Αναζήτηση συνέχεια

Αναζήτηση σε πολύ μεγάλες λίστες

- L λίστα με πολλά στοιχεία
- Ποιά μέθοδος από τις 3 που παρουσιάσαμε είναι γρηγορότερη;
- Χρονομέτρηση εκτέλεσης εντολών
- βιβλιοθήκη (module) time

module time

- time.time() - Η τιμή του χρόνου σε δευτερόλεπτα
- Στο κώδικα python

```
t1=time.time()
a,b= find_two_smallest(count)
t2=time.time()
```

- Η διαφορά `t2-t1` δίνει το χρόνο που έχει ¨περάσει”

Αναζήτηση δύο μικρότερων

- **Μέθοδος 1η**
  - Βρές το μικρότερο
  - αφαίρεσε το,
  - βρες το νέο μικρότερο
  - Χρονομέτρηση για 1000000 στοιχεία: ~ 6 εκατοστά sec
- **Μέθοδος 2η**
  - ταξινομώ
  - βρίσκω τα δύο ελάχιστα
  - Χρονομέτρηση για 1000000 στοιχεία: ~ 60 εκατοστά sec
- **Μέθοδος 3η**
  - “Περπατώ” στα δεδομένα
  - έλεγχω κάθε τιμή και κρατώ στη μνήμη κάθε φορά τις δύο μικρότερες
  - Χρονομέτρηση για 1000000 στοιχεία: ~ 20 εκατοστά sec

Αναζήτηση

ενός συγκεκριμένου στοιχείου

Αναζήτηση

- `L` λίστα αριθμών

- **`L.index(`**στοιχείο**`)`** επιστρέφει τη θέση του “στοιχείου” στην `L`

- Εξετάζουμε κάθε στοιχείο σε σειρά μέχρι να συναντήσουμε το ζητούμενο στοιχείο. (γραμμική αναζήτηση)

Γραμμική αναζήτηση

- Αρχίζοντας από την αρχή της `L`, θέτουμε `i` το δείκτη των στοιχείων της `L`. Ο `i` αρχίζοντας από το 0 διατρέχει αριθμούς και αυξάνει βηματικά χρησιμοποιώντας επαναληπτική διαδικασία **`while`**

- Πραγματοποιούμε έλεγχο αν έχουμε βρεί το το στοιχείο που ψάχνουμε να βρούμε

- Επιστρέφουμε τον πρώτο δείκτη `i` όπου βρήκαμε το στοιχείο που ψάχνουμε

- Λογικός έλεγχος στη χρήση της while:
  - Συνεχίζουμε να αυξάνουμε το i αν δεν έχουμε βρει το στοιχείο που ψάχνουμε
  - Δεν πρέπει ο `i` να ξεπεράσει το μήκος της `L`

Γραμμική Αναζήτηση

```
def linear_search(v, L):
    '''Return the index of the first occurrence of v
       in list L, or return len(L) if v is not in L.
    '''
    i = 0
    while i != len(L) and L[i] != v:
        i = i + 1
    return i
```

Γραμμική Αναζήτηση

```
def linear_search(v, L):
    '''Return the index of the first occurrence of v
       in list L, or return len(L) if v is not in L.
    '''
    i = 0
    while i != len(L) and L[i] != v:
        i = i + 1
    return i
```

- Αν δεν βρει το στοιχείο `v` στη `L` επιστρέφει το μήκος της `L`.

- Δεν υπάρχει λογικό σφάλμα σε αυτό διότι κανένα στοιχείο της `L` δεν έχει θέση ίση με το μήκος της `L`

Γραμμική Αναζήτηση

- Όφελος με την υλοποίηση της αναζήτησης while:

- Δεν χρειάζεται να ψάξουμε όλη τη L. Μόλις βρούμε το στοιχείο που αναζητούμε σταματάμε.

- Υλοποίηση με for:
  - Ο δείκτης `i` διατρέχει μια λίστα με αριθμούς όσο το μήκος της `L`
  - Δεν χρειάζεται ο έλεγχος `i != len(L)` της υλοποίησης με `while`
  - Μειονέκτημα: Δημιουργούμε μια νέα λίστα τόσο μεγάλη όσο και το μέγεθος της `L`

Γραμμική Αναζήτηση

- Αντιμετώπιση: Η επανάληψη με **`for`** διατρέχει τα στοιχεία της `L` και όχι τους δείκτες

- for value in L:
    έλεγχος αν βρήκαμε το στοιχείο

- Ζητούμενο είναι η θέση. Θέτουμε i έναν μετρητή ο οποίος αυξάνει σε κάθε βήμα της επανάληψης

Γραμμική Αναζήτηση

- i=0
  for value in L:
  έλεγχος αν βρήκαμε το στοιχείο
  i+=1

- Ο μετρητής i δεν μπορεί να ξεπεράσει το μήκος της λίστας `L`

Γραμμική Αναζήτηση

```
def linear_search(v, L):
    '''Return the index of the first occurrence of v
    in list L, or return len(L) if v is not in L.
    '''

    i = 0
    for value in L:
        if value == v:
            return i
        i += 1
    return len(L)
```

Γραμμική Αναζήτηση

Παράδειγμα

- ```
  L=range(1000000)
  a= linear_search(10,L) # while - loop
  a= linear_search2(10,L) # for - loop
  ```

- Χρονομέτρηση

  ```
  Search (while-loop): 0.00882149 milliseconds
  Search (for-loop): 0.00405312 milliseconds
  ```
  ```

Γραμμική Αναζήτηση

Επιτάχυνση της αναζήτησης με την επανάληψη while

- Η καθυστέρηση οφείλετε στον διπλό έλεγχο
  - `i != len(L)`
  - `L[i] != v`

- Αν το **`v`** υπάρχει σίγουρα μέσα στη **`L`** δεν χρειάζεται να ελέγχουμε αν ο μετρητής **`i`** γίνει μεγαλύτερος του **`len(L)`**

- Νέος αλγόριθμος: Προσθέτουμε πάντα στο τέλος της **`L`** το στοιχείο που ψάχνουμε

Γραμμική Αναζήτηση

