Algorithms - Assignment 1

Καζγκούτης Αθανάσιος Charbel Al Haddad
Παπαδόπουλος Δημήτριος-Λάζαρος

March 24, 2023

Πρόβλημα 1

Ερώτημα 1

Ο αλγόριθμος που παρατιίθεται παρακάτω δέχεται σαν είσοδο μια συστοιχία n στοιχίων όπου n είναι οι ψήφοι -σε ονοματεπώνυμα-μιας κοινότητας. Στόχος του αλγόριθμου ειναι:

1. να ελέγξει ποιος υποψήφιος έχει τους περισσότερους ψήφους.
2. αμα ξεπερνάει το 50% των συνολικών ψήφων.
3. εφόσων ισχύει το παραπάνω να επιστρέψει στην έξοδο το ονοματεπώνυμο του υποψήφιου .

ΑΛΓΟΡΙΘΜΟΣ 1

```
1  function MajorityFinder(A[1...n])
2   majority_person
3   maxcount = 0
4   count
5   candidate
6   for(i = 1 to n){
7       count = 0
8       candidate = A[i]
9       for(j = 1 to n){
10          if(candidate == A[j])
11              count++}
12      if(count > maxcount)
13          maxcount = count
14          majority_person = temp}
15  if(maxcount > ⌈n/2⌉)
16      return majority_person
17  else
18      return "no person has the_majority"
```

Ανάλυση Αλγορίθμου:

- Στη 1^{η} σείρα ο αλγόριθμος δέχεται τις n ψήφους μέσω μιας συστοιχίας "A[1...n]"
- Στις γραμμές 2-5 γίνονται αρχικοποιήσεις μεταβλητών που χρησιμέυουν στη καταμέτρηση των ψήφων (count,maxcount) και των υποψήφιων(candidate,majority-person),ώστε να εξαχθεί το αποτέλεσμα με επιτυχία.
- Στις γραμμές 6-11 ο αλγόριθμος καταμετρεί όλες τις ψήφους κάθε υποψήφιου..Συγκεκριμένα η αρχική η for χρησιμοποιείται για να προσπελαστούν όλες οι ψήφοι,ενώ η 2η για να γίνει έλεγχος ποιες ψηφοί έχουν το ίδιο ονοματεπώνυμο με την i-στη.Η μεταβλητή count απαριθμεί τις συνολικες ψήφους που έχει ο υποψήφιος της i-στης θέσης του αρχικού πινακα. Επίσης στη μεταβλητή candidate αποθηκέυται το ονοματεπώνυμο του υποψήφιου της i-στης ψήφου.Επειδή έχουμε εμφωλευμένες for η πολυπολοκότητα γίνεται O(n^2).Τέλος να αναφέρουμε οτι ο έλεγχος που γίνεται στη γραμμη 10 είναι γραμμικός με O(κ) οπού κ το μήκος της συμβολοσειράς ,αλλα γίνεται εύκολα αντιληπτό πως η τάξη μεγέθους n≫ κ οπότε ο ελέγχος κάθε συμβολοσείρας θα θεωρείτε οτί έχει πολυπλοκότητα O(1).
- Στις γραμμες 12-14 γίνεται ελεγχός για να βρέθει ποιος υποψήφιος εχει τις περισσοτερες ψήφους.
- Στις τελευταιες γραμμες 15-18 σκοπός του αλγορίθμου είναι να ελεγχθεί αμά ο υποψήφιος που έχει τις περισσότερες ψήφους έχει επίσης και την πλειοψηφία(ποσοστό>50%).
- Ανακεφαλαιώνοντας πρόκειται για έναν ***αργό αλγόριθμο O(n^2)***, καθώς ελέγχονται n φορες όλα τα ονοματεπώνυμα ενώ συγκρίνουμε ολές τις ψήφους μεταξύ τους μια προς μια n φορές .Γεγονώς που θα προσπαθήσουμε να ανατρέψουμε στα επώμενα ερωτήματα χρησιμοποιόντας καλύτερες τεχνικές.

