

## Algoritmi e Strutture Dati

a.a. 2023/24

Compito del 29/05/2024

Cognome: \_\_\_\_\_

Nome:

Matricola: \_\_\_\_\_

E-mail:

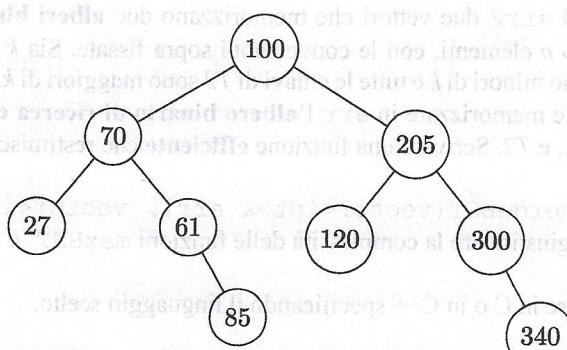
Parte I

(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.

**Esercizi non verranno varcati pienamente.** Con 303 Esercizi si è intenzionato di fornire un corso sia teorico che pratico risolvendo molti esercizi di tipo strutturale e di tipo applicativo.

1. Dato il seguente albero



- a) Eseguire una visita in preordine, una visita in ordine simmetrico e una visita in postordine elencando nei tre casi la sequenza dei nodi incontrati.
  - b) L'albero è un albero binario di ricerca? Giustificare la risposta.

2. La Prof.ssa Raffaetà sostiene di aver sviluppato un algoritmo di complessità

$$T(n) \equiv 7T(n/2) + n^2$$

per determinare tutti i cammini minimi in un grafo orientato e pesato, dove  $n$  rappresenta il numero di nodi del grafo in ingresso. Si dica se l'algoritmo in questione è preferibile o meno all'algoritmo di Floyd-Warshall e perché.

3. Il Prof. Pelillo sostiene di aver sviluppato un algoritmo di complessità

$$T(n) = 8T(n/2) + n^3$$

per determinare una clique massima in un grafo, dove  $n$  rappresenta il numero di nodi del grafo in ingresso. È plausibile che l'algoritmo in questione sia corretto? Perché?

# Algoritmi e Strutture Dati

a.a. 2023/24

## Compito del 29/05/2024

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. Sia arr un vettore di interi di dimensione n. Si assuma che nel vettore arr siano stati inseriti numeri interi positivi provenienti da un **albero binario di ricerca completo T**. In particolare, gli elementi di T sono stati inseriti in arr usando la stessa convenzione che si usa normalmente per la memorizzazione di uno heap in un vettore.
  - a) Scrivere una funzione **efficiente** int maxBST(vector<int>& arr) che restituisce il massimo di arr.
  - b) Siano arr1 ed arr2 due vettori che memorizzano due **alberi binari di ricerca completi T1 e T2** aventi ciascuno n elementi, con le convenzioni sopra fissate. Sia k una chiave intera tale che tutte le chiavi di T1 sono minori di k e tutte le chiavi di T2 sono maggiori di k. Sia arr un vettore di dimensione 2n + 1. Si vuole memorizzare in arr l'**albero binario di ricerca completo T** che si otterrebbe dalla fusione di T1, k, e T2. Scrivere una funzione **efficiente** che restituisce arr. La funzione ha il seguente prototipo:  
vector<int> mergeBST(vector<int>& arr1, vector<int>& arr2, int val)
  - c) Determinare e giustificare la complessità delle funzioni maxBST e mergeBST.

Le funzioni devono essere in C o in C++ specificando il linguaggio scelto.

2. Per iscriversi agli esami gli studenti devono usare il numero di matricola che è rappresentato da una stringa di 7 caratteri con il seguente formato "S#####", dove "#" indica una cifra. Scrivere una procedura **efficiente** per ordinare in modo **decrescente** l'elenco degli iscritti a un esame in base al numero di matricola.  
Il prototipo della procedura è:  
void riordina(vector<string>& arr)

**Valutare e giustificare** la complessità della procedura proposta.

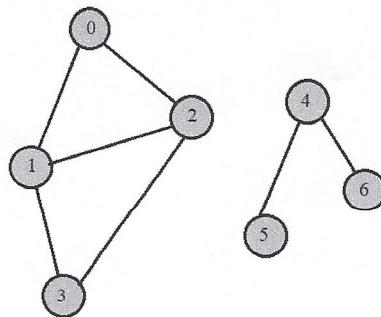
3. Si consideri il seguente algoritmo, che accetta in ingresso un grafo non orientato  $G = (V, E)$ :

```
MyAlgorithm(G)
1. A = ∅
2. for each vertex u ∈ V[G] do
3.     MAKE-SET(u)
4. for each edge (u,v) ∈ E[G] do
5.     if FIND-SET(u) ≠ FIND-SET(v) then
6.         UNION(u,v)
7.         A = A U {(u,v)}
8. if |A| = n - 1 then
9.     return TRUE
10. else
11.     return FALSE
```

