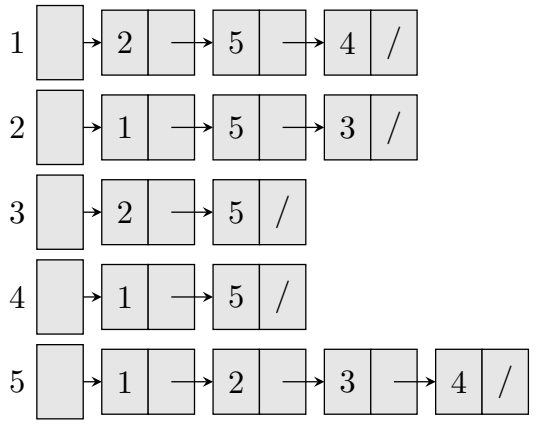
Υπάρχουν δύο καθιερωμένοι τρόποι αναπαράστασης ενός γραφήματος $G = (V, E)$, κατευθυνόμενου ή μη: μέσω ενός *καταλόγου γειτνίασης* (*adjacency list*) ή μέσω ενός *πίνακα γειτνίασης* (*adjacency matrix*). Η αναπαράσταση μέσω καταλόγου γειτνίασης προτιμάται για *αραιά γραφήματα*, στα οποία το $|E|$ είναι πολύ μικρότερο του $|V|^2$. Όταν το γράφημα είναι *πυκνό*, δηλαδή όταν το $|E|$ είναι συγκρίσιμο με το $|V|^2$, ή όταν είναι απαραίτητο να μπορούμε να αποφανθούμε γρήγορα αν δύο κόμβοι συνδέονται μέσω ακμής, η αναπαράσταση μέσω πίνακα γειτνίασης να είναι προτιμότερη.

Η αναπαράσταση μέσω καταλόγων γειτνίασης ενός γραφήματος $G = (V, E)$ συνίσταται σε μια συστοιχία Adj από $|V|$ καταλόγους, έναν για κάθε κόμβο του V. Για κάθε $u \in V$ ο κατάλογος Adj$[u]$ περιέχει όλους τους κόμβους v για τους οποίους υπάρχει ακμή $(u, v) \in E$. Η διάταξη των κόμβων σε κάθε κατάλογο κατά κανόνα είναι αυθαίρετη. Στο Σχήμα 6.4 βλέπουμε τις αναπαραστάσεις του μη κατευθυνόμενου γραφήματος του Σχήματος 6.3 μέσω καταλόγου γειτνίασης και μέσω πίνακα γειτνίασης. Το άθροισμα των μηκών όλων των καταλόγων είναι $2|E|$ αφού αν (u, v) είναι μια μη κατευθυνόμενη ακμή, αυτή εμφανίζεται τόσο στον κατάλογο γειτνίασης του v όσο και του u.
Στο Σχήμα 6.6 βλέπουμε τις αναπαραστάσεις του κατευθυνόμενου γραφήματος του Σχήματος 6.5 μέσω καταλόγου γειτνίασης και μέσω πίνακα γειτνίασης. Το άθροισμα των μηκών όλων των καταλόγων είναι $|E|$ αφού αν (u, v) είναι μια κατευθυνόμενη ακμή, αυτή αναπαριστάται μέσω της παρουσίας του v στον κατάλογο γειτνίασης Adj$[u]$. Τόσο για τα κατευθυνόμενα όσο και για τα μη κατευθυνόμενα γραφήματα, η αναπαράσταση μέσω καταλόγων γειτνίασης απαιτεί μνήμη $\Theta(|V|+|E|)$. Με μικρές

[1] Μια συλλογή $\{A_i\}$ μη κενών συνόλων αποτελεί *διαμέριση* ενός συνόλου A αν τα σύνολα αυτά είναι ανά δύο ξένα μεταξύ τους και η ένωσή τους είναι το A.

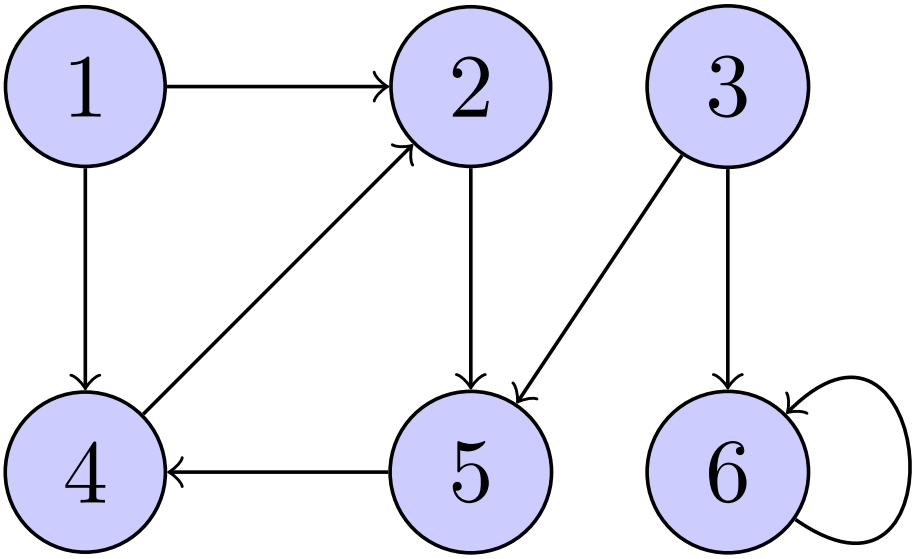


Σχήμα 6.3: Ένα μη κατευθυνόμενο γράφημα με πέντε κόμβους και επτά ακμές.



	1	2	3	4	5
1	0	1	0	1	1
2	1	0	1	0	1
3	0	1	0	0	1
4	1	0	0	0	1
5	1	1	1	1	0

Σχήμα 6.4: Αναπαραστάσεις του μη κατευθυνόμενου γραφήματος του Σχήματος 6.3 μέσω καταλόγου γειτνίασης και μέσω πίνακα γειτνίασης.



Σχήμα 6.5: Ένα κατευθυνόμενο γράφημα με έξι κόμβους και οκτώ ακμές.

τροποποιήσεις η μέθοδος αυτή μπορεί να χρησιμοποιηθεί για την αναπαράσταση *εμβαρών γραφημάτων*, δηλαδή γραφημάτων στα οποία κάθε ακμή φέρει κάποιο *βάρος* που κατά κανόνα δίνεται από μια *συνάρτηση βάρους* $w : E \to \mathbb{R}$. Το βάρος $w(u, v)$ της ακμής $(u, v) \in E$ μπορεί να αποθηκευτεί μαζί με τον κόμβο v στον κατάλογο γειτνίασης του u. Ένα πιθανό μειονέκτημα της αναπαράστασης μέσω καταλόγων γειτνίασης είναι ότι ο ταχύτερος δυνατός τρόπος για να προσδιοριστεί εάν υπάρχει στο γράφημα μια δεδομένη ακμή (u, v) είναι μέσω της αναζήτησης του v στον κατάλογο γειτνίασης $\mathrm{Adj}[u]$. Το μειονέκτημα αυτό μπορεί να αρθεί με την αναπαράσταση του γραφήματος μέσω πίνακα γειτνίασης, αλλά με τίμημα τη δέσμευση ασυμπτωτικά περισσότερης μνήμης.

Για την αναπαράσταση μέσω πίνακα γειτνίασης ενός γραφήματος $G = (V, E)$ υποθέτουμε ότι όλοι οι κόμβοι είναι αριθμημένοι από το 1 έως το $|V|$ κατ' αυθαίρετο τρόπο. Στην περίπτωση αυτή, η αναπαράσταση ενός γραφήματος G συνίσταται σε έναν $|V| \times |V|$ πίνακα $A = (a_{ij})$ με στοιχεία

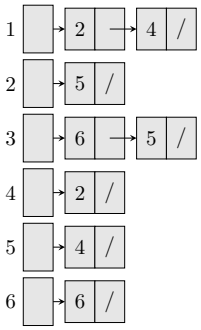
$$a_{ij} = \begin{cases} 1 & \text{εάν } (i, j) \in E, \\ 0 & \text{διαφορετικά.} \end{cases}$$

Ο πίνακας γειτνίασης απαιτεί μνήμη $\Theta(|V|^2)$, ανεξάρτητα από το πλήθος των ακμών του γραφήματος.

6.1 Διερευνήσεις γραφημάτων

Με τον όρο διερεύνηση γραφήματος (graph traversal) εννοούμε τη συστηματική επίσκεψη κάθε κόμβου ενός γραφήματος. Οι διερευνήσεις διαφοροποιούνται ως προς τη σειρά με την οποία εξετάζονται οι κόμβοι του γραφήματος. Διακρίνουμε την *οριζόντια διερεύνηση* (*breadth–first search*) και την *καθοδική διερεύνηση* (*depth–first search*).

Για ένα γράφημα $G = (V, E)$ και έναν δεδομένο αφετηριακό κόμβο s, η οριζόντια διερεύνηση συνίσταται στη συστηματική εξέταση των ακμών του G έτσι ώστε να εντοπιστούν όλοι οι κόμβοι που είναι προσπελάσιμοι από τον s. Η οριζόντια διερεύνηση εντοπίζει πρώτα όλους τους κόμβους σε απόσταση k από τον s, και αφού εξαντλήσει αυτούς τους κόμβους προχωρά στον εντοπισμό κόμβων σε απόσταση $k + 1$. Εδώ, με τον όρο *απόσταση* του κόμβου v εννοούμε τον αριθμό των ακμών σε μια



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Σχήμα 6.6: Αναπαραστάσεις του μη κατευθυνόμενου γραφήματος του Σχήματος 6.5 μέσω καταλόγου γειτνίασης και μέσω πίνακα γειτνίασης.

διαδρομή από τον αφετηριακό κόμβο s στον κόμβο v. Ένας αλγόριθμος οριζόντιας διερεύνησης ενός γραφήματος λειτουργεί χωρίζοντας το σύνολο των κόμβων σε τρία υποσύνολα: αυτούς που έχουν ήδη εξεταστεί, αυτούς που έχουν εντοπιστεί, και αυτούς που δεν είναι ακόμα εντοπισμένοι. Οι κόμβοι χρωματίζονται λευκοί, γκρίζοι ή μαύροι, ανάλογα το με το σύνολο στο οποίο ανήκουν:

- Στην αρχή όλοι οι κόμβοι είναι λευκοί.
- Ένας κόμβος θεωρείται *εντοπισμένος* την πρώτη φορά που συναντάται στη διάρκεια της διερεύνησης, οπότε χρωματίζεται γκρίζος.
- Αν $(u, v) \in E$ και ο κόμβος u είναι μαύρος, τότε ο v είναι είτε γκρίζος είτε μαύρος. Αυτό σημαίνει ότι όλοι οι κόμβοι που γειτνιάζουν με μαύρους κόμβους είναι εντοπισμένοι, ενώ κόμβοι οι οποίοι γειτνιάζουν με γκρίζους κόμβους μπορεί να είναι λευκοί. Οι γκρίζοι κόμβοι αποτελούν το σύνορο μεταξύ των εντοπισμένων και μη εντοπισμένων κόμβων.

Ο αλγόριθμος της οριζόντιας διερεύνησης κατασκευάζει ένα οριζόντιο δένδρο με ρίζα τον αφετηριακό κόμβο s. Όποτε εντοπίζεται ένας λευκός κόμβος v κατά τη σάρωση των γειτονικών κόμβων ενός ήδη εντοπισμένου κόμβου u, ο κόμβος v και η ακμή (u, v) προστίθενται στο δένδρο. Ο κόμβος u ονομάζεται *προκάτοχος* (*predecessor, parent*) του v στο δένδρο. Οι σχέσεις απογόνου και προγόνου στο δένδρο ορίζονται σε σχέση με τη ρίζα s: αν ο κόμβος u είναι σε μια διαδρομή από τη ρίζα s στον κόμβο v, τότε ο u είναι *πρόγονος* του v και ο v είναι *απόγονος* του u.

Ο αλγόριθμος της οριζόντιας διερεύνησης που ακολουθεί υποθέτει ότι το γράφημα παριστάνεται χρησιμοποιώντας καταλόγους γειτνίασης. Χρησιμοποιεί επίσης μια ουρά προτεραιότητας για να αποθηκεύσει τους γκρίζους κόμβους και διατηρεί τρία χαρακτηριστικά για κάθε κόμβο: το χρώμα του, την απόσταση του από τη ρίζα και τον προκάτοχο του. Αν ένας κόμβος v δεν έχει προκάτοχο, είτε γιατί δεν έχει εντοπιστεί ακόμα είτε γιατί πρόκειται για τη ρίζα του δένδρου, αυτό το χαρακτηριστικό είναι κενό (NIL).

```
BreadthFirstSearch(G, s)
 1  for each vertex u ∈ V \ {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each vertex v ∈ Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              Enqueue(Q, v)
18      u.color = BLACK
```

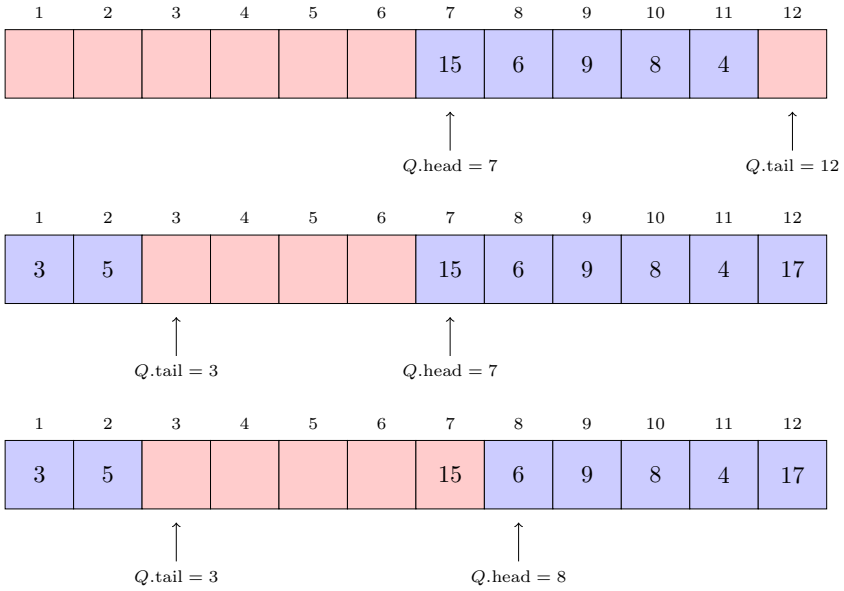
Υπενθυμίζουμε ότι η δομή δεδομένων ουρά προτεραιότητας (priority queue) διατηρεί μια συλλογή στοιχείων στην οποία το στοιχείο που εισάγεται πρώτο είναι αυτό που αφαιρείται πρώτο, αυτό δηλαδή το οποίο έχει παραμείνει στην ουρά για τον μεγαλύτερο χρόνο (first-in, first-out).

Μια ουρά προτεραιότητας υλοποιείται με έναν πίνακα $Q[1\,..\,n]$ με το χαρακτηριστικά Q.size να δηλώνει το μέγεθος της ουράς, Q.head τη θέση του πρώτου στοιχείου της ουράς και Q.tail τη θέση του τελευταίου στοιχείου. Έτσι, τα στοιχεία της ουράς καταλαμβάνουν τις θέσεις Q.head, Q.head $+$ $1, \ldots, Q$.tail $- 1$. Τα στοιχεία της ουράς καταλαμβάνουν τις θέσεις κυκλικά, δηλαδή, η θέση 1 ακολουθεί τη θέση n. Όταν Q.head $= Q$.tail η ουρά είναι άδεια. Στην αρχή, Q.head $= Q$.tail $= 1$. Η λειτουργία εισαγωγής ενός στοιχείου σε μια ουρά είναι η διαδικασία Enqueue, και η λειτουργία αφαίρεσης στοιχείου από την ουρά είναι η διαδικασία Dequeue που εμφανίζονται παρακάτω, χωρίς όμως τους ελέγχους αν η ουρά είναι άδεια κατά την εξαγωγή ενός στοιχείου ή αν είναι πλήρης όταν επιχειρείται η εισαγωγή κάποιου στοιχείου. Και οι δύο διαδικασίες απαιτούν σταθερό χρόνο:

```
Enqueue(Q, x)
    Q[Q.tail] = x
    if Q.tail == Q.size
        Q.tail = 1
    else
        Q.tail = Q.tail + 1

Dequeue(Q)
    x = Q[Q.head]
    if Q.head == Q.size
        Q.head = 1
    else
        Q.head = Q.head + 1
    return x
```

Για παράδειγμα, στο Σχήμα 6.7 βλέπουμε μια ουρά Q με μέγεθος 12 και στοιχεία στις θέσεις $Q[7\,..\,11]$. Το επόμενο στιγμιότυπο παρουσιάζει την ουρά μετά τις πράξεις Enqueue$(Q, 17)$, Enqueue$(Q, 3)$ και Enqueue$(Q, 5)$. Το τελευταίο στιγμιότυπο παρουσιάζει την ουρά μετά την πράξη Dequeue(Q).

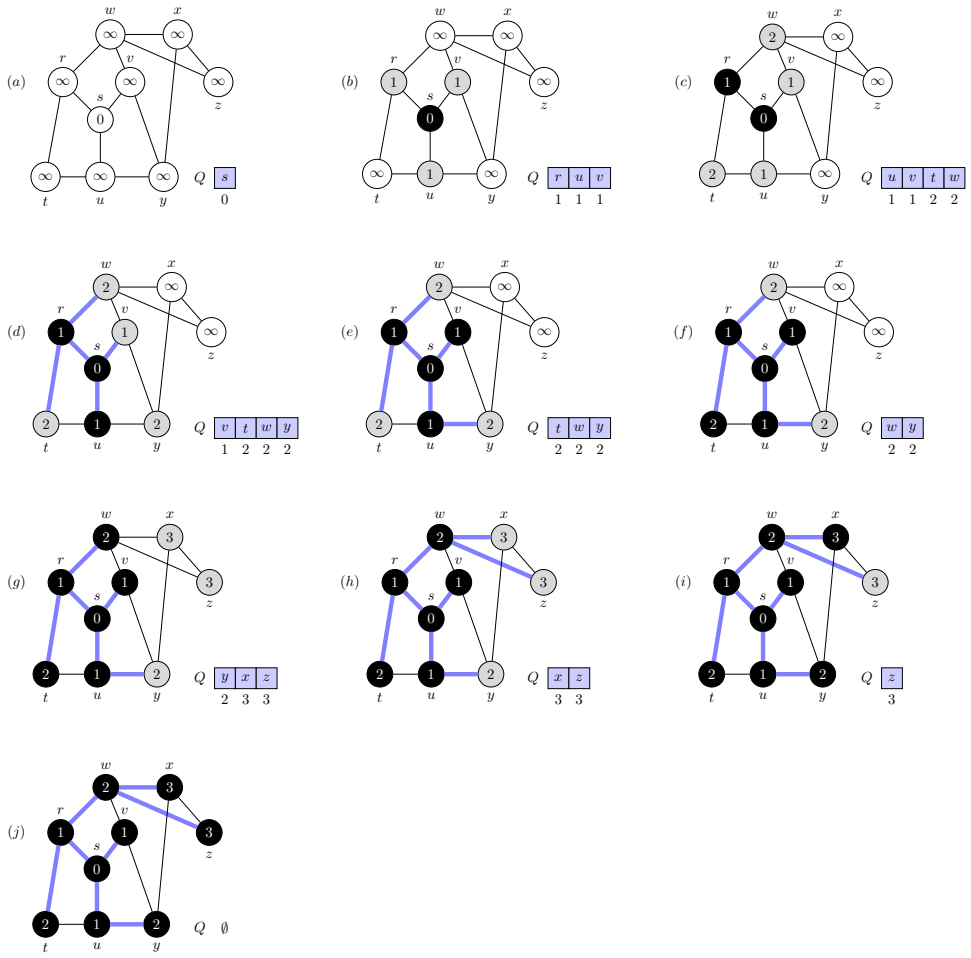


Σχήμα 6.7: Μια ουρά Q με μέγεθος 12 και στοιχεία στις θέσεις $Q[7\,..\,11]$. Το επόμενο στιγμιότυπο παρουσιάζει την ουρά μετά τις πράξεις ENQUEUE$(Q, 17)$, ENQUEUE$(Q, 3)$ και ENQUEUE$(Q, 5)$. Το τελευταίο στιγμιότυπο παρουσιάζει την ουρά μετά την πράξη DEQUEUE(Q).

