Algorithms - Assignment 1

Καζγκούτης Αθανάσιος Charbel Al Haddad
Παπαδόπουλος Δημήτριος-Λάζαρος

April 4, 2023

Πρόβλημα 1

Ερώτημα 1

Ο αλγόριθμος που παρατίθεται παρακάτω δέχεται σαν είσοδο μια συστοιχία (n) στοιχίων όπου n είναι οι ψήφοι (σε ονοματεπώνυμα) μιας κοινότητας. Στόχος του αλγορίθμου ειναι:

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

ΑΛΓΟΡΙΘΜΟΣ 1

```
1 function MajorityFinder1(A[1...n])
2  majority_person //variable to store the person who has the majority
3  maxcount = 0
4  count
5  candidate        //temp variable to store canditate
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    }
13    if(count > maxcount)
14         maxcount = count
15         majority_person = candidate
16 }
17 if(maxcount > ⌊n/2⌋)
18    return majority_person
19 else
20    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 υποψηφίου. Συγκεκριμένα η αρχική η for χρησιμοποιείται για να προσπελαστούν και τα (n) ονόματα της λίστας A, ενώ η 2^{η} ώστε για κάθε όνομα της λίστας να γίνει έλεγχος πόσες ψήφοι έχουν το ίδιο ονοματεπώνυμο με την i-στη. Η μεταβλητή count απαριθμεί τις συνολικές ψήφους που έχει ο υποψήφιος της i-στης θέσης του αρχικού πίνακα. Επίσης στη μεταβλητή candidate αποθηκέυται το ονοματεπώνυμο του υποψηφίου της i-στης ψήφου. Επειδή έχουμε εμφωλευμένο βρόγχο η πολυπολοκότητα είναι $\mathcal{O}(n^2)$. Τέλος να αναφέρουμε ότι ο ελέγχος που γίνεται στη γραμμή 10 έχει γραμμικό χρόνο εκτέλεσης με $\mathcal{O}(k)$ οπού k το μήκος της συμβολοσειράς δηλαδή του ονοματεπώνυμου του υποψηφίου, αλλά γίνεται εύκολα αντιληπτό πως η τάξη μεγέθους $n \gg k$ οπότε ο ελέγχος κάθε συμβολοσειράς θα θεωρείτε ότι έχει πολυπλοκότητα $\mathcal{O}(1)$.
- Στις γραμμες (13-15) γίνεται ελεγχός ώστε αν οι ψήφοι που έχει ο τωρινός candidate ξεπερνούν αυτές που είχε ο majority_person τότε ο candidate γίνεται ο καινούριος majority_person.

- Στις τελευταιες γραμμες (17-20) ο αλγόριθμος ελέγχει αν ο υποψήφιος που έχει τις περισσότερες ψήφους έχει επίσης και την πλειοψηφία (ποσοστό $> 50\%$).
- Ανακεφαλαιώνοντας πρόκειται για έναν ***αργό αλγόριθμο*** $\mathcal{O}(n^2)$, 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 από πολυπλοκότητα $\mathcal{O}(n^2)$ σε $\mathcal{O}(n \log n)$ θα χρησιμοποιήσουμε την στρατιγική ***διαίρει-και-κυρίευε*** (Divide-and-conquer). Συγκεκριμένα ο αλγόριθμος μας βασίζεται στην συγχωνευτική ταξινόμιση (merge sort), ώστε να ταξινομηθούν τα ονοματεπώνυμα σε μια σειρά (στη περίπτωση μας αλφαβητικά). Τέλος έχοντας μια ταξινομημένη λίστα μπορούμε πολύ ευκόλα να ελέγξουμε αν υπάρχει υποψήφιος με περισσότερο από 50% των ψήφων όπως παρουσιάζεται αναλυτικότερα παρακάτω. Ο βοηθητικός αλγοριθμος που θα χρησιμοποιήσουμε:

Merge Sort

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

6 function merge(x[1...k], y[1...l])
7  if(k = 0)
8     return y[1...l]
9  if(l = 0)
10    return x[1...l]
11 if(x[1] ≥ y[1])
12    return x[1] ∘ merge(x[2...k], y[1...l])
13 else
14    return y[1] ∘ merge(x[1...k], y[2...l])
```