Ερώτημα 2

Στη προσπάθεια μας να βελτιώσουμε τον αλγόριθμο πολυπλοκότητας από $0(n^2)$ σε $0(logn)$ θα χρησιμοποιήσουμε την στρατιγική ***διαίρει-και-κυρίευε***(Divide-and-conquer).Συγκεκριμένα ο αλγόριθμος μας βασίζεται στην συγχωνευτική ταξινόμιση (merge sort) ,ωστε να ταξινομηθουν τα ονοματεπώνυμα σε μια σειρά(στη περίπτωση μας αλφαβητικά).Τελός έχωντας μια ταξινομημένη λίστα μπορούμε πολυ ευκόλα να ελέγξουμε αν υπάρχει υποψήφιος με τουλάχιστον 50% των ψήφων όπως παρουσιάζεται αναλυτικότερα παρακάτω. Ο βοηθητικός αλγοριθμος που θα χρησιμοποιήσουμε:

Merge Sort

```
function mergesort(a[1...n])
 if(n > 1)
     return merge(mergesort(a[1...⌊n/2⌋]), mergesort(a[⌊n/2⌋ + 1 ...n]))
 else
     return a

function merge(x[1...k], y[1...l])
 if(k = 0)
     return y[1...l]
 if(l = 0)
     return x[1...l]
 if(x[1] ≥ y[1])
     return x[1] ∘ merge(x[2...k], y[1...l])
 else
     return y[1] ∘ merge(x[1...k], y[2...l])
```

Ανάλυση merge sort για την περίπτωση μας:

- **Αναδρομή**:διαιρούμε το πρόβλημα μας στη μέση $\frac{n}{2}$ διαδοχικά(a=2,b=2),η αναδρομή εξαντλείτε οταν η ταξινομητέα ακολουθία έχει μήκος 1 ≥n, δηλαδή εχεί μονο μια ψήφο με ονοματεπώνυμο που ειναι ήδη ταξινομιμένη.

- **Συγχώνευση(merge)**:η συνάρτηση αυτή συγχωνέυει ταξινομόντας 2 ήδη ταξινομημένους πίνακες ,το βασίκο βήμα της συνάστησης εκτελείτε το πολύ n φορές ενω το υπολογιστικό κοστός c ειναι στη χειρότερη περίπτωση ίσο με το χρόνο που χρειάζεται να προσπελαστή ένα ονοματεπώνυμο ,αυτό συμβαίνει οταν έχουμε να συγκρίνουμε το ίδιο όνομα δηλαδή την ίδια ψήφο .Συγκεκρίμενα πάλι θεώρουμε ότι το μήκος του έχει ενα λογικό εύρος και το c είναι απλα ένας πεπερασμένος αριθμός.Σε κάθε άλλη κατάσταση άν ο πρώτος χαρακτήρας από την πρώτη συμβολοσειρά είναι μεγαλύτερος (ή μικρότερος) από αυτόν της άλλης συμβολοσειράς, τότε η πρώτη συμβολοσειρά είναι μεγαλύτερη (ή μικρότερη) από τη δεύτερη.Σύμφωνα με τα παραπάνω η συνάρτηση merge έχει χρόνο εκτέλεσης $\mathbf{\Theta(n)}$.

- **Συνολικός χρόνος** :Οπως γνωρίζουμε απο την θεωρία (master theory) για a=2,b=2

  $$T(n) = 2T(\frac{n}{2}) + \Theta(n) = cnlogn + cn$$

  αρα θα έχουμε χρόνο εκτέλεσης Θ(nlogn).

- παρακάτω παρουσιάζεται ο βασικός αλγόριθμος του ερωτήματος 2 :

ΑΛΓΟΡΙΘΜΟΣ 2

```
1 function  MajorityFinder2(A[1...n])
2  mergesort(A)
3  for(i = 1 to n){
4      if((n mod 2 == 1) AND (A[i] == A[⌈n/2⌉ − 1 + i]))
5                  return A[i] //end algorithm
6       if ((n mod 2 == 0) AND (A[i] == A[⌈n/2⌉ + i]))
7                  return A[i]//end algorithm
8  }
9  return no majority person found
```

