1η Εργασία - Ανάλυση και Σχεδιασμός Αλγορίθμων

Πρόβλημα 1:

Αρχικά, μας ζητήθηκε να σχεδιάσουμε έναν αλγόριθμο ο οποίος θα εξετάζει αν κάποιος από τους υποψήφιους συγκέντρωσε περισσότερες από τις μισές ψήφους σε ένα συγκεκριμένο μέρος. Έχουμε στη διάθεση μας τα δημοσκοπικά δεδομένα, τα οποία μας δίνουν πληροφορία για αυτό το συγκεκριμένο μέρος. Επίσης, γνωρίζουμε τι δήλωσε κάθε πολίτης ότι προτίθεται να ψηφίσει (το ονοματεπώνυμο των υποψηφίων, εμάς και των αντιπάλων μας).

Ζητούμενο 1:

Για να πετύχουμε τη σχεδίαση αλγορίθμου ο οποίος εκτελείται σε χρόνο $O(n^2)$ θα χρησιμοποιήσουμε δύο εμφωλευμένους βρόχους επανάληψης for.

Procedure find_candidate(A[0,...n-1], n)

```
1:  thesi = -1
2:  for ( i=0 to n/2 )
3:      cnt = 1
4:      for ( j=i+1 to n-1 )
5:          if ( A[i] = A[j] )
6:          cnt=cnt+1
7:      end for
8:      if ( cnt > n/2 )
9:          thesi = i
10: end for
11:  if ( thesi = -1 )
12:     print Nobody
13: else
14:     print A[thesi]
```

Περιγραφή find_candidate:

Η συνάρτηση find_candidate δέχεται ως είσοδο ένα πίνακα (A) που περιέχει την ψήφο των πολιτών και το μέγεθός του (n).

Ξεκινώντας, στη γραμμή 1 ορίζουμε μία μεταβλητή με όνομα thesi και αρχική τιμή -1. Στη γραμμή 2, χρησιμοποιούμε το πρώτο από τα δύο for το οποίο αρχίζει από το i=0 μέχρι το n/2 για να έχουμε πρόσβαση στα μισά στοιχεία του πίνακα A, αφού μετά τα μισά στοιχεία του πίνακα δεν υπάρχει περίπτωση κάποιος υποψήφιος να έχει πάνω από το 50%. Στη γραμμή 3, για κάθε διαφορετική θέση του πίνακα ορίζουμε τον counter ίσο με 1. Στη συνέχεια, στη γραμμή 4 χρησιμοποιούμε το δεύτερο for το οποίο αρχίζει από το j=i+1 μέχρι το n-1 και συγκρίνει το i-οστό στοιχείο με τα επόμενα στοιχεία του πίνακα. Στη γραμμή 5, ελέγχουμε πόσες φορές το i-οστό στοιχείο είναι ίσο με το j-οστό στοιχείο, έτσι ώστε στη γραμμή 7, όταν τελειώσουν οι επαναλήψεις του δεύτερου for, να βρίσκουμε το πλήθος των εμφανίσεων του i-οστού στοιχείου. Το πλήθος αυτό υπολογίζεται στη γραμμή 6 με τη βοήθεια του counter, ο οποίος αυξάνεται κατά 1. Στη γραμμή 8, ελέγχουμε αν ο counter είναι μεγαλύτερος από το μισό πλήθος του πίνακα(n/2). Εάν συμβαίνει αυτό, στη γραμμή 9 ορίζουμε την τιμή της μεταβλητής thesi ίση με το i. Στη γραμμή 10, τελειώνει ο πρώτος βρόχος αφού επαναληφθεί n/2+1 φορές. Στη γραμμή 11, ελέγχουμε εάν η τιμή της μεταβλητής thesi ισούται με -1, επειδή δεν ικανοποιήθηκε η συνθήκη στη γραμμή 8. Εάν συμβαίνει αυτό συμπεραίνουμε ότι δεν υπάρχει υποψήφιος που να συγκέντρωσε περισσότερες από τις μισές ψήφους και έτσι στη γραμμή 12 εκτυπώνουμε το αντίστοιχο μήνυμα. Διαφορετικά στη γραμμή 14 εκτυπώνει τον υποψήφιο που συγκέντρωσε περισσότερες από τις μισές ψήφους.