Η διαδικασία BREADTHFIRSTSEARCH τώρα λειτουργεί ως εξής: στις γραμμές 1–4 κάθε κόμβος πλην του αφετηριακού χρωματίζεται λευκός, ο πρόγονος του ορίζεται να είναι κενός και η απόσταση του από τη ρίζα άπειρη. Στις γραμμές 5–7 ο αφετηριακός κόμβος s χρωματίζεται γκρίζος, ως ο πρώτος κόμβος που εντοπίζεται, ο πρόγονος του ορίζεται, φυσικά, να είναι κενός και η απόσταση του από τη ρίζα να είναι μηδέν. Στις γραμμές 8 και 9 αρχικοποιείται η ουρά προτεραιότητας Q και ο αφετηριακός κόμβος εισάγεται σε αυτή. Οι εντολές της επανάληψης **while** στις γραμμές 10–18 εκτελούνται εφόσον παραμένουν γκρίζοι κόμβοι στην ουρά Q, δηλαδή κόμβοι των οποίων οι γείτονες δεν έχουν ακόμα εξεταστεί. Η γραμμή 11 αφαιρεί από την ουρά τον πρώτο κόμβο u. Οι γραμμές 12–17 εξετάζουν κάθε γειτονικό κόμβο v του u. Αν ο v είναι χρωματισμένος λευκός, τότε στις γραμμές 13–17 χρωματίζεται γκρίζος, χαρακτηρίζεται δηλαδή εντοπισμένος, ο u ορίζεται ως πρόγονος του, η απόσταση του γίνεται κατά ένα μεγαλύτερη από αυτή του u, και, τέλος, εισάγεται στην ουρά Q. Αφού έχουν εξεταστεί όλοι οι γειτονικοί του κόμβοι, στη γραμμή 18 ο κόμβος u χρωματίζεται μαύρος.

Παρατήρηση. Είναι προφανές ότι η σειρά με την οποία εξετάζονται οι κόμβοι ενός γραφήματος κατά την οριζόντια διερεύνηση εξαρτάται από τη σειρά των γειτονικών κόμβων στους καταλόγους γειτνίασης. Οι αποστάσεις όμως των κόμβων από τον αφετηριακό κόμβο είναι μονοσήμαντα ορισμένες.

Το Σχήμα 6.8 παρουσιάζει σχηματικά τη λειτουργία της διαδικασίας BREADTHFIRSTSEARCH για τη διερεύνηση ενός μη κατευθυνόμενου γραφήματος. Οι αποστάσεις από τον αφετηριακό κόμβο s εμφανίζονται ως ετικέτες των κόμβων και κάτω από τους κόμβους στην ουρά προτεραιότητας Q. Οι ακμές που χρωματίζονται μπλε είναι οι ακμές του οριζόντιου δένδρου.
Σχετικά με τον χρόνο εκτέλεσης της διαδικασίας BREADTHFIRSTSEARCH, παρατηρούμε ότι μετά την αρχικοποίηση, κανένας κόμβος δεν χρωματίζεται ξανά λευκός και επομένως ο έλεγχος στην γραμμή 13 εξασφαλίζει ότι κάθε κόμβος εισέρχεται στην ουρά το πολύ μια φορά και αφαιρείται από αυτή το πολύ μια φορά. Τόσο η διαδικασία της εισαγωγής στοιχείου στην ουρά όσο και της αφαίρεσης στοιχείου από αυτή απαιτούν χρόνο $O(1)$, οπότε η BREADTHFIRSTSEARCH αφιερώνει χρόνο $O(|V|)$ σε αυτές. Αφού οι κατάλογοι γειτνίασης ενός κόμβου εξετάζονται μόνο όταν ο κόμβος αφαιρεθεί από την ουρά, αυτό γίνεται το πολύ μια φορά. Το άθροισμα των μηκών των καταλόγων γειτνίασης είναι $\Theta(|E|)$, ο συνολικός χρόνος που αφιερώνεται στην εξέταση των καταλόγων γειτνίασης είναι $O(|V| + |E|)$. Η αρχικοποίηση απαιτεί χρόνο $O(|V|)$ και τελικά ο συνολικός χρόνος εκτέλεσης της



Σχήμα 6.8: Η λειτουργία της διαδικασίας BreadthFirstSearch για τη διερεύνηση ενός μη κατευθυνόμενου γραφήματος. Οι αποστάσεις από τον αφετηριακό κόμβο s εμφανίζονται ως ετικέτες των κόμβων και κάτω από τους κόμβους στην ουρά προτεραιότητας Q. Οι ακμές που χρωματίζονται μπλε είναι οι ακμές του οριζόντιου δένδρου.

BreadthFirstSearch είναι $O(|V| + |E|)$.

6.1.1 Ελάχιστα μονοπάτια

Ορίζουμε το *μήκος βραχύτατης διαδρομής* (*shotest–path distance*) $\delta(s, v)$ από τον αφετηριακό κόμβο s στον κόμβο v τον ελάχιστο αριθμό ακμών σε μια διαδρομή από τον s στον v. Αν δεν υπάρχει διαδρομή από τον s στον v γράφουμε $\delta(s, v) = \infty$. Τα αποτελέσματα που ακολουθούν δείχνουν ότι $v.d = \delta(s, v)$ για κάθε κόμβο v του γραφήματος.

Λήμμα 1. Έστω $G = (V, E)$ ένα οποιοδήποτε γράφημα και $s \in V$. Τότε για οποιαδήποτε ακμή $(u, v) \in E$ ισχύει ότι

$$\delta(s, v) \leq \delta(s, u) + 1.$$

Απόδειξη. Αν ο u είναι προσπελάσιμος από τον s τότε και ο v είναι προσπελάσιμος από τον s. Η βραχύτατη διαδρομή από τον s στον v δεν μπορεί να είναι μεγαλύτερη από τη βραχύτατη διαδρομή από τον s στον u, ακολουθούμενη από την ακμή (u, v). Επομένως η ανισότητα του λήμματος ισχύει. Αν ο u δεν είναι προσπελάσιμος από τον s τότε $\delta(s, u) = \infty$ και επομένως η ανισότητα ισχύει πάλι. □

Λήμμα 2. Έστω $G = (V, E)$ ένα οποιοδήποτε γράφημα και έστω ότι υπολογίζουμε μια οριζόντια διερεύνηση του γραφήματος με τη διαδικασία BreadthFirstSearch χρησιμοποιώντας τον αφετηριακό κόμβο $s \in V$. Τότε, για κάθε $v \in V$ ισχύει $v.d \geq \delta(s, v)$, καθ' όλη τη διάρκεια της διερεύνησης του γραφήματος.

Απόδειξη. Η απόδειξη γίνεται με επαγωγή ως προς τον αριθμό των διαδικασιών Enqueue. Υποθέτουμε λοιπόν ότι $v.d \geq \delta(s, v)$ για κάθε $v \in V$. Μετά την εισαγωγή του αφετηριακού κόμβου s στην ουρά Q έχουμε $s.d = 0 = \delta(s, s)$ και $v.d = \infty \geq \delta(s, v)$, για κάθε $v \in V \setminus \{s\}$. Αυτό σημαίνει ότι $v.d \geq \delta(s, v)$ για κάθε $v \in V$. Έστω v ένας λευκός κόμβος ο οποίος εντοπίζεται κατά την εξέταση των γειτόνων του κόμβου u. Από την επαγωγική υπόθεση έχουμε ότι $u.d \geq \delta(s, u)$. Η γραμμή 15 της διαδικασίας BreadthFirstSearch και το Λήμμα 1 δίνουν

$$v.d = u.d + 1 \geq \delta(s, u) + 1 \geq \delta(s, v).$$

Ο κόμβος v εισάγεται στην ουρά ακριβώς μια φορά, γιατί τότε χρωματίζεται γκρίζος. Επομένως η η τιμή $v.d$ δεν θα αλλάξει. Συνεπώς, η σχέση $v.d \geq \delta(s, v)$ εξακολουθεί να ισχύει. □

Το επόμενο αποτέλεσμα δείχνει ότι οποιαδήποτε στιγμή κατά την οριζόντια διερεύνηση ενός γραφήματος με την BreadthFirstSearch τα χαρακτηριστικά d των κόμβων στην ουρά Q είτε είναι όλα ίσα, είτε σχηματίζουν μια ακολουθία της μορφής $\{k, k, \ldots, k, k + 1, \ldots, k + 1\}$.

Λήμμα 3. Έστω ότι κατά τη διάρκεια της διερεύνησης ενός γραφήματος $G = (V, E)$ με τη διαδικασία BreadthFirstSearch, η ουρά Q περιέχει τους κόμβους $\{v_1, v_2, \ldots, v_r\}$, με v_1 το πρώτο στοιχείο της ουράς και v_r το τελευταίο.Τότε έχουμε $v_r.d \leq v_1.d + 1$ και $v_i.d \leq v_{i+1}.d$, για $i = 1, 2, \ldots, r - 1$.

Απόδειξη. Η απόδειξη γίνεται με επαγωγή ως προς τον αριθμό των διαδικασιών Enqueue και Dequeue στην ουρά Q. Αρχικά, όταν η ουρά περιέχει μόνο τον αφετηριακό κόμβο s το αποτέλεσμα ισχύει τετριμμένα. Για το επαγωγικό βήμα δείχνουμε ότι το αποτέλεσμα ισχύει μετά την εισαγωγή ενός κόμβου στην ουρά και μετά την εξαγωγή ενός κόμβου από αυτήν. Εξετάζουμε πρώτα την περίπτωση της εξαγωγής ενός κόμβου από την ουρά. Όταν ο κόμβος v_1 αφαιρείται από την ουρά, το στοιχείο v_2 γίνεται το πρώτο στοιχείο της ουράς. Από την επαγωγική υπόθεση, $v_1.d \leq v_2.d$. Τότε, $v_1.d \leq v_1.d + 1 \leq v_2.d + 1$, ενώ οι υπόλοιπες ανισότητες δεν αλλάζουν. Επομένως το συμπέρασμα του λήμματος ισχύει με v_2 το πρώτο στοιχείο της ουράς.

Όταν τώρα εισάγεται ένα στοιχείο v στην ουρά που περιέχει ήδη τα $\{v_1, v_2, \ldots, v_r\}$, αυτό γίνεται το στοιχείο v_{r+1}. Αν η ουρά ήταν άδεια, τότε μετά την εισαγωγή του στοιχείου έχουμε $r = 1$ και το αποτέλεσμα ισχύει τετριμμένα. Διαφορετικά, η διαδικασία διερεύνησης έχει αφαιρέσει από την ουρά κάποιο στοιχείο u του οποίου ο κατάλογος γειτνίασης εξετάζεται. Πριν από την αφαίρεση του u

είχαμε $u = v_1$, η επαγωγική υπόθεση ήταν αληθής και επομένως $u.d \leq v_2.d$ και $v_r.d \leq u.d + 1$. Μετά την αφαίρεση του u, το v_1 είναι στην κορυφή της ουράς επομένως $u.d \leq v_1.d$. Συνεπώς, $v_{r+1}.d = v.d = u.d + 1 \leq v_1.d + 1$. Αφού $v_r.d \leq u.d + 1$, έχουμε $v_r.d \leq u.d + 1 = v.d = v_{r+1}.d$, ενώ οι υπόλοιπες ανισότητες δεν αλλάζουν. Επομένως το αποτέλεσμα του λήμματος ισχύει μετά την προσθήκη ενός στοιχείου στην ουρά. □

Συνέπεια του Λήμματος 3 είναι το γεγονός ότι οι τιμές των χαρακτηριστικών d τη στιγμή της εισαγωγής στοιχείων στην ουρά αυξάνουν μονότονα.

Πόρισμα 1. Έστω ότι οι κόμβοι v_i και v_j εισάγονται στην ουρά Q κατά τη διάρκεια διερεύνησης ενός γραφήματος $G = (V, E)$ με τη διαδικασία BreadthFirstSearch. Υποθέτουμε ότι ο v_i εισάγεται πριν τον v_j. Τότε $v_i.d \leq v_j.d$ τη στιγμή που εισάγεται ο κόμβος v_j.

Απόδειξη. Το πόρισμα έπεται από το Λήμμα 3 και το γεγονός ότι σε κάθε κόμβο ανατίθεται ένα πεπερασμένο χαρακτηριστικό d το πολύ μια φορά κατά τη διάρκεια της διερεύνησης. □

Μπορούμε τώρα να αποδείξουμε ότι η διαδικασία BreadthFirstSearch υπολογίζει σωστά τα μήκη βραχύτατων διαδρομών.

Θεώρημα 2. Έστω $G = (V, E)$ ένα οποιοδήποτε γράφημα και έστω ότι υπολογίζουμε μια οριζόντια διερεύνηση του γραφήματος με τη διαδικασία BreadthFirstSearch χρησιμοποιώντας τον αφετηριακό κόμβο s. Κατά τη διάρκεια της διερεύνησης η διαδικασία BreadthFirstSearch εντοπίζει κάθε κόμβο $v \in V$ ο οποίος είναι προσπελάσιμος από τον s, και κατά τον τερματισμό της $v.d = \delta(s, v)$, για κάθε $v \in V$. Επιπλέον, για κάθε κόμβο $v \neq s$, προσπελάσιμο από τον s, μια από τις βραχύτατες διαδρομές από τον s στον v είναι μια βραχύτατη διαδρομή από τον s στον $v.\pi$ ακολουθούμενη από την ακμή $(v.\pi, v)$.

Απόδειξη. Ας υποθέσουμε ότι για κάποιον κόμβο το χαρακτηριστικό d δεν ισούται με το μήκος της βραχύτατης διαδρομής από τον s. Ας υποθέσουμε ότι v είναι ένας τέτοιος κόμβος με ελάχιστη $\delta(s, v)$. Από το Λήμμα 2 έχουμε $v.d \geq \delta(s, v)$ και επομένως $v.d > \delta(s, v)$. Αναγκαστικά $v \neq s$ γιατί $s.d = 0 = \delta(s, s)$. Επιπλέον, ο κόμβος v πρέπει να είναι προσπελάσιμος από τον s γιατί σε διαφορετική περίπτωση $\delta(s, v) = \infty \geq v.d$. Έστω u ο κόμβος αμέσως πριν τον v σε μια βραχύτατη διαδρομή από τον s στον v, έτσι ώστε $\delta(s, v) = \delta(s, u) + 1$. Επειδή $\delta(s, u) < \delta(s, v)$ και εξαιτίας της επιλογής του v έχουμε $u.d = \delta(s, u)$. Συνεπώς,

$$v.d > \delta(s, v) = \delta(s, u) + 1 = u.d + 1. \tag{6.1}$$

Έστω τώρα ότι ο κόμβος u αφαιρείται από την ουρά. Δείχνουμε ότι οποιοσδήποτε και αν είναι ο χρωματισμός του v οδηγούμαστε σε αντίφαση. Πράγματι, αν ο κόμβος v έχει χρωματιστεί λευκός, τότε στη γραμμή 15 της διαδικασίας BreadthFirstSearch έχουμε $v.d = u.d + 1$, σε αντίθεση με τη σχέση (6.1). Αν ο κόμβος v είχε χρωματιστεί μαύρος, τότε είχε επίσης αφαιρεθεί νωρίτερα από την ουρά. Από το Πόρισμα 1 έχουμε $v.d \leq u.d$, το οποίο πάλι έρχεται σε αντίθεση με τη σχέση (6.1). Τέλος, αν ο v έχει χρωματιστεί γκρίζος τότε αυτό έγινε κατά την αφαίρεση από την ουρά κάποιου κόμβου w. Αυτή έγινε νωρίτερα από την αφαίρεση του u. Για τον κόμβο w έχουμε $v.d = w.d + 1$. Από το Πόρισμα 1 όμως $w.d \leq u.d$ και επομένως $v.d = w.d + 1 \leq u.d + 1$, ξανά σε αντίθεση με τη σχέση (6.1). Αποδείξαμε λοιπόν ότι $v.d = \delta(s, v)$ για κάθε $v \in V$.

Είναι επίσης προφανές ότι όλοι οι προσπελάσιμοι από τον s κόμβοι εξετάζονται γιατί, σε διαφορετική περίπτωση, θα είχαμε $\infty = v.d > \delta(s, v)$, για οποιονδήποτε από αυτούς τους κόμβους. Τέλος, στις γραμμές 15–16 της διαδικασίας BreadthFirstSearch βλέπουμε ότι αν $v.\pi = u$ τότε $v.d = u.d + 1$. Αυτό σημαίνει ότι μια βραχύτατη διαδρομή από τον s στον v είναι μια βραχύτατη διαδρομή από τον s στον u ακολουθούμενη από την ακμή (u, v). □

6.1.2 Οριζόντια δένδρα

Το δένδρο που σχηματίζεται από τα χαρακτηριστικά $v.\pi$ και τις ακμές χρωματισμένες μπλε στο Σχήμα 6.8 αποτελούν το οριζόντιο δένδρο της διαδικασίας BreadthFirstSearch. Ακριβέστερα, αν

$G = (V, E)$ είναι ένα γράφημα και s είναι ο αφετηριακός κόμβος τότε ορίζουμε το *υπογράφημα προκατόχων* (*predecessor subgraph*) $G_\pi = (V_\pi, E_\pi)$ του G από τις σχέσεις

$$V_\pi = \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\} \tag{6.2}$$

$$E_\pi = \{(v, \pi, v) : v \in V_\pi \setminus \{s\}\}. \tag{6.3}$$

Το υπογράφημα προκατόχων αποτελείται από τους κόμβους του V οι οποίοι είναι προσπελάσιμοι από τον s, και για κάθε $v \in V_\pi$ το υπογράφημα G_π περιέχει μια βραχύτατη διαδρομή από τον s στον v η οποία είναι επίσης βραχύτατη διαδρομή από τον s στον v στο γράφημα G. Μπορεί να αποδειχθεί ότι το υπογράφημα προκατόχων είναι ένα οριζόντιο δένδρο. Η διαδικασία PrintPath παρακάτω εκτυπώνει τους κόμβους σε μια βραχύτατη διαδρομή από τον αφετηριακό κόμβο s προς τον κόμβο v του γραφήματος G.

```
PrintPath(G, s, v)
    if v == s
        print s
    elseif v.p == NIL
        print "no path from" s "to" v
    else
        PrintPath(G, s, v.π)
        print v
```

Ο χρόνος εκτέλεσης της διαδικασίας PrintPath είναι γραμμικός ως προς τον αριθμό των κόμβων μια και κάθε αναδρομική κλήση αφορά σε μια διαδρομή κατά ένα κόμβο συντομότερη.

6.1.3 Καθοδική διερεύνηση γραφημάτων

Η *καθοδική διερεύνηση γραφημάτων* (*depth–first search*) χρησιμοποιεί τις ακμές των πιο πρόσφατα εντοπισμένων κόμβων για να εντοπίζει νέους κόμβους προτού επιστρέψει στον κόμβο εκκίνησης. Αν παραμένουν κόμβοι ο οποίοι δεν έχουν εντοπιστεί, η καθοδική διερεύνηση επιλέγει έναν από αυτούς και επαναλαμβάνει την ίδια διαδικασία εντοπισμού κόμβων. Η καθοδική διερεύνηση μπορεί να εφαρμοστεί είτε σε κατευθυνόμενα είτε σε μη κατευθυνόμενα γραφήματα.

Όπως και στην οριζόντια διερεύνηση, όποτε εντοπίζεται ένας κόμβους v κατά την εξέταση του καταλόγου γειτνίασης ενός κόμβου u, ο u τίθεται *προκάτοχος* του v δηλαδή, $v.\pi = u$. Κάθε κόμβος αρχικά είναι χρωματισμένος λευκός. Όταν εντοπιστεί χρωματίζεται γκρι, ενώ όταν εντοπιστούν όλοι οι κόμβοι στον κατάλογο γειτνίασης του χρωματίζεται μαύρος.

Παρατήρηση. Σε αντίθεση με την οριζόντια διερεύνηση όπου το υπογράφημα προκατόχων αποτελεί ένα δένδρο, η καθοδική διερεύνηση σχηματίζει ένα υπογράφημα προκατόχων το οποίο μπορεί να περιέχει περισσότερα του ενός δένδρα, να είναι δηλαδή ένα *δάσος* (*forest*). Για τη καθοδική διερεύνηση το υπογράφημα προκατόχων $G_\pi = (V_\pi, E_\pi)$ ικανοποιεί

$$E_\pi = \{(v.\pi, v) : v \in V \text{ και } v.\pi \neq \text{NIL}\}.$$

Ο χρωματισμός των κόμβων με το τρόπο που αναφέρθηκε νωρίτερα εξασφαλίζει ότι κάθε κόμβος θα τοποθετηθεί σε ένα και μόνο δένδρο.

Η καθοδική διερεύνηση ενημερώνει δύο ακόμα χαρακτηριστικά για κάθε κόμβου του γραφήματος, τις λεγόμενες *χρονοσφραγίδες* (*timestamps*). Η χρονοσφραγίδα $v.d$ καταγράφει τον χρόνο πρώτου εντοπισμού του κόμβου v, οπότε και χρωματίζεται γκρίζος, και η χρονοσφραγίδα $v.f$ καταγράφει τον χρόνο κατά τον οποίο ολοκληρώνεται ο έλεγχος του καταλόγου γειτνίασης, οπότε και χρωματίζεται μαύρος. Θα αναφερόμαστε σε αυτόν τον χρόνο ως χρόνο περάτωσης. Οι χρονοσφραγίδες είναι

ακέραιοι αριθμοί μεταξύ 1 και $2|V|$ αφού κάθε κόμβος εντοπίζεται ακριβώς μια φόρα και χρωματίζεται μαύρος επίσης μια φορά μόνο. Για κάθε κόμβο u ισχύει

$$u.d < u.f. \tag{6.4}$$

Ο κόμβος u είναι λευκός πριν τον χρόνο $u.d$, γκρίζος μέχρι τον χρόνο $u.f$ και μαύρος για μεγαλύτερους χρόνους. Η διαδικασία DepthFirstSearch παρακάτω χρησιμοποιεί την καθολική μεταβλητή *time* για τη διαχείριση των χρονοσφραγίδων.

