Ανάλυση και σχεδιασμός αλγορίθμων

Εργασία 1 - Ομάδα 58

Χατζηιωάννου Λαμπρινός, Ευαγγελίδης Νικόλαος, Φιλιππίδης Φοίβος-Παναγιώτης

2023-04-05

Περιεχόμενα

1 Πρόβλημα 1

Σκοπεύετε να βάλετε υποψηφιότητα στις επερχόμενες εκλογές και επιθυμείτε να οργανώσετε όσο το δυνατό πιο αποτελεσματικά τις περιοδείες σας.

Εχετε στα χέρια σας δημοσκοπικά δεδομένα τα οποία σας δίνουν πληροφορία για κάθε χωριό και πόλη της περιφέρειας στην οποία θέτετε υποψηφιότητα. Συγκεκριμένα για κάθε ένα από τα μέρη αυτά, έχετε στα χέρια σας τί δήλωσε κάθε πολίτης, που πήρε μέρος σε δημοσκόπηση, οτι προτίθεται να ψηφίσει (το ονοματεπώνυμο των υποψηφίων, εσάς και των αντιπάλων σας).

Επειδή τα χρονικά περιθώρια είναι πολύ στενά, επιθυμείτε να περιοδεύσετε μόνο στα μέρη εκείνα τα οποία φαίνεται από τα δημοσκοπικά δεδομένα ότι δεν συγκεντρώνετε τουλάχιστον τις μισές ψήφους.

1. Σχεδιάστε έναν αλγόριθμο ο οποίος θα αποφασίζει για ένα μέρος σε χρόνο $O(n^2)$
2. Χρησιμοποιήστε την τεχνική Διαίρει και Βασίλευε για να επιτύχετε το παραπάνω σε χρόνο O(n log n).
3. Υπάρχει αλγόριθμος που να επιτυγχάνει την ίδια εργασία σε χρόνο O(n);

1.0.1 Συμβάσεις στην χρήση των μεταβλητών

Στους αλγορίθμους του πρώτου προβλήματος (και τις αναλύσεις τους) θα δείτε την χρήση μεταβλητών όπως:

- n, που αντιστοιχεί στο μήκος της λίστας εισόδου
- k, που αντιστοιχεί στο πλήθος υποψήφιων στις εκάστοτε εκλογές

1.0.2 Συμβάσεις στην φύση του προβλήματος

Καθώς προσδιορίστηκε πως η ισότητα ελέγχου γραμματοσειρών είναι εκτός του ενδιαφέροντος της εργασίας, θεωρήσαμε είσοδο της μορφής:

$$\text{input array} = \begin{bmatrix}\text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \cdots\end{bmatrix} \tag{1}$$

όπου ο κάθε ακέραιος είναι μοναδικό ID το οποίο αντιστοιχεί (ένα προς ένα) σε υποψήφιο της εκάστοτε περιοχής. Έτσι, όταν αναφερόμαστε στον *υποψήφιο*, δεν αναφερόμαστε ονομαστικά σε εκείνο, αλλά στον αριθμό που του αντιστοιχεί.

Το 0 είναι μή αποδεκτό ID, και για λόγους αποδοτικότητας μνήμης (όπως αυτό καθίσταται εμφανές στο τρίτο υποερώτημα), θεωρούμε πως σε κάθε περίπτωση τα ID που έχουν αποδοθεί στα πολιτικά πρόσωπα είναι συνεχόμενα.

1.1 Υποερωτήμα πρώτο

Γνωρίζουμε ήδη οτι προκειμένου να έχει πολυπλοκότητα ο αλγόριθμος της ταξης του $O(n^2)$ θα πρέπει να έχουμε *ενσωματωμένες* επαναλήψεις.

Όπως φαίνεται στον παρακάτω κώδικα, αξιοποιούμε 2 for. Λεκτικά, αυτά μπορούν να περιγραφούν ως εξής:

Για κάθε υποψήφιο στις εκλογές (εξωτερικό loop), μέτρα όλες τις ψήφους που έχει λάβει. Αν εκείνες είναι μεγαλύτερες από τις μισές ψήφους που καταμετρήθηκαν για την περιοχή, τότε το ζητούμενο του ερωτήματος είναι αληθές.

```
1. Ζήτημα_Ερωτήματος = Ψευδές
2. Για i = 0, ... , k-1:
3.    counter = 0
4.    Για j = 0, ... , n-1:
5.  Εάν i == ψήφοι[j]:
6.                counter += 1
7.          Εάν counter > n/2 :
8.                Ζήτημα_Ερωτήματος = Αληθές
9. Επίστρεψε Ζήτημα_Ερωτήματος
```

