Ανάλυση και σχεδιασμός αλγορίθμων

Εργασία 1

Χατζηιωάννου Λαμπρινός, Ευαγγελίδης Νικόλαος, Φιλιππίδης Φοίβος-Παναγιώτης

<2023-04-05 Wed>

Περιεχόμενα

1 Πρόβλημα 1 **1**
1.1 Εισαγωγικές συμβάσεις . . . 1
1.1.1 Data-matrix . . . 1
1.1.2 Polling-vector . . . 2
1.1.3 **TODO** Province Vector . . . 2
1.1.4 Πλήθος υποψηφίων . . . 2
1.1.5 Συναρτήσεις ελέγχου εντός του κώδικα . . . 2
1.1.6 Μεταβλητή εισόδου: `data_matrix` . . . 3
1.1.7 Μορφή εξόδου . . . 3
1.2 Αλγόριθμος πολυπλοκότητας $O(n^2)$ . . . 3
1.3 Βελτιωμένος αλγόριθμος πολυπλοκότητας $O(n \log n)$ . . . 4
1.4 Εξέταση ύπαρξης αλγορίθμου $O(n)$ . . . 4

2 Πρόβλημα 2 **4**
2.1 Περιγραφή του πίνακα S . . . 5
2.2 Ανάλυση του αλγορίθμου . . . 5

1 Πρόβλημα 1

1.1 Εισαγωγικές συμβάσεις

1.1.1 Data-matrix

Στο πλαίσιο της εξέτασης του προβλήματος, καθώς δεν δίνεται ο τύπος με τον οποίο λαμβάνουμε τα δεδομένα, θα θεωρήσουμε ότι τα δεδομένα, για λόγους απλότητας έρχονται με την μορφή κλασσικού πίνακα ακεραίων `data-matrix`, με τα μη αρνητικά στοιχεία του πίνακα να αποτελούν IDs που αντιστοιχούν σε τοπικά πολιτικά πρόσωπα ενώ τα αρνητικά στοιχεία συνεπάγονται το τέλος δεδομένων για εκείνη την εκλογική περιοχή.

Αυτή η σύμβαση γίνεται γιατί στην μορφή του *κλασσικού* πίνακα $k \times l$ όπου

- k το μέγιστο πλήθος των δειγμάτων (πολιτών που ρωτήθηκαν) μεταξύ όλων των περιοχών
- l το πλήθος των περιοχών

Το πλήθος τον δειγμάτων δεν είναι ίδιο σε κάθε περιοχή. Επομένως θα υπάρχουν κοινά στοιχεία στο τέλος ορισμένων γραμμών, τα οποία δεν θα έχουν σημασία για τις μετρήσεις μας.

Σε μια μορφή *παραδείγματος*:

$$\begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 64 & 64 & 32 & 12 & \cdots & \cdots & \cdots & 21 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 54 & 12 & 22 & 13 & \cdots & -2 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

Στην δεύτερη γραμμή φαίνεται ότι sample size $= k$ και επομένως οι αριθμοί συνεχίζουν (με μη αρνητική τιμή μέχρι το τέλος). Στην τέταρτη, όμως, γραμμή το -2 φανερώνει πως τα δεδομένα που συλλέχθηκαν τελειώνουν στην προηγούμενη στήλη, επιτρέποντας μας να πάμε κατευθείαν στην επόμενη γραμμή. Αυτό αν και παρουσιάζει μία μικρή βελτίωση στην ταχύτητα του προγράμματος μας, αποσκοπεί κυρίως στην μη προσπέλαση άχρηστων σε εμάς δεδομένων.

1.1.2 Polling-vector

Την ίδια στιγμή, θα αποθηκεύουμε τα δεδομένα μας σε ένα διάνυσμα `polling-vector` όπου `polling-vector[`
θα είναι οι ψήφοι που έχουν συλλεχθεί κατά την δημοσκόπηση για τον υποψήφιο με το εκάστοτε ID

1.1.3 TODO Province Vector

1.1.4 Πλήθος υποψηφίων

Χωρίς βλάβη της γενικότητας μπορούμε να περιορίσουμε το πλήθος των υποψηφίων σε 1000 (δηλαδή θεωρώντας πως δεν θα υπάρξει ID μεγαλύτερο του 999). Προφανώς, αν κάποια είσοδος απαιτούσε ακόμα μεγαλύτερο πλήθος υποψηφίων αυτό δεν θα επηρέαζε καθόλου τον αλγόριθμο, μόνο θα απαιτούσε κατάλληλη αλλαγή του κώδικα.

1.1.5 Συναρτήσεις ελέγχου εντός του κώδικα

Μέσα στον κώδικα θα δείτε ορισμένες συναρτήσεις οι οποίες ορίστηκαν κυρίως για τον έλεγχο της λογικής και λειτουργίας του αλγορίθμου. Αν και εύκολα κατανοητές και *self-described*, κρίναμε καλό το να τις αφήσουμε εδώ για την αποφυγή της οποιας παρεξήγησης.