DepthFirstSearch(G)

```
1  for each vertex u ∈ V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ V
6      if u.color == WHITE
7          DFS-Visit(G, u)
```

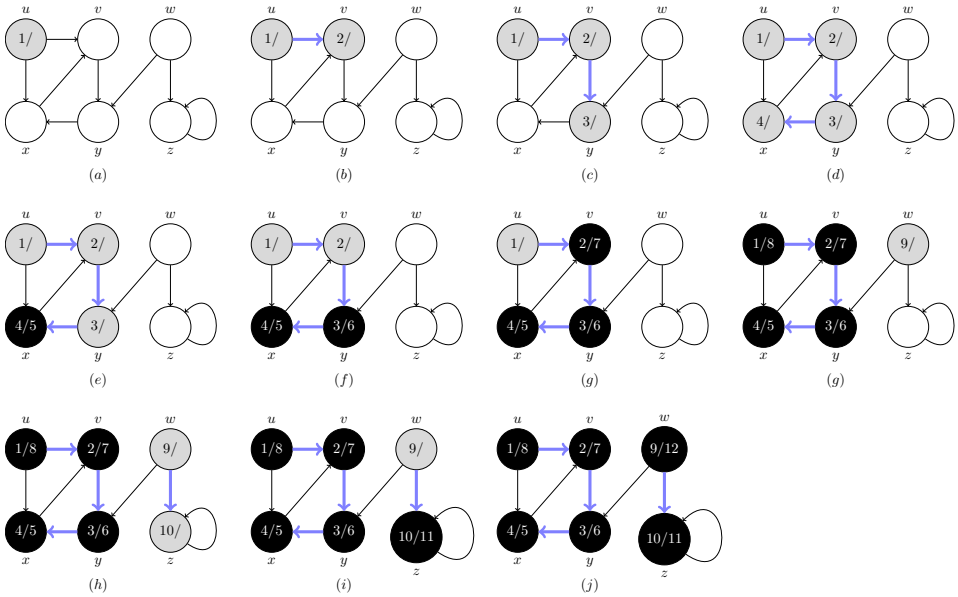
DFS-Visit(G, u)

```
1  time = time + 1
2  u.d = time
3  u.color = GRAY
4  for each vertex v ∈ Adj[u]
5      if v.color == WHITE
6          v.π = u
7          DFS-Visit(G, v)
8  time = time + 1
9  u.f = time
10 u.color = BLACK
```

Στις γραμμές 1–3 οι κόμβοι χρωματίζονται λευκοί και οι προκάτοχοι τους ορίζονται να είναι κενοί. Η γραμμή 4 αρχικοποιεί την καθολική μεταβλητή *time* για τη διαχείριση των χρονοσφραγίδων. Οι γραμμές 5–7 εξετάζουν τους λευκούς κόμβους με τη διαδικασίας DFS-Visit. Η κλήση DFS-Visit(G, u) κάνει τον κόμβο u τη ρίζα ενός νέου δένδρου στο δάσος των καθοδικών δένδρων. Όταν DFS-Visit(G, u) επιστρέψει ο κόμβος u θα έχει κάποιο χρόνο εντοπισμού $u.d$ και κάποιο χρόνο περάτωσης $u.f$, τον χρόνο ολοκλήρωσης του ελέγχου των κόμβων στον κατάλογο γειτνίασης του u. Στις γραμμές 1–3 στη διαδικασία DFS-Visit ενημερώνεται η καθολική μεταβλητή *time*, ορίζεται ο χρόνος εντοπισμού του κόμβου u και χρωματίζεται γκρι. Στις γραμμές 4–7 κάθε γειτονικός κόμβος v του u, αν είναι λευκός, ορίζεται ο προκάτοχος του εξετάζεται αναδρομικά. Όταν κάθε ακμή που εξέρχεται του u έχει εξεταστεί, η μεταβλητή *time* ενημερώνεται, ο χρόνος ολοκλήρωσης ελέγχου του κατάλογου γειτνίασης $u.f$ καταγράφεται, και χρωματίζεται μαύρος.

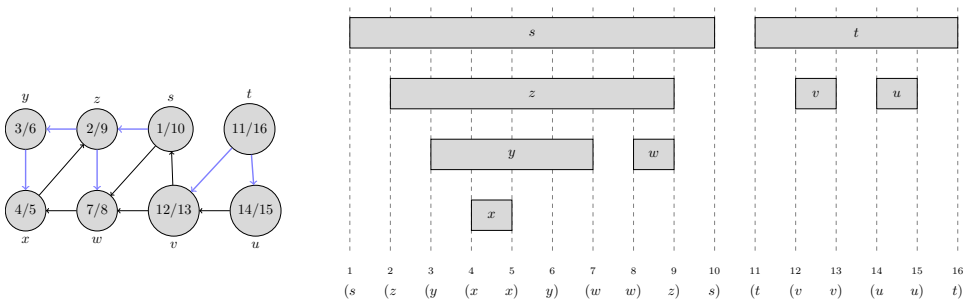
Τα αποτελέσματα της καθοδικής διερεύνησης εξαρτώνται από τη σειρά με την οποία εμφανίζονται οι κόμβοι στους καταλόγους γειτνίασης. Ο χρόνος αρχικοποίησης είναι $\Theta(|V|)$ ενώ η διαδικασία DFS-Visit καλείται ακριβώς μια φορά για κάθε κόμβο. Μια και το άθροισμα των μηκών των καταλόγων γειτνίασης είναι $\Theta(|E|)$ τότε ο χρόνος εκτέλεσης των γραμμών 4–7 στη διαδικασία DFS-Visit είνια $O(|V|+|E|)$. Συνεπώς ο χρόνος εκτέλεσης της διαδικασίας DepthFirstSearch είναι επίσης $O(|V|+|E|)$.

Για το κατευθυνόμενο γράφημα του Σχήματος 6.3 η λειτουργία της διαδικασίας DepthFirstSearch φαίνεται στο Σχήμα 6.9. Η διαδικασία της καθοδικής διερεύνησης αποκαλύπτει πολύτιμες πληροφορίες για τη δομή του γραφήματος. Το υπογράφημα προκατόχων αποτελεί ένα δάσος με τα καθοδικά δένδρα να αντιστοιχούν στις αναδρομικές κλήσεις της διαδικασίας DFS-Visit. Ειδικότερα, $u = v.\pi$



Σχήμα 6.9: Η λειτουργία της διαδικασίας DepththFirstSearch για τη διερεύνηση του κατευθυνόμενου γραφήματος του Σχήματος 6.5. Με μπλέ χρώμα σημειώνονται οι ακμές των καθοδικών δένδρων.

αν και μόνο αν η διαδικασία DFS-Visit(G, v) κλήθηκε κατά τη διάρκεια της διάνυσης του καταλόγου γειτνίασης του κόμβου u. Επιπλέον, ο κόμβος v είναι απόγονος του u αν και μόνο αν εντοπίζεται κατά τη διάρκεια που ο u είναι χρωματισμένος γκρι. Μια άλλη σημαντική ιδιότητα της καθοδικής διερεύνησης είναι ότι οι χρόνοι εντοπισμού και περάτωσης έχουν *δομή παρενθέσεων* (*parenthesis structure*). Αυτό σημαίνει ότι αν παραστήσουμε τον εντοπισμό του κόμβου u με μια αριστερή παρένθεση "$(u$'" και τον χρόνο περάτωσης με μια δεξιά παρένθεση "$u)$" τότε η χρονική ακολουθία των πράξεων εντοπισμού και περάτωσης είναι σωστή συντακτικά με την έννοια ότι οι διαδοχικές παρενθέσεις είναι σωστά φωλιασμένες. Ένα παράδειγμα της δομής παρενθέσεων από τη καθοδική διερεύνηση ενός κατευθυνόμενου γραφήματος φαίνεται στο Σχήμα 6.10.



Σχήμα 6.10: Το αποτέλεσμα της καθοδικής διερεύνησης ενός κατευθυνόμενου γραφήματος, τα χρονικά διαστήματα μεταξύ εντοπισμού και περάτωσης κάθε κόμβου και αντίστοιχη παρενθετική έκφραση.

Θεώρημα 3. Σε οποιαδήποτε καθοδική διερεύνηση ενός γραφήματος $G = (V, E)$, για κάθε ζεύγος κόμβων u και v, ισχύει μία και μόνο μία από τις ακόλουθες τρεις συνθήκες:

- τα διαστήματα $[u.d, u.f]$ και $[v.d, v.f]$ είναι ξένα μεταξύ τους και κανένας από τους κόμβους u και v δεν είναι απόγονους του άλλου στο καθοδικό δάσος,
- το διάστημα $[u.d, u.f]$ εμπεριέχεται τελείως στο διάστημα $[v.d, v.f]$ και ο u είναι απόγονος του v σε ένα καθοδικό δένδρο, ή

- το διάστημα $[v.d, v.f]$ εμπεριέχεται τελείως στο διάστημα $[u.d, u.f]$ και ο v είναι απόγονος του u σε ένα καθοδικό δένδρο.

Απόδειξη. Ας υποθέσουμε ότι $u.d < v.d$. Αν $v.d < u.f$ τότε $v.d < u.f$, δηλαδή ο v εντοπίστηκε ενόσω ο u ήταν ακόμα γκρίζος. Αυτό σημαίνει ότι ο v είναι απόγονος του u. Επιπλέον, αφού ο v εντοπίστηκε μετά τον u όλες οι εξερχόμενες από αυτόν ακμές εξερευνούνται και ο v περατώνεται προτού η διερεύνηση επιστρέψει στον u για να τον περατώσει. Αυτό σημαίνει ότι το διάστημα $[v.d, v.f]$ εμπεριέχεται πλήρως στο διάστημα $[u.d, u.f]$. Στην περίπτωση που $u.f < v.d$, έχουμε από την ανισότητα (6.4) ότι τα διαστήματα $[u.d, u.f]$ και $[v.d, v.f]$ είναι ξένα μεταξύ τους. Αυτό σημαίνει ότι κανένας κόμβος δεν εντοπίστηκε ενόσω ο άλλος ήταν γκρίζος και επομένως κανείς τους δεν είναι απόγονος του άλλου. Η περίπτωση $v.d < u.d$ είναι εντελώς ανάλογη. Επαναλαμβάνουμε το ίδιο επιχείρημα με το u στη θέση του v και αντίστροφα. □

Ένα άμεσο πόρισμα του παραπάνω θεωρήματος είναι το αποτέλεσμα

Πόρισμα 2. Ο κόμβος v είναι γνήσιος απόγονος του u στο καθοδικό δάσος ενός γραφήματος G αν και μόνο αν $u.d < v.d < v.f < u.f$.

6.1.4 Τοπολογική ταξινόμηση

Η *τοπολογική ταξινόμηση* (*topological sort*) ενός κατευθυνόμενου άκυκλου γραφήματος $G = (V, E)$, ή "ΚΑΓ" όπως λέγεται μερικές φορές (directed acyclig graph, dag), είναι η διάταξη των κόμβων σε μια γραμμική αλληλουχία τέτοια ώστε αν αν το γράφημα G περιέχει μια ακμή (u, v), ο κόμβος u να εμφανίζεται πριν τον v σε αυτή την αλληλουχία. Είναι προφανές ότι αν το γράφημα περιέχει κύκλους είναι αδύνατον να υπάρχει τέτοια αλληλουχία. Τα κατευθυνόμενα γραφήματα χρησιμοποιούνται σε πολλές εφαρμογές που απαιτούν να υποδειχθεί κάποια σειρά προτεραιότητας για ορισμένα γεγονότα. Στο Σχήμα 6.11 βλέπουμε ένα παράδειγμα που αφορά στη σειρά με την οποία κάποιος φορά τα ρούχα του το πρωί. Μια κατευθυνόμενη ακμή (u, v) στο ΚΑΓ του Σχήματος 6.11 υποδεικνύει ότι το ένδυμα u πρέπει να φορεθεί πριν από το ένδυμα v. Το Σχήμα Σχήμα 6.11(b) αναπαριστά το τοπολογικά ταξινομημένο ΚΑΓ με τη μορφή γραμμικής αλληλουχίας των κόμβων έτσι ώστε όλες οι κατευθυνόμενες ακμές να έχουν φορά από τα αριστερά προς τα δεξιά. Οι κόμβοι παρουσιάζονται κατά φθίνουσα σειρά ως προς το χρόνο περάτωσης.

Ο αλγόριθμος για την τοπολογική ταξινόμηση ενός ΚΑΓ παρουσιάζεται στη διαδικασία TopologicalSort παρακάτω. Ο χρόνος εκτέλεσης της διαδικασίας αυτής είναι ο χρόνος εκτέλεσης της καθοδικής διερεύνησης δηλαδή, $\Theta(|V| + |E|)$.

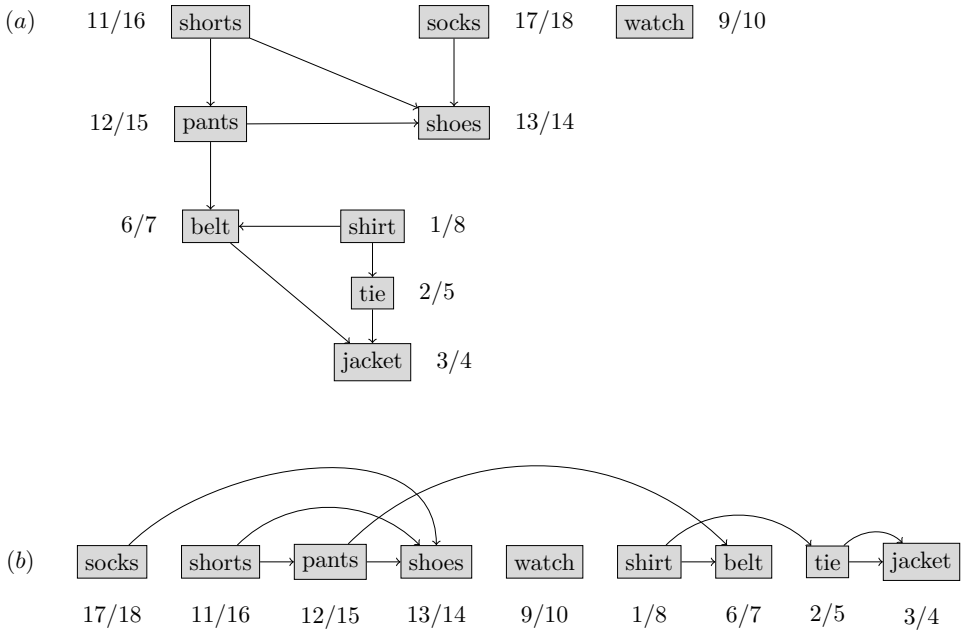
TopologicalSort(G)

1 DepthFirstSearch(G) to compute times $v.f$ for every $v \in V$
2 as $v.f$ is computed, insert it onto the front of a linked list
3 **return** the linked list of vertices

6.1.5 Ισχυρά συνδεδεμένες συνιστώσες

Έστω $G = (V, E)$ ένα κατευθυνόμενο γράφημα. Μια *ισχυρά συνδεδεμένη συνιστώσα* (*strongly connected component*) του G είναι ένα μέγιστο σύνολο κόμβων $C \subseteq V$ τέτοιο ώστε για κάθε ζεύγος κόμβων u και v στο C να ισχύει $u \rightsquigarrow v$ και $v \rightsquigarrow u$. Με άλλα λόγια, οι κόμβοι u και v να είναι προσπελάσιμοι ο ένας από τον άλλο. Στο Σχήμα 6.12 παρουσιάζεται ένα γράφημα με τρεις συνδεδεμένες συνιστώσες, χρωματισμένες με διαφορετικά χρώματα, και οι χρόνοι εντοπισμού και περάτωσης που υπολογίστηκαν κατά τη καθοδική διερεύνηση του γραφήματος.

Ο αλγόριθμος για την εύρεση των ισχυρά συνδεδεμένων συνιστωσών χρησιμοποιεί το *ανάστροφο γράφημα* $G^T = (V, E^T)$, για το οποίο ορίζουμε $E^T = \{(u, v) : (v, u) \in E\}$. Αυτό σημαίνει ότι το G^T περιέχει τις ίδιες ακμές, αλλά η φορά κάθε ακμής είναι ανεστραμμένη. Είναι προφανές



Σχήμα 6.11: (a) Τοπολογική ταξινόμηση ενδυμάτων. Μια κατευθυνόμενη ακμή (u, v) υποδεικνύει ότι το ένδυμα u πρέπει να φορεθεί πριν από το ένδυμα v. Δίπλα σε κάθε κόμβο αναγράφονται οι χρόνοι εντοπισμού και περάτωσης. (b) Το ίδιο γράφημα τοπολογικά ταξινομημένο με τους κόμβους διατεταγμένους από αριστερά προς τα δεξιά κατά φθίνουσα σειρά ως προς το χρόνο περάτωσης.

ότι τα γραφήματα G και G^T έχουν ακριβώς τις ίδιες ισχυρά συνδεδεμένες συνιστώσες: οι κόμβοι u και v είναι αμοιβαία προσπελάσιμοι στο G και μόνο αν είναι αμοιβαία προσπελάσιμοι στο G^T. Ο αλγόριθμος χρησιμοποιεί επίσης το λεγόμενο *γράφημα συνιστωσών* (*component graph*) $G^{SCC} = (V^{SCC}, E^{SCC})$, το οποίο ορίζεται ως εξής: Έστω ότι το γράφημα G έχεις τις ισχυρά συνδεδεμένες συνιστώσες $C_1, C_2, \ldots, C_k$. Τότε $V^{SCC} = \{v_1, v_2, \ldots, v_k\}$ όπου $v_i \in C_i$ για $i = 1, 2, \ldots, k$. Η ακμή $(v_i, v_j) \in E^{SCC}$ αν το γράφημα G περιέχει την κατευθυνόμενη ακμή (x, y) για κάποια $x \in C_i$ και $y \in C_j$. Με άλλα λόγια, αν ομαδοποιήσουμε όλους του κόμβους μιας ισχυρά συνδεδεμένης συνιστώσας αφαιρώντας τις μεταξύ τους ακμές, το γράφημα που προκύπτει είναι το G^{SCC}. Το γράφημα G^{SCC} για το γράφημα του Σχήματος 6.12 φαίνεται στο Σχήμα 6.13.

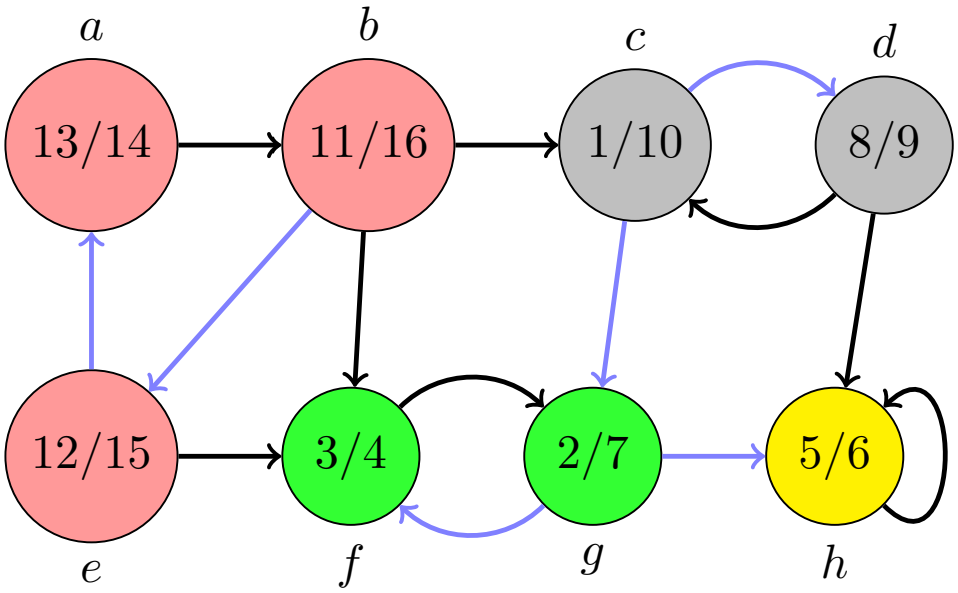
StronglyConnectedComponents(G)

1 call DepthFirstSearch(G) to compute finish times
2 create G^T
3 call DepthFirstSearch(G^T)
(consider vertices in decrasing order of $u.f$)
4 output the vertices of each tree in the depth-first forest
as a separate strongly connected component

Δείχνουμε κατ' αρχάς ότι το γράφημα συνιστωσών είναι άκυκλο. Αυτό χρησιμοποιείται στη διαδικασία StronglyConnectedComponents για να εξετάσει του κόμβους του με μια διαδικασία τοπολογικής ταξινόμησης.

