Θέματα προετοιμασίας β’ (ΚΕΦ. 11-15) στο μάθημα «ΑΛΓΟΡΙΘΜΟΙ ΚΑΙ ΠΟΛΥΠΛΟΚΟΤΗΤΑ»

Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Πανεπιστήμιο Ιωαννίνων, Άρτα

τελευταία ενημέρωση: 15/1/2022

Κεφάλαιο 11 (Διαίρει και Βασίλευε)

1. Απαντήστε με ΣΩΣΤΟ/ΛΑΘΟΣ
   a. Τα προβλήματα που επιλύονται με την τεχνική διαίρει και βασίλευε τυπικά επιλύονται με τη χρήση αναδρομής.
   b. Στην τεχνική διαίρει και βασίλευε το αρχικό πρόβλημα χωρίζεται σε μικρότερα επικαλυπτόμενα υποπροβλήματα.
   c. Η χρονική πολυπλοκότητα του αλγορίθμου merge-sort είναι O(n^2).
   d. Ο ταχύτερος αλγόριθμος πολλαπλασιασμού δύο n-bit ακεραίων έχει πολυπλοκότητα O(n^2).
   e. Το master θεώρημα εφαρμόζεται σε αναδρομικές εξισώσεις.
2. Με βάση το master θεώρημα:

$$T(n) = \begin{cases} c & \text{if } n < d \\ aT(n/b) + f(n) & \text{if } n \geq d \end{cases}$$

$1.\ \text{if } f(n) \text{ is } O(n^{\log_b a - \varepsilon}), \text{ then } T(n) \text{ is } \Theta(n^{\log_b a})$

$2.\ \text{if } f(n) \text{ is } \Theta(n^{\log_b a} \log^k n), \text{ then } T(n) \text{ is } \Theta(n^{\log_b a} \log^{k+1} n)$

$3.\ \text{if } f(n) \text{ is } \Omega(n^{\log_b a + \varepsilon}), \text{ then } T(n) \text{ is } \Theta(f(n)),$
$\text{provided } af(n/b) \leq \delta f(n) \text{ for some } \delta < 1.$

   υπολογίστε την πολυπλοκότητα των ακόλουθων αναδρομικών εξισώσεων.
   a. T(n) = 2T(n/2) + n
   b. T(n) = 7T(n/3) + n
   c. T(n) = 8T(n/2) + n^2
   d. T(n) = T(n/2) + nlogn
   e. T(n) = T(n/2) + n^2
3. Γράψτε μια αναδρομική εξίσωση που να περιγράφει το χρόνο εκτέλεσης της merge sort.
4. Συμπληρώστε τον ακόλουθο κώδικα συγχώνευσης δύο ταξινομημένων ακολουθιών έτσι ώστε η συγχώνευση να εκτελείται σε γραμμικό χρόνο.

```
def merge(list1, list2):
    ...

list1 = [1, 3, 5, 7, 9]
list2 = [0, 2, 4, 6, 8]

merged_list = merge(list1, list2)

print ("Συγχώνευση λιστών : " + str(merged_list))
```

5. Σε ένα πρόβλημα μεγιστοποίησης δύο συναρτήσεων f1 και f2 προκύπτουν οι ακόλουθες λύσεις με κάθε ζεύγος τιμών να αναφέρει πρώτα την τιμή του f1 και στη συνέχεια την τιμή του f2: (5,6), (3,9), (4,5), (10,1), (7,2), (9,0), (0,9). Ποιες είναι οι μη κυριαρχούμενες λύσεις;
6. Τι είναι το πρόβλημα μέγιστου συνόλου (maximaset); Τι πολυπλοκότητα έχει η επίλυσή του με διαίρει και βασίλευε;

Κεφάλαιο 12 (Δυναμικός Προγραμματισμός)

1. Απαντήστε με ΣΩΣΤΟ/ΛΑΘΟΣ
   a. Η αποθήκευση αποτελεσμάτων υποπροβλημάτων είναι βασικό χαρακτηριστικό του δυναμικού προγραμματισμού.
   b. Ο δυναμικός προγραμματισμός δίνει λύσεις σε προβλήματα για τα οποία φαίνεται η επίλυσή τους να απαιτεί εκθετικό χρόνο, σε πολύ καλύτερο χρόνο.
   c. Ο merge-sort είναι ένας αλγόριθμος δυναμικού προγραμματισμού.
2. Καταγράψτε όλες τις μη κενές υποσυμβολοσειρές και όλες τις μη κενές υποακολουθίες της λέξης «ΤΑΞΗ».
3. Συμπληρώστε τον ακόλουθο κώδικα που υπολογίζει αναδρομικά όλες τις υποακολουθίες της συμβολοσειράς που δέχεται ως παράμετρο (BRUTE FORCE).

```
subsequences = []

def all_subsequences(string, index, c):
    ...

all_subsequences("ΤΑΞΗ", 0, "")
print(subsequences)
```

