Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή ΗΜ&ΜΥ
Αλγόριθμοι και Πολυπλοκότητα
7ο εξάμηνο
Ακαδημαϊκή περίοδος: 2015-2016

1η Σειρά Γραπτών Ασκήσεων

Ψαρουδάκη Ελένη
Α.Μ.: 03112080

28 Οκτωβρίου 2015

Άσκηση 1:Ασυμπτωτικός Συμβολισμός, Αναδρομικές Σχέσεις

(α)

Μας ζητήθηκε να ταξινομήσουμε σε αύξουσα σειρά τάξης μεγέθους κάποιες συναρτήσεις. Προκειμένου να γίνει η διάταξη λάβαμε υπόψιν την διάταξη που υπάρχει στις διαφάνειες η οποία είναι ως εξής

$$\mathcal{O}(1) \subset \mathcal{O}(\log^* n) \subset \mathcal{O}(\log n) \subset$$
$$\subset \mathcal{O}(\text{poly}(\log n)) \subset \mathcal{O}\sqrt{n} \subset \mathcal{O}(n) \subset \mathcal{O}(n \log n) \subset$$
$$\subset \mathcal{O}(poly(n)) \subset \mathcal{O}(n^{\log n}) \subset \mathcal{O}(2^n) \subset \mathcal{O}(3^n) \subset$$
$$\subset \mathcal{O}(n!) \subset \mathcal{O}(n^n) \subset \mathcal{O}(A(n))$$

Επίσης κάτι που μάθαμε στο μάθημα είναι ότι όταν δυσκολευόμαστε να βρούμε την τάξη μιας συνάρτησης μπορουμε να λογαριθμήσουμε την συνάρτηση λαμβάνοντας υπόψιν ότι πλέον οι σταθερές "μετράνε". Με βάση αυτά έχουμε:

1. $$\sum_{k=1}^{n} i2^{-k} = 2^{-n}(-n + 2^{n+1} - 2) = \mathcal{O}(1)$$
   παίρνοντας το όριο στο άπειρο

2. $$\log \binom{n}{logn} = \Theta(\log^2 n)$$
   διότι
   $$(n/k)^k \leq \binom{n}{k} \leq (ne/k)^k$$
   και εφαρμόζουμε την προηγούμενη σχέση για $k = logn$ και με λογαρίθμιση προκύπτει το ζητούμενο.

3. $$\log^4 n = o(\sqrt{n})$$
   διότι
   $$\forall \epsilon > 0 : \log^d n = o(n^\epsilon)$$
   και εφαρμόζουμε την προηγούμενη σχέση για
   $$\epsilon = \frac{1}{2}$$

4. $$\frac{\log(n!)}{\log^3 n} = \Theta(\frac{n}{log^2 n})$$
   διότι
   $$\log(n!) = \Theta(n \log n)$$

5. (α')
   $$\log(\binom{2n}{n}) \leq \log(\frac{2n^{2n}}{n^n(2n-n)^{2n-n}}) = \log 2^{2n} = \log 4^n = \Theta(n)$$

διότι ισχύει η ταυτότητα

$$\binom{n}{k} \leq \frac{n^n}{k^k(n-k)^{n-k}}$$

(μπορεί να αποδειχθεί με επαγωγή)

(β′)

$$n2^{2^{2^{100}}} = \mathcal{O}(n)$$

6.

$$n^2 = \mathcal{O}(n^2)$$

7.

$$\frac{n^3}{\log^{10} n} \leq \frac{n^3}{\sqrt{n}} = \Theta(n^{\frac{5}{2}})$$

8.

$$\sum_{k=1}^{n} k^3 = \Theta(n^4)$$

9.

$$\binom{n}{6} = \frac{n!}{6!(n-6)!} = \mathcal{O}(n^6)$$

10.

$$(\log 16n)^{\log 16n} = \Theta((16n)^{\log\log 16n})$$

διότι ισχύει η ταυτότητα

$$a^{\log_b c} = c^{\log_b a}$$

11.