Λήμμα 4. Έστω C και C' δύο διακριτές ισχυρά συνδεδεμένες συνιστώσες του κατευθυνόμενου γραφήματος $G = (V, E)$, και $u, v \in C$, $u', v' \in C'$. Υποθέτουμε ότι το γράφημα G περιέχει μια διαδρομή $u \rightsquigarrow u'$. Τότε το G δεν μπορεί να περιέχει μια διαδρομή $v' \rightsquigarrow v$.

Απόδειξη. Αν το γράφημα G περιέχει μια διαδρομή $v' \rightsquigarrow v$ τότε περιέχει επίσης τις διαδρομές $u \rightsquigarrow u' \rightsquigarrow v'$ και $v' \rightsquigarrow v \rightsquigarrow u$. Αυτό σημαίνει ότι οι κόμβοι u και u' είναι αμοιβαία προσπελάσιμοι



Σχήμα 6.12: Ένα γράφημα με τρεις συνδεδεμένες συνιστώσες, χρωματισμένες με διαφορετικά χρώματα, και οι χρόνοι εντοπισμού και περάτωσης που υπολογίστηκαν κατά τη καθοδική διερεύνηση του.

το οποίο έρχεται σε αντίθεση με το γεγονός ότι οι ισχυρά συνδεδεμένες συνιστώσες C και C' είναι διακριτές. □

Για τη συνέχεια, όταν αναφερόμαστε σε χρόνους εντοπισμού και χρόνους περάτωσης θα εννοούμε αυτούς που υπολογίστηκαν κατά την πρώτη κλήση της διαδικασίας StronglyConnectedComponents. Οι έννοιες αυτές επεκτείνονται και σε σύνολα κόμβων: αν $U \subseteq V$ τότε $d(U)$ και $f(U)$ είναι ο μικρότερος χρόνος εντοπισμού και ο μεγαλύτερος χρόνος περάτωσης, αντίστοιχα, οποιουδήποτε κόμβου του U. Επομένως,

$$d(U) = \min\{u.d : u \in U\}, \quad f(U) = \max\{u.f : u \in U\}.$$

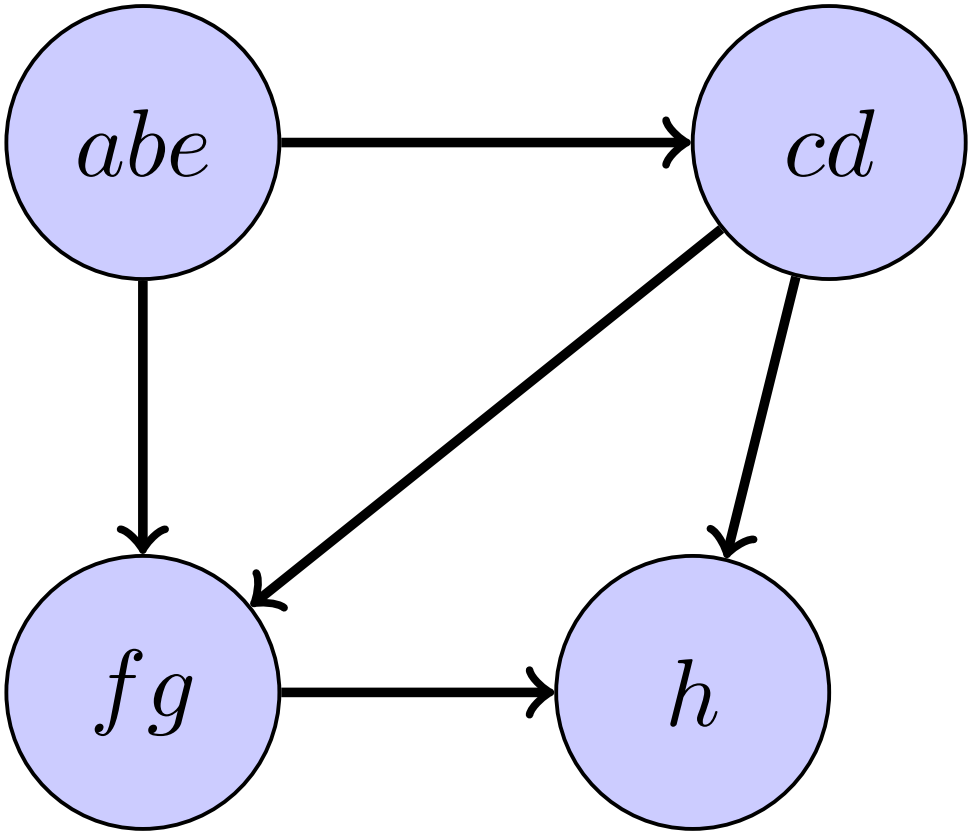
Για την απόδειξη του επόμενου αποτελέσματος θα χρησιμοποιήσουμε (χωρίς απόδειξη) το λεγόμενο *θεώρημα της λευκής διαδρομής* (*white–path theorem*):

Θεώρημα 4 (Θεώρημα της λευκής διαδρομής). Στο καθοδικό δάσος ενός γραφήματος G ο κόμβος v είναι απόγονος του κόμβου u αν και μόνο αν στον χρόνο εντοπισμού $u.d$ υπάρχει μια διαδρομή από τον u στον v η οποία αποτελείται μόνο από λευκούς κόμβους.

Λήμμα 5. Έστω C και C' δύο διακριτές ισχυρά συνδεδεμένες συνιστώσες του κατευθυνόμενου γραφήματος $G = (V, E)$. Υποθέτουμε ότι υπάρχει ακμή $(u, v) \in E$ τέτοια ώστε $u \in C$ και $v \in C'$. Τότε $f(C') > f(C)$.

Απόδειξη. Θεωρούμε πρώτα την περίπτωση $d(C') < d(C)$. Έστω x ο πρώτος κόμβος που εντοπίζεται στη συνιστώσα C'. Στο χρόνο $x.d$ όλοι οι κόμβοι των C και C' είναι λευκοί. Στον ίδιο χρόνο το γράφημα G περιέχει μια διαδρομή από τον κόμβο x προς κάθε κόμβο της συνιστώσας C' η οποία αποτελείται μόνο από λευκούς κόμβους. Στον ίδιο χρόνο, επειδή $(u, v) \in E$, για κάθε κόμβο $w \in C$ υπάρχει ένα μονοπάτι από τον κόμβο x στον κόμβο w το οποίο αποτελείται από λευκούς κόμβους μόνο: $x \rightsquigarrow u \to v \rightsquigarrow w$. Από το θεώρημα της λευκής διαδρομής, όλοι οι κόμβοι των συνιστωσών C και C' καθίστανται απόγονοι του x στο καθοδικό δένδρο. Από το Πόρισμα 2, ο κόμβος x έχει τον μεγαλύτερο χρόνο περάτωσης από όλους τους απογόνους του και επομένως έχουμε ότι $x.f = f(C') > f(C)$.

Ας υποθέσουμε τώρα ότι $d(C') > d(C)$. Έστω y ο πρώτος κόμβος που εντοπίζεται στη συνιστώσα C, έτσι ώστε $y.d = d(C)$. Στο χρόνο $y.d$ όλοι οι κόμβοι στη συνιστώσα C είναι λευκοί και το γράφημα G



Σχήμα 6.13: Το γράφημα G^{SCC} για το γράφημα του Σχήματος 6.12.

περιέχει μια διαδρομή από τον y προς κάθε κόμβου του C αποτελούμενη μόνο από λευκούς κόμβους. Από το θεώρημα της λευκής διαδρομής όλοι οι κόμβοι της συνιστώσας C γίνονται απόγονοι του y στο καθοδικό δένδρο, και από το Πόρισμα 2 έχουμε ότι $y.f = f(C)$. Επειδή $d(C') > d(C) = y.d$, όλοι οι κόμβου στη συνιστώσα C' είναι λευκοί στον χρόνο $y.d$. Αφού υπάρχει ακμή (u, v) από τη συνιστώσα C' στη συνιστώσα C, προκύπτει από το Λήμμα 4 ότι δεν μπορεί να υπάρχει διαδρομή από τη συνιστώσα C στη συνιστώσα C'. Επομένως, κανένας κόμβος στη συνιστώσα C' δεν είναι προσπελάσιμος από τον y. Αυτό σημαίνει ότι στον χρόνο $y.f$ όλοι οι κόμβοι του C' είναι λευκοί. Συνεπώς, για κάθε κόμβο $w \in C'$ έχουμε $w.f > y.f$ από το οποίο παίρνουμε ότι $f(C') > f(C)$. □

Πόρισμα 3. Έστω C και C' δύο διακριτές ισχυρά συνδεδεμένες συνιστώσες του κατευθυνόμενου γραφήματος $G = (V, E)$ με $f(C) > f(C')$. Τότε το σύνολο ακμών E^T δεν περιέχει καμμία ακμή (v, u) τέτοια ώστε $u \in C'$ και $v \in C$.

Απόδειξη. Από το Λήμμα 5 έχουμε ότι αν $f(C') < f(C)$ τότε δεν υπάρχει ακμή $(u, v) \in E$ με $u \in C'$ και $v \in C$. Μια και οι ισχυρά συνδεδεμένες συνιστώσες των G και G^T είναι οι ίδιες, αν δεν υπάρχει ακμή $(u, v) \in E$ τότε δεν υπάρχει ακμή $(v, u) \in E^T$ με $u \in C'$ και $v \in C$. □

Το παραπάνω αποτέλεσμα αποτελεί το κυριότερο συστατικό για την απόδειξη ότι η διαδικασία StronglyConnectedComponents υπολογίζει σωστά τις ισχυρές συνδεδεμένες συνιστώσες ενός γραφήματος. Πράγματι, ας εξετάσουμε τι γίνεται κατά τη καθοδική διερεύνηση του γραφήματος G^T:

- Η διερεύνηση ξεκινά από κάποιον κόμβο x του οποίου ο χρόνος περάτωσης $x.f$ είναι μέγιστος. Ο κόμβος αυτός ανήκει σε κάποια ισχυρά συνδεδεμένη συνιστώσα C. Επειδή ο χρόνος $x.f$ είναι μέγιστος, ο χρόνος $f(C)$ είναι μέγιστος πάνω σε όλες τις συνιστώσες.

- Όταν η διερεύνηση ξεκινά από τον κόμβο x, επισκέπτεται όλους τους κόμβους στη συνιστώσα C. Από το Πόρισμα 3 το γράφημα G^T δεν περιέχει ακμές από κόμβους του C προς κόμβους σε άλλες συνδεδεμένες συνιστώσες και επομένως η διερεύνηση από το C δεν επισκέπτεται

κόμβους σε άλλες συνδεδεμένες συνιστώσες. Αυτό σημαίνει ότι το καθοδικό δένδρο με ρίζα τον κόμβο x περιέχει ακριβώς τους κόμβους της συνιστώσας C.

- Έχοντας ολοκληρώσει τη διερεύνηση των κόμβων της συνιστώσας C, η δεύτερη καθοδική διερεύνηση επιλέγει ως ρίζα του νέου καθοδικού δένδρου έναν κόμβο από κάποια συνιστώσα C' για την οποία ο χρόνος περάτωσης $f(C')$ είναι μέγιστος πάνω από όλες τις συνιστώσες εκτός της C. Όπως πριν η διερεύνηση επισκέπτεται όλους τους κόμβους της συνιστώσας C'. Πάλι από το Πόρισμα 3, αν κάποιες ακμές στο γράφημα G^T εξέρχονται από τη συνιστώσα C' προς κάποια άλλη συνιστώσα, αναγκαστικά καταλήγουν στη συνιστώσα C τους οποίους η δεύτερη καθοδική διερεύνηση έχει ήδη επισκευθεί.

- Γενικότερα, όταν η καθοδική διερεύνηση του G^T επισκέπτεται οποιαδήποτε ισχυρά συνδεδεμένη συνιστώσα, όλες οι εξερχόμενες από αυτή ακμές καταλήγουν σε κόμβους τους οποίους η διερεύνηση έχει επισκευθεί. Επομένως, κάθε καθοδικό δένδρο αντιστοιχεί σε ακριβώς μια συνδεδεμένη συνιστώσα.

Αυτά τα συμπεράσματα μπορούν να συνοψιστούν στο θεώρημα

Θεώρημα 5. Η διαδικασία StronglyConnectedComponents υπολογίζει σωστά τις ισχυρά συνδεδεμένες συνιστώσες του κατευθυνόμενου γραφήματος G.

6.2 Ασκήσεις

1. Σε μια γιορτή, οι προσκεκλημένοι καθώς προσέρχονται χαιρετούν ο ένας τον άλλο δια χειραψίας, και κάθε καλεσμένος θυμάται πόσε χειραψίες αντάλλαξε. Στο τέλος της γιορτής ο αριθμός των χειραψιών αθροίζεται. Απόδείξτε ότι το άθροισμα είναι άρτιο. Αυτό είναι το *λήμμα της χειραψίας*: σε κάθε μη κατευθυνόμενο γράφο $G = (V, E)$ ισχύει ότι

$$\sum_{v \in V} \text{βαθμός}(v) = 2|E|.$$

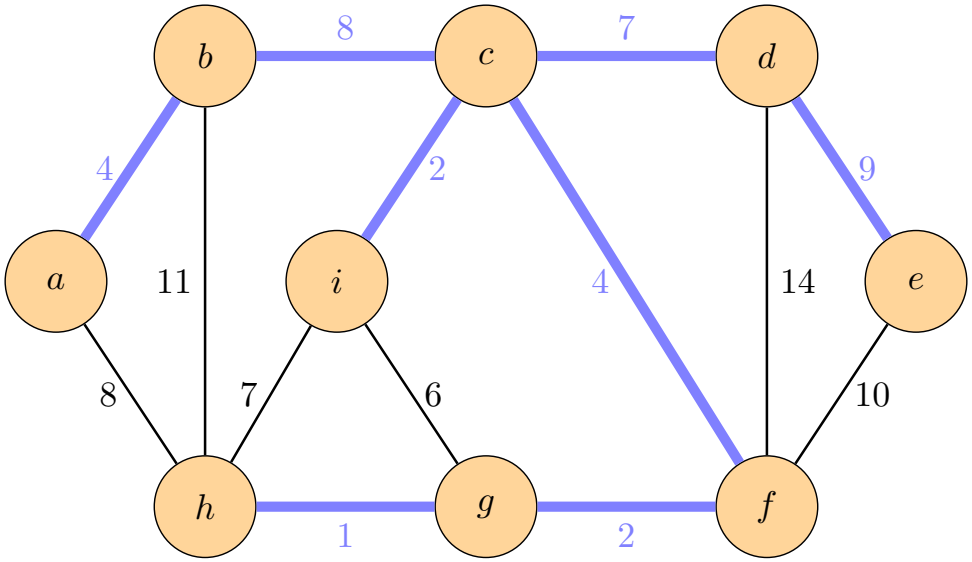
2. Δείξτε ότι αν ένα γράφημα περιλαμβάνει μια διαδρομή μεταξύ δύο κόμβων u και v τότε περιλαμβάνει μια απλή διαδρομή μεταξύ των u και v. Δείξτε ότι αν ένα κατευθυνόμενο γράφημα περιλαμβάνει έναν κύκλο, τότε περιλαμβάνει έναν απλό κύκλο.

3. Δείξτε ότι για οποιοδήποτε συνδεδεμένο μη κατευθυνόμενο γράφημα $G = (V, E)$ ισχύει ότι $|E| \geq |V| - 1$.

   Απόδειξη. Η πρόταση είναι προφανής για ένα συνδεδεμένο, μη κατευθυνόμενο γράφημα με $n = |V| \leq 3$. Θα υποθέσουμε λοιπόν ότι $n \geq 4$ και θα αποδείξουμε την πρόταση με επαγωγή σε άτοπο. Έστω $G = (V, E)$ ένα συνδεμένο, μη κατευθυνόμενο γράφημα με τον μικρότερο αριθμό κόμβων n για το οποίο $|E| \leq n - 2$. Δείχνουμε κατ' αρχάς ότι ένα τέτοιο γράφημα περιέχει έναν κόμβο με βαθμό ένα. Πράγματι, σε διαφορετική περίπτωση θα είχαμε, από την Άσκηση 1, ότι

$$2|E| = \sum_{v \in V} \text{βαθμός}(v) > 2n,$$

   γιατί βαθμός$(v) > 1$ για κάθε $v \in V$. Τότε $|E| > n > n - 2$, το οποίο έρχεται σε αντίθεση με την υπόθεσή μας. Συνεπώς υπάρχει τουλάχιστον ένας κόμβος v με βαθμό ένα. Έστω (v, w) η μοναδική ακμή του E η αποία εκκινεί από τον v. Θεωρούμε τώρα το γράφημα G' με σύνολο κόμβων $V \setminus \{v\}$ και σύνολο ακμών $E \setminus \{(v, w)\}$. Το G' είναι ένα συνδεδεμένο, μη κατευθυνόμενο γράφημα με $n - 1$ κόμβους για το οποίο $|E| \leq n - 2 - 1 = (n - 1) - 2$, το οποίο έρχεται σε αντίθεση με το γεγονός ότι το γράφημα G ήταν ελάχιστο. Συνεπώς, $|E| \geq n - 1 = |V| - 1$.



Σχήμα 6.14: Ένα ελαφρύτατο συνδετικό δένδρο ενός συνδεδεμένου γραφήματος. Οι μπλε ακμές είναι οι ακμές του ελαφρύτατου συνδετικού δένδρου με συνολικό βάρος 37.

4. Για κάποια δεδομένη αναπαράσταση ενός κατευθυνόμενου γραφήματος μέσω καταλόγων γειτνίασης, πόσος χρόνος απαιτείται για να υπολογιστεί ο βαθμός εξόδου κάθε κόμβου; Πόσος χρόνος απαιτείται για να υπολογιστούν οι βαθμοί εισόδου;

5. Δείξτε ότι, εφόσον δίνεται ένας πίνακας γειτνίασης για κάποιο κατευθυνόμενο γράφημα G, τότε για να διαπιστωθεί αν το G περιέχει έναν *καθολικό τερματικό κόμβο*, δηλαδή, έναν κόμβο με βαθμό εισόδου $|V| - 1$ και βαθμό εξόδου μηδέν, αρκεί χρόνος $O(|V|)$.

6. Υπολογίστε τα χαρακτηριστικά d και π των κόμβων σε μια οριζόντια διερεύνηση του γραφήματος στο Σχήμα 6.5 χρησιμοποιώντας ως αφετηριακό κόμβο τον κόμβο 3.

7. Υπολογίστε τα χαρακτηριστικά d και π των κόμβων σε μια οριζόντια διερεύνηση του γραφήματος στο Σχήμα 6.8 χρησιμοποιώντας ως αφετηριακό κόμβο τον κόμβο u. Υποθέστε ότι οι κόμβοι εμφανίζονται στους καταλόγους γειτνίασης με αλφαβητική σειρά.

8. Πως τροποποιείται η διαδικασία BreadthFirstSearch αν το γράφημα παριστάνεται μέσω πίνακα γειτνίασης; Ποιός είναι ο χρόνος εκτέλεσης της διαδικασίας BreadthFirstSearch τώρα;

6.3 Ελαφρύτατα συνδετικά δένδρα

Έστω $G = (V, E)$ ένα μη κατευθυνόμενο, συνδεδεμένο γράφημα. Θυμόμαστε ότι ένα μη κατευθυνόμενο γράφημα λέγεται συνδεδεμένο αν οι όλοι οι κόμβοι του συνδέονται ανά δύο με κάποια διαδρομή. Υποθέτουμε ακόμα ότι κάθε ακμή $(u, v) \in E$ χαρακτηρίζεται από ένα βάρος $w(u, v)$, για κάποια συνάρτηση βάρους $w : E \to \mathbb{R}$. Αναζητούμε ένα άκυκλο υποσύνολο $T \subseteq E$ το οποίο να συνδέει όλους τους κόμβους και του οποίου το συνολικό βάρος

$$w(T) = \sum_{(u,v)\in E} w(u, v)$$

να είναι ελάχιστο. Αφού το T είναι άκυκλο και συνδέει όλους τους κόμβους θα πρέπει να σχηματίζει ένα δένδρο, το οποίο ονομάζετα *συνδετικό δένδρο* (*spanning tree*), δείτε το Σχήμα 6.14. Το πρόβλημα του προσδιορισμού του δένδρου T ονομάζεται *πρόβλημα του ελαφρύτατου συνδετικού δένδρου* (*minimum spanning tree problem*).