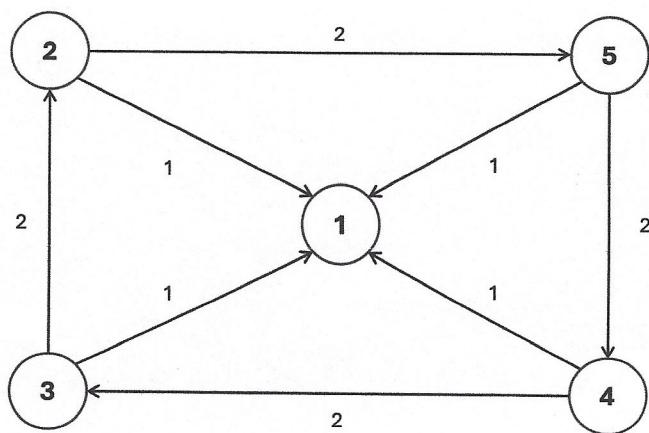
dove  $|A|$  denota il numero di elementi (cardinalità) dell'insieme A.

In quale caso MyAlgorithm restituisce TRUE? In questo caso, cosa conterrà l'insieme A alla fine dell'algoritmo? (Giustificare formalmente le risposte.)

Si simuli inoltre accuratamente l'esecuzione dell'algoritmo sul grafo seguente, mostrando l'evoluzione delle strutture dati coinvolte passo dopo passo:



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:

- si indichi l'ordine con cui vengono estratti i vertici
- si riempia la tabella seguente con i valori dei vettori  $d$  e  $\pi$ , iterazione per iterazione:

	vertice 1		vertice 2		vertice 3		vertice 4		vertice 5	
	$d[1]$	$\pi[1]$	$d[2]$	$\pi[2]$	$d[3]$	$\pi[3]$	$d[4]$	$\pi[4]$	$d[5]$	$\pi[5]$
dopo inizializzazione	0	NIL	$\infty$	NIL	$\infty$	NIL	$\infty$	NIL	$\infty$	NIL
1 iterazione 1	0	NIL	-1	1	-1	1	-1	1	-1	1
2 iterazione 2	0	NIL	-1	1	-3	2	-1	1	1	2
3 iterazione 3	0	NIL	-1	1	-3	2	-5	3	1	2
4 iterazione 4	0	NIL	-1	1	-3	2	-5	3	-7	4
5 iterazione 5	0	NIL	-1	1	-3	2	-5	3	-7	4

## PART I

1) PREORDINE: 100 - 70 - 27 - 61 - 85 - 205 - 120 - 300 - 340

SIMMETRICO: 27 - 70 - 61 - 85 - 100 - 120 - 205 - 300 - 340

POSTORDINE: 27 - 85 - 61 - 70 - 120 - 340 - 300 - 205 - 100

NO BST, 61 < 70 MA STA ALLA SUA DESTRA DOVE CI SONO I NODI CON CHIAVE MAGGIORE IN UN BST.

2)  $T(n) = 7 T\left(\frac{n}{2}\right) + n^2$

$\alpha = 7 \quad b = 2 \quad d = \log_2 7 \approx 2,81$

$f(n) \approx g(n) \quad f(n) = \Theta(n^d)$

2° CASO  
 $T(n) = \Theta(n^d \log n) = \Theta(n^2 \log n)$

FLOYD-WARSHALL =  $\Theta(n^3)$

T(n) è preferibile asintoticamente a FLOYD-WARSHALL

3)  $T(n) = 8 T\left(\frac{n}{2}\right) + n^3$

$\alpha = 8 \quad b = 2 \quad \log_2 8 = 3 \quad g(n) = n^3$

$f(n) \approx g(n)$

2° CASO

$f(n) = \Theta(n^d) \Rightarrow n^3 = \Theta(n^3)$

$T(n) = \Theta(n^d \log n) = \Theta(n^3 \log n)$

Determinare una clique massima in un grafo è un problema NP-completo. Gli algoritmi esistenti per questo problema hanno complessità esponenziale nel caso generale, non polinomiale o quasi-polinomiale come quelle calcolate. Quindi non è plausibile che l'algoritmo sia corretto.