Στην ανάλυση του παραπάνω αλγορίθμου βλέπουμε ότι:

- Το πρώτο for εκτελείται k φορές. Αν λάβουμε υπόψιν μας την χειρότερη περίπτωση[1], δηλαδή να έχουμε τους μέγιστους υποψήφιους, τότε πρακτικά μιλάμε για n διαφορετικούς υποψηφίους.
- Το ενωματωμένο (nested) for loop, καθώς αποτελείται από πράξεις σταθερού χρόνου $O(1)$, και επαναλαμβάνεται n φορές έχει πολυπλοκότητα $O(n)$

Έτσι, η πολυπλοκότητα, συνολικά του αλγορίθμου:

$$O(k * n) \overset{k \leq n}{=} O(n^2) \tag{2}$$

[1] Κατόπιν διευκρίνησης που δόθηκε ότι $k \leq n$

1.2 Υποερωτήμα δεύτερο

Για να χτίσουμε αλγόριθμο πολυπλοκότητας $O(n \log n)$, είναι προφανές πως η divide and conquer λύση μας θα βασίζεται στην διαίρεση των δεδομένων εισόδου, και την αναδρομική λειτουργία επί των .

1.2.1 Συνάρτηση πλειοψηφία

Η γενική συνάρτηση που θα εκμεταλλευτούμε είναι η πλειοψηφία, η οποία δεδομένων δεικτών αριστερού και δεξιού άκρου εντός της λίστας εισόδου ψήφοι, επιστρέφει την τιμή του πλειοψηφικού στοιχείου της, αν αυτό υπάρχει. Αν δεν υπάρχει τότε επιστρέφει 0 (ψευδές).

Το παραπάνω βασίζεται στο, αποδείξιμο, γεγονός ότι για να είναι ένα στοιχείο πλειοψηφικό μίας λίστας, χρειάζεται να είναι το πλειοψηφικό είτε του δεξιού είτε του αριστερού μισού της.

```
Συνάρτηση πλειοψηφία(αριστερά, δεξιά, ψήφοι):

  // Αν μόνο ενα στοιχείο εντός της λίστας τότε
  // είναι το πλειοψηφικό στοιχείο της.
  Εάν αριστερά == δεξιά:
    επίστρεψε ψήφοι[δεξιά]

  // Μοίρασε την λίστα στην μέση, και κατόπιν αναδρομικά
  // εξέτασε το δεξί και το αριστερό κομμάτι.
  μέση              = (αριστερά + δεξιά) // 2
  αριστερή_πλειοψηφία = πλειοψηφία(αριστερά, μέση, ψήφοι)
  δεξί_πλειοψηφία     = πλειοψηφία(μέση+1, δεξιά, ψήφοι)

  // Εάν πλειοψηφία και κοινός υποψήφιος σε πλειοψηφία
  Εάν αριστερή_πλειοψηφία == δεξιά_πλειοψηφία:
    επίστρεψε αριστερή_πλειοψηφία

  // Εάν υπάρχει πλειοψηφία στο αριστερό μισό
  Εάν αριστερή_πλειοψηφία:
    επαναλήψεις = 0.
    Για j = αριστερά ,,, δεξιά:
      Αν ψήφοι[j] == αριστερη_πλειοψηφία:
        επαναλήψεις += 1.
    Αν επαναλήψεις > (δεξιά - αριστερά) // 2:
      επίστρεψε αριστερή_πλειοψηφία

  // Αντίστοιχα, για πλειοψηφία στο δεξί μισό
  Εάν δεξιά_πλειοψηφία:
    επαναλήψεις = 0.
    Για j = αριστερά ,,, δεξιά:
      Αν ψήφοι[j] == δεξιά_πλειοψηφία
        επαναλήψεις += 1.
    Αν επαναλήψεις > (δεξιά - αριστερά) // 2:
      επίστρεψε δεξιά_πλειοψηφία

  Επίστρεψε 0
```

1.2.2 Ανάλυση και επαλήθευση

Η εγκυρότητα της παραπάνω συνάρτησης/αλγορίθμου, επαληθεύεται με την μέθοδο της επαγωγής. Για την χρονική πολυπλοκότητα της έχουμε:

$$T(n) = T(n/2) + O(n) \tag{3}$$

Χαρακτηριστική επίλυσης τύπου DC, η οποία για $n \to \infty$ ξέρουμε ότι συνεπάγεται πολυπλοκότητα $O(n \log n)$.