Οι δύο αλγόριθμοι που θα δούμε χρησιμοποιούν μια άπληστη προσέγγιση για την επίλυση του προβλήματος. Αυτή η προσέγγιση αποτυπώνεται στο γεγονός ότι επεκτείνουν το συνδετικό δένδρο κατά μία ακμή κάθε φορά. Οι αλγόριθμοι διαχειρίζονται ένα σύνολο ακμών A τηρώντας την ακόλουθη αναλλοίωτη συνθήκη βρόχου:

> Πριν από κάθε επανάληψη το A αποτελεί υποσύνολο κάποιου ελαφρύτατου συνδετικού δένδρου.

Σε κάθε βήμα προσδιορίζουμε μια ακμή (u, v) η οποία μπορεί να προστεθεί στο σύνολο A χωρίς να παραβιαστεί η παραπάνω συνθήκη, με την έννοια ότι το σύνολο $A \cup \{(u, v)\}$ αποτελεί επίσης υποσύνολο κάποιου ελαφρύτατου συνδετικού δένδρου. Μια τέτοια ακμή ονομάζεται *ασφαλής* (*safe*) για το σύνολο A. Η διαδικασία GenericMST αποτελεί πρότυπο ενός τέτοιου αλγόριθμου:

```
GenericMST(G, w)
1  A = ∅
2  while A is not a spanning tree
3      Find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

Η αναλλοίωτη συνθήκη χρησιμοποιείται ως εξής:

Αρχικός έλεγχος. Μετά τη γραμμή 1 το σύνολο A ικανοποιεί τη συνθήκη κατά τετριμμένο τρόπο.

Έλεγχος διατήρησης. Η επαναληπτική διαδικασία στις γραμμές 2–4 προσθέτει μόνο ασφαλείς ακμές και επομένως διατηρεί την αναλλοίωτη συνθήκη.

Επιβεβαίωση αποτελέσματος. Όλες οι ακμές που έχουν προστεθεί στο A ανήκουν σε κάποιο ελαφρύτατο συνδετικό δένδρο και επομένως το σύνολο A που επιστρέφεται στη γραμμή 5 θα πρέπει να είναι ένα ελαφρύτατο συνδετικό δένδρο.

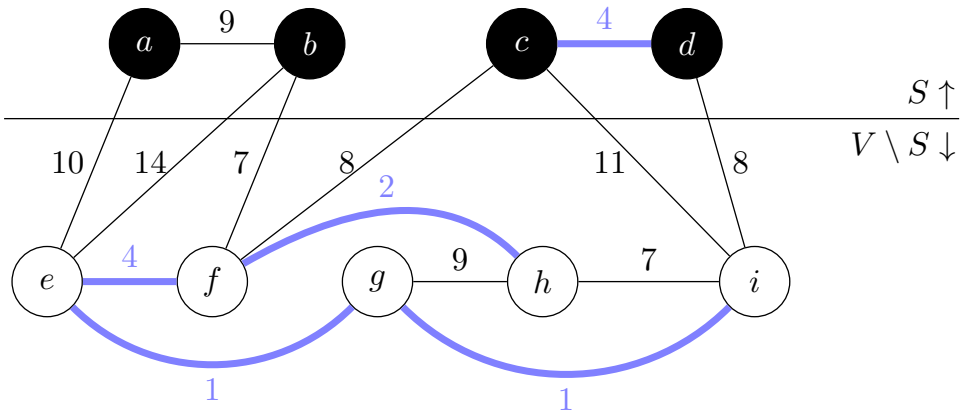
Το λεπτό σημείο της παραπάνω διαδικασίας είναι φυσικά η εύρεση μιας ασφαλούς ακμής. Η ύπαρξη της είναι εξασφαλισμένη γιατί κατά την εκτέλεση της γραμμής 3 η αναλλοίωτη συνθήκη ορίζει ότι υπάρχει ένα συνδετικό δένδρο T τέτοιο ώστε $A \subseteq T$. Κατά τη διάρκεια της επαναληπτικής διαδικασίας στις γραμμές 2–4 το σύνολο A θα πρέπει να είναι γνήσιο υποσύνολο του T, και επομένως θα πρέπει να υπάρχει κάποια ακμή $(u, v) \in T$ τέτοια ώστε $(u, v) \notin A$ και η (u, v) να είναι ασφαλής για το A.

Στο υπόλοιπο της ενότητας παραθέτουμε έναν κανόνα για την αναγνώριση ασφαλών ακμών. Αυτός ο κανόνας χρησιμοποιείται και από τους δύο αλγόριθμους που επιλύουν το πρόβλημα του ελαφρύτατου συνδετικού δένδρου. Θα χρειαστούμε τους παρακάτω ορισμούς.

Ορισμός. *Τομή* (*cut*) $(S, V \setminus S)$ ενός μη κατευθυνόμενου γραφήματος $G = (V, E)$ είναι μια διαμέριση του V. Λέμε ότι μια ακμή $(u, v) \in E$ *διασχίζει* (*crosses*) την τομή $(S, V \setminus S)$ αν ένα από τα άκρα της βρίσκεται στο S και το άλλο στο $V \setminus S$. Λέμε ότι μια τομή *σέβεται* (*respects*) ένα σύνολο ακμών A εάν καμιά ακμή του συνόλου A δεν διασχίζει την τομή. Μια ακμή που διασχίζει κάποια τομή χαρακτηρίζεται ως *ελαφρά* (*light edge*) ως προς τη διάσχιση της τομής εάν το βάρος της είναι το ελάχιστο μεταξύ των βαρών όλων των ακμών που διασχίζουν την τομή. Εν γένει, λέμε ότι μια ακμή που διαθέτει κάποια ιδιότητα είναι *ελαφρά* ως προς αυτή την ιδιότητα εάν το βάρος της είναι το ελάχιστο μεταξύ των βαρών όλων των ακμών που διαθέτουν αυτή την ιδιότητα.

Ένα παράδειγμα τομής παρουσιάζεται στο Σχήμα 6.15. Το υποσύνολο A των ακμών του γραφήματος απεικονίζεται με μπλέ χρώμα. Η τομή $(S, V \setminus S)$ σέβεται το A αφού καμιά μπλε ακμή δεν διασχίζει την τομή.

Θεώρημα 1. Έστω $G = (V, E)$ συνδεδεμένο, μη κατευθυνόμενο γράφημα με συνάρτηση βάρους $w : E \to \mathbb{R}$. Έστω A ένα υποσύνολο του E το οποίο εμπεριέχεται σε ένα ελαφρύτατο συνδετικό



Σχήμα 6.15: Οι κόμβοι του συνόλου S απεικονίζονται με μαύρο χρώμα ενώ οι κόμβοι του $V \setminus S$ με λευκό. Οι ακμές που διασχίζουν την τομή είναι αυτές που συνδέουν λευκούς με μαύρους κόμβους. Η ακμή (b, f) είναι η μοναδική ελαφρά ακμή ως προς τη διάσχιση της τομής.

δένδρο για το G, έστω $(S, V \setminus S)$ οποιαδήποτε τομή του G η οποία σέβεται το A, και έστω (u, v) μια ελαφρά ακμή ως προς τη διάσχιση της $(S, V \setminus S)$. Η ακμή (u, v) είναι ασφαλής για το A.

Απόδειξη. Έστω T ένα ελαφρύτατο συνδετικό δένδρο το οποίο περιέχει το A. Υποθέτουμε ότι το T δεν περιέχει την ελαφρά ακμή (u, v), διαφορετικά το θεώρημα έχει αποδειχθεί. Θα κατασκευάσουμε ένα άλλο συνδετικό δένδρο T' το οποίο να περιέχει το $A \cup \{(u, v)\}$, αποδεικνύοντας έτσι ότι η ακμή (u, v) είναι ασφαλής για το A.

Η ακμή (u, v) σχηματίζει κύκλο μαζί με τις ακμές της απλής διαδρομής p από το u στο v στο συνδετικό δένδρο T. Αφού οι κόμβοι u και v είναι σε διαφορετικές πλευρές της τομής $(S, V \setminus S)$, τουλάχιστον μια ακμή του T εμπεριέχεται στη διαδρομή p και διασχίζει την τομή. Αν μια τέτοια ακμή είναι η (x, y) τότε αυτή δεν ανήκει στο σύνολο A γιατί η τομή σέβεται το A. Αφού η ακμή (x, y) βρίσκεται στο μοναδικό μονοπάτι στο T από το u στο v , η αφαίρεση της διαιρεί το T σε δύο μέρη. Η προσθήκη της ακμής (u, v) σχηματίζει ένα νέο συνδετικό δένδρο $T' = (T \setminus \{(x, y)\}) \cup \{(u, v)\}$.

Δείχνουμε ότι το T' είναι ελαφρύτατο συνδετικό δένδρο. Αφού η ακμή (u, v) είναι ελαφρά ως προς τη διάσχιση της τομής $(S, V \setminus S)$ και η (x, y) την δισχίζει, έχουμε ότι $w(u, v) \leq w(x, y)$. Επομένως,

$$w(T') = w(T) - w(x, y) + w(u, v) \leq w(T).$$

Όμως το T είναι ελαφρύτατο συνδετικό δένδρο, συνεπώς $w(T) \leq w(T')$. Αυτό δείχνει ότι το T' είναι επίσης ελαφρύτατο συνδετικό δένδρο.

Δείχνουμε τώρα ότι η ακμή (u, v) είναι ασφαλής για το A. Έχουμε $A \subseteq T'$ γιατί $A \subseteq T$ και $(x, y) \notin A$, επομένως $A \cup \{(x, y)\} \subseteq T'$. Αφού το T' είναι ελαφρύτατο συνδετικό δένδρο η ακμή (u, v) είναι ασφαλής για το A. □

Παρατήρηση. Μπορούμε να καταλάβουμε τη λειτουργία της διαδικασίας GenericMST χρησιμοποιώντας το προηγούμενο θεώρημα:

- Το σύνολο A είναι πάντα άκυκλο, ως ένα υποσύνολο ενός ελαφρύτατου συνδετικού δένδρου.
- Το γράφημα $G_A = (V, A)$ αποτελεί δάσος και κάθε μια από τις συνδεδεμένες συνιστώσες του είναι ένα δένδρο. Οποιαδήποτε ακμή (u, v) ασφαλής για το A, ενώνει διαφορετικές συνιστώσες του G_A αφού το $A \cup \{(u, v)\}$ πρέπει να είναι άκυκλο.
- Οι εντολές της επανάληψης **while** στις γραμμές 2–4 της διαδικασίας GenericMST εκτελούνται ακριβώς $|V| - 1$ φορές αφού ανακαλύπτει μια από τις $|V| - 1$ ακμές του ελαφρύτατα συνδετικού δένδρου κάθε φορά.

- Αρχικά, όπου $A = \emptyset$, υπάρχουν ακριβώς $|V|$ δένδρα στο γράφημα G_A και κάθε επανάληψη μειώνει τον αριθμό τους κατά ένα. Όταν το δάσος περιέχει ένα μόνο δένδρο, η διαδικασία τερματίζεται.

Το αποτέλεσμα που ακολουθεί είναι άμεση συνέπεια του Θεωρήματος 1:

Πόρισμα. Έστω $G = (V, E)$ συνδεδεμένο, μη κατευθυνόμενο γράφημα με συνάρτηση βάρους $w : E \to \mathbb{R}$. Έστω A ένα υποσύνολο του E το οποίο εμπεριέχεται σε ένα ελαφρύτατο συνδετικό δένδρο για το G και $C = (V_C, E_C)$ μια συνδεδεμένη συσνιστώσα στο δάσος $G_A = (V, A)$. Αν (u, v) είναι ελαφρά ακμή η οποία συνδέει τη συνιστώσα C με κάποια άλλη συνιστώσα στο G_A τότε η (u, v) είναι ασφαλής για το A.

Απόδειξη. Η τομή $(V_C, V \setminus V_C)$ σέβεται το A και η ακμή (u, v) είναι ελαφρά ως προς αυτή την τομή. Επομένως η ακμή (u, v) είναι ασφαλής για το A. □

6.3.1 Δομές δεδομένων για παράσταση ξένων συνόλων

Ορισμένες εφαρμογές, συμπεριλαμβανομένης και αυτής του προσδιορισμού των συνδεδεμένων συνιστωσών ενός μη κατευθυνόμενου γραφήματος, περιλαμβάνουν την ομαδοποίηση n διακριτών στοιχείων σε μια συλλογή ξένων συνόλων, δηλαδή σύνολα των οποίων η τομή ανά δύο είναι κενή. Σημαντικές λειτουργίες είναι η εύρεση του συνόλου στο οποίο ανήκει ένα δεδομένο στοιχείο και η συνένωση δύο συνόλων.

Μια *δομή δεδομένων ξένων συνόλων* (*disjoint–set data structure*) τηρεί μια συλλογή $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ ξένων δυναμικών συνόλων. Το κάθε σύνολο προσδιορίζεται μέσω κάποιου *αντιπροσώπου* (*representative*) του, που είναι κάποιος μέλος του συνόλου. Οι λειτουργίες που υποστηρίζονται είναι οι εξής:

MakeSet(x), η οποία δημιουργεί ένα νέο σύνολο με μοναδικό στοιχείο το x. Δεδομένου του ότι τα σύνολα είναι ξένα μεταξύ τους θα πρέπει το x να μην περιέχεται ήδη σε κάποιο άλλο σύνολο.

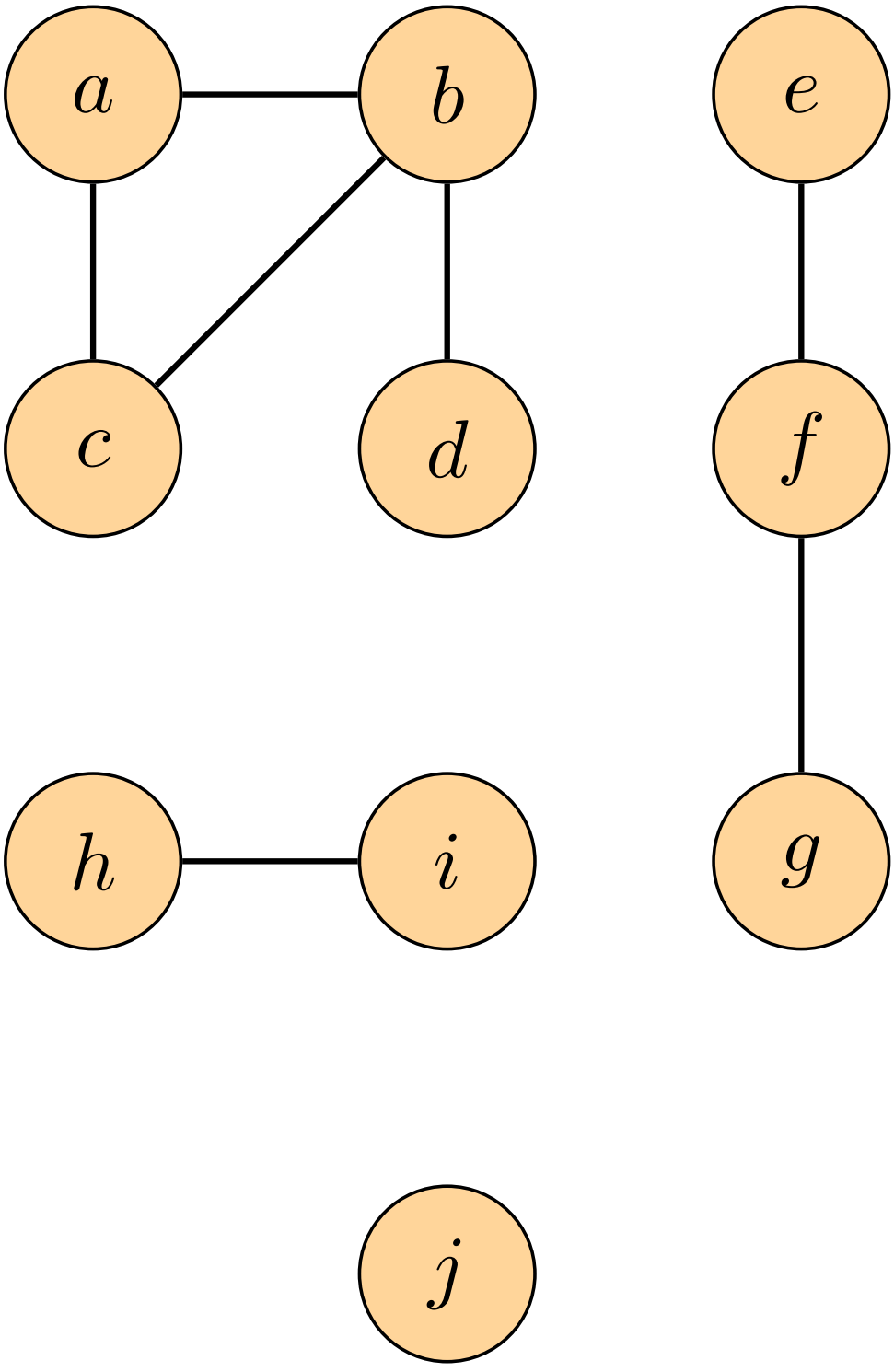
Union(x, y), η οποία συνενώνει τα σύνολα που περιέχουν τα στοιχεία x και y, έστω S_x και $S - y$, σε ένα νέο σύνολο το οποίο αποτελεί την ένωση αυτών των συνόλων. Τα σύνολα προϋποτίθενται ξένα μεταξύ τους πριν τη συνένωση. Ο αντιπρόσωπος του συνόλου $S_x \cup S_y$ μπορεί να είναι οποιοδήποτε μέλος του. Τα σύνολα S_x και S_y αφαιρούνται από τη συλλογή μετά την πράξη της συνένωσης.

FindSet(x), η οποία επιστρέφει έναν δείκτη προς τον αντιπρόσωπο του (μοναδικού) συνόλου που περιέχει το στοιχείο x.

Ως ένα παράδειγμα δείχνουμε πως μπορούμε να χρησιμοποιήσουμε αυτές τις λειτουργίες στον υπολογισμό των συνδεδεμένων συνιστωσών ενός μη κατευθυνόμενου γραφήματος $G = (V, E)$. Όταν οι ακμές του γραφήματος δεν μεταβάλλονται με τον χρόνο οι συνδεδεμένες συνιστώσες υπολογίζονται ταχύτερα μέσω της καθοδικής διερεύνησης. Στις περιπτώσεις όμως που οι ακμές προστίθενται δυναμικά είμαστε αναγκασμένοι να τηρούμε τις συνδεδεμένες συνιστώσες ως συλλογή ξένων συνόλων. Η διαδικασία ConnectedComponents