4. Τι σημαίνει το χαρακτηριστικό «βελτιστότητα υποπροβλημάτων» στο δυναμικό προγραμματισμό;
5. Αναφέρατε 4 προβλήματα που λύνονται αποδοτικά με δυναμικό προγραμματισμό.
6. Δίνονται οι δύο συμβολοσειρές X=GACCGACT και Y=AGCACGTC. Συμπληρώστε τον ακόλουθο πίνακα με τον οποίο ο αλγόριθμος δυναμικού προγραμματισμού εντοπίζει τη μέγιστη κοινή ακολουθία ανάμεσα στις δύο συμβολοσειρές.

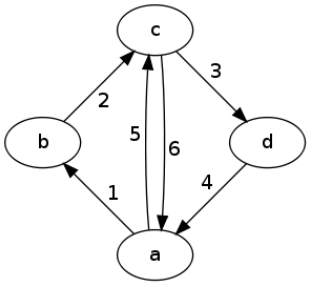
			G	A	C	C	G	A	C	T
		-1	0	1	2	3	4	5	6	7
	-1									
A	0									
G	1									
C	2									
A	3									
C	4									
G	5									
T	6									
C	7									

Κεφάλαιο 13 (Γράφοι και διασχίσεις)

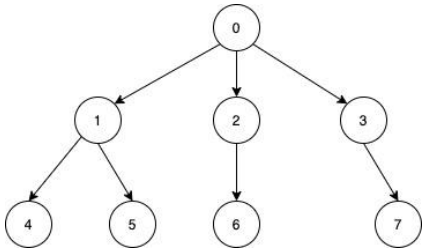
1. Απαντήστε με ΣΩΣΤΟ/ΛΑΘΟΣ
   a. Η αναπαράσταση γράφων με πίνακα γειτονικότητας προτιμάται σε αραιούς γράφους.
   b. Η πολυπλοκότητα χώρου για γράφο που αναπαρίσταται με πίνακα γειτονικότητας είναι O(V^2).
   c. Η πολυπλοκότητα χώρου για γράφο που αναπαρίσταται με λίστα γειτονικότητας είναι O(V+E).
   d. Αν ένας γράφος αναπαρίσταται με πίνακα γειτονικότητας ο εντοπισμός του εάν μια κορυφή συνδέεται απευθείας με μια άλλη κορυφή είναι ταχύτερος από ότι αν ο γράφος αναπαρίσταται με λίστα γειτονικότητας.
   e. Αν ένας γράφος αναπαρίσταται με πίνακα γειτονικότητας ο εντοπισμός των γειτονικών κορυφών μιας κορυφής είναι ταχύτερος από ότι αν ο γράφος αναπαρίσταται με λίστα γειτονικότητας.
   f. Μια απλή διαδρομή σε έναν γράφο είναι μια σειρά από κορυφές και ακμές που ξεκινά από μια κορυφή και τελειώνει σε μια κορυφή.
   g. Κάθε κύκλος σε ένα γράφο αποτελεί και μια διαδρομή.
   h. Η αναδρομική υλοποίηση του αλγορίθμου DFS (αναζήτηση πρώτα κατά βάθος) είναι εύκολη.

i. Ο αλγόριθμος DFS υλοποιείται με χρήση ουράς.
j. Η πολυπλοκότητα του αλγορίθμου DFS είναι O(V+E).
k. Οι αλγόριθμοι DFS και BFS έχουν την ίδια πολυπλοκότητα.
l. Ο αλγόριθμος DFS εντοπίζει τη διαδρομή με το ελάχιστο πλήθος ακμών μεταξύ δύο κορυφών.
m. Ο αλγόριθμος DFS και ο αλγόριθμος BFS μπορούν να χρησιμοποιηθούν για την εύρεση των συνεκτικών συνιστωσών ενός γραφήματος.
n. Ο αλγόριθμος DFS έχει μικρότερες απαιτήσεις μνήμης από ότι ο αλγόριθμος BFS.