1.2.3 Τελική μορφή αλγόριθμου

Έτσι, αν θεωρήσουμε για λόγους συνέπειας πως θέλουμε το πρόγραμμα μας να επιστρέφει Αληθή ή Ψευδή τιμή βάση της ύπαρξης πλειοψηφίας στο εξεταζόμενο μέρος, ο τελικός αλγόριθμος είναι ο εξής:

```
1. Αν πλειοψηφία(0, n-1, ψήφοι)
2.    Ζήτημα_Ερωτήματος = Αληθές
3. Αλλιώς
4.    Ψευδές
```

1.3 Υποερωτήμα τρίτο

Υπάρχει γραμμικός αλγόριθμος, τον οποίο και παραθέτουμε παρακάτω. Εκμεταλλευόμενοι την έλλειψη περιορισμού στην μνήμη, κατασκευάζουμε μία λίστα X, μήκους k, ώστε να χωράει τον αριθμό ψήφων του κάθε υποψηφίου, και ελέγχουμε ένα ένα τα στοιχεία της λίστα εισόδου ψήφοι, ενημερώνοντας κατάλληλα την X.

Σε κάθε βήμα ελέγχουμε αν το ενημερωμένο στοιχείο της X πληρεί τις προϋποθέσεις πλειοψηφίας, για την επιστροφή αληθούς τιμής.

```
1. Ζήτημα_Ερωτήματος = Ψευδές
2. Για i = 0, ... , k-1:
3.    X[i] = 0
4. Για i = 0, ... , n-1:
5.    X[ψήφοι[i]] += 1
6.    Εάν X[ψήφοι[i]] > n/2:
7.       Ζήτημα_Ερωτήματος = Αληθές
```

2 Πρόβλημα 2

Ο κώδικας που δόθηκε είναι της μορφής.
Algorithm 1

```
1. for i = 0,...,k do
2.    H[i] = 0
3. end for
4. for j = 1,...,n do
5.    H[T[j]] = H[T[j]] + 1
6. end for
7. for i = 1,...,k do
8.    H[i] = H[i] + H[i − 1]
9. end for
```

```
10. for j = n,...,1 do
11.    S[H[T [j]]] = T [j]
12.    H[T[j]] = H[T[j]] − 1
13. end for
```

2.1 Περιγραφή του αλγορίθμου

Ο δοθείς αλγόριθμος είναι μια παραλλαγή του counting sort, αλγόριθμου με πολυπλοκότητα $O(n+k)$, χρήσιμου, όμως, μόνο όταν γνωρίζουμε το k (μέγιστη πιθανή τιμή των στοιχείων εντός του πίνακα) και εκείνο είναι επαρκώς μικρό, ώστε να μπορούμε να δεσμέυσουμε αρκετά μεγάλο χώρο για την λίστα H.

Έχουμε

1. Πρώτο loop, block complexity $O(k)$: Αρχικοποίηση του πίνακα H, ούτως ώστε όλα του τα στοιχεία να είναι μηδενικά
2. Δεύτερο loop, block complexity $O(n)$: Ουσιαστικά διαμόρφωση του πίνακα H ούτως ώστε το στοιχείο $H[n]$ να μας δείχνει πόσες φορές υπήρξε το n στον πίνακα εισόδου T
3. Τρίτο loop, block complexity $O(k)$: Διαμόρφωση του πίνακα H ούτως ώστε το $H[n]$, να μας δείχνει το πλήθος των στοιχείων τιμής μικρότερης η ίσης του n υπήρξε στον πίνακα εισόδου T
4. Τέταρτο loop, block complexity $O(n)$: Εισαγωγή του πίνακα S: Για κάθε στοιχείο του αρχικού πίνακα T, χρησιμοποιούμε τον πίνακα H και το τοποθετούμε στην κατάλληλη θέση. Αναλυτικότερα, γνωρίζοντας από το H πόσα στοιχεία βρίσκονται πριν από αυτό στην ταξινομιμένη εκδοχή του πίνακα, το βάζουμε στην μέγιστη θέση που μπορεί να μπεί, ενώ παράλληλα μειώνουμε το κελί του H που αντιστοιχεί στην υπο εξέταση τιμή, ώστε τυχόν επαναλήψεις του να μπουν πριν από εκείνο. Έτσι τελικά, ο S είναι ο T με τα στοιχεία του σε αύξουσα σειρά.

2.2 Ανάλυση του αλγορίθμου

Ο αλγόριθμος αποτελείται από μία σειρά δομών επανάληψης, χωρίς καμία να είναι ενσωματωμένη σε άλλη. Η πολυπλοκότητα κάθε μίας εξ αυτών, καθώς περιλαμβάνει μόνο απλές πράξεις (πολυπλοκότητας $O(1)$) εύκολα φαίνεται ότι είναι:

- $O(k)$
- $O(n)$
- $O(k)$
- $O(n)$

Έτσι, συνολικά ο αλγόριθμος έχει πολυπλοκότητα $O(n+k)$