Ζητούμενο 2:

Για να πετύχουμε χρόνο πολυπλοκότητας O(nlogn) με την χρήση της τεχνικής διαίρει και βασίλευε χρησιμοποιούμε τον αλγόριθμο find_majority, ο οποίος διαίρει στη μέση τον αρχικό πίνακα μέσω των 2 αναδρομικών κλήσεων. Η μία κλήση είναι για να βρεθεί ο υποψήφιος που το ονοματεπώνυμό του εμφανίζεται περισσότερες από τις μισές φορές στο πρώτο μισό του πίνακα A (0 έως middle) και η δεύτερη κλήση για να βρεθεί ο υποψήφιος στο δεύτερο μισό του πίνακα (middle+1 έως n-1). Αυτό επιτυγχάνεται και με την κλήση της συνάρτησης votes η οποία βρίσκει τις εμφανίσεις του κάθε υποψηφίου στον αντίστοιχο υποπίνακα, σε γραμμικό χρόνο O(n). Μόλις ολοκληρωθεί αυτό, γίνεται σύγκριση μεταξύ των δύο αυτών υποψηφίων. Εάν είναι ίσοι τότε επιστρέφουμε τον υποψήφιο αυτό. Διαφορετικά, ελέγχουμε πόσες φορές εμφανίζεται ο αριστερός υποψήφιος στον αρχικό πίνακα απο (0,...n-1) και πόσες ο δεξιός. Αν κάποιος από αυτούς εμφανίστηκε περισσότερες από τις μισές φορές επιστρέφουμε τον αντίστοιχο υποψήφιο, διαφορετικά επιστρέφουμε ότι δεν υπάρχει τέτοιος υποψήφιος.

Ο χρόνος εκτέλεσης της find_majority είναι:

T(n)=T(n/2)+T(n/2)+O(n)=2T(n/2)+O(n)

Όπου 2T(n/2) είναι ο χρόνος για τις δύο αναδρομικές κλήσεις και O(n) ο χρόνος εκτέλεσης της συνάρτησης votes. Εφαρμόζοντας το Θεώρημα Κυριαρχίας προκύπτει ο χρόνος πολυπλοκότητας O(nlogn).

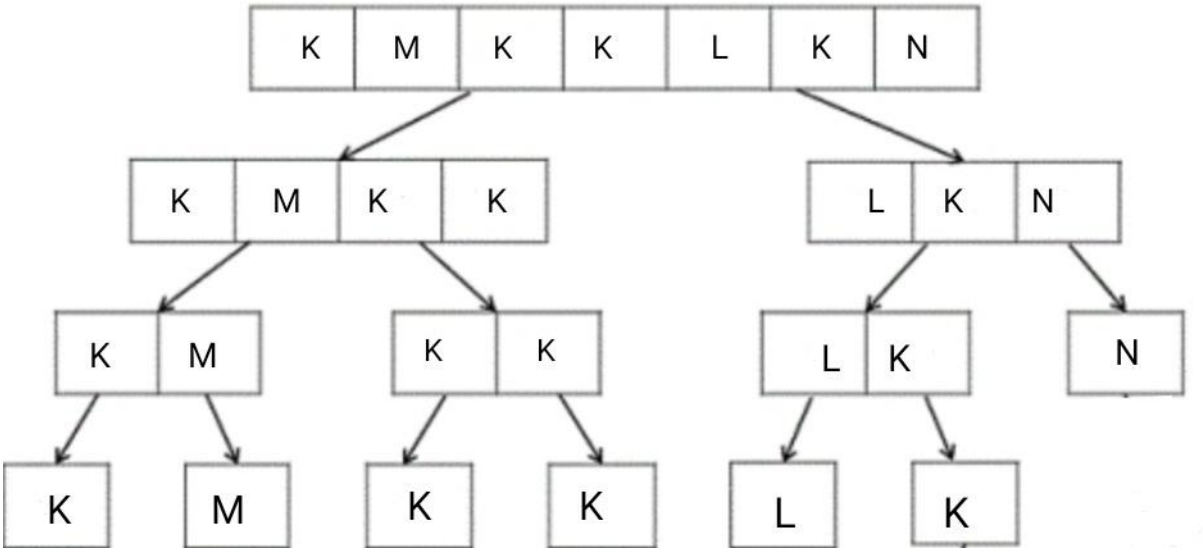
Procedure votes(A , name , start , finish)

```
1: cnt=0
2:  for(i=start to finish)
3:       if(A[i]=name)
4:                cnt=cnt+1
5:  end for
6:  return cnt
7:  end procedure
```

