Αλγόριθμοι και Πολυπλοκότητα
Ασκήσεις Φροντιστηρίου

Εαρινό Εξάμηνο 2009

Κάτια Παπακωνσταντινοπούλου

1. Έστω $a > 1$ και $f(n) = O(\log_a(n))$. Δείξτε ότι $f(n) = O(\log n)$.

2. Δείξτε ότι $\log(n!) = O(n \log n)$.

3. Ποια είναι η πολυπλοκότητα του παρακάτω τμήματος προγράμματος;

```
1.  for i = 1 to n do
2.      for j = 1 to (i+1)/2 do
3.          x = x + 1
4.      end for
5.  end for
```

---

4. Να δειχθεί ότι $3^n \neq O(2^n)$.

5.

α. Δείξτε ότι $2^{n+1} = O(2^n)$.

β. Είναι $2^{2n} = O(2^n)$;

γ. Δείξτε ότι αν $f(n) = O(n)$ τότε $2^{f(n)}$ δεν είναι $O(2^n)$.

6. Να δοθεί ο καλύτερος O συμβολισμός για τις ακόλουθες συναρτήσεις:

α. $2 \log n - 4n + 3n \log n$

β. $2 + 4 + \ldots + 2n$

γ. $2 + 4 + 8 + \ldots + 2^n$

7. Δείξτε ότι για πραγματικούς αριθμούς n, a και μη αρνητικό ακέραιο b ισχύει $(n + a)^b = \Theta(n^b)$.

8. Να υπολογίσετε την πολυπλοκότητα του αλγορίθμου 'ταξινόμηση Φυσαλίδας' (BubbleSort).

BUBBLE SORT (A)
1. for $i = n$ to 1 do
2. for $j = 1$ to $i - 1$ do
3. if $a_j < a_{j+1}$ then
4. $swap(a_j, a_{j+1})$
5. end if
6. end for
7. end for

---

9. Να υπολογίσετε την πολυπλοκότητα του αλγορίθμου 'ταξινόμηση με Εισαγωγή' (Insertion Sort).

INSERTION SORT (A)
1. for $i = 1$ to n do
2. $val = a_i$
3. $j = i - 1$
4. while $j \geq 1$ and $a_j < val$ do
5. $a_{j+1} = a_j$
6. $j = j - 1$
7. end while
8. $a_{j+1} = val$
9. end for

---

10. Να υπολογίσετε την πολυπλοκότητα του αλγορίθμου 'ταξινόμηση με Επιλογή' (Selection Sort).

SELECTION SORT (A)
1. for $i = 1$ to $n - 1$ do
2. $max = i$
3. for $j = i + 1$ to n do
4. if $a_j > a_{max}$ then
5. $max = j$
6. end if
7. end for
8. $swap(a_{max}, a_i)$
9. end for

---

11. Υπολογίστε το πλήθος των κόμβων ενός πλήρους δυαδικού δέντρου ύψους n με δύο τρόπους, λαμβάνοντας υπόψη κάθε φορά μία από τις εξής παρατηρήσεις:

α. πλήθος κόμβων = πλήθος εσωτερικών κόμβων + πλήθος φύλλων,

β. πλήθος κόμβων = 1 (ρίζα) + πλήθος κόμβων των δύο υποδέντρων που ξεκινούν από τα παιδιά της ρίζας.

12. Να λύσετε την αναδρομική εξίσωση $T_n = T_{n-1} + 2$ για $n \geq 1$ και $T_1 = 1$.

---

13. Να επιλυθεί η αναδρομική εξίσωση $4a_n - 2a_{n-1} + 17a_{n-2} - 4a_{n-3} = 0$ με $a_0 = 1, a_1 = -1$ και $a_2 = 2$.

14. Να επιλυθεί η αναδρομική εξίσωση $a_n = a_{n-1} + n^2$ με $a_0 = 0$.

15. Δείξτε ότι το άθροισμα $\sum_{k=0}^{n} \frac{1}{k^2}$ φράσεται από πάνω από μια σταθερά.

16. Χρησιμοποιώντας τη μέθοδο Master να υπολογίσετε ασυμπτωτικά όρια στις παρακάτω αναδρομές:

α. $T(n) = 4T(n/2) + n$

β. $T(n) = 4T(n/2) + n^2$

γ. $T(n) = 4T(n/2) + n^3$

17. Μπορούμε να εφαρμόσουμε τη μέθοδο Master στην ακόλουθη αναδρομή;

$$T(n) = 2T(n/2) + n \log n$$

18. Ποια είναι η πολυπλοκότητα των αλγορίθμων Binary Search και Merge Sort;
Γράψτε τις αντίστοιχες αναδρομικές εξισώσεις και λύστε τις με τη βοήθεια της μεθόδου Master.

