

**Algoritmi e Strutture Dati**  
a.a. 2020/21

**Compito del 26/08/2021**

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

Nome: \_\_\_\_\_

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

E-mail: \_\_\_\_\_

**Parte I**  
(30 minuti; ogni esercizio vale 2 punti)

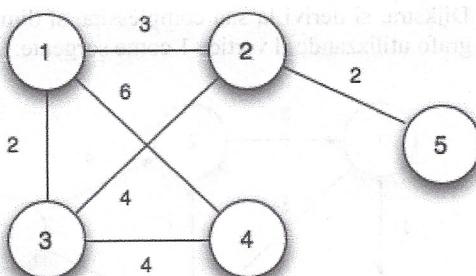
**Avvertenza:** Si giustifichino tecnicamente tutte le risposte. In caso di discussioni poco formali o approssimative gli esercizi non verranno valutati pienamente.

1. Scrivere l'algoritmo build-Min-Heap e simulare la sua esecuzione sull'array  $\langle 35, 42, 2, -1, 27 \rangle$ .
2. Un algoritmo ricorsivo  $\mathcal{A}$  determina i cammini minimi tra tutte le coppie di vertici in un grafo pesato e ha complessità pari a:

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

dove  $n$  rappresenta il numero di vertici del grafo. Si stabilisca se  $\mathcal{A}$  è asintoticamente più efficiente dell'algoritmo di Floyd-Warshall.

3. Si supponga di eseguire l'algoritmo di Prim sul seguente grafo, utilizzando il vertice 1 come sorgente:



- a) Con quale ordine verranno estratti i vertici?
- b) Si determini l'albero di copertura minimo restituito dall'algoritmo.

# Algoritmi e Strutture Dati

a.a. 2020/21

## Compito del 26/08/2021

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

Nome: \_\_\_\_\_

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

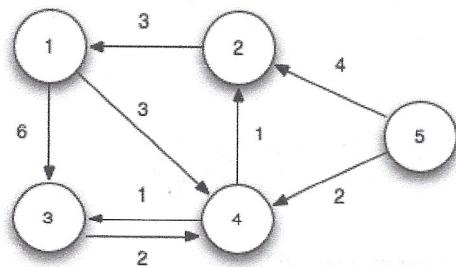
E-mail: \_\_\_\_\_

### 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. Si consideri un albero binario  $T$ , i cui nodi  $x$  hanno i campi  $left$ ,  $right$ ,  $p$  e  $key$  che rappresentano il figlio sinistro, il figlio destro, il padre e la chiave del nodo, rispettivamente. Un cammino è una sequenza di nodi  $x_0, x_1, \dots, x_n$  tale che per ogni  $i = 0, \dots, n-1$  vale  $x_{i+1} \rightarrow p = x_i$ . Il cammino è detto *terminabile* se  $x_n \rightarrow l = NULL$  oppure  $x_n \rightarrow r = NULL$ . Diciamo che l'albero è *3-bilanciato* se tutti i cammini terminabili che partono dalla radice di  $T$  hanno lunghezze che differiscono al più di 3.
  - a. Scrivere una funzione **efficiente** in C  $bal(u)$  che dato in input la radice  $u$  di un albero  $T$  verifica se è *3-bilanciato* e ritorna 1 se  $T$  è *3-bilanciato*, 0 altrimenti.
  - b. Valutare la complessità della funzione, indicando eventuali relazioni di ricorrenza.
2. Si scriva un algoritmo **efficiente** che, ricevuto in ingresso un insieme di  $n$  intervalli chiusi  $[a_i, b_i]$ , con  $a_i, b_i$  numeri interi,  $a_i \leq b_i$  e  $1 \leq i \leq n$ , stabilisca se la loro unione è un intervallo (nel qual caso, restituisca tale intervallo) o meno.  
Calcolare e giustificare la complessità dell'algoritmo proposto.  
Si devono scrivere le eventuali procedure/funzioni ausiliarie utilizzate.
3. 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  $\pi$ , 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	$\infty$
1	iterazione 1	0	NIL	$\infty$	NIL	6	1	3	1	$\infty$
4	iterazione 2	0	NIL	4	4	4	3	1	$\infty$	NIL
3	iterazione 3	0	NIL	4	4	4	3	1	$\infty$	NIL
2	iterazione 4	0	NIL	4	4	4	3	1	$\infty$	NIL
5	iterazione 5	0	NIL	4	4	4	3	1	$\infty$	NIL

4. Si definiscano le classi di complessità P, NP ed NPC e si enunci e dimostri il teorema fondamentale della NP-completezza. Si formuli inoltre il problema CLIQUE e si mostri che è NP-completo.