Procedure find_majority(A[0,..,n-1] , start , finish)

```
1:  if (start = finish)
2:      return A[start]
3:  end if
4:  middle = (start+finish)/2
5:  majority1 = find_majority(A, start, middle)
6:  majority2 = find_majority(A, middle+1, finish)
7:  if (majority1 = majority2)
8:      return majority1
9:  end if
10: cnt1 = votes(A, majority1, start, finish)
11: cnt2 = votes(A, majority2, start, finish)
12: if (cnt1 > (finish-start+1)/2)
13:     return majority1
14: else if (cnt2 > (finish-start+1)/2)
15:     return majority2
16: else
17:     return “Nobody”
18: end if
19: end procedure
```

Παράδειγμα εκτέλεσης find_majority A[]={K,M,K,K,L,K,N }(n=7):



Αρχικά start=0 και finish=6, είναι άνισα οπότε γίνεται αναδρομική κλήση για να βρεθεί ο υποψήφιος με περισσότερες από n/2 εμφανίσεις στο πρώτο μισό του πίνακα (0 εως 3). Τώρα, για start=0 και finish=3, είναι άνισα οπότε γίνονται αναδρομικές κλήσεις από 0 έως 1 και από 2 έως 3. Για την αναδρομική κλήση από 0 έως 1, start=0 και finish=1, είναι άνισα οπότε ξανά γίνονται αναδρομικές κλήσεις για τη θέση 0 και για τη θέση 1. Σε αυτές τις δύο αναδρομές start=finish οπότε προκύπτει ότι, majority1=K και majority2=M. Αυτά τα 2 στοιχεία είναι άνισα και το πλήθος τους δεν είναι μεγαλύτερο από το μισό μέγεθος του υποπίνακα απο 0 έως 1, οπότε majority1=Nobody. Συνεχίζουμε με την αναδρομική κλήση από 2 έως 3, οπότε start=2 και finish=3, είναι άνισα οπότε ξανά γίνονται αναδρομικές κλήσεις για το στοιχείο στη θέση 2 και για το στοιχείο στη θέση 3. Σε αυτές τις δύο αναδρομές start=finish, οπότε majority1=K και majority2=K, είναι ίσα οπότε προκύπτει ότι majority2=K. Τέλος, γίνεται σύγκριση majority1=Nobody και majority2=K, είναι άνισα οπότε μέσω της συνάρτησης votes μετράμε το πλήθος των εμφανίσεων του καθενός ξεχωριστά στο πρώτο μισό του πίνακα, δηλαδή από 0 έως 3. Τελικά προκύπτει ότι ο υποψήφιος για το πρώτο μισό του πίνακα είναι ο majority1=K, αφού εμφανίζεται περισσότερες από τις μισές φορές. Αντίστοιχα προκύπτει ότι ο δεύτερος υποψήφιος για το δεύτερο μισό του πίνακα A είναι majority2=Nobody(4 εως 6). Γίνεται σύγκριση μεταξύ των δύο αυτών υποψήφιων και αφού δεν είναι ίσοι ελέγχουμε το πλήθος των εμφανίσεων του καθενός στον αρχικό πίνακα A, δηλαδή από 0 έως 6. Τελικά προκύπτει ότι ο υποψήφιος με περισσότερες από n/2 ψήφους είναι ο K.

Ζητούμενο 3:

Ναι, υπάρχει αλγόριθμος που να επιτυγχάνει την ίδια εργασία σε χρόνο O(n) και για να το πετύχουμε αυτό χρησιμοποιήσαμε τον αλγόριθμο Boyer-Moore majority vote, ο οποίος βρίσκει το ονοματεπώνυμο του υποψηφίου που εμφανίζεται περισσότερες από τις μισές φορές στον πίνακα A σε γραμμικό χρόνο O(n). Στη συνέχεια, μέσω ενός for βρίσκουμε το πλήθος των εμφανίσεων του υποψηφίου αυτού σε γραμμικό χρόνο O(n). Αυτό χρειάζεται έτσι ώστε να επαληθεύσουμε ότι όντως αυτός ο υποψήφιος συγκέντρωσε περισσότερες από τις μισές ψήφους, αφού υπάρχει περίπτωση ο υποψήφιος να μην τηρεί το κριτήριο αυτό.