19. Ποιό είναι το μέγιστο και το ελάχιστο πλήθος στοιχείων σε σωρό ύψους h;

20. Δείξτε ότι σε σωρό με n στοιχεία, το ύψος είναι $\lfloor \log n \rfloor$.

21. Ένας πίνακας σε φθίνουσα διάταξη είναι σωρός; Δικαιολογήστε την απάντησή σας.

22. Είναι οι παρακάτω ακολουθίες σωροί;

α. (34,29,20,7,18,3,9,4,6,5,10,4)

β. 'complexity'

23. Να εισάγετε το στοιχείο 17 στον πρώτο σωρό του σχήματος 3.4 στη σελίδα 52 των σημειώσεων.

24. Να διαγράψετε τη ρίζα του σωρού που προέκυψε στην προηγούμενη άσκηση.

25. Να ταξινομηθούν με τη βοήθεια σωρού τα παρακάτω στοιχεία: 6,9,7,4,5,8.

26. Ενα πλήρες δυαδικό δέντρο είναι εύκολο να αναπαρασταθεί με ένα πίνακα. Οι κόμβοι του δέντρου έχουν μια φυσική διάταξη: κατεβαίνοντας γραμμή-γραμμή από τη ρίζα προς τα φύλλα και σε κάθε επίπεδο από αριστερά προς τα δεξιά. Αν υπάρχουν n κόμβοι, τότε αυτή η διάταξη καθορίζει τις θέσεις τους $1, \ldots, n$ μέσα στον πίνακα.

α. Δείξτε ότι ο κόμβος στη θέση k του πίνακα έχει γονέα τον κόμβο στη θέση $\lfloor k/2 \rfloor$ και παιδιά τους κόμβους στις θέσεις $2k$ και $2k + 1$.

β. Αν η αναπαράσταση γίνεται με πλήρες d-αδικό δέντρο, ποιές είναι οι αντίστοιχες θέσεις;

27. Θεωρήστε ένα πλήρες δυαδικό δέντρο 7 κόμβων, οι οποίοι είναι αριθμημένοι από το 1 ως το 7 όπως σε ένα δυαδικό σωρό. Δώστε την αναπαράσταση του με μια λίστα γειτνίασης, καθώς και μια ισοδύναμη αναπαράσταση με πίνακα γειτνίασης.

28. Δίνεται η αναπαράσταση ενός κατευθυνόμενου γράφου με μια λίστα γειτνίασης. Πόσο χρόνο απαιτεί ο υπολογισμός

α. των εσωτερικών βαθμών

β. των εξωτερικών βαθμών

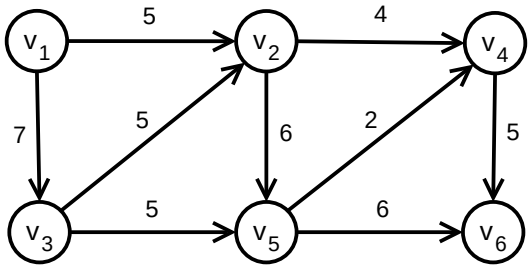
όλων των κόμβων;

29. Να δείξετε ότι το άθροισμα των βαθμών των κόμβων ενός μη κατευθυνόμενου γράφου με m ακμές είναι $2m$.

30. Να δείξετε ότι το άθροισμα των εξωτερικών βαθμών των κόμβων ενός κατευθυνόμενου γράφου με m ακμές είναι m.

31. Ποιος είναι ο χρόνος εκτέλεσης της πρώτα-κατά-πλάτος αναζήτησης σε ένα γράφο που παριστάνεται με πίνακα γειτνίασης; Υποθέστε ότι ο αλγόριθμος έχει τροποποιηθεί κατάλληλα ώστε να χειρίζεται είσοδο αυτής της μορφής.

32. Θεωρήστε τον παρακάτω γράφο:



Να βρεθεί το συντομότερο μονοπάτι από τον κόμβο v_1 στον κόμβο v_6 με τον αλγόριθμο:

α. του Dijkstra

β. του Bellman

33. Έστω κατευθυνόμενος ακυκλικός γράφος με βάρη $G = (V, A, W)$ με $|V| = n$. Να δοθεί ένας αλγόριθμος πολυωνυμικής πολυπλοκότητας που βρίσκει το μέγιστο μονοπάτι από τον κόμβο v_1 στον κόμβο v_n. Θεωρούμε ότι οι κόμβοι $v_1, v_2, \ldots, v_n$ είναι τοπολογικά ταξινομημένοι.