## PART II

```
1) int max BST (vector<int>& arr) {
    return arr.at (arr.size () - 1)
}
```

$$T(n) = \Theta(1)$$

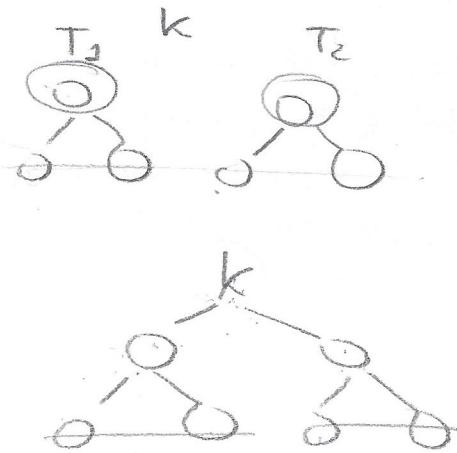
```
vector<int> mergeBST (vector<int>& arr1, vector<int>& arr2, int val) {
    vector<int> ris;
    ris.push_back (val);
    size_t sx = 0, j = 1, dx = 0;
    [ ] O(1)
    while (sx < arr1.size ()) {
        ← OGNI ELEMENTO DI ARR1 e ARR2 È INSERITO
        UNA SOLA VOLTA IN RIS
        size_t end = sx + j;
        while (sx < end) {
            ris.push_back (arr1[sx]);
            sx++;
            ← NON C'È RISOLUZIONE
        }
        while (dx < end) {
            ris.push_back (arr2[dx]);
            dx++;
        }
        j *= 2;
    }
    return ris;
}
```

$$T(n) = \Theta(2m) = \Theta(n)$$

### ALTRA SOLUZIONE

```
vector<int> mergeBST (vector<int>& arr1, vector<int>& arr2, int val) {
    vector<int> arr;
    arr.reserve (arr1.size () + arr2.size () + 1);
    arr.insert (arr.end (), arr1.begin (), arr1.end ());
    arr.push_back (val);
    arr.insert (arr.end (), arr2.begin (), arr2.end ());
    return arr;
}
```

$$T(n) = O(n)$$



(3)

$\text{FIND-SET}(4) \neq \text{FIND-SET}(6)$

$\text{UNION}(4,6): \{0,1,2,3\} \cup \{4,5,6\}$

$A = \{(0,1), (0,2), (1,3), (4,5), (4,6)\}$

Poiché  $|A| < m-1 = 6$ , l'algoritmo restituisce FALSE

Il grafo è disconnesso e non è possibile costruire uno spanning tree per tutta il grafo. A contiene uno spanning tree per ciascuna componente连通的 del grafo.

4) Dijkstra ( $G, w, s$ )

$Q = V[G]$

$\text{INIT-SS}(G, s)$

$S = \emptyset$

while  $Q \neq \emptyset$

$u = \text{EXTRACT-MIN}(Q)$

$S = S \cup \{u\}$

for each  $v$  in  $\text{adj}[u]$

$\text{RELAX}(u, v, w(u, v))$

return  $(d, G_T)$

$\text{RELAX}(G, u, v, w(u, v))$

if  $d[v] > d[u] + w(u, v)$

$d[v] = d[u] + w(u, v)$

CORRETTITÀ

COSA SI INTENDE DI MOSTRARE

Alla fine dell'algoritmo:  $d[u] = S(s, u) \quad \forall u \in V$

$G_T$  è un albero di cammini minimi

PROCEDIMENTO

Dimostriamo solamente la prima parte.

Per assurdo  $\exists u \in V$  tale che al momento dell'estrazione da  $Q$ ,  $d[u] \neq S(s, u)$  e questo accade per la prima volta.

1)  $M \neq S$  perché dopo la INIT-SS sicuramente  $d[s] = S(s, s)$

2) Quindi, al momento di estrarre  $u$ , l'insieme  $S$  dei vertici già esbatti sarà  $\neq \emptyset$  perché almeno vi sarà il vertice  $s$

3) Faccio un disegno con l'insieme  $S$  contenente i vertici  $s, x$  collegati da un cammino minimo. Disegno l'insieme  $Q = V \setminus S$  con i vertici  $y, u$  collegati da un cammino minimo.