Procedure find_candidate(A[0,...,n-1] , n)

```
1:  majority = A[0], cnt1 = 1
2:   for ( i=1 to n-1 )
3:       if ( A[i] = majority )
4:            cnt1 = cnt1+1
5:       else if ( cnt1 = 0 )
6:             majority = A[i]
7:             cnt1 = 1
8:       else
9:            cnt1=cnt1-1
10:  end for
11:  cnt2 = 0
12:  for ( i=0 to n-1 )
13:     if ( majority = A[i] )
14:          cnt2++
15:  end for
16:  if ( cnt2 > n/2 )
17:     print majority
18:  else
19:     print Nobody
```

Η συνάρτηση find_candidate δέχεται ως είσοδο ένα πίνακα (A) που περιέχει την ψήφο των πολιτών και το μέγεθός του (n).

Ξεκινώντας, στη γραμμή 1 ορίζουμε μία μεταβλητή majority με αρχική τιμή το πρώτο στοιχείο του πίνακα A, καθώς και μία μεταβλητή cnt1 (counter1) με αρχική τιμή 1. Στη γραμμή 2, χρησιμοποιούμε ένα for το οποίο αρχίζει από το i=1 μέχρι το n-1 για να έχουμε πρόσβαση σε όλα τα στοιχεία του πίνακα, για τα οποία ελέγχουμε στη γραμμή 3 εάν ισούνται με την μεταβλητή majority. Εάν συμβαίνει αυτό, στη γραμμή 4, αυξάνουμε την τιμή του counter1 κατά 1. Διαφορετικά, ελέγχουμε στη γραμμή 5 εάν το counter1 ισούται με 0. Εάν επαληθεύεται ο έλεγχος αυτός, ορίζουμε στην γραμμή 6 την τιμή του majority ίση με την τιμή του στοιχείου που βρίσκεται στην θέση i του πίνακα A και στη γραμμή 7 ορίζουμε την τιμή του counter1 ίση με 1. Διαφορετικά, στη γραμμή 9 μειώνουμε την τιμή του counter1 κατά 1. Στη γραμμή 10 τελειώνει ο πρώτος βρόχος αφού επαναληφθεί n φορές. Στη συνέχεια, στη γραμμή 11 ορίζουμε μία μεταβλητή με όνομα cnt2 (counter2) και αρχική τιμή 0. Στη γραμμή 12, χρησιμοποιούμε ένα δεύτερο for το οποίο αρχίζει από το i=0 μέχρι το n-1 για να έχουμε πρόσβαση σε όλα τα στοιχεία του πίνακα, για τα οποία ελέγχουμε στη γραμμή 13 εάν η μεταβλητή majority ισούται με αυτά. Εάν συμβαίνει αυτό, στη γραμμή 14, αυξάνουμε την τιμή του counter2 κατά 1. Στη γραμμή 15 τελειώνει ο δεύτερος βρόχος αφού επαναληφθεί n φορές. Αφού βρήκαμε την τιμή του counter2, ελέγχουμε στη γραμμή 16 εάν το πλήθος αυτό είναι μεγαλύτερο από το n/2, δηλαδή τις μισές ψήφους. Εάν συμβαίνει αυτό, στη γραμμή 17 εκτυπώνουμε το majority, δηλαδή το όνομα του υποψηφίου που συγκέντρωσε περισσότερες από τις μισές ψήφους. Διαφορετικά, εάν δεν ξεπέρασε το πλήθος των μισών ψήφων, τότε στη γραμμή 19 εκτυπώνουμε το μήνυμα ότι δεν βρέθηκε κάποιος υποψήφιος που να έχει περισσότερες από τις μισές ψήφους.