Ανάλυση merge sort για την περίπτωσή μας:

- **Αναδρομή**: Διαιρούμε το πρόβλημά μας κάθε φορά στη μέση ($\frac{n}{2}$) διαδοχικά ($a = 2, b = 2$). Η αναδρομή εξαντλείτε οταν η ταξινομητέα ακολουθία έχει μήκος $l \leq 1$, δηλαδή έχει μόνο μια ψήφο με ονοματεπώνυμο που ειναι ήδη ταξινομιμένη.

- **Συγχώνευση(merge)**: Η συνάρτηση αυτή συγχωνέυει ταξινομώντας 2 ήδη ταξινομημένους πίνακες. Το βασικό βήμα της συνάρτησης εκτελείτε το πολύ (n) φορές ενώ το υπολογιστικό κοστός c είναι στη χειρότερη περίπτωση ίσο με το χρόνο που χρειάζεται να προσπελαστεί ένα ονοματεπώνυμο. Συγκεκρίμενα πάλι θεωρούμε ότι έχει γραμμικό χρόνο εκτέλεσης με $\mathcal{O}(k)$ οπού k το μήκος της συμβολοσειράς δηλαδή του ονοματεπώνυμου του υποψηφίου, αλλά γίνεται εύκολα αντιληπτό πως η τάξη μεγέθους $n \gg k$ οπότε ο ελέγχος κάθε συμβολοσειράς θα θεωρείτε ότι έχει πολυπλοκότητα $\mathcal{O}(1)$. Ουσιαστικά η σύγκριση γίνεται με τον εξής τρόπο 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 $\mathcal{O}(cn) = \mathcal{O}(n)$.

Συνολικός χρόνος: Όπως γνωρίζουμε από την θεωρία (Master Theorem) για $a = 2, b = 2$

$$T(n) = 2T(\frac{n}{2}) + f(n) = 2T(\frac{n}{2}) + \mathcal{O}(n)$$

,όπου $f(n) = \mathcal{O}(n)$ καθώς:

1. Ο χρόνος που απαιτείται για να δημιουργηθούν τα υποπροβλήματα είναι $\mathcal{O}(1)$.
2. Ο χρόνος για να γίνει συγχώνευση (merge) είναι $\mathcal{O}(n)$.

Αρα $d = 1$ οπότε επειδή $d = \log_b a$ και

$$T(n) = \left\{ \begin{array}{ll} \mathcal{O}(n^d) & \text{if } d > log_b a \\ \mathcal{O}(n^d log_b n) & \text{if } d = log_b a \\ \mathcal{O}(n^{log_b a}) & \text{if } d < log_b a \end{array} \right\}$$