## PARTE I

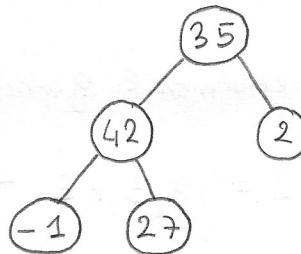
1) build Min Heap (Array A) {

A.heapsize = A.length

for  $i = \text{Floor}(A.length/2)$  DOWN TO 1  
min-heapify(A, i)

}

A	35	42	2	-1	27
	1	2	3	4	5



	35	42	2	-1	27
	1	2	3	4	5
↑					

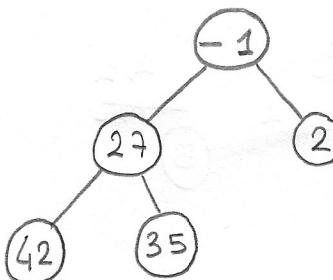
$i = 2$

	35	-1	2	42	27
	1	2	3	4	5
↑					

$i = 1$

	-1	35	2	42	27
	1	2	3	4	5
↑					

$i = 1$



$$T(m) = O(m)$$

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

$$a=3 \quad b=2 \quad d=\log_2 3 \approx 1,5 \dots < 2 \quad f(n)=n^2 \quad g(n)=n^{\log_2 3}$$

$$g(n) < f(n)$$

3° CASO DEL TEOREMA MASTER

$$f(n) = \Omega(n^{d+\varepsilon}) \quad \exists \varepsilon > 0$$

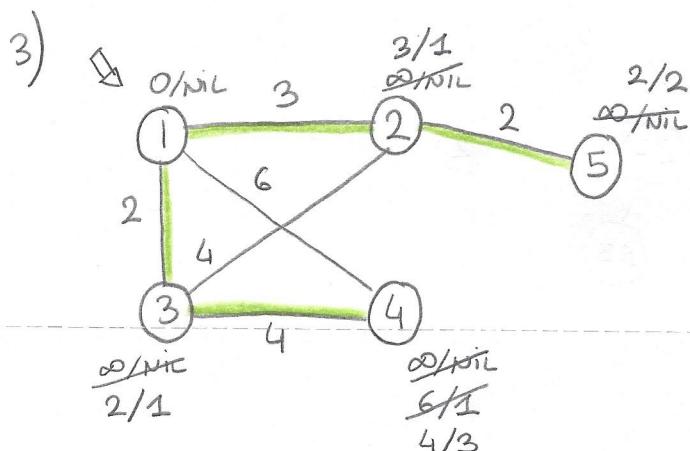
$$\exists c < 1 \exists \text{ per } n \text{ sufficientemente grande } \alpha f\left(\frac{n}{b}\right) \leq c \cdot f(n)$$

$$n^2 = \Omega\left(n^{\log_2 3 + \varepsilon}\right) \Rightarrow 2 = \log_2 3 + \varepsilon \Rightarrow \varepsilon = 2 - \log_2 3 > 0$$

$$3\left(\frac{n}{2}\right)^2 \leq c n^2 \Rightarrow 3 \frac{n^2}{4} \leq c n^2 \Rightarrow \frac{3}{4} n^2 \leq c n^2 \Rightarrow c \geq \frac{3}{4} \Rightarrow \frac{3}{4} \leq c < 1$$

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

A NON È ASINTOTICAMENTE PIÙ EFFICIENTE DI FLOYD-WARSHALL POICHÉ LA COMPLESSITÀ DI FW È  $T(n) = \Theta(n^3)$ .



1 - 3 - 2 - 5 - 4

## PART II

```

1) typedef struct Node {
    int key;
    Node* left;
    Node* right;
    Node* p;

    Node(int k, Node* padre=nullptr, Node* sx=nullptr, Node* dx=nullptr) {
        key=k;
        p=padre;
        left=sx;
        right=dx;
    }
} * PNode;

```

```

typedef struct Tree {
    PNode root;
    Tree(PNode r=nullptr) : root(r) {}
} * PTree;

```

```

bool tre_bil(PNode u) {
    int min, max;
    tre_bil(u, min, max);
    return (max - min <= 3);
}