$$2^{logn^3} = (2^{logn})^{lognlogn} = \mathcal{O}n^{log^2 n}$$

12. (α′)

$$n\sum_{k=0}^{n} \binom{n}{k} = \mathcal{O}(n2^n)$$

(β′)

$$\sum_{k=1}^{n} k2^k = 2(1 + (n-1)2^n) = \mathcal{O}(n2^n)$$

13.

$$\binom{2n}{n/4} \leq \frac{2n^{2n}}{(n/4)^{n/4}(2n-n/4)^{2n-n/4}} = (\frac{2^2}{(3/4)^{3/4}(1/4)^{1/4}})^n = \mathcal{O}(7.1^n)$$

διότι ισχύει η ταυτότητα

$$\binom{n}{k} \leq \frac{n^n}{k^k(n-k)^{n-k}}$$

(μπορεί να αποδειχθεί με επαγωγή)

14.

$$\sqrt{n!} = \Theta(\sqrt{n!})$$

διότι

$$\mathrm{f(n)} = \mathrm{o(g(n))} \Leftrightarrow \sqrt{\mathrm{f(n)}} = \mathrm{o}(\sqrt{\mathrm{g(n)}})$$

(b)

1. $$T(n) = 3T(\frac{n}{4}) + n\log^5 n$$

   Παρατηρούμε ότι
   $$n^{\log_b a} = n^{\log_4 3} = n^{0.7} < n\log^5 n$$
   άρα βρισκόμαστε στην τρίτη περίπτωση του Master Theorem. Επίσης η συνθήκη
   $$af(\frac{n}{b}) < f(n)$$
   ισχύει για μεγάλες τιμές του n ασυμπτωτικά άρα
   $$T(n) = \Theta(n\log^5 n)$$

2. $$T(n) = 4T(\frac{n}{4}) + n\log^5 n$$

   Παρατηρούμε ότι
   $$n^{\log_b a} = n^{\log_4 4} = n$$
   Όμως η συνάρτηση $n\log^5 n$ δεν είναι πολυωνυμικά μεγαλύτερη από την συνάρτηση n επομένως το Master Theorem δεν εφαρμόζεται. Άρα θα πρέπει να καταφύγουμε σε δέντρο αναδρομών. Το δέντρο έχει ύψος $\log_4 n$ επειδή το μέγεθος κάθε υποπροβλήματος για κάθε κόμβο του δέντρου σε βάθος i είναι ίσο με $\frac{n}{4^i}$. Σε κάθε επίπεδο του δέντρου έχουμε 4 καινούριους κόμβους. Αναπτύσσοντας λίγο την αναδρομική σχέση βλέπουμε ότι η πολυπλότητά της δίνεται από το άθροισμα
   $$\sum_{i=0}^{\log_4 n-1} n\log\frac{n}{4^i} = \Theta(n\log^6 n)$$

3. $$T(n) = 5T(\frac{n}{4}) + n\log^5 n$$

   Παρατηρούμε ότι
   $$n^{\log_b a} = n^{\log_4 5} = n^{1.16} < n\log^5 n$$
   άρα βρισκόμαστε στην πρώτη περίπτωση του Master Theorem. Επομένως
   $$T(n) = \Theta(n^{\log_4 5})$$

4. $$T(n) = 2T(\frac{n}{3}) + \frac{n}{\log^5 n}$$

   Παρατηρούμε ότι
   $$n^{\log_b a} = n^{\log_3 2} = n^{0.63} < \frac{n}{\log^5 n}$$
   άρα βρισκόμαστε στην τρίτη περίπτωση του Master Theorem. Επίσης η συνθήκη
   $$af(\frac{n}{b}) < f(n)$$
   ισχύει για μεγάλες τιμές του n ασυμπτωτικά άρα
   $$T(n) = \Theta(\frac{n}{\log^5 n})$$

5.