Θα έχουμε: $T(n) = \mathcal{O}(n \log n)$. 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/2⌉)
4  {   if(A[i] == A[⌊n/2⌋ + i)))
5          return A[i] //end algorithm
6  }
7  return no majority person found
```

Ανάλυση Αλγορίθμου

- Στην 1^{η} γραμμή ο αλγόριθμος δέχεται τις (n) ψήφους μέσω μιας συστοιχίας "A[1...n]".
- Στη 2^{η} γραμμή καλούμε την συνάρτηση mergesort για να ταξινομήσουμε τον πίνακα Α δηλαδή τα ονοματεπώνυμα των υποψηφίων κατά αλφαβητική σειρά σε $\mathcal{O}(n \log n)$.
- Στις επόμενες γραμμές ο αλγόριθμος εξετάζει ($\lceil \frac{n}{2} \rceil$) φορές για διαστήματα με μήκος $\lfloor \frac{n}{2} \rfloor + 1^{**}$ αν είναι ο ίδιος υποψήφιος στο πρώτο και στο τελευταίο στοιχείο του διαστήματος αναζήτησης, εκμεταλευόμενοι το γεγονός οτι ο πίνακας Α είναι ταξινομημένος. Η διαδικασία τερματίζει είτε όταν βρεθεί υποψήφιος με πλειοψηφία είτε όταν θά έχουν προσπελαστεί όλα τα δυνατά διαστήματα χώρις κάποιο επιθυμητό αποτέλεσμα. Η συγκεκριμένη διαδικασία είναι γραμμική και έχει πολυπλοκότητα $\mathcal{O}(cn) = \mathcal{O}(n)$. Αφού όπως και στο ερώτημα 1 ο ελέγχος που γίνεται για σύγκριση 2 συμβολοσειρών έχει γραμμικό χρόνο εκτέλεσης $\mathcal{O}(k)$ οπού k το μήκος της συμβολοσειράς δηλαδή του ονοματεπώνυμου του υποψηφίου, αλλά λόγω της τάξης μεγέθους $n \gg k$ ο ελέγχος για την ισότητα των συμβολοσειρών θα θεωρείτε ότι έχει πολυπλοκότητα $\mathcal{O}(1)$.
- Παρακάτω παρουσιάζεται ένα παράδειγμα για καλύτερη κατανόηση της διαδικασίας. Έστω ότι έχουμε 2 πίνακες 10 και 11 στοιχείων αντίστοιχα ώστε να δούμε τις περιπτώσεις που το (n) μπορεί να είναι είτε άρτιος είτε περιττός, όπου είναι αποθηκευμένοι ψήφοι με 2 μόνο γράμματα για χάριν απλότητας. Το πρώτο γράμμα συμβολίζει το όνομα και το δεύτερο το επίθετο.

	1	2	3	4	5	6	7	8	9	10
ΑΡΤΙΟΣ Α	ΔΠ	ΔΠ	ΑΚ	ΑΒ	ΧΠ	ΧΠ	ΑΚ	ΔΠ	ΑΒ	ΑΚ

	1	2	3	4	5	6	7	8	9	10	11
ΠΕΡΙΤΤΟΣ Π	ΧΠ	ΑΒ	ΧΠ	ΧΠ	ΑΚ	ΧΠ	ΧΠ	ΔΠ	ΧΠ	ΑΒ	ΑΚ

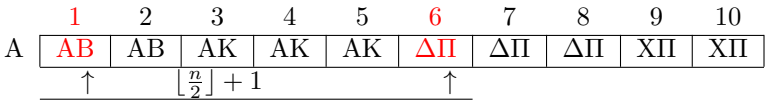
Με την κλήση της merge sort οι πίνακες ταξινομούνται ως εξής:

	1	2	3	4	5	6	7	8	9	10
Α	ΑΒ	ΑΒ	ΑΚ	ΑΚ	ΑΚ	ΔΠ	ΔΠ	ΔΠ	ΧΠ	ΧΠ

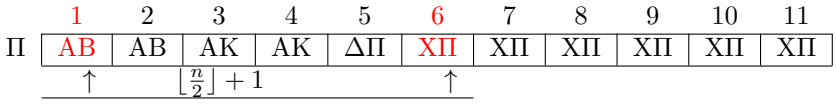
	1	2	3	4	5	6	7	8	9	10	11
Π	ΑΒ	ΑΒ	ΑΚ	ΑΚ	ΔΠ	ΧΠ	ΧΠ	ΧΠ	ΧΠ	ΧΠ	ΧΠ

Στη συνέχεια ξεκινάει ο έλεγχος για διαστήματα με μήκος $\lfloor \frac{n}{2} \rfloor + 1$:

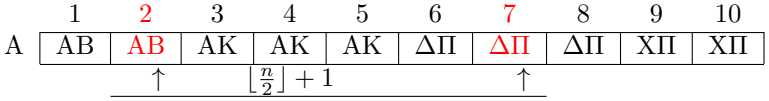
Για $i = 1$



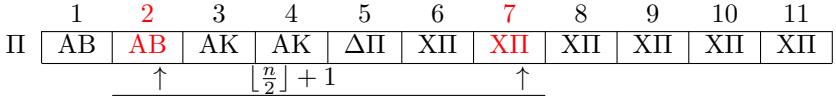