Disegno l'arco  $(x, y)$

a) CASO 1:  $s$  non raggiunge  $u$ , ma allora  $S(s, u) = \infty = d[u]$  dalla INIT

b) CASO 2: cammino minimo tra  $s, u$

4) Al momento di estrarre  $x$ , per ipotesi  $d[x] = \delta(y, x)$

5) Dopo la RELAX  $(u, v, w(u, v))$ ,  $d[y] = \delta(s, y)$

6) Ora però poniamo Dijkstra eseguirà il vertice  $u$ , sarà quindi vero che  $d[u] \leq d[v]$

7)  $\delta(s, y) \leq \delta(s, u)$ . Per ipotesi  $s, \dots, y$  era un cammino minimo. Essendo da  $y$  a  $u$  pesi  $\geq 0$  allora per certo  $\delta(s, y) \leq \delta(s, u)$

8)  $\delta(s, u) \leq d[u]$  LIMITE INFERIORE

(2)

```

2) void noddine (vector<string>& arr)
    int dim = arr.size();
    vector<string> aux(dim);
    for (size_t d=6; d>=1; d--) {
        if (d%2 == 0)
            countingsort(arr, aux, d)
        else
            countingsort(aux, arr, d)
    }
}

```

```

void countingsort (vector<string>& tab, vector<string>& out, size_t d) {
    size_t dim = tab.size();
    vector<int> occ(10, 0);  $\Theta(1)$ 
    for (size_t i=0; i < dim; i++)  $\Theta(n)$ 
        occ[tab[i][d] - '0']
    for (int i=8; i >= 0; i--)  $\Theta(1)$ 
        occ[i] += occ[i+1]
    for (size_t i=dim-1; i >= 0; i--) {
        out[occ[tab[i][d] - '0'] - 1] = tab[i];  $\Theta(n)$ 
        occ[tab[i][d] - '0']--;
    }
}

```

$$T(m) = \Theta(6m) \Rightarrow \Theta(m)$$

### 3) QUANDO RESTITUISCE TRUE?

- Restituisce TRUE quando il grafo è connesso, poiché un grafo connesso e senza cicli ha esattamente  $m-1$  archi in un albero di copertura (spanning tree).  
La condizione  $|A|=m-1$  verifica se il grafo è connesso  
(Modifica di Kruskal?)
  - COSA CONTIENE A IN QUEL CASO?  
A contiene gli archi di uno spanning tree del grafo, ossia un sottoinsieme di  $|E|$  archi che collega tutti i  $m$  vertici senza cicli
- DIMOSTRAZIONE FORMALE
- Ogni vertice viene inizialmente isolato ( $\text{MAKE-SET}(u)$ )
  - Durante l'iterazione su ogni arco  $(u,v)$ , l'algoritmo verifica se  $u$  e  $v$  sono già connessi tramite  $\text{FIND-SET}(u) \neq \text{FIND-SET}(v)$
  - Se non sono connessi, li unisce  $\text{UNION}(u,v)$  e aggiunge  $(u,v)$  a A.
  - Se G è connesso, dopo aver processato tutti gli archi, l'insieme A contenrà  $m-1$  archi che collegano tutti i nodi
  - Se G non è connesso, alcuni vertici rimarranno isolati e l'insieme A avrà meno di  $m-1$  archi. In questo caso, l'algoritmo restituisce FALSE

$\text{MAKE-SET}(u) : \{\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}\}$

$\text{FIND-SET}(0) \neq \text{FIND-SET}(1)$

$\text{UNION}(0,1) : \{\{0,1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}\}$

$A = \{(0,1)\}$

$\text{FIND-SET}(0) \neq \text{FIND-SET}(2)$

$\text{UNION}(0,2) : \{\{0,1,2\}, \{3\}, \{4\}, \{5\}, \{6\}\}$

$A = \{(0,1), (0,2)\}$

$\text{FIND-SET}(1) = \text{FIND-SET}(2) \rightarrow \text{NESSUNA AZIONE}$

$\text{FIND-SET}(1) \neq \text{FIND-SET}(3)$

$\text{UNION}(1,3) : \{\{0,1,2,3\}, \{4\}, \{5\}, \{6\}\}$

$A = \{(0,1), (0,2), (1,3)\}$

$\text{FIND-SET}(2) = \text{FIND-SET}(3) \rightarrow \text{NESSUNA AZIONE}$

$\text{FIND-SET}(4) \neq \text{FIND-SET}(5)$

$\text{UNION}(4,5) : \{\{0,1,2,3\}, \{4,5\}, \{6\}\}$

$A = \{(0,1), (0,2), (1,3), (4,5)\}$