Algoritmi e Strutture Dati

a.a. 2023/24

Compito del 12/06/2024

Cognome: _____ Nome: _____

Matricola: _____ E-mail: _____

Parte I

(30 minuti; ogni esercizio vale 2 punti)

Avvertenza: Si giustifichino tecnicamente tutte le risposte. In caso di discussioni poco formali o approssimative gli esercizi non verranno valutati pienamente.

1. Completare la seguente tabella indicando la complessità delle operazioni che si riferiscono a un dizionario di *n* elementi. Per l'operazione **Cancellazione** si assuma di essere sull'elemento *x* a cui si applica l'operazione.

	Ricerca	**Minimo**	**Cancellazione**	**Massimo**	**Costruzione**
Tabelle Hash con liste di collisione (caso medio)*					
Max-Heap					

*La Tabella Hash ha dimensione m e il fattore di carico è α e le liste sono doppiamente concatenate.

2. Sia *G* un grafo non orientato con *n* vertici ed *m* archi. Si stabilisca se le seguenti affermazioni sono corrette:

   a) Se $m \geq n - 1$ allora *G* è connesso
   b) Se $m = n - 1$ allora *G* è un albero
   c) Se $m \leq n - 1$ allora *G* è aciclico

   In caso affermativo si fornisca una dimostrazione, altrimenti un controesempio.

3. Un nuovo algoritmo per determinare un albero di copertura minimo in un grafo pesato ha complessità

$$T(m) = 3T\left(\frac{m}{3}\right) + \frac{m}{3}$$

   dove *m* rappresenta il numero di archi del grafo in ingresso. Si dica se l'algoritmo in questione è preferibile o meno all'algoritmo di Kruskal e perché.

Algoritmi e Strutture Dati

a.a. 2023/24

Compito del 12/06/2024

Cognome: _____ Nome: _____

Matricola: _____ E-mail: _____

Parte II

(2.5 ore; ogni esercizio vale 6 punti)

Avvertenza: Si giustifichino tecnicamente tutte le risposte. In caso di discussioni poco formali o approssimative gli esercizi non verranno valutati pienamente.

1. Siano T_1 e T_2 due **alberi binari di ricerca** tali che tutte le chiavi memorizzate in T_1 sono minori delle chiavi memorizzate in T_2.
   a. Scrivere una funzione in C o in C++ `PTree MergeBST(PTree T1, PTree T2)` che presi in input T_1 e T_2 restituisca un albero binario di ricerca T le cui chiavi sono tutte e sole le chiavi contenute in T_1 e T_2.
   b. Determinare la complessità della funzione proposta nel caso migliore e nel caso peggiore.

   La rappresentazione dell'albero binario di ricerca è tramite puntatori e utilizza i tipi `PTree` e `PNode`.

2. Un giovane amante del trekking vuole organizzare la prossima camminata e ha raccolto nel vettore *alture* l'altitudine di n punti che vorrebbe raggiungere. Dal punto in posizione $0 <= i < n$, il giovane può raggiungere qualsiasi altro punto in posizione $i < j < n$, data una condizione: l'altitudine del punto in posizione j deve essere minore o uguale a quella del punto in posizione i. Una sequenza di punti consecutivi forma un percorso, la cui lunghezza è il numero dei punti.
   a. fornire una caratterizzazione ricorsiva della lunghezza l_i di un percorso di lunghezza massima che abbia come ultimo punto *alture[i]*;
   b. tradurre tale definizione in un algoritmo di programmazione dinamica con il metodo bottom-up che determina la lunghezza del percorso più lungo percorribile dal giovane.
   c. valutare e giustificare la complessità dell'algoritmo.