$$T(n) = 2T(\frac{n}{7}) + 4T(\frac{n}{9}) + n$$

Βλέπουμε ότι δεν μπορεί να εφαρμοστεί καμία περίπτωση του Master Theorem, οπότε πρέπει να καταφύγουμε σε μια καλή μαντεψιά ή να κάνουμε δέντρο αναδρομών. Παρατηρώντας την αναδρομική σχέση βλέπουμε ότι το φορτίο των αναδρομικών κλήσεων ολοένα και μειώνεται και φαίνεται να κυριαρχεί ο σταθερός όρος n. Για να δείξουμε ότι ο αλγόριθμος είναι $\mathcal{O}(n)$ θα πρέπει να αποδείξουμε ότι $T(k) < ck$ για κάθε $k < n$ για κάποιο c. Ισχύει ότι

$$T(n) < \frac{2cn}{7} + \frac{4cn}{9} + n = (1 + \frac{46c}{63})n \leq cn$$

αν ισχύει ότι $c > \frac{63}{17}$. Επίσης είναι προφανές ότι $T(n) = \Omega(n)$ διότι

$$T(n) = T(\frac{n}{6}) + 3T(\frac{n}{5}) + n \geq n$$

Άρα $T(n) = \Theta(n)$.

6.

$$T(n) = T(n-1) + \log n$$

Βλέπουμε ότι η παραπάνω αναδρομική σχέση περιγράφει ένα αλγόριθμο που μειώνει λίγο το αρχικό πρόβλημα και στη συνέχεια κάνει χρόνο $\log n$ προκειμένου να λύσει το κάθε υποπρόβλημα. Άρα υποψιαζόμαστε χρόνο $\Theta(n \log n)$. Επομένως πρέπει να αποδείξουμε ότι $T(n) = \mathcal{O}(n \log n)$ και $T(n) = \Omega(n \log n)$. Αρχικά θα αποδείξουμε το πάνω φράγμα. Έστω ότι $T(n) \leq cn \log n$ για κάποια σταθερά c. Ισχύει ότι:

$$\begin{aligned}
T(n) &= T(n-1) + \log n \\
&\leq c(n-1)\log(n-1) + \log n \\
&= cn\log(n-1) - c\log(n-1) + \log n \\
&\leq cn\log(n-1) - c\log(\frac{n}{2}) + \log n \\
&(\log(n-1) \geq \log(\frac{n}{2}) \text{ for } n \geq 2) \\
&= cn\log(n-1) - c\log n + c + \log n \\
&\leq cn\log n - c\log n + c + \log n \\
&\leq cn\log n
\end{aligned}$$

Η παραπάνω σχέση ισχύει αν

$$\begin{aligned}
-c\log n + c + \log n &\leq 0 \\
c &\leq \log n(c-1) \\
\log n &\geq \frac{c}{c-1}
\end{aligned}$$

που ισχύει για $c = 2$ και $n \geq 4$. Τώρα πρέπει να δείξουμε ότι $T(n) = \Omega(n \log n)$. Έστω ότι $T(n) \geq cn \log n + dn$ για κάποιες σταθερέςc, d.Ισχύει ότι:

$$\begin{aligned}
T(n) &= T(n-1) + \log n \\
&\geq c(n-1)\log(n-1) + d(n-1) + \log n \\
&= cn\log(n-1) - c\log(n-1) + dn - d + \log n \\
&\geq cn\log(\frac{n}{2}) - c\log(n-1) + dn - d + \log n \\
&(\log(n-1) \geq \log(\frac{n}{2}) \text{ for } n \geq 2) \\
&= cn\log n - cn - c\log(n-1) + dn - d + \log n \\
&\geq cn\log n
\end{aligned}$$

Η παραπάνω σχέση ισχύει αν

$$-cn - c\log(n-1) + dn - d + \log n \geq 0$$

Όμως

$$-cn - c\log(n-1) + dn - d + \log n > \\ > -cn - c\log(n-1) + dn - d + \log(n-1)$$

Επομένως αρκεί να βρούμε σταθερές c,d για τις οποίες να ισχύει

$$-cn - c\log(n-1) + dn - d + log(n-1) \geq 0 \\ \Leftrightarrow -cn - c\log(n-1) + dn - d + \log(n-1) \geq 0 \\ (d-c)n \geq (c-1)\log(n-1) + d$$