34. Να βρείτε το δέντρο επικάλυψης ελάχιστου κόστους του παρακάτω γράφου $G = (V, E)$ χρησιμοποιώντας:

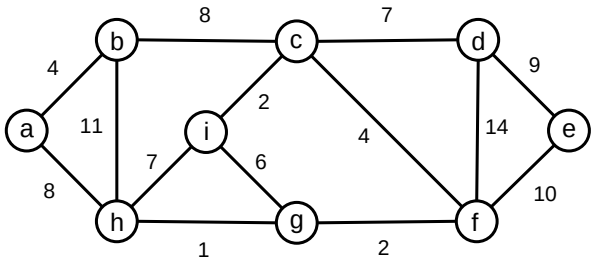
α. τον αλγόριθμο *Prim* και τους γραμμικούς πίνακες $near[v], dist[v],\ v \in V$, τέτοιους ώστε αν T είναι το τρέχον δέντρο επικάλυψης:

$$near[v] = \begin{cases} v & \text{αν } v \in T \\ 0 & \text{αν } v \notin T \text{ και δεν υπάρχει γείτονας του } v \text{ στο } T \\ u & \text{αν } u \text{ είναι ο πλησιέστερος γείτονας του } v \text{ στο } T \end{cases}$$

και

$$dist[v] = \begin{cases} w(v,u) & \text{αν } near[v] = u \\ +\infty & \text{αν } near[v] = 0 \\ 0 & \text{αν } near[v] = v \end{cases}$$

β. τον αλγόριθμο *Kruskal* και τις δομές Union και Find για τον έλεγχο κυκλικότητας.



35. Έστω e η μέγιστου βάρους ακμή σε κάποιο κύκλο ενός γράφου $G = (V, E)$. Δείξτε ότι το δέντρο επικάλυψης ελάχιστου κόστους του G δεν περιέχει την e.

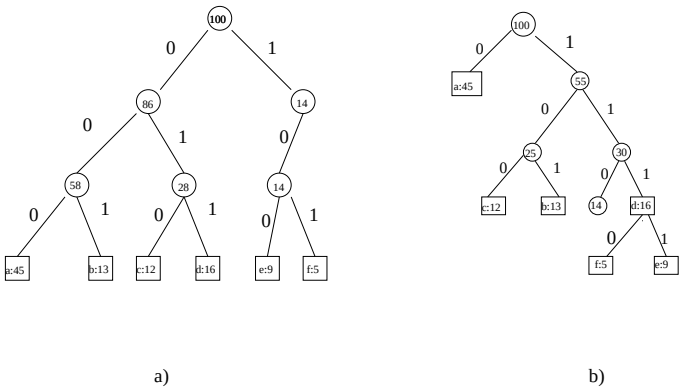
36. Κώδικες Huffman.

Οι κώδικες Huffman χρησιμοποιούνται για να συμπιέζουμε δεδομένα. Ο άπληστος (greedy) αλγόριθμος Huffman χρησιμοποιεί έναν πίνακα από τη συχνότητα εμφάνισης του κάθε συμβόλου και με αυτό τον τρόπο φτιάχνει ένα βέλτιστο τρόπο για να αναπαραστήσουμε κάθε σύμβολο με μια δυαδική συμβολοακολουθία. Έστω πως έχουμε ένα αρχείο με 100.000 χαρακτήρες όπου οι συχνότητες εμφάνισης είναι οι ακόλουθες:

	a	**b**	**c**	**d**	**e**	**f**
Συχνότητα (σε χιλιάδες)	45	13	12	16	9	5
Σταθερού μήκους κωδικός	000	001	010	011	100	101
Μεταβλητού μήκους κωδικός	0	101	100	111	1101	1100

Στην περίπτωση της σταθερού μήκους κωδικοποίησης, χρησιμοποιούμε 3 δυαδικά ψηφία για κάθε αρχικό σύμβολο. Μπορούμε ωστόσο να βελτιώσουμε την κωδικοποίηση, αν τα πιο συχνά χρησιμοποιούμενα σύμβολα τα κωδικοποιούμε με λιγότερα δυαδικά ψηφία, ενώ τα σπανιότερα σύμβολα τα κωδικοποιούμε με περισσότερα ψηφία. Επίσης, χρησιμοποιούμε κωδικοποίση στην οποία καμία κωδικοποίηση συμβόλου δεν αποτελεί πρόθεμα καμίας άλλης κωδικοποίησης κάποιου άλλου συμβόλου. Αυτή η ιδιότητα είναι χρήσιμη για την αποκωδικοποίηση. Για παράδειγμα να έχουμε την κωδικοποιημένη ακολουθία 001011101, αυτή έχει προέλθει από τα *aabd*.

