Algoritmi e Strutture Dati

a.a. 2020/21

Compito del 14/06/2021

Cognome: _____ Nome: _____

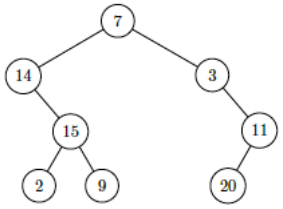
Matricola: _____ E-mail: _____

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. Dato il seguente albero



   Eseguire una visita in preordine, una visita in ordine simmetrico, una visita in postordine e una visita in ampiezza elencando nei quattro casi la sequenza dei nodi incontrati.

2. Un algoritmo ricorsivo $\mathcal{A}$ è in grado di stabilire se, all'interno di un grafo orientato e pesato G, è presente almeno un ciclo negativo (rispondendo "TRUE" in caso affermativo e "FALSE" in caso contrario). La sua complessità è data da

$$T(n) = 16T(n/2) + 3n^4 + 2n^3$$

   dove n rappresenta il numero di vertici del grafo. Esistono algoritmi più efficienti di $\mathcal{A}$ per risolvere il problema dato?

3. Il prof. M. I. L. Lennium sostiene di aver dimostrato le seguenti affermazioni:

   (a) CLIQUE è riducibile polinomialmente a ISOMORFISMO-DI-GRAFI;

   (b) ISOMORFISMO-DI-GRAFI è riducibile polinomialmente a CICLO-NEGATIVO (ovvero, il problema decisionale discusso nell'esercizio precedente).

   Se fossero vere entrambe, cosa potremmo dedurre? Perché? Cosa potremmo dedurre, invece, dalla due affermazioni prese singolarmente?

Algoritmi e Strutture Dati

a.a. 2020/21

Compito del 14/06/2021

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. Uno *heap* ternario è simile ad uno *heap* binario, tranne che per il fatto che i nodi interni possono avere fino a tre figli.
   a. Spiegare come si può rappresentare uno *heap* ternario in un vettore.
   b. Qual è l'altezza di uno *heap* ternario di n elementi (in funzione di n)?
   c. Si scriva la funzione `heap-extract-max()` per gli *heap* ternari. Calcolare la complessità computazionale di tale funzione.
   d. Si scriva la funzione `max-heap-insert()` per gli *heap* ternari. Calcolare la complessità computazionale di tale funzione.

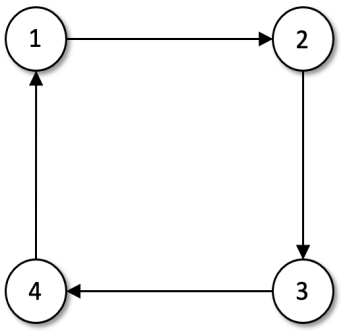
2. I numeri di Fibonacci sono definiti dalla seguente ricorrenza:
   $F_0 = 0$
   $F_1 = 1$
   $F_i = F_{i-1}+F_{i-2}$ per $i \geq 2$

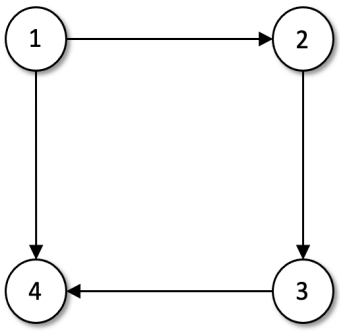
   Scrivere un algoritmo di **programmazione dinamica** con tempo O(n) per calcolare n-esimo numero di Fibonacci. Quale schema è stato utilizzato (top-down o bottom-up)?
   Giustificare il calcolo della complessità.

3. Si stabilisca quale problema risolve il seguente algoritmo, che accetta in ingresso un grafo orientato $G = (V, E)$ (`Q` rappresenta una coda con priorità con campo chiave `d`):

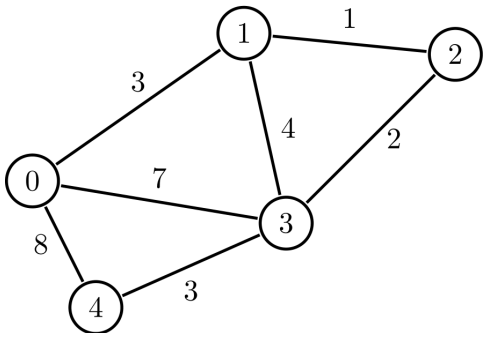
```
MyAlgorithm(G)

1.  for each u ∈ V[G]
2.     for each x ∈ V[G] \ {u}
3.        d[x] = +∞
4.     d[u] = 0
5.     Q = V[G]
6.     for i = 1 to |V[G]|
7.        v = ExtractMin(Q)
8.        for each z ∈ V[G]
9.           if (v,z) ∈ E[G] then
10.             if d[z] > d[v] + 1 then
11.                d[z] = d[v] + 1
12.    for each x ∈ V[G]
13.       if d[x] = +∞ then
14.          return FALSE
15. return TRUE
```

Si dimostri la correttezza dell'algoritmo e si determini la sua complessità computazionale assumendo che la coda con priorità `Q` sia implementata mediante un array lineare. Si dica inoltre cosa restituisce `MyAlgorithm` in presenza dei seguenti grafi. Perché?





4. Si scriva l'algoritmo di Prim, si dimostri la sua correttezza, si fornisca la sua complessità computazionale e si simuli accuratamente la sua esecuzione sul seguente grafo (utilizzando il vertice 0 come "sorgente"):



In particolare:

a) si indichi l'ordine con cui vengono estratti i vertici
b) si riempia la tabella seguente con i valori dei vettori key e π, iterazione per iterazione:

	vertice 0		vertice 1		vertice 2		vertice 3		vertice 4	
	key[0]	**π[0]**	**key[1]**	**π[1]**	**key[2]**	**π[2]**	**key[3]**	**π[3]**	**key[4]**	**π[4]**
dopo inizializzazione										
iterazione 1										
iterazione 2										
iterazione 3										
iterazione 4										
iterazione 5										