2. Με τι ισούται το άθροισμα των βαθμών (degrees) ενός μη κατευθυνόμενου γράφου;
3. Ποιο είναι το άνω όριο για το πλήθος των ακμών σε ένα μη κατευθυνόμενο γράφο χωρίς self-loops και παράλληλες ακμές;
4. Δείξτε την αναπαράσταση του ακόλουθου γραφήματος α) με πίνακα γειτονικότητας και β) με λίστα γειτονικότητας.



5. Περιγράψτε λεκτικά το πως μπορεί να χρησιμοποιηθεί ο αλγόριθμος DFS για τον εντοπισμό ενός κύκλου σε έναν κατευθυνόμενο γράφο.
6. Δίνεται ο ακόλουθος γράφος (δέντρο). Ποια είναι η διαδρομή που επιστρέφει ο DFS και ποια η διαδρομή που επιστρέφει ο BFS;



7. Δίνεται ο ακόλουθος κώδικας που υπολογίζει τη διαδρομή που εντοπίζει ο αλγόριθμος αναζήτησης πρώτα κατά πλάτος (breadth first search) από μια κορυφή αφετηρία s προς μια κορυφή προορισμό e. Τροποποιήστε τον κώδικα έτσι ώστε να υλοποιεί τον αλγόριθμο αναζήτησης πρώτα κατά βάθος (depth first search).

```
from collections import deque


def bfs(g, s, e):
    """Αναζήτηση πρώτα κατά πλάτος από το s στο  e"""
    frontier = deque()  # μέτωπο αναζήτησης
    frontier.append(s)
    visited = set()  # κορυφές που έχουν επισκεφτεί
    visited.add(s)
    prev = {s: None}  # για κάθε κορυφή, η προηγούμενη κορυφή
    while frontier:  # όσο το μέτωπο αναζήτησης δεν είναι κενό
        current_node = frontier.popleft()
        for next_node in g[current_node]:
            if not next_node in visited:
                frontier.append(next_node)
                visited.add(next_node)
                prev[next_node] = current_node

    # κατασκευή του μονοπατιού από το e στο s
```

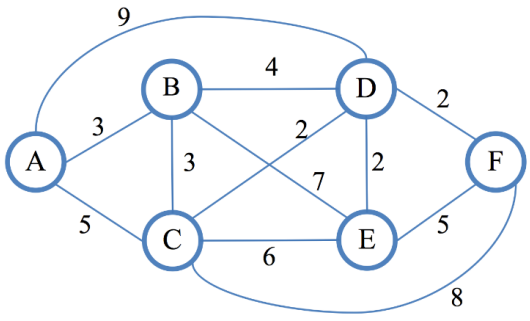
```
    path = []
    at = e
    while at != None:
        path.append(at)
        at = prev[at]

    # αντιστροφή του μονοπατιού για να προκύψει το μονοπάτι από το s στο e
    path = path[::-1]

    # αν τα s και e είναι συνδεδεμένα επιστροφή του μονοπατιού
    if path[0] == s:
        return path
    return []

graph = {"A": ["B", "C"], "B": ["C", "D"], "C": ["D"], "D": ["E"], "E": []}

path = bfs(graph, "A", "E")
print("BFS: ", path)
```

```
>>> ['A', 'B', 'D', 'E']
```

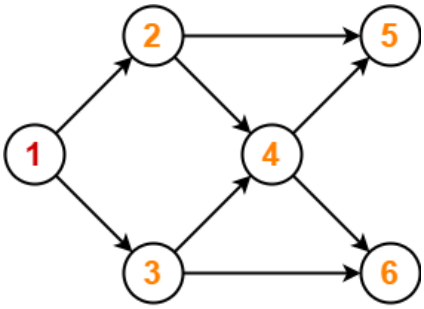
8. Γράψτε συνάρτηση με ορίσματα ένα κατευθυνόμενο γράφο G και μια κορυφή του γράφου s, που να επιστρέφει το μικρότερο αριθμό βημάτων (διασχίσεις ακμών) που απαιτούνται για τη μετάβαση από την κορυφή s προς όλες τις άλλες κορυφές του γράφου.