Για να γίνει η κωδικοποίηση και αποκωδικοποίση, χρησιμοποιείται ένα δυαδικό δέντρο, όπου στα φύλλα του έχουμε τα αρχικά σύμβολα. Η κωδικοποίηση γίνεται αν διατρέξουμε το δέντρο από τη ρίζα προς τα φύλλα και



Σχήμα 1: Η κωδικοποίηση Huffman σταθερού μήκους a) και μεταβλητού μήκους b).

οι ακμές έχουν ετικέτες 0 ('πήγαινε αριστερά') και 1 ('πήγαινε δεξιά'). Για τις παραπάνω συχνότητες, παίρνουμε το παρακάτω δέντρο του Σχήματος 1.

Μία βέλτιστη κωδικοποίηση είναι αυτή που προκύπτει από ένα *πλήρες* δυαδικό δέντρο. Αν θεωρήσουμε πως έχουμε C σύμβολα, τότε το πλήρες δυαδικό δέντρο που δίνει τη βέλτιστη κωδικοποίηση θα έχει $|C|$ φύλλα και $|C|-1$ εσωτερικούς κόμβους. Το κόστος του δέντρου για την κωδικοποίηση είναι

$B(T) = \sum_{c \in C} f(c) d_T(c)$

όπου $f(c)$ είναι η συχνότητα του συμβόλου c και $d_T(c)$ είναι το βάθος του συμβόλου c, άρα και το μήκος της κωδικοποίησής του.

Ο Huffman κατασκεύασε έναν άπληστο (greedy) αλγόριθμο για να βρίσκει τη βέλτιστη λύση. Έστω C το σύνολο των n συμβόλων και ότι για κάθε σύμβολο $c \in C$, $f(c)$ είναι η συχνότητα εμφάνισής του. Ο αλγόριθμος φτιάχνει από κάτω-προς-τα-πάνω το δέντρο. Ξεκινάει από τα $|C|$ φύλλα και κάνει $|C|-1$ διαδικασίες 'συγχώνευσης' για να κατασκευάσει το τελικό δέντρο. Μια ουρά Q χρησιμοποιείται για να βρίσκει τις μικρότερες συχνότητες και να τις συγχωνεύει. Το αποτέλεσμα είναι ένα νέο αντικείμενο του οποίου η συχνότητα είναι το άθροισμα των συχνοτήτων που συγχωνεύτηκαν.

Αλγόριθμος 1 HUFFMAN(C)

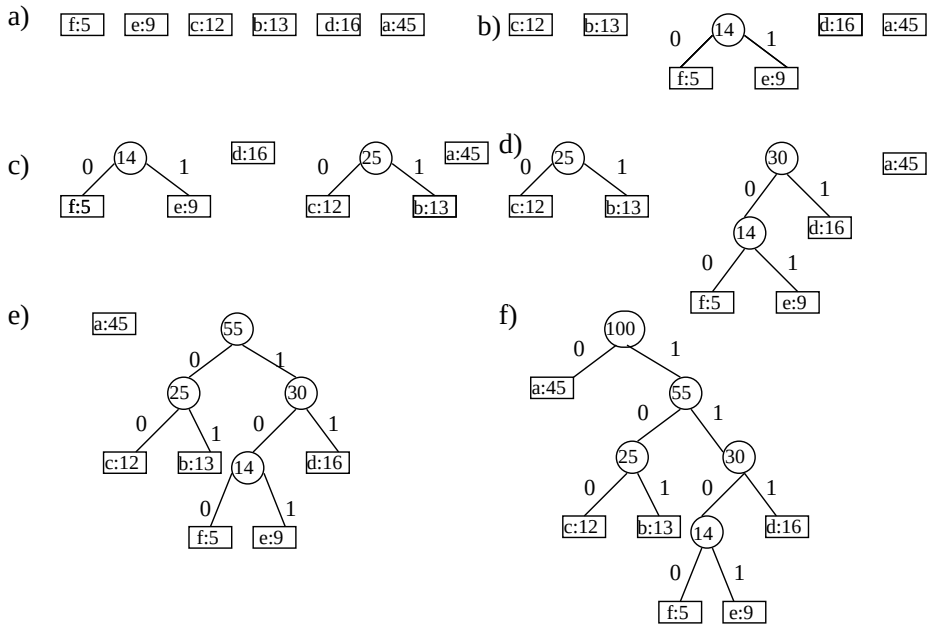
```
1: n = |C|
2: Q = C
3: for i = 1 to n − 1 do
4:     δημιούργησε ένα νέο κόμβο z
5:     left[z] = x = EXTRACT − MIN(Q)
6:     right[z] = y = EXTRACT − MIN(Q)
7:     f[z] = f[x] + f[y]
8:     INSERT(Q, z)
9: end for
10: return EXTRACT-MIN(Q) (* Η ρίζα του δέντρου *)
```