που ισχύει για $c = 1, d = 2$ και $n2$. Εφόσον αποδείξαμε και το πάνω αλλά και το κάτω φράγμα έχουμε ότι

$$T(n) = \Theta(n \log n)$$

7.

$$T(n) = 2T(\sqrt{n}) + \Theta(\log n)$$

Για τη συγκεκριμένη σχέση θα εφαρμόσουμε αλλαγή μεταβλητών προκειμένου να την φέρουμε σε μια πιο εύκολη μορφή ώστε να εφαρμόσουμε το Master Theorem. Θέτουμε $m = \log n$ και η παραπάνω σχέση μετασχηματίζεται σε

$$T(2^m) = 2T(2^{\frac{m}{2}}) + m$$

Στη συνέχεια η αναδρομική σχέση γράφεται ως εξής:

$$S(m) = 2S(\frac{m}{2}) + \Theta(m),$$

έχοντας θέσει $S(m) = T(2^m)$. Εφαρμόζοντας το Master Theorem παρατηρούμε ότι

$$m^{log_b a} = \Theta(m)$$

άρα είμαστε στη δεύτερη περίπτωση και επομένως

$$S(m) = m \log m$$

και με πίσω αντικατάσταση έχουμε ότι

$$T(n) = \Theta(\log n \log\log n)$$

8.

$$T(n) = T(n/4) + \sqrt{n}$$

Παρατηρούμε ότι

$$n^{\log_b a} = n^{\log_4 1} = 1 < \sqrt{n}$$

άρα βρισκόμαστε στην τρίτη περίπτωση του Master Theorem. Επίσης η συνθήκη

$$af(\frac{n}{b}) < f(n)$$

ισχύει για μεγάλες τιμές του n ασυμπτωτικά άρα

$$T(n) = \Theta(\sqrt{n})$$

Άσκηση 2:Ταξινόμηση

(α)

1. Θα χρησιμοποιήσουμε την ιδέα της quicksort. Ωστόσο καθώς δεν θέλουμε να ταξινομίσουμε πλήρως το δέντρο αλλά να τον ταξινομήσουμε σε k μέρη, η αναδρομική διαδικασία θα έχει κόστος / ύψος δέντρου $\log k$. Ως pivot θα παίρνουμε κάθε φορά το μεσσαίο στοιχείο. Για την εύρεση του μεσαίου θα χρησιμοποιήσουμε την quickselect με κόστος $\mathcal{O}(n)$. Για κάθε επίπεδο γίνονται επιπλέον n συγκρίσεις για την αντιμετάθεση των στοιχείων($2^i \cdot \frac{n}{2^i}$). Επομένως θα έχουμε συνολικό κόστος $\mathcal{O}(n \log k)$).

   Ο αλγόριθμος μας είναι βέλτιστος καθώς ο χρόνος εκτέλεσης της χειρότερης περίπτωσης είναι $\Omega(n \log k)$. Αυτό συμβαίνει διότι για την εύρεση του μεσαίου στοιχείου απαιτείται χρόνος τουλάχιστον $\Omega(\frac{n}{2^i})$, οπού i το επίπεδο και άρα $\Omega(n)$ για κάθε επίπεδο. Επομένως καθώς το ύψος του δέντρου δεν αλλάζει απατείται $\Omega(n \log k)$.

2. Το πρόβλημα μας ισοδυναμεί με την ταξινόμηση k πινάκων χωριστά. Για την ταξινόμηση κάθε πίνακα $\frac{n}{k}$ στοιχείων, χρησιμοποιώ quicksort με κόστος $\mathcal{O}(\frac{n}{k} \log \frac{n}{k})$. Επομένως για τους k πίνακες θα έχω $\mathcal{O}(n \log \frac{n}{k})$.

   Κάθε στοιχείο έχει $\frac{n}{k}$ πιθανές θέσεις άρα στην χειρότερη περίπτωση απαιτούνται $\log \frac{n}{k}$ συγκρίσεις. Επομένως για κάθε υποπίνακα έχω κόστος $\frac{n}{k} \log \frac{n}{k}$ και έτσι προκύπτει πως στην χειρότερη έχω για τον χρόνο εκτέλεσης $\Omega(k \log \frac{n}{k})$.