Ανάλυση Αλγορίθμου

- Στη πρωτη σειρα ο πίνακας A δέχεται τους ψηφούς σε ονοματεπώνυμα (συμβολοσειρές).
- Στη 2η γραμμή καλούμε την συνάρτηση mergesort για να ταξινομήσουμε τον πίνακα A κατα αλφαβητική σειρά(Θ(nlogn))
- Στις επόμενες 2 γραμμές ο αλγόριθμος εξετάζει το σενάριο να έχουμε ταυτόχρονα ισοβαθμία 50% 2 υποψήφιων.Αυτο συμβαίνει οταν αρχικα εχουμε αρτίο αριθμό ψήφων και οι πρωτες μισές ψήφοι του ταξινομημένου πίνακα ανήκουν στον ένα υποψήφιο και οι άλλες μισές στον αλλο ,ολο αυτο μοντελοποιείται με έναν βρόχο ελέγχου που απαιτεί πεπέρασμενο χρόνο 0(1) αφου ειναι 4 έλεγχεει.
- Στις γραμμες 6-10 ο αλγόριθμος αναζητεί αμα υπάρχει διάστημα μήκους $\frac{n}{2}$ που να υπάρχει ο ίδιος υποψήφιος ελέγχοντας μονο το 1ο και το $\frac{n}{2}$+1 ψήφο εκμεταλεύονας το γεγονός οτι ο πίνακας A είναι ταξινομημένος . Ο συνολίκος χρόνος που απαιτεί η συγκεκριμένη διαδικασία είναι T($\frac{n}{2}$) και ειναί γραμμική η προσπέλαση άρα O(n).
- επειδή η γραμμές 6-10 δεν συνδέονατaι με κάποια αναδρομική σχέση μεσα στην συνάρτηση merge η **συνολικη πολυπλοκότητα** του 2ου αλγορίθμου είναι nlogn +n και αρα **0(nlogn).**
- Ανακεφαλαιώνοντας ο αλγόριθμος αυτός σαφώς εμφανίζεται ποιο βελτιωμένος από τον πρώτο καθώς εχούμε ελλατώσει τις συγκρίσεις σε μεγάλο βαθμό με την βοήθεια του devide and conquer αλλά και πάλι εμφανίζονται αρκετές επαναλήψεις ανάγνωσης ψήφου .

Ερώτημα 3

Στο ερώτημα 1 μετράμε την συχνότητα που εμφανίζεται κάθε υποψήφιος στην λίστα χρησιμοποιώντας εμφολευμένο βρόγχο και για αυτό πετυχαίνουμε χρόνο εκτέλεσης O(ν2). Όμως αν αφαιρέσουμε τον εσωτερικό βρόγχο μπορούμε να πετύχουμε χρονο εκτέλεσης O(ν). Η ιδέα είναι να χρησιμοποιήσουμε hashmap για να μετρήσουμε και να αποθηκεύσουμε την συχνότητα εμφάνισης του κάθε υποψηφίου. Το hashmap μας επιτρέπει την αναζήτηση του υποψηφίου και την είσοδο στοιχείων σε χρόνο O(1)**.

ΑΛΓΟΡΙΘΜΟΣ 2

```
1 function MajorityFinder3(A[1...n])
2  HashMap T
3  for(i = 1 to n)
4      if(T.search(A[i]) == false)
5          T.put([A[i], 1)
6      else
7          T[A[i]] = T[A[i]] + 1
8      if(T[A[i]] > ⌈n/2⌉)
9          return A[i]
```

Ανάλυση Αλγορίθμου

- Στο βήμα 2 δημιουργούμε μία μεταβλητή τύπου hashmap όπου η αποθήκευση γίνεται σε ζευγάρια key - value όπου key είναι ο υποψήφιος και value ο αριθμός εμφάνισης του στην λίστα.
- Στα βήματα 3 - 9 διατρέχουμε την λίστα A για να αναζητήσουμε τον υποψήφιο A[i] στο hashmap.
- Συγκεκριμένα στα βήματα 4-5 η εντολή T.search αναζητεί στο hashmap τον υποψήφιο A[i] και αν δεν υπάρχει τον τοποθετεί με την εντολή T.put και τοποθετεί σαν value την συχνότητα εμφάνισης που είναι 1.
- Στα βήματα 6-7 αν ο υποψήφιος A[i] υπάρχει στο hashmap το value του συγκεκριμένου δηλαδή η συχότητα εμφάνισης του αυξάνεται κατά 1.
- Τέλος στα βήματα 8-9 ελέγχουμε αν η συχότητα εμφάνισης του υποψηφίου A[i] είναι μεγαλύτερη από 50% τότε επιστρέφουμε το όνομα αυτού του υποψηφίου.

Επειδή τόσο οι εντολές T.search και T.put για ένα hashmap τρέχουν σε O(1)** και χρησιμοποιούμε ένα βρόγχο που επαναλαμβάνεται (n) φορές, πετυχαίνουμε συνολικό χρόνο O(n).