   Il prototipo della funzione è il seguente:

```
int percorso_piu_lungo(vector<int>& alture)
```

3. Sia $G = (V, E)$ un grafo orientato e pesato, con funzione peso $w : E \rightarrow \mathrm{R}$, che non contenga cicli negativi e cappi. Si assuma che i vertici siano numerati da 1 a n, ovvero $V = \{1, \dots, n\}$, e sia W la matrice di dimensione n x n definita come segue:

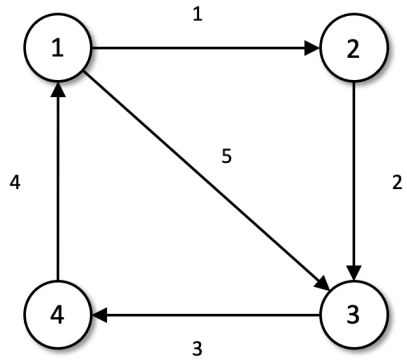
$$W[i,j] = \begin{cases} 0 & se\ i = j \\ w(i,j) & se\ i \neq i\ e\ (i,j) \in E \\ +\infty & se\ i \neq j\ e\ (i,j) \notin E \end{cases}$$

   Si stabilisca quali problemi risolvono i due algoritmi riportati di seguito (che prendono in ingresso la matrice W associata al grafo), se ne dimostri la correttezza e se ne calcoli la complessità.

```
MyAlgorithm1(W)
1.  n = n.ro di righe di W
2.  alloca spazio per un vettore D
    di dimensione n
3.  D[1] = 0
4.  for i = 2 to n
5.    D[i] = +∞
6.  for k = 1 to n
7.    for i = 1 to n
8.      for j = 1 to n
9.        D[j] = min{ D[j], D[i] + W[i,j] }
10. return D
```

```
MyAlgorithm2(W)
1.  n = n.ro di righe di W
2.  alloca spazio per una matrice D
    di dimensione n x n
3.  for i = 1 to n
4.    for j = 1 to n
5.      D[i,j] = W[i,j]
6.  for k = 1 to n
7.    for i = 1 to n
8.      for j = 1 to n
9.        D[i,j] = min{ D[i,j], D[i,k] + D[k,j] }
10. return D
```

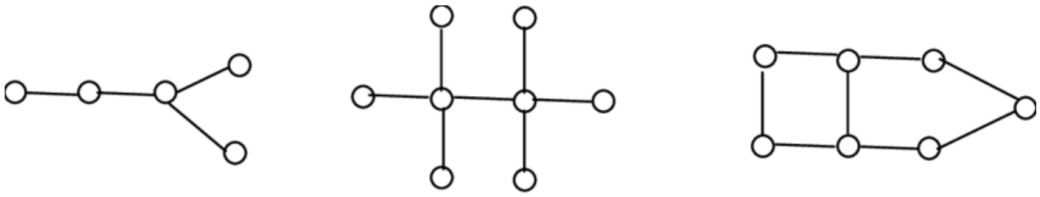
Cosa restituiscono in uscita i due algoritmi in presenza del grafo seguente? (Scrivere esplicitamente il *vettore D* nel primo caso e la *matrice D* nel secondo. Non è necessario riportare la simulazione degli algoritmi.)



4. Il seguente algoritmo accetta in ingresso la matrice di adiacenza di un grafo non orientato e restituisce TRUE o FALSE. Qual è la funzione svolta dall'algoritmo e qual è la sua complessità?

```
MyAlgorithm3(A)
1.  n = n.ro di righe di A
2.  crea due matrici n x n B e C
3.  for i = 1 to n
4.     for j = 1 to n
5.        B[i,j] = 0
6.        for k = 1 to n
7.           B[i,j] = A[i,k]*A[k,j]
8.  for i = 1 to n
9.     for j = 1 to n
10.       C[i,j] = 0
11.       for k = 1 to n
12.          C[i,j] = B[i,k]*A[k,j]
13. sum = 0
14. for i = 1 to n
15.    sum = sum + C[i,i]
16. if sum = 0 then
15.    return TRUE
16. else
17.    return FALSE
```

Cosa restituirebbe l'algoritmo nei casi seguenti? Perché?



G_1 G_2 G_3