(β)

1. Έχουμε ένα πίνακα με n στοιχεία. Από αυτά το κάθε στοιχείο βρίσκεται σε απόσταση το πολύ k από την σωστή του θέση. Επομένως υπάρχουν $2k$ πιθανές θέσεις για το κάθε στοιχείο. Αρχικά παίρνουμε τα πρώτα $2k$ στοιχεία του πίνακα και τα ταξινομούμε. Ξέρουμε πως τα πρώτα k στοιχεία θα βρίσκονται στις τελικές τους θέσεις. Ταξινομούμε τώρα τα στοιχεία $k+1$ έως $3k$. Πλέον τα στοιχεία $k+1$ έως $2k$ θα βρίσκονται στις τελικές τους θέσεις. Συνεχίζουμε την διαδικασία μέχρι να ταξινομήσουμε και τα τελευταία στοιχεία, που μπορεί να είναι λιγότερα από $2k$.

   Σε κάθε βήμα ταξινομούνται $2k$ στοιχεία (με εξαίρεση ίσως το τελευταίο βήμα) με κόστος $\mathcal{O}(k \log k)$ και τουλάχιστον k από αυτά θα βρίσκονται στην τελική τους θέση. Η συγκεκριμένη διαδικασία επαναλαμβάνεται n/k φορές, επομένως το συνολικό κόστος θα είναι $\mathcal{O}(n \log k)$.

2. Αν θεωρήσουμε πως το κάθε στοιχείο έχει 2k πιθανές θέσεις και έχουμε n στοιχεία τότε πρέπει στην χειρότερη περίπτωση να κάνουμε $\log k$ συγκρίσεις για το κάθε στοιχείο που ισοδυναμεί με πολυπλοκότητα τουλάχιστον $\Omega(n \log k)$.

Άσκηση 3: Αναζήτηση

(α)

Για αρχή θεωρούμε πως δεν υπάρχει περιορισμός στον αριθμό των φίλτρων από ταοποία μπορεί να πιει ο κάθε εθελοντής, καθώς πολύ μικρή ποσότητα από το κάθε φίλτρο οδηγεί στην εμφάνιση υπερδυνάμεων.

Για τη λύση του ζητούμενου προβλήματος θα χρησιμοποιήσουμε δυαδική αναπαράσταση. Αριθμούμε κάθε φίλτρο με έναν ακέραιο από το δυαδικό 1 (0000 0000 0000 0000 0001) μέχρι και το δυαδικό 1000000 (1111 0100 0010 0100 0000). Επομένως για την αναπαράσταση όλων των φίλτρων είνι απαραίτητα $\log_2(1000000) = 20$ bits. Θα αντιστοιχίσουμε σε κάθε bit έναν εθελοντή. Ο καθένας από αυτούς θα πίνει απο το ή όχι από

το κάθε φίλτρο ανάλογα με το αν η δυαδική του αναπαράσταση έχει στο αντίστοιχο bit του εθελοντή 1. Έτσι ο συνδιασμός των εθελοντών που θα εμφανίζουν υπερδυνάμεις θα δείχει τη δυαδική απεικόνιση της αρίθμησης του φίλτρου.

Επομένως, σύμφωνα με τη λύση μας, ο ελάχιστος αριθμός εθελοντών που απαιτείται είναι 20 και ο καθένας θα αξιοποιείται όπως περιγράφηκε παραπάνω.

(b)

Για την εύρεση τη ελάχιστης μέγιστης απόστασης που θα διανύσουμε σε μία μέρα, θα εκτελέσουμε δυαδική αναζήτηση. Απαραίτητη είναι λοιπόν η εύρεση άνω και κατω ορίων. Για το κάτω όριο θα βρούμε τη μέγιστη απόσταση και τον μέσο όρο των αποστάσεων που θα διανύσουμε σε μία μέρα και από αυτά θα κρατήσουμε το μέγιστο. Για το άνω όριο βρίσκουμε το άθροισμα όλων των αποστάσεων. Η εύρεση των ορίων έχει κόστος $\Theta(n)$. Ωστόσο αυτή η χρονική πολυπλοκότητα δεν επηρεάζει τη συνολική πολυπλοκότητου αλγορίθμου, καθώς όπως θα δούμε στη συνέχεια η δυαδική αναζήτηση έχει μεγαλύτερη πολυπλοκότητα.