**Για να ισχύει ότι η αναζήτηση και η τοποθέτηση του ονόματος(string) του υποψηφίου τρέχει σε χρόνο O(1) πρέπει να αποφεύγονται οι συγκρούσεις. Ο καλύτερος τρόπος για να αποφεύγονται οι συγκρούσεις είναι χρησιμοποιώντας μια καλή hash function η οποία κατανέμει τα ονόματα των υποψηφίων ομοιόμορφα στο hashmap.

Πρόβλημα 2

Ερώτημα 1

Algorithm 1

Έστω πίνακας T με στοιχεία n θετικούς ακεραίους με εύρος [0,...,k] (k ακέραιος)

```
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
```

Ανάλυση Αλγορίθμου

- Στα βήματα (1-3) δημιουργείται ο πίνακας H με k στοιχεία (δηλαδή με μέγεθος ίσο με το εύρος των αριθμών) και αρχικοποιούνται όλα τα στοιχεία του με 0.

- Στα βήματα (4-6) χρησιμοποιείται ο πίνακας H για να μετρήσει πόσες φορές εμφανίζεται κάθε αριθμός στη λίστα T. Συγκεκριμένα διατρέχει τη λίστα T και αυξάνει το μετρητή H[T[j]] κάθε φορά που συναντά ένα στοιχείο T[j].

- Στα βήματα (7-9) εκτελεί μια σωρευτική καταμέτρηση, αθροίζει δηλαδή το H[i] με το προηγούμενό του το H[i - 1] ώστε κάθε στοιχείο του πίνακα H να αποθηκεύει το άθροισμα του στοιχείου αυτού με όλα τα προηγούμενά του. Έτσι τελικά το κάθε στοιχείο H[i] δείχνει πόσοι αριθμοί προηγούνται του αριθμού i δηλαδη του T[j].

- Στα βήματα (10-13) δημιουργείται ο τελικός πίνακα S, ο οποίος θα περιέχει τα στοιχεία του πίνακα T ταξινομημένα. Συγκεκριμένα ξεκινάει από το τέλος του πίνακα T και τοποθετεί κάθε στοιχείο στη θέση που του αντιστοιχεί στον πίνακα S με βάση τον πίνακα καταμετρητών H. Στη συνέχεια μειώνει κατά 1 τον μετρητή του στοιχείου που τοποθέτησε.

Παρακάτω ακολουθεί ένα παράδειγμα με $n = 9$ και $k = 3$. Έστω ο πίνακας T:

	1	2	3	4	5	6	7	8	9
T	3	2	2	1	3	0	0	2	3

Μετά τα βήματα (1-3) ο πίνακας H είναι ο εξής:

	0	1	2	3
H	0	0	0	0

Μετά τα βήματα (4-6) ο πίνακας H έχει αποθηκεύσει πόσες φορές εμφανίζονται οι αριθμοί του πίνακα T:

	0	1	2	3
H	2	1	3	3

Μετά τα βήματα (7-9) ο πίνακας H έχει αποθηκεύσει τα σωρευτικά αθροίσματα:

	0	1	2	3
H	2	3	6	9

Μετά τα βήματα (10-13) ο πίνακας S που προκύπτει είναι ο ταξινομημένος πίνακας T:

	1	2	3	4	5	6	7	8	9
S	0	0	1	2	2	2	3	3	3

Ερώτημα 2

Εντολές	costs (c)	times (t)
1 for i = 0,...,k do	c1	k+2
2 H[i] = 0	c2	k+1
3 end for		
4 for j=1,...,n do	c3	n+1
5 H[T[j]] = H[T[j]]+1	c4	n
6 end for		
7 for i =1,...,k do	c5	k+1
8 H[i]= H[i]+H[i-1]	c6	k
9 fend for		
10 for j = n,...,1 do	c7	n+1
11 S[H[T[j]]] = T[j]	c8	n
12 H[T[j]] = H[T[j]]-1	c9	n
13 end for		

$$T(n) = \sum_{a=1}^{9} c_a \cdot t_a = c_1 \cdot (k+2) + c_2 \cdot (k+1) + c_3 \cdot (n+1) + c_4 \cdot n + c_5 \cdot (k+1) + c_6 \cdot k$$
$$+ c_7 \cdot (n+1) + c_8 \cdot n + c_9 \cdot n \implies$$

Όπου οι $c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9$ είναι κάποιες σταθερές

$$\implies T(n) = \mathcal{O}(n+k)$$