```
ConnectedComponents(G)
1  for every v ∈ G
2      MakeSet(v)
3  for every edge (u, v) ∈ E
4      if FindSet(u) ≠ FindSet(v)
5          Union(u, v)
```

αρχικά τοποθετεί κάθε κόμβο v σε δικό του, ξεχωριστό σύνολο. Στη συνέχεια, για κάθε ακμή (u, v) συνενώνει τα σύνολα που περιέχουν τους u και v. Αφού εξεταστούν όλες οι ακμές, δύο κόμβοι βρίσκονται στην ίδια συνδεδεμένη συνιστώσα αν και μόνο αν οι κόμβοι βρίσκονται στο ίδιο σύνολο, δείτε την Άσκηση 6. Η διαδικασία SameComponent



Σχήμα 6.16: Ένα γράφημα με τέσσερεις συνδεδεμένες συνιστώσες $\{a, b, c, d\}$, $\{e, f, g\}$, $\{h, i\}$ και $\{j\}$.

Ακμή	Συλλογή ξένων συνόλων									
Αρχικά σύνολα	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(a,c)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(h,i)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(a,b)	$\{a,b,c,d\}$				$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(e,f)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$
(b,c)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

SAMECOMPONENT(u, v)
1 **if** FINDSET(u) == FINDSET(v)
2 **return** TRUE
3 **else**
4 **return** FALSE

αποφαίνεται αν δύο κόμβοι ανήκουν στην ίδια συνδεδεμένη συνιστώσα. Ο πίνακας παρακάτω παρουσιάζει τα στιμιότυπα της κατασκευής των ξένων συνόλων μέσω της διαδικασίας CONNECTEDCOMPONENTS για το γράφημα στο Σχήμα 6.16, το οποίο έχει τέσσερεις συνδεδεμένες συνιστώσες.

6.4 Ο αλγόριθμος του Kruskal

Ο αλγόριθμος του Kruskal ανήκει στην κατηγορία των άπληστων αλγόριθμων. Στον αλγόριθμο του Kruskal το σύνολο A του γενικού αλγόριθμου GENERICMST είναι ένα δάσος. Η ασφαλής ακμή που προστίθεται στο A είναι πάντα μια ακμή (u, v) ελάχιστου βάρους η οποία συνδέει δύο διαφορετικές συνιστώσες. Έστω C_1 και C_2 τα δένδρα που συνδέονται μεταξύ τους μέσω αυτής της ακμής (u, v). Δεδομένου ότι η (u, v) θα πρέπει να είναι μια ελαφρά ακμή ως προς τη σύνδεση του C_1 με κάποιο άλλο δένδρο, το Πόρισμα 1 συνεπάγεται ότι η (u, v) είναι ασφαλής για το C_1.

Η υλοποίηση βασίζεται στην τήρηση κάποιων ξένων συνόλων. Κάθε τέτοιο σύνολο περιέχει τους κόμβους ενός δένδρου του τρέχοντος δάσου. Η λειτουργία FINDSET(u) επιστρέφει ένα στοιχείο–αντιπρόσωπο του συνόλου στο οποίο ανήκει ο u. Μπορούμε να προσδιορίσουμε αν δύο κόμβοι u και v ανήκουν στο ίδιο δένδρο ελέγχοντας αν το στοιχείο FINDSET(u) συμπίπτει με το FINDSET(v). Η συνένωση των δένδρων πραγματοποιείται με τη διαδικασία UNION.

KRUSKAL(G, w)
1 $A = \emptyset$
2 **for** every $v \in V$
3 MAKESET(v)
4 Sort the list of edges into increasing order by weight w
5 **for** every edge $(u, v) \in E$ in decreasing order by weight
6 **if** FINDSET$(u) \neq$ FINDSET(v)
7 $A = A \cup \{(u, v)\}$
8 UNION(u, v)
9 **return** A

Ένα παράδειγμα της λειτουργίας του αλγόριθμου του Kruskal φαίνεται στο Σχήμα 6.17. Συνοπτικά:

- Στις γραμμές 1–3 ορίζεται ως αρχικό σύνολο A το κενό σύνολο και δημιουργούνται $|V|$ δένδρα, καθένα από τα οποία περιέχει έναν κόμβο.
- Στη γραμμή 4 διατάσσονται οι ακμές του συνόλου E κατά μη φθίνουσα σειρά ως προς το βάρος.

- Ο βρόχος στις γραμμές 5–8 ελέγχει, για κάθε ακμή (u, v), αν τα άκρα της u και v ανήκουν στο ίδιο δένδρο.
- Αν ανήκουν, τότε η προσθήκη της στο δάσος θα δημιουργήσει κύκλο και συνεπώς η ακμή απορρίπτεται. Σε αντίθετη περίπτωση, η ακμή προστίθεται στο δάσος A και οι κόμβοι των δύο δένδρων συγχωνεύονται στη γραμμή 8.

6.5 Ο αλγόριθμος του Prim

Μια βασική ιδιότητα του αλγόριθμου του Prim είναι ότι οι ακμές που προστίθενται στο σύνολο A του γενικού αλγόριθμου GenericMST σχηματίζουν πάντα ένα ακριβώς δένδρο και όχι δάσος δένδρων. Όπως φαίνεται και στο Σχήμα 6.18 το δένδρο ξεκινά από έναν αυθαίρετο κόμβο r και επεκτείνεται μέχρι να καλύψει όλους τους κόμβους του συνόλου V. Σε κάθε βήμα προστίθεται στο A μια ελαφρά ακμή ως προ τη σύνδεση του A με κάποιον απομονωμένο κόμβο του γραφήματος $G_A = (V, A)$. Με βάση το Πόρισμα 1 προστίθενται πάντα ακμές οι οποίες είναι ασφαλείς για το A. Έτσι, όταν ο αλγόριθμος τερματιστεί, οι ακμές του συνόλου A συνιστούν ένα ελαφρύτατο συνδετικό δένδρο. Όπως και στον αλγόριθμο του Kruskal, η στρατηγική είναι άπληστη, γιατί σε κάθε βήμα το δένδρο επεκτείνεται κατά μια ακμή η οποία το επιβαρύνει με την ελάχιστη δυνατή ποσότητα.

Στον ψευδοκώδικα που ακολουθεί το συνδεδεμένο γράφημα G και ο ριζικός κόμβος r του προς ανάπτυξη ελαφρύτατου συνδετικού δένδρου αποτελούν στοιχεία εισόδου του αλγόριθμου. Κατά την εκτέλεση τπυ αλγόριθμου όλοι οι κόμβοι που δεν ανήκουν στο δένδρο είναι καταχωρισμένοι σε μια ουρά προτεραιότητας ελαχίστου Q με βάση ένα χαρακτηριστικό *key*. Για κάθε κόμβο v το χαρακτηριστικό $v.key$ είναι το ελάχιστο από τα βάρη των ακμών που συνδέουν τον v με κάποιον κόμβο του δένδρου. Αν δεν υπάρχει τέτοια ακμή τότε $v.key = \infty$. Το χαρακτηριστικό $v.\pi$ υποδεικνύει τον προκάτοχο του v στο δένδρο. Κατά τη διάρκεια της εκτέλεσης το σύνολο A της διαδικασίας GenericMST τηρείται ως

$$A = \{(v, v.\pi) : v \in V \setminus \{r\} \setminus Q\}.$$

Όταν ο αλγόριθμος τερματίσει η ουρά προτεραιότητας ελαχίστου Q είναι κενή. Επομένως το ελαφρύτατο συνδετικό δένδρο είναι το

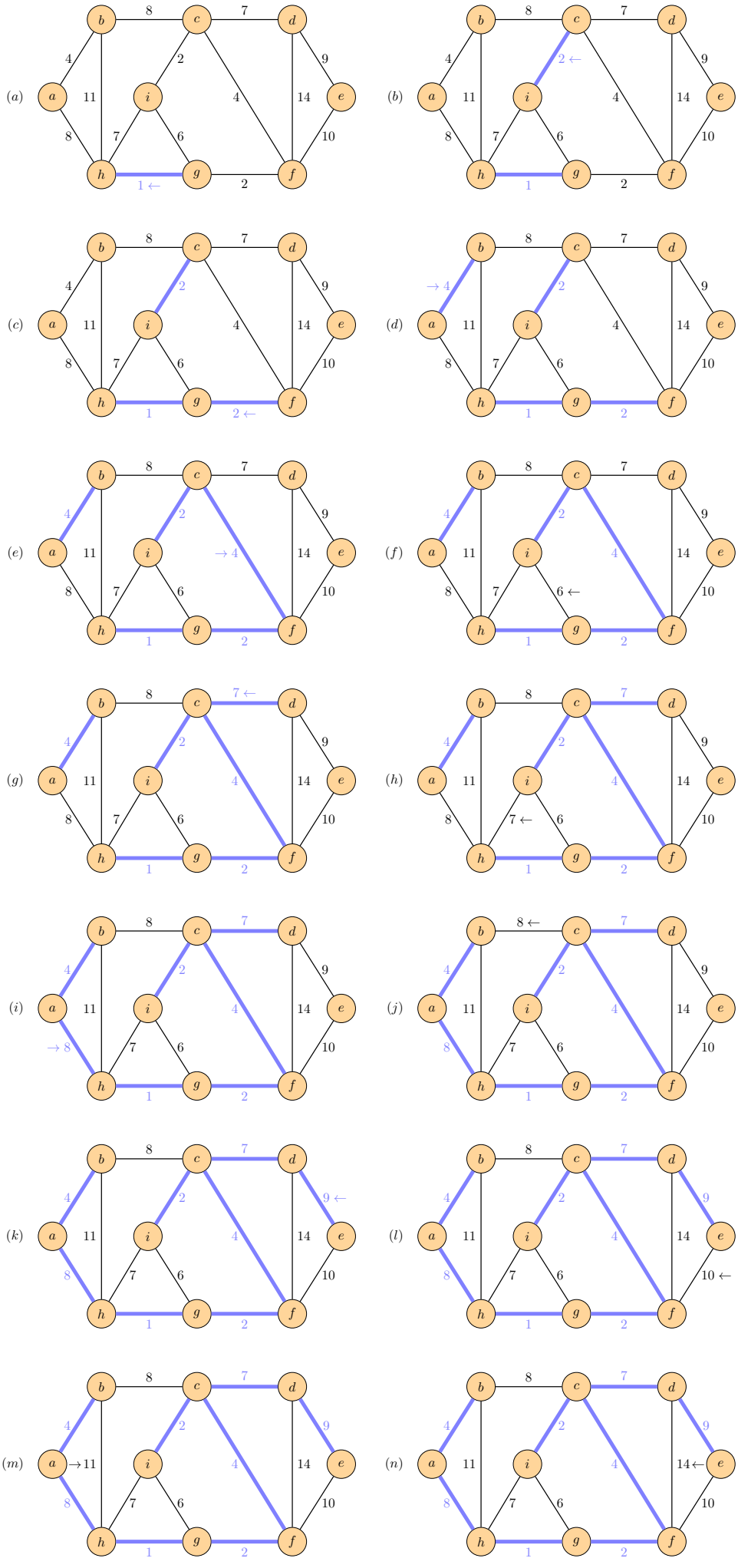
$$A = \{(v, v.\pi) : v \in V \setminus \{r\}\}.$$

Prim(G, w, r)