Στη συνεχεια αναζητούμε τη ελάχιστη μέγιστη απόσταση l που μπορούμε να διανύσουμε σε μια μέρα. Για την εκάστοτε πιθανή απόσταση l εκτελούμε την παρακάτω συνάρτηση.

Algorithm 1 Function F_S

```
1: function Function F_S(l)
2:     Sum ← 0
3:     days ← 1
4:     for i ← 1 to n do
5:         if (Sum + d[i]) < l then
6:             Sum ← Sum + d[i]
7:         else
8:             Sum ← d[i]
9:             days ← days + 1
```

Μετά την εκτέλεση της παραπάνω συνάρτησης συγκρίνουμε το αποτέλεσμα της μεταβλητής $days$ με το επιθυμητό αριθμό ημερών ταξιδιού k.

- Αν το k είναι μικρότερο τότε συνεχίζουμε την αναζήτηση με νέο άνω όριο την τιμή της μεταβλητής $days$.
- Αν το k είναι μεγαλύτερο τότε συνεχίζουμε την αναζήτηση με νέο κάτω όριο την τιμή της μεταβλητής $days + 1$

Ο συνολικός χρόνος εκτέλεσης ισούται με τον αριθμό των επαναλήψεων επί το χρόνο εκτέλεσης της κάθε επανάληψης. Σε κάθε επανάληψη/κλήση της συνάρτησης έχουμε κόστος/χρόνο εκτέλεσης $\Theta(n)$. Ο αριθμός των επαναλήψεων ισούται με $\log_2 upper\_limit - lower\_limit$. Το $upper\_limit - lower\_limit$ είναι άνω φραγμένο από το άθροισμα των αποστάσεων. Επομένως, η χρονική πολυπλοκότητα του αλγορίθμου μας θα είναι:

$$O(n \log_2 \sum_{i=1}^{n} d_i)$$

Άσκηση 4: Απόλυτη Πλειοψηφία και Ισχυρή Πληοψηφία

(α)

Στην άσκηση μας ενδιαφέρει για αρχή να διαπιστώσουμε αν υπάρχει ή όχι απόλυτη πλειοψηφία. Ύπαρξης απόλυτης πλειοψηφίας ισοδυναμεί με $n/2 + 1$ βουλευτές για το

κόμμα με την πλειοψηφία. Θα χρησιμοποιήσουμε την παρακάτω διαδικασία. Χωρίζουμε όλους τους βουλευτές σε ζευγάρια(Ο πρώτος με το δεύτερο, ο τρίτος με τον τέταρτο κλπ). Αν το n είναι περιττός αριθμός θα περισσέψει ο τελευταίος βουλευτής. Ζητάμε από τους δύο βουλευτές να χαιρετηθούν. Αν ανήκουν στο ίδιου κόμμα κρατάμε τη θεση του πρώτου εκ των δύο βουλευτών. Αν ανήκουν σε διαφορετικό δεν κρατάμε τη θέση κανενός. Αν ο αριθμός των ζευγαριών είναι μονός κρατάμε και τη θέση του τελευταίου βουλευτή. Επαναλαμβάνουμε τη συγκεκριμένη διαδικασία αποκλεισμού μέχρι να μείνει μόνο ένας βουλευτής.