Στο τρέχον παράδειγμα τα στάδια κατασκευής του δέντρου είναι αυτά που φαίνονται στο Σχήμα 2.

α. Ποιά είναι μία βέλτιστη κωδικοποίηση Huffman για την παρακάτω ακολουθία συχνοτήτων εμφάνισης, βασισμένη στους πρώτους 8 αριθμούς Fibonacci, δηλ. $a:1, b:1, c:2, d:3, e:5, f:8, g:13, h:21$.

β. Γενικεύστε για τους n πρώτους αριθμούς Fibonacci.

37. Γενικεύστε τον αλγόριθμο Huffman για τρία ψηφία κωδικοποίησης (κωδικοποίηση με τα ψηφία 0,1 και 2).



Σχήμα 2: Κατασκευή δέντρου κωδικοποίησης Huffman.

38. Θεωρήστε το παρακάτω στιγμιότυπο του προβλήματος του σακιδίου: Έχουμε $n = 5$ αντικείμενα με βάρη $a = \{8, 12, 15, 16, 13\}$ και αξίες $c = \{27, 9, 30, 16, 6.5\}$ αντίστοιχα και ένα σακίδιο χωρητικότητας $b = 45$.

Ζητάμε ένα υποσύνολο των αντικειμένων που χωράει στο σακίδιο και έχει μέγιστη αξία.

α. Να επιλυθεί το πρόβλημα στη συνεχή έκδοχή του με έναν άπληστο αλγόριθμο

β. Να επιλυθεί το πρόβλημα στην εκδοχή 0-1 με έναν άπληστο αλγόριθμο

γ. Να επιλυθεί το πρόβλημα στην εκδοχή 0-1 με δυναμικό προγραμματισμό

39. Απόσταση σύνταξης.

Οι ελεγκτές ορθογραφίας χρειάζονται έναν τρόπο να αξιολογούν πόσο 'κοντά' είναι μία λέξη σε μία άλλη. Ένας τρόπος είναι η έννοια της *απόστασης σύνταξης* (edit distance). Η απόσταση σύνταξης είναι ο ελάχιστος αριθμός 'συντακτικών διορθώσεων' που μπορούμε να κάνουμε ώστε να πάμε από τη μία λέξη στην άλλη. Μία συντακτική διόρθωση είναι ένα από τα παρακάτω:

- εισαγωγή ενός χαρακτήρα
- διαγραφή ενός χαρακτήρα
- μετατροπή ενός χαρακτήρα σε ένα άλλο

Ισοδύναμα, μπορούμε να ορίσουμε την απόσταση σύνταξης ως εξής. Μία 'ευθυγράμμιση' δύο λέξεων είναι ένας τρόπος να βάλουμε μία λέξη πάνω από την άλλη έτσι ώστε τα άκρα τους να πέσουν στην ίδια στήλη, συμπληρώνοντας όσα κενά θέλουμε μεταξύ χαρακτήρων των δύο λέξεων. Για παράδειγμα, παρακάτω φαίνονται δύο ευθυγραμμίσεις των λέξεων sunny και snowy (τα κενά συμβολίζονται με παύλα):

```
s - n o w y          - s n o w - y

s u n n - y          s u n - - n y
```

Το κόστος μίας ευθυγράμμισης είναι ο αριθμός των στηλών στις οποίες οι δύο λέξεις διαφέρουν. Στο παραπάνω παράδειγμα, η αριστερή ευθυγράμμιση έχει κόστος 3 ενώ η δεξιά 5. Η απόσταση σύνταξης είναι το ελάχιστο κόστος μεταξύ όλων των ευθυγραμμίσεων δύο λέξεων. Στο παραπάνω παράδειγμα, η απόσταση σύνταξης είναι 3.

Ζητάμε έναν αλγόριθμο που να βρίσκει την απόσταση σύνταξης μεταξύ δύο λέξεων x, y.