1. printNonNilCandidates Πολύ απλή συνάρτηση, πολυπλοκότητας $O(n)$, όπου n το μήκος του διανύσματος εισόδου `myvector`. Ο ρόλος της είναι απλά να τυπώνει τις ψήφους όλων εκείνων που συγκέντρωσαν τουλάχιστον μια ψήφο.

```
def printNonNilCandidates(myvector):
    "Simply iterate through the vector and print candidates that got at least one vote"
    for i in range(len(myvector)):
        if myvector[i] > 0:
            print(f"Candidate {i}:\t {myvector[i]}")
```

2. candidateWinsTheProvince Εξίσου απλή συνάρτηση, πολυπλοκότητας και πάλι $O(n)$, όπου n το μήκος του διανύσματος εισόδου `myvector`, η οποία βοηθά στον έλεγχο του κατα πόσο κάποιος κέρδισε την εκάστοτε εκλογική περιφέρεια.

```
def printNonNilCandidates(myvector):
    "Simply iterate through the vector and print candidates that got at least one vote"
    for i in range(len(myvector)):
        if myvector[i] > 0:
            print(f"Candidate {i}:\t {myvector[i]}")
```

1.1.6 Μεταβλητή εισόδου: `data_matrix`

Ουσιαστικά είναι η μεταβλητή που αλλάζει...
Περίπτωση απλή

```
data_matrix = [[12, 12, 52],
               [13, 2, 15],
               [12, 51, 2]]
```

Γενικά καλύτερη περίπτωση

```
data_matrix = [[12, -12, 52],
               [13, 2, 15],
               [12, 51, -2]]
```

1.1.7 Μορφή εξόδου

Ο αλγόριθμος μας είναι σχεδιασμένος να επιστρέφει ένα boolean vector, με μήκος ίσο με το πλήθος των γραμμών του `data_matrix`. Θετική τιμή σε κάποιο στοιχείο της εξόδου σημαίνει πως υπάρχει πολιτικός, στην περιοχή που αντιστοιχεί εκείνη η γραμμή, που συγκεντρώνει πάνω από το 50% των ψήφων.

1.2 Αλγόριθμος πολυπλοκότητας $O(n^2)$

Ο αρχικός αλγόριθμος είναι αυτός που έρχεται φυσικά σε όποιον ακούσει το πρόβλημα και δεν προϋποθέτει καμία ενέργεια επεξεργασίας των δεδομένων εισόδου:

```
Γ           :     _    [          ] = Ψ
Γ           :
    Γ                                :
        Α            < 0:
            Δ                              ,
           [                   ] += 1
    Γ               :
        Α    [    ] >       :
                  _    [          ] = Α
            Τ
```

Όπως φανερώνει και ο αλγόριθμος σε ψευδογλώσσα, σε big O notation, η πολυπλοκότητα του αλγορίθμου μπορεί αρχικά να εκφραστεί ως:

$$O(\text{number of areas} \times \text{number of samples per area} + \text{number of areas} \times \text{number of candidates}) \quad (1)$$

Θέτοντας ως άνω φράγμα για αυτές τις ποσότητες το n, δηλαδή να ισχύει:

$$\text{number of areas, number of samples per area, number of candidates} \leq n \quad (2)$$

Καθίσταται προφανές ότι η πολυπλοκότητα του κώδικα είναι $O(n^2)$
Παρακάτω δίνεται ο κώδικας του παραπάνω αλγορίθμου σε python

```
def o_squared_complexity_algorithm():
    "First solution, non-optimized, to the given problem"
    number_of_provinces = len(data_matrix)
    province_vector = [0] * number_of_provinces
    for electional_province in range(number_of_provinces):
```

```
        # This should be a rather logical assumption for the working circumstances
        candidate_votes_vector = [0] * 1000
        # This variable is useful to determine
        stop_point = len(data_matrix[electional_province])
        # For every column (point of data at that time)
        for data_point_index in range(stop_point):
            # Agreed upon hypothesis: Valid IDs need to be non-negative
            if data_matrix[electional_province][data_point_index] < 0:
                stop_point = data_point_index
                break
            candidate_votes_vector[data_matrix[electional_province][data_point_index]] += 1
        province_vector[electional_province] = candidatewinstheprovince(candidate_votes_vector, s
        if province_vector[electional_province]:
            print(f"Someone wins province {electional_province}")
```

1.3 Βελτιωμένος αλγόριθμος πολυπλοκότητας $O(n \log n)$

Για να βελτιώσουμε τον αλγόριθμο, έχοντας βοήθεια και από την εκφώνηση όσον αφορά την Divide and Conquer φύση της λύσης με πολυπλοκότητα $O(n \log n)$, αυτό που μπορούμε να κάνουμε είναι:

```
   _      =    (         )
     _      == 1:
    Γ                              :
        A      :
           T
           [                ] += 1

  _      =           _      // 2                  (         )
  _      =                          _      // 2.

Π         1,             _
Π         1,             _
```

1.4 Εξέταση ύπαρξης αλγορίθμου $O(n)$

2 Πρόβλημα 2

Ο κώδικας που δόθηκε είναι της μορφής.

```
Algorithm 1
for i = 0,...,k do
    H[i] = 0
end for
for j = 1,...,n do
    H[T[j]] = H[T[j]] + 1
end for
for i = 1,...,k do
    H[i] = H[i] + H[i - 1]
end for
for j = n,...,1 do
    S[H[T [j]]] = T [j]
```

```
12.     H[T[j]] = H[T[j]] - 1
13. end for
```

2.1 Περιγραφή του πίνακα S

2.2 Ανάλυση του αλγορίθμου