```
1  for every u ∈ V
2      u.key = ∞
3      u.π = NIL
4  r.key = 0
5  Q = V
6  while Q ≠ ∅
7      u = ExtractMin(Q)
8      for every v ∈ Adj[u]
9          if v ∈ Q and w(u, v) < v.key
10             v.π = u
11             v.key = w(u, v)
```

Η λειτουργία του αλγόριθμου του Prim για το γράφημα του Σχήματος 6.14 αναπαρίσταται γραφικά στο Σχήμα 6.18. Συνοπτικά:

- Στις γραμμές 1–5 ορίζεται η τιμή του χαρακτηριστικού key για όλους του κόμβους το ∞, εκτός του ριζικού κόμβου r του οποίου τίθεται ίσο με μηδέν ώστε να εξετατεί πρώτος. Ο προκάτοχος όλων των κόμβων τίθεται NIL και ορίζεται η ουρά προτεραιότητας ελαχίστου Q ώστε να


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)
(k)
(l)
(m)
(n)

περιέχει αρχικά το σύνολο όλων των κόμβων. Ο αλγόριθμος τηρεί την ακόλουθη αναλλοίωτη συνθήκη:

1. Πριν από κάθε επανάληψη του βρόχου **while** στις γραμμές 6-11
$$A = \{(v, v.\pi) : v \in V \setminus \{r\} \setminus Q\}.$$
2. Οι κόμβοι που έχουν ήδη συμπεριληφθεί στο ελαφρύτατο συνδετικό δένδρο είναι εκείνοι που ανήκουν στο σύνολο $V \setminus Q$.
3. Για όλους τους κόμβους $v \in Q$, αν $v.\pi \neq$ NIL τότε $v.\text{key} < \infty$ και το χαρακτηριστικό $v.\text{key}$ ισούται με το βάρος μια ελαφράς ακμής $(v, v.\pi)$ ως προς τη σύνδεση του v με κόμβους που ανήκουν ήδη στο ελαφρύτατο συνδετικό δένδρο.

- Στη γραμμή 7 προσδιορίζεται ως κόμβος $u \in Q$ το άκρο μιας ελαφράς ακμής ως προς τη διάσχιση της τομής $(V \setminus Q, Q)$ (εκτός από την πρώτη επανάληψη όπου έχουμε $u = r$). Με την αφαίρεση του κόμβου u από το σύνολο Q ο u προστίθεται αυτομάτως στο σύνολο $V \setminus Q$ των κόμβων που ανήκουν στο δένδρο, ενώ η ακμή $(v, v.\pi)$ προστίθεται στο A.

- Ο βρόχος **for** στις γραμμές 8–11 ενημερώνει το χαρακτηριστικό key όλων των κόμβων v που γειτνιάζουν με τον u αλλά δεν ανήκουν στο δένδρο. Η ενημέρωση αυτή εξασφαλίζει την τήρηση του τρίτου μέρους της αναλλοίωτης συνθήκης.

6.6 Ασκήσεις

1. Έστω (u, v) μια ακμή του συνδεδεμένου γγραφήματος G. Δείξτε ότι η ακμή (u, v) ανήκει σε κάποιο ελαφρύτατο συνδετικό δένδρο του G.

2. Δείξτε με ένα αντιπαράδειγμα ότι η παρακάτω εικασία, το αντίστροφο του Θεωρήματος 1, είναι εσφαλμένη: Έστω $G = (V, E)$ συνδεδεμένο, μη κατευθυνόμενο γράφημα με συνάρτηση βάρους $w : E \to \mathbb{R}$. Έτσω A ένα υποσύνολο του E το οποίο εμπεριέχεται σε ένα ελαφρύτατο συνδετικό δένδρο για το G, έστω $(S, V \setminus S)$ οποιαδήποτε τομή του G η οποία σέβεται το A, και έστω (u, v) μια ασφαλής για το A ακμή η οποία διασχίζει την $(S, V \setminus S)$. Τότε η (u, v) είναι ελαφρά ακμή ως προς τη διάσχιση της τομής αυτής.

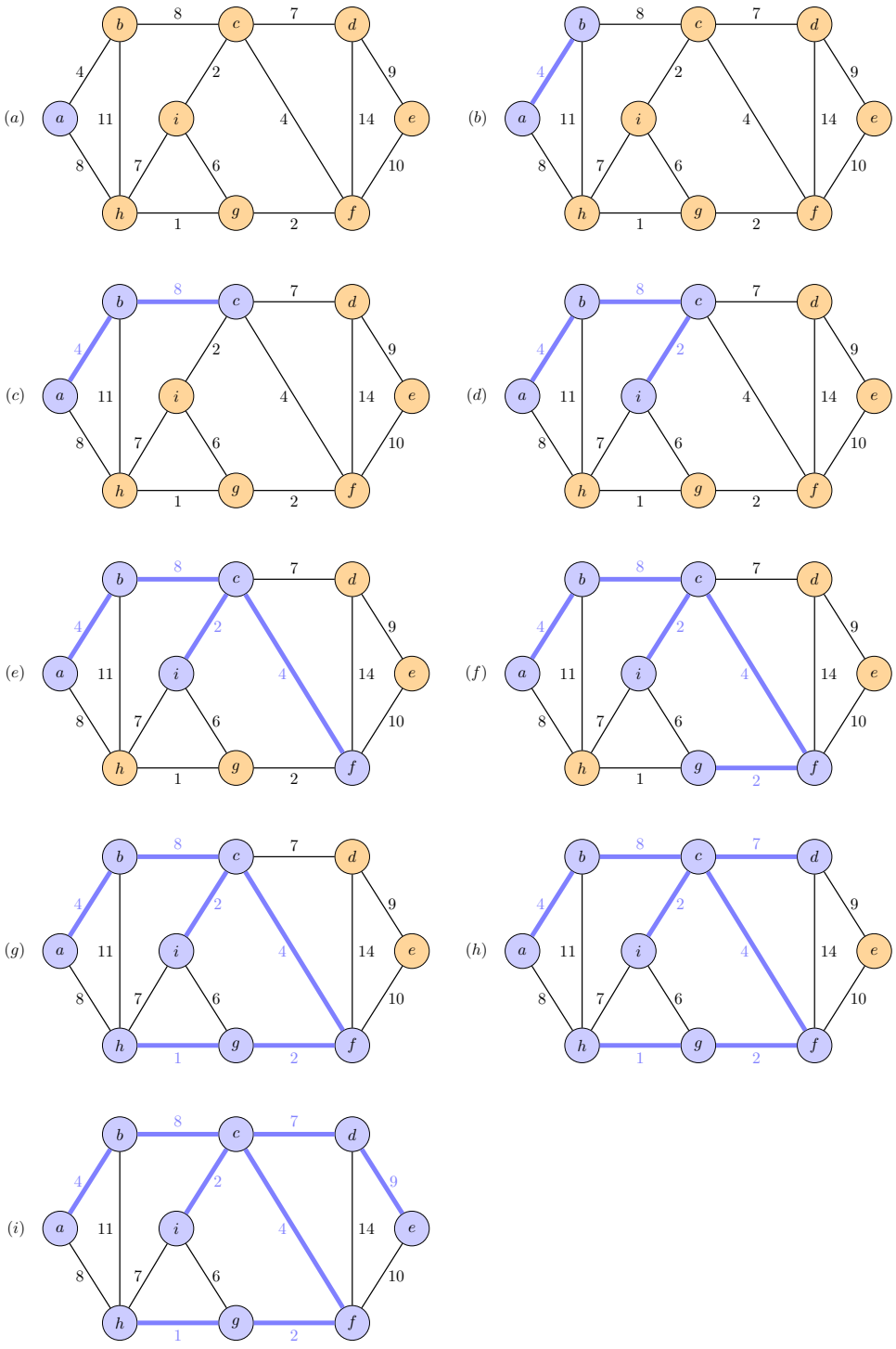
3. Δείξτε ότι αν η ακμή (u, v) εμπεριέχεται σε κάποιο ελαφρύτατο συνδετικό δένδρο τότε είναι ελαφρά ως προς τη διάσχιση κάποιας τομής του γραφήματος.

4. Έστω e μεγιστοβαρής ακμή σε κάποιο κύκλο του συνδεδεμένου γραφήματος G. Δείξτε ότι υπάρχει ελαφρύτατο συνδετικό δένδρο του G το οποίο δεν περιλαμβάνει την ακμή e.

5. Δείξτε ότι ένα γράφημα έχει μοναδικό ελαφρύτατο συνδετικό δένδρο εάω για κάθε τομή του γραφήματος υπάρχει μοναδική ελαφρά ακμή ως προς τη διάσχιση της τομής αυτής. δείξτε ότι το αντίστροφο δεν ισχύει παραθέτοντας ένα αντιπαράδειγμα.

6. Δείξτε ότι κατά την εκτέλεση της ConnectedComponents δύο κόμβοι ανήκουν στην ίδια συνδεδεμένη συνιστώσα αν και μόνο αν ανήκουν στο ίδιο σύνολο.

   Απόδειξη. Αν οι κόμβοι u και v είναι στην ίδια συνδεδεμένη συνιστώσα τότε υπάρχει μια διαδρομή από τον έναν στον άλλο. Αν συνδέονται μέσω μιας μόνο ακμής τότε τοποθετούνται στον ίδιο σύνολο όταν εξετάζεται η συγκεκριμένη ακμή. Αφού εξετάζονται όλες οι ακμές του μονοπατιού κάθε κόμβος του θα τοποθετηθεί στο ίδιο σύνολο. Αντίστροφα, έστω u και v δύο κόμβοι του ίδιου συνόλου. Αφού κάθε κόμβος αρχικά τοποθετείται στο δικό του σύνολο, κάποια ακολουθία ακμών οδήγησε στη συνένωση των συνόλων που περιείχαν τους u και v. Μεταξύ αυτών των ακμών πρέπει να υπάρχει μια διαδρομή από το u στο v και επομένως οι u και v ανήκουν στην ίδια συνδεδεμένη συνιστώσα. □



Σχήμα 6.18: Η λειτουργία του αλγόριθμου του Prim για το γράφημα του Σχήματος 6.14. Ο αφετηριακός κόμβος είναι ο a. Οι σκιασμένες ακμές είναι αυτές που ανήκουν στο δάσος A. Σε κάθε βήμα οι κόμβοι στο δένδρο καθορίζουν μια τομή του γραφήματος και μια ελαφρά ως προς την τομή ακμή προστίθεται στο δένδρο.

7. Έστω $G = (V, E)$ ένα μη κατευθυνόμενο γράφημα με k συνδεδεμένες συνιστώσες. Πόσες φορές καλείται η διαδικασία FindSet κατά την εκτέλεση της διαδικασίας ConnectedComponents; Η διαδικασία Union;

   Απόδειξη. Η διαδικασία FindSet καλείται δύο φορές για κάθε ακμή του γραφήματος επομένως συνολικά καλείται $2|E|$ φορές. Αφού κάθε κόμβος αρχικά τοποθετείται στο δικό του σύνολο, αφού στο τέλος έχουμε μόνο k από αυτά, και αφού κάθε κλήση της διαδικασίας Union μειώνει τον αριθμό τους κατά ένα, η διαδικασία Union καλείται ακριβώς $|V| - k$ φορές. □

8. Υποθέτουμε ότι εκτελούμε τη διαδικασία ConnectedComponents για το ακατεύθυντο γράφημα $G = (V, E)$, όπου το σύνολο κόμβων είναι $V = \{a, b, c, d, e, f, g, h, i, j, k\}$, και ότι οι ακολουθίες του συνόλου E εξετάζονται με τη σειρά (d, i), (f, k), (g, i), (b, g), (a, h), (i, j), (d, k), (b, j), (d, f), (g, j), (a, e), (i, d). Βρείτε τους κόμβους κάθε συνδεδεμένης συνιστώσας μετά από κάθε επανάληψη.

6.7 Ελαφρύτατες διαδρομές

Έστω $G = (V, E)$ ένα κατευθυνόμενο γράφημα όπου κάθε ακμή $(u, v) \in E$ χαρακτηρίζεται από ένα βάρος $w(u, v)$, για κάποια συνάρτηση βάρους $w : E \to \mathbb{R}$. Ορίζουμε το *βάρος* $w(p)$ μιας διαδρομής $p = \langle v_0, v_1, \ldots, v_k \rangle$ ως το άθροισμα των βαρών των ακμών της, δηλαδή

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

Ορίζουμε ως *βάρος ελαφρύτατης διαδρομής* από τον κόμβο u στον κόμβο v την ποσότητα

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{αν υπάρει διαδρομή από το } u \text{ στο } v, \\ \infty & \text{διαφορετικά.} \end{cases}$$

Μια *ελαφρύτατη διαδρομή* από τον κόμβο u στον κόμβο v είναι μια διαδρομή με βάρος $w(p) = \delta(u, v)$.

Θα επιλύσουμε αρχικό το πρόβλημα των *ελαφρύτατων ομοαφετηριακών διαδρομών* (*single–source shortest path problem*) στο οποίο θέλουμε να βρούμε τις ελαφρύτατες διαδρομές από κάποιον αφετηριακό κόμβο $s \in V$ προς κάθε άλλο κόμβο $v \in V$. Ξεκινάμε δείχνοντας τη βέλτιστη υποδομή του προβλήματος με την έννοια ότι μια ελαφρύτατη διαδρομή μεταξύ δύο κόμβων εμπεριέχει άλλες ελαφρύτατες διαδρομές.

Λήμμα 1. Έστω $G = (V, E)$ κατευθυνόμενο γράφημα με συνάρτηση βάρους $w : E \to \mathbb{R}$ και $p = \langle v_0, v_1, \ldots, v_k \rangle$ ελαφρύτατη διαδρομή από τον κόμβο v_0 στον κόμβο v_k. Για κάθε i και j τέτοια ώστε $0 \le i \le j \le k$, έστω $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ η υπο-διαδρομή της p από τον κόμβο v_i στον κόμβο v_j. Τότε η p_{ij} αποτελεί ελαφρύτατη διαδρομή από τον κόμβο v_i στον κόμβο v_j.

Απόδειξη. Αναλύουμε τη διαδρομή p στη μορφή $v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$, έτσι ώστε $w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$. Αν υπάρχει διαδρομή p'_{ij} από τον κόμβο v_i στον κόμβο v_j τέτοια ώστε $w(p'_{ij}) < w(p_{ij})$ τότε η διαδρομή

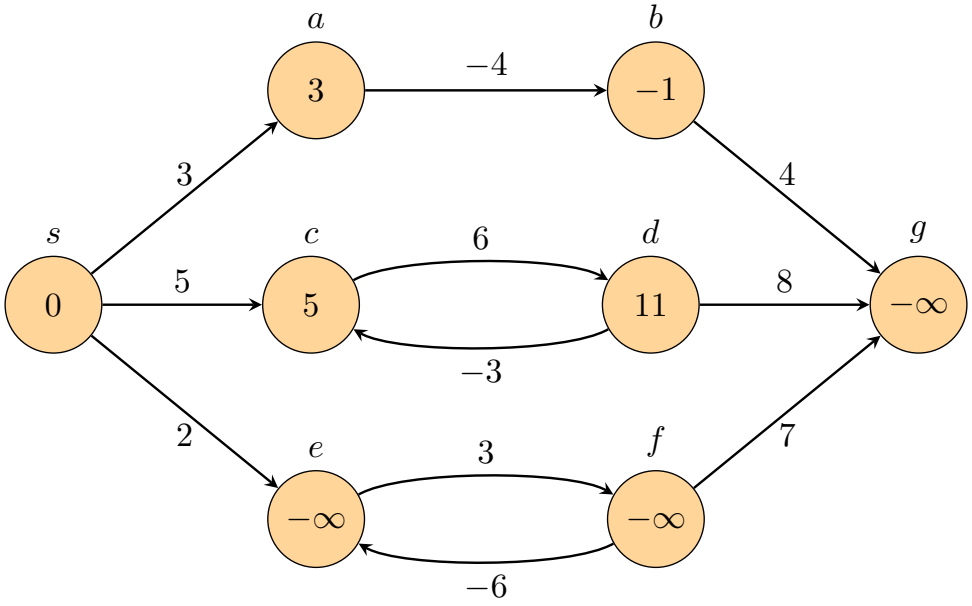
$$v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p'_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$$

έχει βάρος $w(p_{0i}) + w(p'_{ij}) + w(p_{jk})$, μικρότερο δηλαδή του βάρους της ελαφρύτατης διαδρομής p από τον κόμβο v_0 στον κόμβο v_k, το οποίο δεν μπορεί να συμβαίνει. Άρα η p_{ij} αποτελεί ελαφρύτατη διαδρομή από τον κόμβο v_i στον κόμβο v_j. □

Παρατήρηση. Αν το γράφημα $G = (V, E)$ δεν περιέχει κύκλους με αρνητικό βάρος, προσπελάσιμους από κάποιον αφετηριακό κόμβο s, τότε η ποσότητα $\delta(s, v)$ είναι καλά ορισμένη για οποιοδήποτε κόμβο $v \in V$. Σε διαφορετική περίπτωση οι ελαφρύτατες διαδρομές δεν είναι καλά ορισμένες: καμιά διαδρομή από τον αφετηριακό κόμβο s προς οποιοδήποτε κόμβο ενός κύκλου με αρνητικό βάρος δεν

μπορεί να είναι ελαφρύτατη. Οποιοδήποτε μονοπάτι μπορεί να γίνει ελαφρύτερο διανύοντας τον κύκλο με αρνητικό βάρος. Επομένως, αν σε κάποια διαδρομή από τον s στον v υπάρχει κύκλος με αρνητικό βάρος θέτουμε $\delta(s, v) = -\infty$.

Στο Σχήμα 6.19 αναπαρίσταται ένα κατευθυνόμενο γράφημα με αφετηριακό κόμβο s και αρνητικά βάρη ακμών. Εντός κάθε κόμβου αναγράφεται το βάρος ελαφρύτατης διαδρομής από τον κόμβο s. Επειδή οι κόμβοι e και f σχηματίζουν έναν κύκλο αρνητικού βάρους προσπελάσιμο από τον s, τα βάρη ελαφρύτατης διαδρομής για αυτούς είναι $-\infty$. Επειδή ο κόμβος g είναι προσπελάσιμος από έναν κόμβο με βάρος ελαφρύτατης διαδρομής $-\infty$, το βάρος ελαφρύτατης διαδρομής του είναι επίσης $-\infty$.



Σχήμα 6.19: Κατευθυνόμενο γράφημα με αφετηριακό κόμβο s και αρνητικά βάρη ακμών. Εντός κάθε κόμβου αναγράφεται το βάρος ελαφρύτατης διαδρομής από τον κόμβο s. Επειδή οι κόμβοι e και f σχηματίζουν έναν κύκλο αρνητικού βάρους προσπελάσιμο από τον s, τα βάρη ελαφρύτατης διαδρομής για αυτούς είναι $-\infty$. Ο κόμβος g είναι προσπελάσιμος από έναν κόμβο με βάρος ελαφρύτατης διαδρομής $-\infty$ οπότε το βάρος ελαφρύτατης διαδρομής του είναι $-\infty$.

Αντίστοιχα επιχειρήματα δείχνουν ότι μια ελαφρύτατη διαδρομή δεν μπορεί να περιέχει κύκλο θετικού βάρους αφού αν αφαιρέσουμε τον κύκλο παίρνουμε μια διαδρομή με την ίδια αφετηρία και προορισμό χαμηλότερου βάρους. Αν τώρα αφαιρέσουμε από κάποια διαδρομή έναν κύκλο μηδενικού βάρους παίρνουμε μια άλλη διαδρομή με το ίδιο βάρος. Συνεπώς, αν υπάρχει ελαφρύτατη διαδρομή από κάποιον αφετηριακό κόμβο s προς κάποιον κόμβο v η οποία περιλαμβάνει κύκλο μηδενικού βάρους, τότε υπάρχει επίσης μια άλλη ελαφρύτατη διαδρομή η οποία δεν περιλαμβάνει αυτόν τον κύκλο. Επομένως μπορούμε να υποθέσουμε χωρίς περιορισμό της γενικότητας ότι οι ελαφρύτατες διαδρομές που βρίσκουμε δεν περιέχουν κύκλους. Οποιαδήποτε άκυκλη διαδρομή σε ένα γράφημα $G = (V, E)$ περιλαμβάνει το πολύ $|V|$ διαφορετικούς κόμβους και επομένως περιλαμβάνει το πολύ $|V| - 1$ ακμές. Συνεπώς μπορούμε να περιοριστούμε σε ελαφρύτατες διαδρομές οι οποίες περιέχουν το πολύ $|V| - 1$ ακμές.

Για την αναπαράσταση των ελαφρύτατων διαδρομών σε ένα γράφημα $G = (V, E)$ τηρούμε για κάθε κόμβο $v \in V$ το χαρακτηριστικό του προκατόχου $v.\pi$, το οποίο είτε παραπέμει σε άλλο κόμβο είτε είναι κενό. Για οποιονδήποτε κόμβο v με μη κενό χαρακτηριστικό προκατόχου μπορούμε να χρησιμοποιήσουμε τη διαδικασία PrintPath(G, s, v) που χρησιμοποιήσαμε στην οριζόντια διερεύνηση για να εκτυπώσουμε μια ελαφρύτατη διαδρομή από τον κόμβο s στον κόμβο v.

Μπορεί να αποδειχθεί ότι οι τιμές των χαρακτηριστικών π που παράγονται από τους αλγόριθμους εύρεσης ελαφρύτατων διαδρομών που θα δούμε παρακάτω έχουν την ιδιότητα ότι κατά τον τερματισμό τους το υπογράφημα προκατόχων $G_\pi = (V_\pi, E_\pi)$ με

$$V_\pi = \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\},$$
$$E_\pi = \{(v.\pi, v) \in E : v \in V_\pi \setminus \{s\}\}$$

συνιστά ένα δένδρο ελαφρύτατων διαδρομών, δηλαδή ένα δένδρο με ρίζα τον αφετηριακό κόμβο s και το οποίο περιλαμβάνει μια ελαφρύτατη διαδρομή από τον s προς κάθε άλλο κόμβο προσπελάσιμο από τον s. Αν υποθέσουμε ότι το G δεν περιέχει κανέναν κύκλο αρνητικού βάρους τότε ένα *δένδρο ελαφρύτατων διαδρομών* με ρίζα τον s είναι ένα κατευθυνόμενο υπογράφημα $G' = (V', E')$ όπου $V' \subseteq V$ και $E' \subseteq E$, τέτοιο ώστε

1. το V' είναι το σύνολο των κόμβων του G που είναι προσπελάσιμοι από τον s,
2. το G' συνιστά δένδρο με ρίζα τον s, και
3. για κάθε κόμβο $v \in V'$ η μοναδική απλή διαδρομή από τον s μέχρι τον v στο G' αποτελεί ελαφρύτατη διαδρομή από τον s στον v στο γράφημα G.

Μια ακόμα τεχνική η οποία χρησιμοποιείται στους αλγόριθμους εύρεσης ελαφρύτατων διαδρομών είναι αυτή της *χαλάρωσης* (*relaxation*). Για κάθε κόμβο v τηρούμε το χαρακτηριστικό $v.d$ το οποίο αντιπροσωπεύει ένα άνω φράγμα του βάρους μιας ελαφρύτατης διαδρομής από τον αφετηριακό κόμβο s μέχρι τον v. Η ποσότητα αυτή ονομάζεται *προεκτίμηση ελαφρύτατης διαδρομής* (*shortest–path estimate*). Η απόδοση αρχικών τιμών σε αυτό το χαρακτηριστικό γίνεται με την παρακάτω διαδικασία:

INITIALIZESINGLESOURCE(G, s)
1 **for** every $v \in V$
2 $\quad v.d = \infty$
3 $\quad v.\pi = \text{NIL}$
4 $s.d = 0$

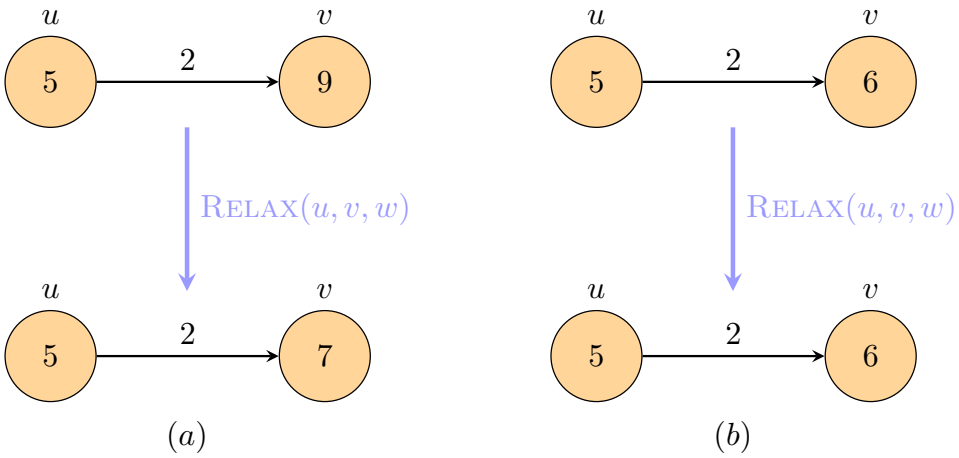
Η διαδικασία της χαλάρωσης της ακμής (u, v) έχει ως εξής: ελέγχουμε αν μπορούμε να βελτιώσουμε την τρέχουσα ελαφρύτατη διαδρομή για τον κόμβο v διερχόμενοι μέσω του κόμβου u. Αν πράγματι μπορούμε τότε ενημερώνουμε τα χαρακτηριστικά $v.\pi$ και $v.d$:

RELAX(u, v, w)
1 **if** $v.d < u.d + w(u, v)$
2 $\quad v.d = u.d + w(u, v)$
3 $\quad v.\pi = u$

Στο Σχήμα 6.20 παρουσιάζονται δύο παραδείγματα χαλάρωσης μιας ακμής. Στο ένα από αυτά μειώθηκε η προεκτίμηση ελαφρύτατης διαδρομής, στο άλλο όχι. Η διαδικασία της χαλάρωσης αποτελεί τον μοναδικό τρόπο μείωσης της προεκτίμησης ελαφρύτατης διαδρομής στους αλγόριθμους εύρεσης ελαφρύτατων διαδρομών που θα δούμε παρακάτω. Οι αλγόριθμοι διαφέρουν μεταξύ τους ως προς το πόσες φορές εκτελούν την πράξη της χαλάρωσης σε κάθε ακμή και ως προς τη σειρά με την οποία διατρέχουν τις ακμές. Για να αποδείξουμε την ορθότητα των αλγορίθμων αυτών θα χρειαστούμε τα παρακάτω αποτελέσματα:

Τριγωνική ανισότητα.
Για κάθε ακμή $(u, v) \in E$ έχουμε $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Άνω φράγμα του βάρους ελαφρύτατης διαδρομής.
Για κάθε $v \in V$ έχουμε $v.d \geq \delta(s, v)$ και μόλις το χαρακτηριστικό $v.d$ εξισωθεί με αυτή την τιμή δεν μεταβάλλεται ποτέ.



Σχήμα 6.20: Η χαλάρωση της ακμής (u, v) με βάρος 2. Η προεκτίμηση της ελαφρύτατης διαδρομής εμφανίζεται εντός των κόμβων.

Ανυπαρξία διαδρομής.
Αν δεν υπάρχει διαδρομή από τον κόμβο s μέχρι τον κόμβο v τότε ισχύει $v.d = \delta(s, v) = \infty$.

Ιδιότητα σύγκλισης.
Αν $s \rightsquigarrow u \rightarrow v$ είναι κάποια ελαφρύτατη διαδρομή στο G για κάποιους κόμβους $u, v \in V$ και αν $u.d = \delta(s, u)$ σε οποιαδήποτε χρονική στιγμή πριν από τη χαλάρωση της ακμής (u, v), τότε $u.d = \delta(s, u)$ για όλες τις χρονικές στιγμές μετά τη χαλάρωση.

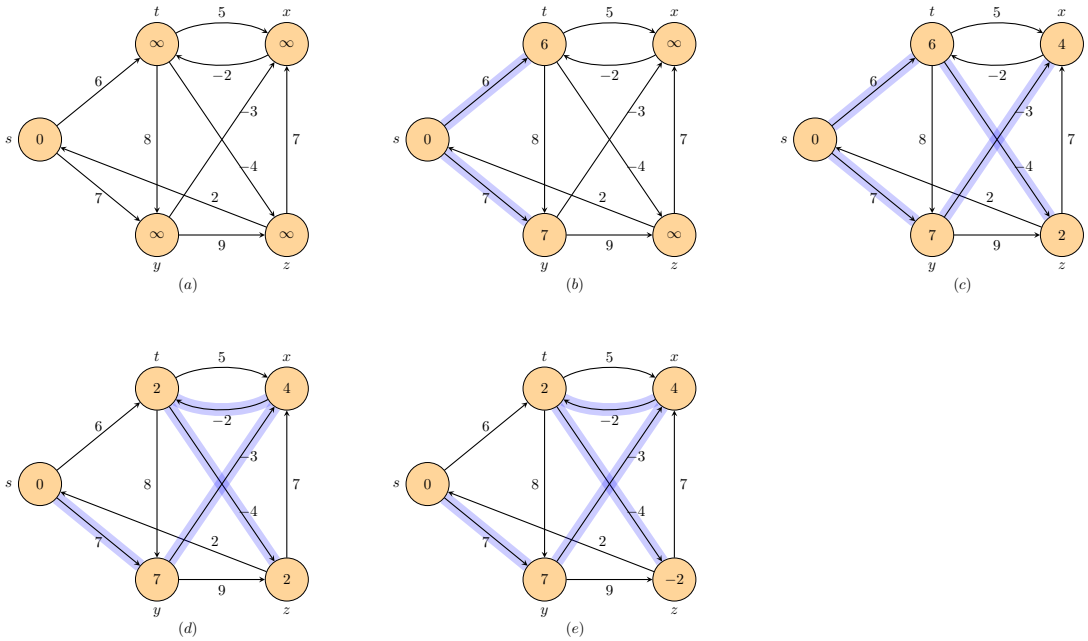
Χαλάρωση διαδρομής.
Αν η $p = \langle v_0, v_1, \ldots, v_k \rangle$ είναι κάποια ελαφρύτατη διαδρομή από τον κόμβο $s = v_0$ μέχρι τον κόμβο v_k, και οι ακμές της p υποστούν χαλάρωση με τη σειρά $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, τότε $v_k.d = \delta(s, v_k)$. Η ιδιότητα αυτή ισχύει ανεξάρτητα από όποιες άλλες πράξεις χαλάρωσης λαμβάνουν χώρα.