$AB \neq \Delta\Pi$



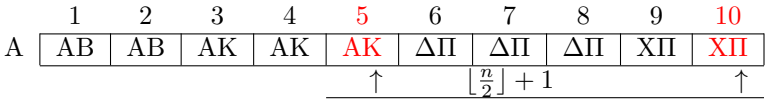
$AB \neq X\Pi$
Τώρα για $i = 2$:



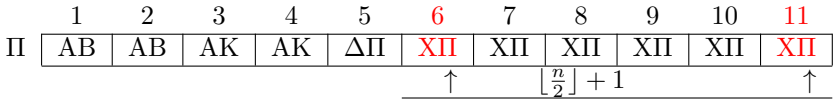
$AB \neq \Delta\Pi$



$AB \neq X\Pi$
Όμοια για $i = 3$ και $i = 4$
Τελικά για $i = 5$ για τον άρτιο πίνακα



$AK \neq X\Pi$ Αρα κανένας υποψήφιος του πίνακα Α δεν έχει πλειοψηφία μεγαλύτερη από 50%.
Ο πίνακας Π θα έχει μία ακόμα επανάληψη για $i = 6$ γιατί χρησιμοποιούμε άνω όριο στον βρόγχο επανάληψης ($\lceil \frac{n}{2} \rceil$):



$X\Pi = X\Pi$ Αρα στο παράδειγμα μας ο ΧΠ έχει πλειοψηφία πάνω απο 50% στον περιττό πίνακα.

- Ανακεφαλαιώνοντας ο αλγόριθμος αυτός σαφώς εμφανίζεται πιο βελτιωμένος από τον πρώτο καθώς έχουμε ελλατώσει τις συγκρίσεις σε μεγάλο βαθμό με την βοήθεια του devide and conquer αλλά και πάλι εμφανίζονται αρκετές επαναλήψεις ανάγνωσης ψήφου.
Συνολικά η πολυπλοκότητα είναι $\mathcal{O}(nlogn + n) = \mathcal{O}(nlogn)$

**Χρησιμοποιούμε κάτω οριο ($\lfloor\ \rfloor$) ώστε να συμπεριλάβουμε τις περιπτώσεις για άρτιο και περιτττό (n) σε μια έκφραση.

Ερώτημα 3

Στο ερώτημα 1 μετράμε την συχνότητα που εμφανίζεται κάθε υποψήφιος στην λίστα χρησιμοποιώντας εμφολευμένο βρόγχο και για αυτό πετυχαίνουμε χρόνο εκτέλεσης $\mathcal{O}(n^2)$. Όμως αν αφαιρέσουμε τον εσωτερικό βρόγχο μπορούμε να πετύχουμε χρονο εκτέλεσης $\mathcal{O}(n)$. 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 $\mathcal{O}(1)$**.

ΑΛΓΟΡΙΘΜΟΣ 3

```
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 $\mathcal{O}(1)$** και χρησιμοποιούμε ένα βρόγχο που επαναλαμβάνεται (n) φορές, πετυχαίνουμε συνολικό χρόνο $\mathcal{O}(n)$.

**Για να ισχύει ότι η αναζήτηση και η τοποθέτηση του ονόματος(string) του υποψηφίου, τρέχει σε χρόνο $\mathcal{O}(1)$ πρέπει να αποφεύγονται οι συγκρούσεις. Ο καλύτερος τρόπος για να αποφεύγονται οι συγκρούσεις είναι χρησιμοποιώντας μια καλή hash function η οποία να κατανέμει τα ονόματα των υποψηφίων ομοιόμορφα στο hashmap. Σε περίπτωση συγκρούσεων θα γίνεται η διαδικασία του κατακερματισμού.

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		

Ο συνολικός χρόνος που απαιτείται προκύπτει απο το άθροισμα των επιμέρους χρόνων κάθε εντολής:

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

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

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