Στην παραπάνω διαδικασία στηριζόμαστε στο γεγονός πως αν υπάρχει πλειοψηφία ο βουλευτής που θα μείνει στην διαδικασία αποκλεισμού θα πρέπει να ανήκει στο ζητούμενο κόμμα. Αν υπάρχει πλοψηφία, θα πρέπει να συνεχίσει να υπάρχει και μετά την διαδικασία του αποκλεισμού. Αυτό μπορεί να δικαιολογηθεί καθώς αν από τη σύγκριση προκύψει πως οι δύο βουλευτές ανήκουν σε διαφορετικό κόμμα μόνο ένας από αυτούς θα μπορούσε να ανήκει στο κόμμα της πλειοψηφίας. Σε κάθε κλήση της διαδικασίας έχουμε το πολύ x/2 βουλευτές σε σχέση με την προηγούμενη κλήση. Όταν μείνει μόνο ένας βουλευτής (έστω Β), αυτός θα ανήκει στην πλειοψηφία, αν αυτή υπάρχει.

Στη συνέχεια ελέγχουμε κάθε βουλευτή με εξαίρεση αυτόν που μας έμεινε με τον Β+1 βουλευτή που ήταν το αρχικό του ζευγάρι (έχουμε κρατήσει την θέση του Β). Ελέγχουμε με τον Β+1 έτσι ώστε να μην αναγκάσουμε ένα ζευγάρι βουλευτών να χαιρετηθεί παραπάνω από μία φορές. Αν οι δύο βουλευτές ανήκουν στο ίδιο κόμμα, προσθέτουμε ένα στον μετρητή, ο οποίος έχει αρχικοποιηθεί στο 2. Αν το τελικό νούμερο είναι μεγαλύτερο του n/2, τότε υπάρχει πλειοψηφία και σε αυτή ανήκουν οι βουλευτές που αν χαιρετηθούν με τον Β+1 χαμογελούν.

Στην ειδική περίπτωση που ο βουλευτής Β είναι και ο τελευταίος, ελέγχουμε όλους τους άλλους μαζί του, καθώς αυτό σημαίνει πως δεν έχει χαιρετηθεί με κανέναν.

Αυτός ο αλγόριθμος είναι την τάξης $O(n)$, καθώς κάθε φορά απαιτούνται για τους αποκλισμούς βουλευτών $n/2+n/4+...$ συγκρίσεις και $n-2$ συγκρίσεις για τον έλεγχο και τον εντοπισμό των βουλευτών στην περίπτωση πλειοψηφίας. Επομένως απαιτούνται το πολύ $2n$ συγκρίσεις.

(β)

Ξέρουμε πως αν βρισκόμαστε στη δεύτερη περίπτωση, η πιθανότητα να διαλέξουμε 0 είναι μεγαλύτερη του $p = 2/3$, ενώ η πιθανότητα να διαλέξουμε 1 είναι μικρότερη του $(1 - p =)1/3$. Πιθανότητα λάθους υπάρχει αν διαλέξουμε τυχαία συνεχόμενα μηδενικά στοιχεία, ενώ βρισκόμαστε στη δεύτερη περίπτωση. Αυτή ισούται με

$$p \cdot p \cdot p \cdot .... \leq 0.1 \rightarrow (2/3)^x \leq 0.1 \rightarrow x \geq 6$$

Επομένως αρκεί να επιλέξουμε 6 στοιχεία από τον πίνακα με τυχαίο τρόπο. Αυτό θα μας δώσει πιθανότητα λάθους μικρότερη του 10% με σταθερό χρόνο 6 ελέγχων, δηλαδή $O(1)$.

Ο ελάχιστος χρόνος εκτέλεσης χειρότερης περίπτωσης ενός ντετερμινιστικού αλγορίθμου είναι n, καθώς αν βρισκόμαστε στην περίπτωση που όλα τα στοιχεία είναι μηδενικά, πρέπει να ελέγξουμε όλα τα στοιχεία για απαντήσουμε με βεβαιότητα.

Άσκηση 5: Πολυκατοικίες χωρίς Θέα

Algorithm 2 Πολυκατοικίες χωρίς θέα

```
1: B[1] ← 0
2: for i ← 2 to n do
3:     if A[i] < A[i − 1] then
4:         B[i] ← i − 1
5:     else
6:         k ← B[i − 1]
7:         while (k! = 0) or (A[i] >= A[k]) do
8:             k ← B[i − 1]
9:         B[i] ← k
```

Με την πρώτη ματιά ο αλγόριθμος φαίνεται να είναι της τάξης $O(n^2)$. Θα αποδείξουμε ότι είναι επίσης και $O(n)$.

