Algoritmi e Strutture Dati

a.a. 2023/24

Compito del 16/01/2025

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. Scrivere l'algoritmo build-Max-Heap e simulare la sua esecuzione sull'array <-37, -9, 15, 8, 70>

2. Un algoritmo ricorsivo MyMST è in grado di determinare un albero di copertura minimo all'interno di un grafo pesato G. La sua complessità è pari a:

   $$T(m) = 9T(m/3) + 7m + \log m$$

   dove m rappresenta il numero di archi del grafo. Esistono algoritmi più efficienti di MyMST per risolvere il problema dato?

3. Sia $G = (V, E)$ un grafo non orientato e siano u e v due vertici collegati da un cammino p. Si stabilisca se le seguenti affermazioni sono vere o false, giustificando tecnicamente la risposta:
   a. "Se G è aciclico, allora non esiste un altro cammino tra u e v, oltre p."
   b. "Se G è connesso, allora esistono necessariamente altri cammini tra u e v, oltre p."
   c. "Se G è un albero, allora il numero di cammini tra u e v dipende dal numero di archi in G."

Algoritmi e Strutture Dati

a.a. 2023/24

Compito del 16/01/2025

Cognome: _____ Nome: _____

Matricola: _____ E-mail: _____

Parte II

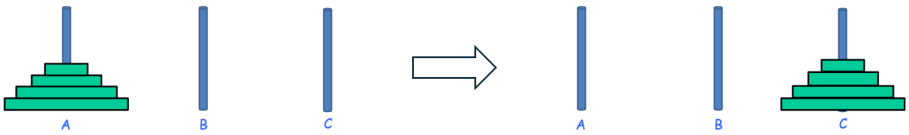
(2.5 ore; ogni esercizio vale 6 punti)

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

1. Dato un albero binario i cui nodi *x* hanno i campi **left**, **right** e **key**, dove **key** è un numero intero:
   a. definire l'**altezza** di un nodo *x*;
   b. scrivere una funzione **efficiente** in C o C++ che ritorna il numero di nodi per i quali la chiave x->key è minore o uguale dell'altezza del nodo. Il prototipo della funzione è:
   `int lessHeight(PNode r)`
   c. valutare la complessità della funzione, **indicando eventuali relazioni di ricorrenza e mostrando la loro risoluzione**;
   d. specificare il linguaggio di programmazione scelto.

2. Per ordinare l'array A[1..n], si ordina in modo ricorsivo il sottoarray A[1.. n-1] e poi si inserisce A[n] nel sottoarray ordinato A[1.. n-1].
   a. Scrivere lo pseudocodice per questa variante **ricorsiva** dell'insertion sort.
   b. Fornire una ricorrenza per il suo tempo di esecuzione nel caso peggiore e risolverla in modo formale.
   c. Quale è il tempo di esecuzione nel caso migliore? Mostrare un esempio di input che determina il caso migliore.

3. Il problema della *torre di Hanoi* è un classico rompicapo matematico nel quale sono dati tre paletti e un certo numero di dischi di grandezza diversa. Il gioco inizia ponendo i dischi su un paletto in ordine decrescente, in modo da formare un cono. Lo scopo del gioco è portare tutti i dischi su un paletto diverso potendo spostare solo un disco alla volta e potendo mettere un disco solo su un altro più grande, mai su uno più piccolo. Nella configurazione riportata in figura il numero di dischi è 4, il paletto di partenza è A e il paletto di arrivo è C. Il paletto B viene utilizzato come paletto di "appoggio".



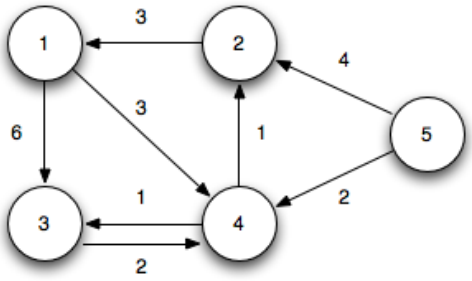
Il seguente algoritmo rappresenta una soluzione ricorsiva al problema:

```
procedure hanoi (n: integer; partenza, appoggio, arrivo:  palo);
begin
if (n = 1) then muoviDisco(partenza, arrivo)
else begin
   hanoi(n-1, partenza, arrivo, appoggio);
   muoviDisco(partenza, arrivo);
   hanoi(n-1, appoggio, partenza, arrivo);
end
```

dove `muoviDisco(partenza,arrivo)` è una semplice procedura che sposta il disco situato in cima al palo di partenza su quello di arrivo e ha complessità costante.

Qual è la complessità dell'algoritmo? Giustificare formalmente la risposta.

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										