Πρόβλημα 2:

Ζητούμενο 1:

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
```

Στον παραπάνω αλγόριθμο υλοποιούνται 4 διαφορετικοί βρόχοι for, με σκοπό στο τέλος του τέταρτου βρόχου να συμπληρωθεί ο πίνακας S μεγέθους n με τα στοιχεία του πίνακα T σε αύξουσα σειρά. Αυτό επιτυγχάνεται ως εξής:

Στον **πρώτο βρόχο** for αρχικοποιείται με 0 ο πίνακας H μεγέθους k+1, όπου το k είναι το μέγιστο στοιχείο του πίνακα T.

Ο **δεύτερος βρόχος** for χρησιμοποιείται για την μέτρηση των εμφανίσεων του κάθε αριθμού στον πίνακα T χρησιμοποιώντας την εντολή H[T[j]] = H[T[j]] + 1.

Το πλήθος του κάθε αριθμού καταχωρείται στον πίνακα H στην αντίστοιχη θέση του αριθμού.

Παραδείγματος χάρη,

Είσοδος:

T[1]=1 T[2]=6 T[3]=5 T[4]=4 T[5]=5 T[6]=5 T[7]=4 T[8]=4

Έξοδος:

H[0]=0 H[1]=1 H[2]=0 H[3]=0 H[4]=3 H[5]=3 H[6]=1

Στον **τρίτο βρόχο** διατρέχει τον πίνακα H για να αποθηκεύσει τον αριθμό των στοιχείων που είναι μικρότερα ή ίσα με κάθε στοιχείο στον πίνακα T. Μετρώντας το πλήθος αυτό μπορούμε να προσδιορίσουμε τη σωστή θέση του στοιχείου στον ταξινομημένο πίνακα S[ ].

Έξοδος:

H[0]=0 H[1]=1 H[2]=1 H[3]=1 H[4]=4 H[5]=7 H[6]=8

Στο συγκεκριμένο παράδειγμα δεν υπάρχουν στοιχεία μικρότερα ή ίσα του 0, υπάρχει ένα στοιχείο μικρότερο ή ίσο του 1 κτλ.

Στον **τέταρτο βρόχο** καταχωρείται κάθε στοιχείο του T στην σωστή του θέση στον ταξινομημένο πίνακα S, χρησιμοποιώντας τον πίνακα H που λειτουργεί σαν δείκτης. Στην συνέχεια, μειώνει το πλήθος του στοιχείου T[j] στον πίνακα H κατά 1. Αυτό γίνεται έτσι ώστε σε περίπτωση που το στοιχείο βρεθεί δύο ή περισσότερες φορές στον πίνακα T να τοποθετηθεί στον πίνακα S στην προηγούμενη του θέση.

Έξοδος:

H[4]=3 H[4]=2 H[5]=6 H[5]=5 H[4]=1 H[5]=4 H[6]=7 H[1]=0

S[1]=1 S[2]=4 S[3]=4 S[4]=4 S[5]=5 S[6]=5 S[7]=5 S[8]=6

Ζητούμενο 2:

Ο πρώτος βρόχος εκτελείται (k+1) φορές, ο δεύτερος και ο τέταρτος βρόχος εκτελούνται n φορές, ενώ ο τρίτος βρόχος εκτελείται k φορές. Οι γραμμές 2,5,8,11 και 12 εκτελούνται σε σταθερό χρόνο O(1), για αυτό δεν τις λαμβάνουμε υπόψη. Οπότε ο χρόνος εκτέλεσης του παραπάνω αλγόριθμου είναι:

$T(n) = k+1+n+k+n = O(n+k)$

, δηλαδή έχει γραμμικό χρόνο ως προς το μέγεθος n του πίνακα T και το μέγιστο του στοιχείο k.

Είναι αποτελεσματικός όταν το εύρος των τιμών εισόδου [0,..,k] δεν είναι σημαντικά μεγαλύτερο από τον αριθμό των στοιχείων εισόδου. Σε περίπτωση που το “k” είναι πολύ μεγαλύτερο από το πλήθος των στοιχείων του T, δηλαδή μεγαλύτερο της τάξης του n, τότε η χρήση μνήμης που απαιτείται από τον αλγόριθμο γίνεται αναποτελεσματική.

Ευστρατίου Ευστράτιος, ΑΕΜ: 10489

Περατικού Νιόβη, ΑΕΜ: 10494

Θεοφίλου Αθηνά, ΑΕΜ: 10146