```
def linear_search(v, L):
    '''Return the index of the first occurrence of v in
    list L, or return len(L) if v is not in L.
    '''
    L.append(v)          ⟶  Προσθέτω το στοιχείο v στο τέλος της L
    i = 0

    while L[i] != v:     ⟶  Αφαιρώ το έλεγχο i != len(L)
        i = i + 1

    L.pop()              ⟶  Αφαιρώ το τελευταίο στοιχείο

    return i
```

Αναζήτηση

σε string ή λεξικό

- Σε “ακολουθιακές” μορφές όπως η λίστα μπορούμε να εφαρμόσουμε ανάλογα, τη γραμμική αναζήτηση

- Αναζήτηση χρησιμοποιώντας επανάληψη **`while`** ή **`for`**

- Το λεξικό χρειάζετε προσοχή διότι οι δείκτες των στοιχείων (τα κλειδιά) δεν είναι φυσικοί αριθμοί όπως στις λίστες, strings, …

Αναζήτηση σε string

```
def linear_search(v, L):
    '''Return the index of the first occurrence of v
    in string L, or return len(L) if v is not in L.
    '''
    i = 0
    for value in L:
        if value == v:
            return i
        i += 1
    return len(L)
```

- Η συνάρτηση που θεωρήσαμε για την αναζήτηση σε λίστες με επαναλήψη `for` εφαρμόζεται χωρίς μετατροπές για τα string

Αναζήτηση σε λεξικό

```
def linear_search(v, L):
    '''Return the index of the first occurrence of v
       in dict L, or return len(L) if v is not in L.
    '''
    i = 0
    while i != len(L) and L[i] != v:
        i = i + 1
    return i
```

Συντακτικό λάθος αν L λεξικό

- Η συνάρτηση που θεωρήσαμε για την αναζήτηση σε λίστες δεν μπορεί να εφαρμοστεί χωρίς αλλαγές. Οι δείκτες δεν είναι φυσικοί αριθμοί.
- Αν θέλουμε να χρησιμοποιήσουμε την παραπάνω μορφή του αλγορίθμου αναζήτησης πρέπει να χρησιμοποιήσουμε τις μεθόδους **`values()`** και **`keys()`** για λεξικά

Αναζήτηση σε λεξικό

```
def linear_search(v, L):
    '''Return the index of the first occurrence of v
       in list L, or return a message if v is not in L.
    '''
    i = 0
    while i != len(L) and list(L.values())[i] != v:
        i = i + 1
    if list(L.values())[i] == v:
       return list(L.keys())[i]
    else:
       return 'I haven't found it'
```

- Αν θέλουμε να χρησιμοποιήσουμε την παραπάνω μορφή του αλγορίθμου αναζήτησης πρέπει να χρησιμοποιήσουμε τις μεθόδους **`values()`** και **`keys()`** για λεξικά

Αναζήτηση σε string

```
def linear_search(v, L):
    '''Return the index of the first occurrence of v
    in list L, or return len(L) if v is not in L.
    '''
    i = 0
    for value in L:
        if value == v:
            return i
        i += 1
    return len(L)
```

- Η συνάρτηση που θεωρήσαμε για την αναζήτηση σε λίστες με επαναλήψη `for` εφαρμόζεται χωρίς μετατροπές για τα string

Αναζήτηση
με νέα κριτηρια

- L=[(1,0),(0,1),(1,1),(3,2),(1,2),(-2,2)]

- Θέλουμε να βρούμε το μεγαλύτερο σημείο ως προς την ευκλείδια απόσταση.

- Η max δεν δουλεύει σωστά

Αναζήτηση
με νέα κριτηρια

```
def find_smallest(L):
    ''' Βρείτε το μικρότερο στοιχείο σε μια λίστα
        με tuples, σύμφωνα με την Ευκλείδια απόσταση
    '''
Υποθέτουμε ότι η λίστα είναι μη κενή
θέτουμε ως το μικρότερο να είναι το πρώτο στοιχείο
Διατρέχουμε τη λίστα και
        αν βρούμε μικρότερο (ως προς την ευκλείδια απόσταση)
        θέτουμε αυτό ως το νέο μικρότερο
```

Αναζήτηση με νέα κριτηρια

```
def find_smallest(L):
    ''' Βρείτε το μικρότερο στοιχείο σε μια λίστα
        με tuples, σύμφωνα με την Ευκλείδια απόσταση
    '''
    minimum=L[0]
    min_value=minimum[0]**2+minimum[1]**2
    for i in range(1,len(L)):
        x=L[i]
        s=x[0]**2+x[1]**2
        if s<min_value:
            minimum=L[i]
            min_value=s
    return minimum
```

Αναζήτηση με νέα κριτήρια

```
def find_smallest(L):
    ''' Βρείτε το μικρότερο στοιχείο σε μια λίστα
        με tuples, σύμφωνα με την Ευκλείδια απόσταση
    '''
    minimum=L[0]
    for i in range(1,len(L)):
        if my_less(L[i],minimum):
            minimum=L[i]
    return minimum

def my_less(x,y):
    ''' x,y είναι tuple. Επιστρέφει True αν το x είναι μικρότερο από το y
σύμφωνα με την Ευκλείδια απόσταση.
    '''
    v=x[0]**2+x[1]**2
    w=y[0]**2+y[1]**2
    return v<=w
```