Κεφάλαιο 14 (Συντομότερες διαδρομές)

1. Απαντήστε με ΣΩΣΤΟ/ΛΑΘΟΣ
    a. Μια υποδιαδρομή της συντομότερης διαδρομής είναι η ίδια μια συντομότερη διαδρομή.
    b. Ο αλγόριθμος του Dijkstra είναι ένας «άπληστος» αλγόριθμος.
    c. Ο αλγόριθμος του Dijkstra λειτουργεί ορθά και σε γράφους με αρνητικά βάρη ακμών.
    d. Ο αλγόριθμος των Bellman-Ford λειτουργεί ορθά και σε γράφους με αρνητικά βάρη ακμών.
    e. Ο αλγόριθμος του Dijkstra μπορεί να εντοπίζει κύκλους αρνητικού βάρους σε γράφους.
    f. Η αναζήτηση των συντομότερων διαδρομών σε έναν γράφο μπορεί να επιταχυνθεί αν γνωρίζουμε ότι ο γράφος είναι DAG (Κατευθυνόμενος Ακυκλικός Γράφος).
2. Εφαρμόστε τον αλγόριθμο του Dijkstra για την εύρεση των συντομότερων διαδρομών στον ακόλουθο γράφο. Καταγράψτε για κάθε κορυφή v όλες τις τιμές που σταδιακά λαμβάνει η ετικέτα D[v] (δλδ η απόσταση της κορυφής v από την κορυφή αφετηρίας).



3. Ποια είναι η πολυπλοκότητα του αλγορίθμου του Dijkstra;
4. Ποια είναι η πολυπλοκότητα του αλγορίθμου των Bellman-Ford;
5. Ποια είναι η πολυπλοκότητα του αλγορίθμου των Floyd-Warshall;
6. Με ποια κριτήρια επιλέγεται ο καταλληλότερος αλγόριθμος ανάμεσα στους αλγόριθμους: Dijkstra, Bellman-Ford, Floyd-Warshall;
7. Για ποιο λόγο ενδείκνυται η χρήση ουράς προτεραιότητας στην υλοποίηση του αλγορίθμου του Dijkstra;
8. Παραθέστε όλες τις τοπολογικές ταξινομήσεις για τον ακόλουθο γράφο.



Κεφάλαιο 15 (Δέντρα επικάλυψης ελάχιστου κόστους)

1. Απαντήστε με ΣΩΣΤΟ/ΛΑΘΟΣ
    a. Το ελάχιστο συνεκτικό δέντρο ενός γράφου είναι πάντα μοναδικό.
    b. Το ελάχιστο συνεκτικό δέντρο ενός γράφου είναι μοναδικό αν τα βάρη του γράφου είναι διακριτά.
    c. Το ελάχιστο συνεκτικό δέντρο ενός γράφου μπορεί να περιλαμβάνει κύκλους.
    d. Ο αλγόριθμος του Prim είναι παρόμοιος με τον αλγόριθμο του Dijkstra.
    e. Ο αλγόριθμος του Prim είναι ένας άπληστος αλγόριθμος.
    f. Ο αλγόριθμος του Kruskal ξεκινά από μια κορυφή και σταδιακά κατασκευάζει από αυτή το ελάχιστο συνεκτικό δέντρο.
    g. Ο αλγόριθμος του Boruvka έχει περισσότερες ομοιότητες με τον αλγόριθμο του Kruskal παρά με τον αλγόριθμο του Prim.
    h. Οι ακμές του συνεκτικού δέντρου όλων των συνεκτικών γραφημάτων με 20 κορυφές είναι 19.
2. Ποια δομή δεδομένων επιταχύνει την εκτέλεση του αλγορίθμου του Kruskal εφόσον ο αλγόριθμος υλοποιηθεί κάνοντας χρήση της;
3. Ποια είναι η πολυπλοκότητα του αλγορίθμου του Prim;
4. Ποια είναι η πολυπλοκότητα του αλγορίθμου του Kruskal;
5. Υπολογίστε το ελάχιστο συνεκτικό δέντρο για τον ακόλουθο γράφο α) με τον αλγόριθμο του Prim ξεκινώντας από την κορυφή a και β) με τον αλγόριθμο του Kruskal. Και για τους δύο αλγόριθμους καταγράψτε τις ακμές που ορίζουν το ελάχιστο συνεκτικό δέντρο με τη σειρά με την οποία ο αλγόριθμος τις προσαρτά σε αυτό.