Σύμφωνα με την εκφώνηση στον πίνακα B αποθηκεύεται κάθε φορά η θέση του κτηρίου που θα πυροβολήσουν οι κάτοικοι της πολυκατοικίας. Συγκρίνουμε λοιπόν το ύψος του κάθε κτηρίου (i) με αυτό του αμέσως προηγούμενου $(i-1)$. Αν το εκάστοτε κτήριο i είναι πιο χαμηλό, τότε οι κάτοικοι θα πυροβολύσουν την προηγούμενη πολυκατοικία και το $B[i]$ μπορεί να συμπληρωθεί με σταθερό κόστος μίας σύγκρισης.

Διαφορετικά το κτήριο i είναι πιο ψηλό από το αμέσως προηγούμενο του. Αν το $i-1$ ήταν το ψηλότερο μέχρι τότε, δηλαδή $[i-1]=0$, τότε πάλι μπορούμε να συμπληρώσουμε το $B[i]=0$ με σταθερό κόστος. Αν όχι γίνεται η σύγκριση μεταξύ του i κτηρίου και του κτηρίου που θα πυροβολήσουν οι κάτοικοι του $i-1$, δηλαδή του $B(i-1)$ κτηρίου. Έστω ότι αυτό είναι η πολυκατοικία k. Αν το κτήριο k είναι ψηλότερο από το i τότε έχουμε βρεί τη ζητούμενη πολυκατοικία. Αν όχι, επαναλαμβάνουμε την παραπάνω διαδικασία, τοποθετώντας κάθε φορά στο k το (k) μέχρι να πληρείται κάποια από τις δύο προυποθέσεις.

Το πρόβλημα συνεχίζει να φαίνεται $O(n^2)$. Αν κάνουμε όμως μία εκτενέστερη διερεύνηση παρατηρούμε τα παρακάτω:

- Δεν χρειάζεται κάθε φορά να ελέγχουμε όλα τα προηγούμενα στοιχεία, πράγμα που θα καθιστούσε τον αλγόριθμο της τάξης του (n^2)
- Κατά τη διάρκεια εκτέλεσης του αλγορίθμου θα γίνουν το πολύ $2n$ συγκρίσεις μεταξύ κτηρίων.

Το κόστος προέρχεται από τις συγκρίσεις μεταξύ των κτηρίων. Έστω ότι βρισκόμαστε στο i βήμα. Γίνονται x συγκρίσεις και εμείς θέλουμε να μοιράσουμε το κόστος των συγκρίσεων αυτών στα στοιχεία του πίνακα. Ξεκινάμε χρεώνοντας το στοιχείο i και όλα τα ενδιάμεσα σημεία της διαδρομής. Δηλαδή αν γίνει μόνο μία σύγκριση, χρεώνεται μόνο το i στοιχείο, ενώ αν γίνουν περισσότερες, το i,$i-1$ κλπ, με εξαίρεση το τελικό k, χρεώνουμε ένα για καθε σύγκριση. Ένα στοιχείο χρεώνεται επομένως μόνο αν είναι το i ή κάποιο ενδιάμεσο. Ωστόσο σύμφωνα με τον αλγόριθμο μας, δεν υπάρχει λόγος να ελεγχθεί κανένα ενδιάμεσο βήμα πάνω από μία φορά καθόλη τη διάρκεια εκτέλεσης.

Επομένως αφού γίνονται $2n$ συγκρίσεις μπορούμε να θεωρήσουμε $O(n)$, δηλαδή γραμμικό χρόνο.

Στην περίπτωση που μετρήσουμε και το κόστος των συγκρίσεων με το 0, για την περίπτωση που έχουμε το ψηλότερο κτήριο τότε έχουμε το πολύ 3n συγκρίσεις, πράγμα που υποδεικνύει και πάλι γραμμικό χρόνο.

Σημειώσεις

-
-