Ιδιότητα υπογραφήματος .
Αν $v.d = \delta(s, v)$ για όλους τους κόμβους $v \in V$, τότε το υπογράφημα προκατόχων αποτελεί δένδρο ελαφρύτατων διαδρομών με ρίζα τον κόμβο s.

6.8 Ο αλγόριθμος των Bellman–Ford

Ο αλγόριθμος των Bellman–Ford δέχεται ως είσοδο ένα κατευθυνόμενο γράφημα $G = (V, E)$ με κάποιον αφετηριακό κόμβο s και κάποια συνάρτηση βάρους $w : E \rightarrow \mathbb{R}$. Επιστρέφει μια τιμή λογικού τύπου η οποία υποδεικνύει αν υπάρχει ή όχι κύκλος αρνητικού βάρους προσπελάσιμος από τον s. Αν δεν υπάρχει, τότε υπολογίζει τις ελαφρύτατες διαδρομές και τα βάρη τους. Η διαδικασία BELLMANFORD χαλαρώνει τις ακμές σταδιακά, μειώνοντας την προεκτίμηση $v.d$ μέχρι να αποκτήσει την τιμή $\delta(s, v)$.

BELLMANFORD(G, w, s)
1 INITIALIZESINGLESOURCE(G, s)
2 **for** $i = 1$ **to** $|V| - 1$
3 **for** every edge $(u, v) \in E$
4 RELAX(u, v, w)
5 **for** every edge $(u, v) \in E$
6 **if** $v.d > u.d + w(u, v)$
7 **return** FALSE
8 **return** TRUE



Σχήμα 6.21: Η λειτουργία του αλγόριθμου των Bellman–Ford. Ο αφετηριακός κόμβος είναι ο s. Εντός των κόμβων αναγράφονται οι τιμές των αντίστοιχων χαρακτηριστικών d. Οι σκιασμένες ακμές υποδεικνύουν τις τιμές του χαρακτηριστικού προκατόχου: αν η ακμή (u, v) είναι σκιασμένη, τότε $v.\pi = u$.

Μετά την απόδοση αρχικών τιμών στα χαρακτηριστικά d και π όλων των κόμβων, ο αλγόριθμος διατρέχει όλες τις ακμές $|V| - 1$ φορές. Κάθε διέλευση αντιπροσωπεύει μια επανάληψη του βρόχου **for** στις γραμμές 2–4 και συνίσταται σε μια πράξης χαλάρωσης σε κάθε ακμή. Στη συνέχεια ο αλγόριθμος ελέγχει αν υπάρχει κάποιος κύκλος αρνητικού βάρους στις γραμμές 5–8 και επιστρέφει την ενδεδειγμένη τιμή λογικού τύπου. Είναι φανερό ότι ο αλγόριθμος των Bellman–Ford έχει χρόνο εκτέλεσης $O(|V||E|)$ αφού η απόδοση αρχικών τιμών απαιτεί χρόνο $\Theta(|V|)$ και κάθε μια από τις $|V| - 1$ διελεύσεις των ακμών στις γραμμές 2–4 απαιτεί χρόνο $\Theta(|E|)$. Στο Σχήμα 6.21 βλέπουμε τη λειτουργία του αλγόριθμου σε ένα γράφημα με πέντε κόμβους. Ο αφετηριακός κόμβος είναι ο s.
Θα δείξουμε καταρχάς ότι στην περίπτωση που δεν υπάρχουν κύκλοι αρνητικού βάρους, ο αλγόριθμος υπολογίζει σωστά τα βάρη ελαφρύτατων διαδρομών για όλους τους κόμβους που είναι προσβάσιμοι από την αφετηρία.

Λήμμα 2. Έστω $G = (V, E)$ ένα κατευθυνόμενο γράφημα με κάποιον αφετηριακό κόμβο s και κάποια συνάρτηση βάρους $w : E \to \mathbb{R}$. Υποθέτουμε ότι το G δεν περιέχει κύκλους αρνητικού βάρους. Τότε, μετά από $|V| - 1$ επαναλήψεις του βρόχου στις γραμμές 2–4 της διαδικασίας BellmanFord έχουμε $v.d = \delta(s, v)$, για όλους τους κόμβους $v \in V$ που είναι προσπελάσιμοι από τον s.

Απόδειξη. Έστω $v \in V$ ένας κόμβος προσπελάσιμος από τον s και $p = \langle v_0, v_1, \ldots, v_k \rangle$, όπου $v_0 = s$ και $v_k = v$, ελαφρύτατη διαδρομή από τον s στον v. Οι ελαφρύτατες διαδρομές είναι απλές, επομένως η διαδρομή p έχει το πολύ $|V| - 1$ ακμές. Συνεπώς, $k \leq |V| - 1$. Κάθε μία από τις $|V| - 1$ επαναλήψεις του βρόχου **for** στις γραμμές 2–4 της διαδικασίας BellmanFord χαλαρώνει $|E|$ ακμές. Μεταξύ των ακμών που χαλαρώνουν στην i-στή επανάληψη, για $i = 1, 2, \ldots, k$, είναι η (v_{i-1}, v_i). Από την ιδιότητα χαλάρωσης διαδρομής έχουμε ότι $v.d = v_k.d = \delta(s, v_k) = \delta(s, v)$. □

Πόρισμα 1. Έστω $G = (V, E)$ ένα κατευθυνόμενο γράφημα με κάποιον αφετηριακό κόμβο s και κάποια συνάρτηση βάρους $w : E \to \mathbb{R}$. Τότε, για κάθε κόμβο $v \in V$ υπάρχει διαδρομή από τον s στον v αν και μόνο αν κατά την περάτωση της διαδικασίας BellmanFord έχουμε $v.d < \infty$.

Απόδειξη. Ας υποθέσουμε πρώτα ότι ο κόμβος v είναι προσπελάσιμος από τον s. Τότε υπάρχει ελαφρύτατη διαδρομή από τον s στον v μήκους $\delta(s, v)$. Το μήκος της διαδρομής είναι πεπερασμένο

αφού περιέχει το πολύ $|V| - 1$ ακμές. Τότε από το Λήμμα 2 έχουμε ότι $v.d = \delta(s, v) < \infty$ κατά την περάτωση της διαδικασίας BellmanFord. Ας υποθέσουμε τώρα ότι $v.d < \infty$ κατά την περάτωση της διαδικασίας BellmanFord. Το χαρακτηριστικό $v.d$ μειώνεται μονότονα κατά τη διαδικασία BellmanFord και η τιμή του αλλάζει μόνο όταν $u.d + w(u, v) < v.d$, για κάποιον γειτονικό κόμβο u του v. Την ίδια στιγμή, $v.\pi = u$, δηλαδή ο κόμβος u γίνεται προκάτοχος του v στο υπογράφημα προκατόχων. Το υπογράφημα προκατόχων είναι ένα δένδρο με ρίζα τον κόμβο s και περιέχει μια διαδρομή προς τον κόμβο v. Οι ακμές της διαδρομής είναι ακμές του γραφήματος G επομένως υπάρχει μια διαδρομή από τον s στον v στο γράφημα G. □

Θεώρημα 1 (Ορθότητα του αλγόριθμου των Bellman–Ford). Έστω $G = (V, E)$ ένα κατευθυνόμενο γράφημα με κάποιον αφετηριακό κόμβο s και κάποια συνάρτηση βάρους $w : E \to \mathbb{R}$. Αν το G δεν περιέχει κύκλους αρνητικού βάρους προσπελάσιμους από τον s τότε η διαδικασία BellmanFord επιστρέφει την τιμή True, θέτει $v.d = \delta(s, v)$ για κάθε κόμβο $v \in V$ και το υπογράφημα προκατόχων G_π καθίσταται δένδρο ελαφρύτατων διαδρομών με ρίζα τον s. Αν το γράφημα G περιέχει κύκλους αρνητικού βάρους προσπελάσιμους από τον s τότε η διαδικασία BellmanFord επιστρέφει την τιμή False.

Απόδειξη. Υποθέτουμε πρώτα ότι το γράφημα G δεν περιέχει κύκλους με αρνητικά βάρη προσπελάσιμους από τον s. Δείχνουμε ότι κατά την περάτωση της διαδικασίας BellmanFord ισχύει $v.d = \delta(s, v)$ για κάθε κόμβο $v \in V$. Αν ο κόμβος v είναι προσπελάσιμος από τον s, τότε το Λήμμα 2 αποδεικνύει αυτόν τον ισχυρισμό. Αν ο v δεν είναι προσπελάσιμος από τον s ο ισχυρισμός αποδεικνύεται από την ιδιότητα της ανυπαρξίας διαδρομής.

Οι ιδιότητες του υπογραφήματος προκατόχων και το γεγονός ότι κατά την περάτωση της διαδικασίας BellmanFord ισχύει $v.d = \delta(s, v)$ για κάθε κόμβο $v \in V$ συνεπάγονται ότι το G_π είναι δένδρο ελαφρύτατων διαδρομών. Κατά την περάτωση της διαδικασίας έχουμε για κάθε ακμή $(u, v) \in E$ ότι

$$v.d = \delta(s, v) \leq \delta(s, u) + w(u, v) = u.d + w(u, v),$$

και επομένως η συνθήκη στη γραμμή 6 της διαδικασίας BellmanFord δεν ισχύει ποτέ. Συνεπώς ο αλγόριθμος των Bellman–Ford επιστρέφει την τιμή True. Έστω τώρα ότι το γράφημα G περιέχει κάποιον κύκλο με αρνητικό βάρος, προσπελάσιμο από τον s. Αν ο κύκλος είναι ο $c = \langle v_0, v_1, \ldots, v_k \rangle$ με $v_0 = v_k$, έχουμε

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0. \tag{6.5}$$

Υποθέτουμε ότι ο ο αλγόριθμος των Bellman–Ford επιστρέφει την τιμή False. Τότε $v_i d \leq v_{i-1}.d + w(v_{i-1}, v_i)$, για $i = 1, 2, \ldots, k$. Αθροίζοντας αυτές τις ανισότητες πάνω σε όλες τις ακμές του κύκλου έχουμε

$$\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

Όμως $v_0 = v_k$ το οποίο συνεπάγεται ότι

$$\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d.$$

Οι σχέσεις αυτές μαζί με το γεγονός ότι από το Πόρισμα 1 έχουμε ότι $v.d < \infty$, μας δίνουν

$$0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i),$$

σχέση η οποία αντιβαίνει την ανισότητα (6.5). Επομένως, στην περίπτωση που το γράφημα G δεν περιέχει κύκλους με αρνητικό βάρος προσπελάσιμους από τον s, η διαδικασία BellmanFord επιστρέφει την τιμή True, διαφορετικά την τιμή False. □

6.9 Ο αλγόριθμος του Dijsktra

Ο αλγόριθμος του Dijkstra[2] επιλύει το πρόβλημα των ομοαφετηριακών ελαφρύτατων διαδρομών για το κατευθυνόμενο γράφημα $G = (V, E)$ υπό την πρϋπόθεση ότι η συνάρτηση βάρους $w : E \to \mathbb{R}$ παίρνει μη αρνητικές τιμές, δηλαδή $w(u, v) \geq 0$ για κάθε ακμή $(u, v) \in E$.

Ο αλγόριθμος του Dijkstra τηρεί ένα σύνολο κόμβων S για τους οποίους τα βάρη των ελαφρύτατων διαδρομών από τον αφετηριακό κόμβο s έχουν ήδη προσδιοριστεί. Σε κάθε βήμα επιλέγει έναν κόμβο $u \in V \setminus S$ με την ελάχιστη προεκτίμηση ελαφρύτατης διαδρομής, τον προσθέτει στο σύνολο S, και χαλαρώνει τις ακμές που ξεκινούν από τον u. Για την τήρηση του συνόλου S ο αλγόριθμος του Dijkstra χρησιμοποιεί μια ουρά προτεραιότητας ελαχίστου με κλειδιά τις προεκτιμήσεις ελαφρύτατων διαδρομών. Αποτελεί μια γενίκευση του αλγόριθμου της οριζόντιας διερεύνησης σε γράφους με βάρη. Η διαδικασία DIJKSTRA φαίνεται παρακάτω:

DIJKSTRA(G, w, s)
1 INITIALIZESINGLESOURCE(G, s)
2 $S = \emptyset$
3 $Q = \emptyset$
4 **for** each vertex $u \in V$
5 INSERT(Q, u)
6 **while** $Q \neq \emptyset$
7 $u =$ EXTRACTMIN(Q)
8 $S = S \cup \{u\}$
9 **for** each vertex $v \in$ Adj$[u]$
10 RELAX(u, v, w)
11 **if** RELAX decrased $v.d$
12 DECREASEKEY$(Q, v, v.d)$

Στην γραμμή 1 αρχικοποιούνται τα χαρακτηριστικά d και π και το σύνολο S είναι αρχικά κενό. Ο αλγόριθμος τηρεί την αναλλοίωτη συνθήκη $Q = V \setminus S$ πριν κάθε επανάληψη του βρόχου **while** στις γραμμές 6–12. Οι γραμμές 3–5 αρχικοποιούν την ουρά προτεραιότητας ελαχίστου με όλους τους κόμβους του V. Σε κάθε επανάληψη του βρόχου **while** αφαιρείται κάποιος κόμβος u από την ουρά Q και προστίθεται στο σύνολο S. Ο κόμβος u έχει την μικρότερη προεκτίμηση ελαφρύτατης διαδρομής όλων των κόμβων του συνόλου $V \setminus S$. Οι γραμμές 9–12 χαλαρώνουn κάθε ακμή (u, v) που εξέρχεται από τον u, ενημερώνοντας το χαρακτηριστικό $v.d$ και τον προκάτοχο $v.\pi$, στην περίπτωση που το βάρος της τρέχουσας διαδρομής προς τον v ελαττώνεται, αν η διαδρομή διανύσει την ακμή (u, v). Όποτε η διαδικασία της χαλάρωσης αλλάξει τα χαρακτηριστικά d και π η κλήση της διαδικασίας DECREASEKEY στη γραμμή 12 ενημερώνει την ουρά προτεραιότητας ελαχίστου. Ο αλγόριθμος δεν εισάγει κόμβους στην ουρά Q μετά τις γραμμές 4–5, και κάθε κόμβος αφαιρείται από την ουρά Q και προστίθεται στο σύνολο S ακριβώς μια φορά, συνεπώς οι εντολές του βρόχου **while** εκτελούνται ακριβώς $|V|$ φορές.

Αφού ο αλγόριθμος του Dijkstra επιλέγει για προσθήκη στο σύνολο S εκείνον τον κόμβο του $V \setminus S$ που βρίσκεται πιο κοντά στο S, με την έννοια της μικρότερης προεκτίμησης ελαφρύτατης διαδρομής, μπορεί να θεωρηθεί ως άπληστος αλγόριθμος. Αν και οι άπληστοι αλγόριθμοι δεν δίνουν πάντα βέλτιστα αποτελέσματα, για τον αλγόριθμο του Dijkstra μπορούμε να δείξουμε ότι υπολογίζει ελαφρύτατες διαδρομές. Το Θεώρημα 2 δείχνει ότι $u.d = \delta(s, u)$ κάθε φορά που ένας κόμβος u προστίθεται στο σύνολο S.

Θεώρημα 2 (Ορθότητα του αλγόριθμου του Dijkstra). Έστω $G = (V, E)$ ένα κατευθυνόμενο γράφημα με κάποιον αφετηριακό κόμβο s και κάποια συνάρτηση βάρους $w : E \to \mathbb{R}$. Κατά την περάτωση του αλγόριθμου του Dijkstra για το γράφημα G έχουμε $u.d = \delta(s, u)$ για κάθε κόμβο $u \in V$.

[2] Edsger Wybe Dijkstra (1930-2002). Ολλανδός επιστήμονας της πληροφορικής με σημαντική προσφορά στη θεωρία γραφημάτων, στην ανάλυση αλγορίθμων και στις γλώσσες προγραμματισμού. Τιμήθηκε με το βραβείο Turing το 1972.

Απόδειξη. Θα δείξουμε ότι στην αρχή κάθε επανάληψης του βρόχου **while** στις γραμμές 6–12 έχουμε $v.d = \delta(s, v)$ για κάθε $v \in S$. Ο αλγόριθμος τερματίζεται όταν $S = V$ οπότε $v.d = \delta(s, v)$ για κάθε $v \in V$. Η απόδειξη γίνεται με επαγωγή ως προς τον αριθμό των επαναλήψεων του βρόχου **while** , ο οποίος ισούται με $|S|$ στην αρχή κάθε επανάληψης. Όταν $|S| = 0$ τότε $S = \emptyset$ και ο ισχυρισμός ισχύει τετριμμένα. Το ίδιο ισχύει και όταν $|S| = 1$ οπότε $S = \{s\}$ και $s.d = \delta(s, s) = 0$.

Για το επαγωγικό βήμα, υποθέτουμε ότι $v.d = \delta(s, v)$ για κάθε $v \in S$. Ο αλγόριθμος εξάγει έναν κόμβο u από την ουρά $Q = V \setminus S$. Επειδή ο κόμβος u εισάγεται στο σύνολο S πρέπειο να δείξουμε ότι $u.d = \delta(s, u)$. Αν δεν υπάρχει διαδρομή από τον s στον u τότε η ιδιότητα της ανυπαρξίας διαδρομής παρέχει το ζητούμενο αποτέλεσμα. Αν υπάρχει διαδρομή από τον s στον u, έστω y ο πρώτος κόμβος στην ελαφρύτατη διαδρομή από τον s στον u ο οποίος δεν ανήκει στο σύνολο S, και έστω $x \in S$ ο προκάτοχός του σε αυτή τη διαδρομή. Επειδή ο y δεν εμφανίζεται νωρίτερα από τον u στην ελαφρύτατη διαδρομή και επειδή όλα τα βάρη των ακμών είναι μη αρνητικά, έχουμε ότι $\delta(s, y) \leq \delta(s, u)$. Εξαιτίας του γεγονότος ότι η κλήση ExtractMin στη γραμμή 7, επέστρεψε τον κόμβο u ως τον κόμβο με το ελάχιστο χαρακτηριστικό d στο σύνολο $V \setminus S$, έχουμε επίσης ότι $u.d \leq y.d$, οπότε η ιδιότητα του άνω φράγματος δίνει ότι $\delta(s, u) \leq u.d$.

Αφού $x \in S$, η επαγωγική υπόθεση συνεπάγεται ότι $x.d = \delta(s, x)$. Κατά τη διάρκεια της επανάληψης **while** η οποία προσέθεσε τον κόμβο x στο σύνολο S, η ακμή (x, y) υπέστη χαλάρωση. Από την ιδιότητα της σύγκλισης το χαρακτηριστικό $y.d$ έλαβε την τιμή $\delta(s, y)$ εκείνη τη στιγμή. Συνπώς,

$$\delta(s, y) \leq \delta(s, u) \leq u.d \leq y.d, \quad \text{και} \quad y.d = \delta(s, y),$$

έτσι ώστε $\delta(s, y) = \delta(s, u) = u.d. = y.d$. Επομένως, $u.d = \delta(s, u)$, και από την ιδιότητα του άνω φράγματος, η τιμή αυτής παραμένει η ίδια. □

Πόρισμα 2. Έστω $G = (V, E)$ ένα κατευθυνόμενο γράφημα με κάποιον αφετηριακό κόμβο s και κάποια συνάρτηση βάρους $w : E \to \mathbb{R}$. Κατά την περάτωση του αλγόριθμου του Dijkstra για το γράφημα G, το υπογράφημα προκατόχων G_π είναι ένα δένδρο ελαφρύτατων διαδρομών με ρίζα τον κόμβο s.

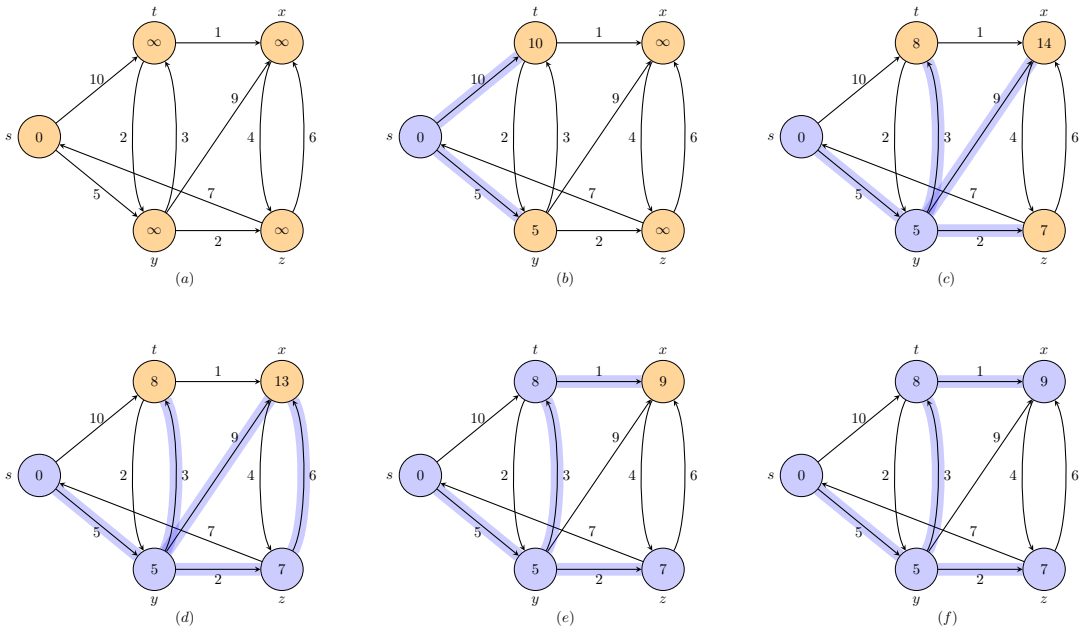
Απόδειξη. Άμεση από το Θεώρημα 2 και τις ιδιότητες του G_π. □

Ένα παράδειγμα της διαδικασίας Dijkstra φαίνεται στο Σχήμα 6.22. Ο χρόνος εκτέλεσης εξαρτάται από την υλοποίηση της ουράς προτεραιότητας ελαχίστου. Οι διαδικασίες Insert και ExtractMin καλούνται ακριβώς μια φορά για κάθε κόμβο, ενώ η διαδικασίας DecreaseKey καλείται το πολύ $|E|$ φορές. Η συνηθισμένη υλοποίηση της ουράς προτεραιότητας με πίνακες απαιτεί χρόνο $O(1)$ για τις διαδικασίες Insert και ExtractMin και χρόνο $O(|V|)$ για τη διαδικασία DecreaseKey. Συνολικά λοιπόν ο χρόνος εκτέλεσης είναι $O(|V|^2 + E)$.

6.10 Ασκήσεις

1. Εκτελέστε τον αλγόριθμο των Bellman–Ford στο γράφημα του σχήματος 6.21. Προσδιορίστε τα χαρακτηριστικά d και π μετά από κάθε διέλευση.

2. Εκτελέστε τον αλγόριθμο του Dijkstra στο γράφημα του σχήματος 6.21. Προσδιορίστε τα χαρακτηριστικά d και π.



Σχήμα 6.22: Η λειτουργία της διαδικασίας Dijkstra.

Βιβλιογραφία

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Reading, Mass.: Addison-Wesley, 1974.

[2] Thomas H. Cormen et al. *Introduction to algorithms*. 3rd ed. MIT Press, 2009.

[3] David Gries. *The Science of Programming*. Monographs in Computer Science. Springer New York, 1989.

[4] John MacCormick. *Nine Algorithms That Changed the Future: The Ingenious Ideas That Drive Today's Computers*. Princeton University Press, 2011.

[5] John C. Mitchell. *Foundations for Programming Languages*. Foundations of computing. MIT Press, 1996.