Algoritmi e Strutture Dati

a.a. 2022/23

Compito del 11/9/2023

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.

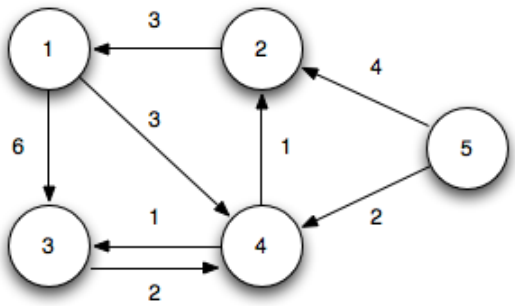
1. Si disegni l'albero binario di ricerca la cui visita in post-ordine ha come risultato 1, 12, 7, 16, 19, 25, 20, 13. Poi si effettui la cancellazione del nodo 13 e si disegni l'albero risultante.

2. Un algoritmo ricorsivo $\mathcal{A}$ determina un albero di copertura minimo in un grafo pesato connesso e ha complessità pari a:

$$T(m) = 16T(m/4) + 16m$$

dove m rappresenta il numero di archi del grafo. Si stabilisca se $\mathcal{A}$ è asintoticamente più efficiente dell'algoritmo di Kruskal.

3. Si scriva l'algoritmo di Floyd-Warshall e si supponga di volerlo utilizzare sul seguente grafo:



Si produca: (a) la matrice iniziale che dovrà essere data in ingresso all'algoritmo e (b) la matrice generata dall'algoritmo *dopo la prima iterazione*.

Algoritmi e Strutture Dati

a.a. 2022/23

Compito del 11/9/2023

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. Sia *T* un albero generale i cui nodi contengono interi e hanno campi: **key**, **left-child** e **right-sib**.
   a. Si scriva una funzione **efficiente** in C o C++ che verifichi la seguente proprietà: per ogni nodo *u*, le chiavi dei figli del nodo *u* devono avere valori non decrescenti considerando i figli di *u* da sinistra verso destra. Il prototipo della funzione è
      `bool isNonDec(PNodeG r)`
      Restituisce `true` se la proprietà è verificata altrimenti `false`.
   b. Valutare e giustificare la complessità della funzione
   c. Specificare il linguaggio di programmazione scelto e la definizione di `PNodeG`.

2. Sia H_1 un vettore di lunghezza *2n* contenente un max-heap di interi, di dimensione *n*, secondo lo schema visto a lezione (e nel libro di testo). Sia H_2 un vettore di lunghezza *n* contenente un max-heap di interi di lunghezza *n*, secondo lo schema visto a lezione (e nel libro di testo). Si consideri il problema di trasformare il vettore H_1 in un vettore **ordinato** contenente tutti gli elementi degli heap H_1 ed H_2, senza allocare altri vettori ausiliari.

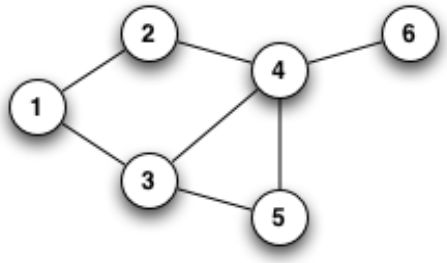
   a. Fornire lo pseudocodice di un algoritmo **efficiente** per risolvere il problema proposto utilizzando, tra le procedure viste a lezione, solamente max-heapify.
   b. Determinarne e giustificare la complessità.

3. Si stabilisca quale problema risolve il seguente algoritmo, dove $G = (V, E)$ rappresenta un grafo non orientato:

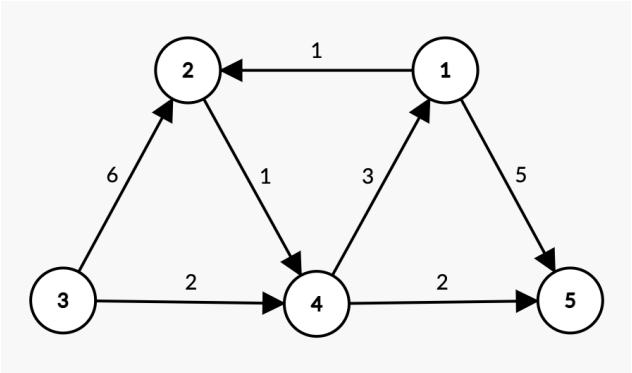
```
MyAlgorithm(G)
1. A = ∅
2. for each u ∈ V[G] do
3.   if MyFunction(G,A,u) then
4.      A = A ∪ {u}
5. return V[G] \ A                /* restituisci i vertici di G non appartenenti ad A */

MyFunction(G,A,u)
1. for each v ∈ A do
2.    if (u,v) ∈ E[G] then
3.      return FALSE
4. return TRUE
```

   Si dimostri la correttezza dell'algoritmo e si determini la sua complessità.

   Si simuli infine la sua esecuzione sul seguente grafo e si mostri (con alcuni esempi) che il risultato finale può dipendere dall'ordine in cui vengono estratti i vertici al passo 2 di `MyAlgorithm`. In particolare, si determini l'ordine di estrazione che consente all'algoritmo di restituire in uscita il massimo numero di vertici.



4. Si scriva l'algoritmo di Dijkstra, si derivi la sua complessità, si dimostri la sua correttezza e si simuli la sua esecuzione sul seguente grafo utilizzando il vertice 1 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 d e π, iterazione per iterazione:

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