```

```

void tre_bil(u, int &min, int &max) {
    int minh, maxh, minsx, maxsx, mindx, maxdx;
    if (u == nullptr) {
        min = -1;
        max = -1;
    } else {
        tre_bil(u->left, minsx, maxsx);
        tre_bil(u->right, mindx, maxdx);
        min = (minsx <= mindx ? minsx : mindx) + 1;
        max = (maxsx <= maxdx ? maxsx : maxdx) + 1;
    }
}

```

$$T(m) = \begin{cases} c & \text{se } k \leq 1 \\ T(k) + T(m-k-1) + d & \text{se } k > 1 \end{cases} \Rightarrow T(m) = \Theta(m)$$

2) bool unione (std::vector<std::pair<int, int>>& arr, std::pair<int, int> & ns) {  
 size\_t i;  
 int endtemp;  
 mergesort(arr, {}, arr.size());  
 endtemp = arr[0].second;  
 i = 1;  
 while (i < arr.size() && endtemp + 1 ≥ arr[i].first) {  
 if (arr[i].second > endtemp) {  
 endtemp = arr[i].second;  
 }  
 i++;  
 }  
 if (i == arr.size()) {  
 ns = {arr[0].first, endtemp};  
 return true;  
 }  
 return false;  
}

3) DIJKSTRA ( $G, \omega, s$ )

INIT-SS ( $G, s$ )

$Q \leftarrow V[G]$

$S \leftarrow \emptyset$

while  $Q \neq \emptyset$

$u \leftarrow \text{EXTRACT MIN}(Q)$

$S \leftarrow S \cup \{u\}$

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

RELAX ( $u, v, \omega(u, v)$ )

return ( $d, G_T$ )

INIT-SS ( $G, s$ )

for each  $u \in V[G]$

$d[u] = +\infty$

$\pi[u] = \text{NIL}$

$d[s] = 0$

RELAX ( $u, v, \omega(u, v)$ )

if  $d[v] > d[u] + \omega(u, v)$  then

$d[v] = d[u] + \omega(u, v)$

$\pi[v] = u$

3) T(DISKSTRA) :

• HEAP  $O(m \log n)$  :- GRAFO SPARSO :  $m \approx n = O(n \log n)$ - GRAFO DENSO :  $m \approx n^2 = O(n^2 \log n)$ 

• ARRAY :

- GRAFO SPARSO :  $m \approx n = O(n^2)$ - GRAFO DENSO :  $m \approx n^2 = O(n^2)$ 

## CORRETTEZZA DI DISKSTRA

Sia  $G = (V, E)$  un grafo orientato pesato sugli archi, cioè con  $w : E \rightarrow \mathbb{R}$  tale che  $\forall (u, v) \in E : w(u, v) \geq 0$ .

Allora, alla fine dell'algoritmo di Dijkstra, si ha :

1)  $\forall v \in V : d[v] = S(s, v)$ 2)  $G_T$  è un albero di cammini minimi

Dimostriremo la correttezza del primo punto attraverso la dimostrazione di un'affermazione più forte ma equivalente, cioè che per ogni  $u \in V$ , al momento dell'estrazione di  $u$ , risulta  $d[u] = S(s, u)$ . La dimostrazione avviene per assurdo e si avvale di 9 osservazioni.

## DIMOSTRAZIONE

Supponiamo per assurdo che esista un vertice  $u \in V$  tale che al momento della sua estrazione  $d[u] \neq S(s, u)$ , e che  $u$  sia il primo vertice per cui questo accade.

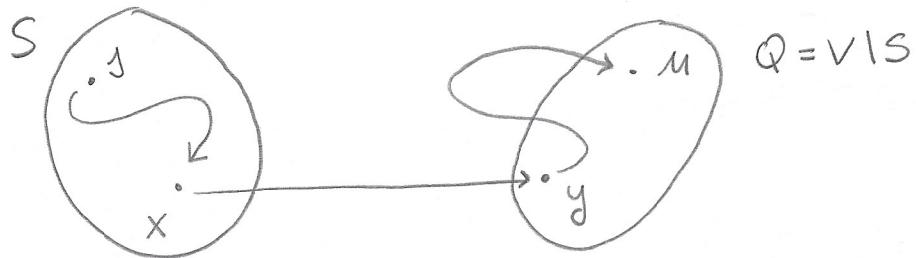
## OSSERVAZIONI

1)  $u$  non può essere la sorgente: dopo  $\text{INIT\_ss}(G, s)$ ,  $d[s] = 0 = S(s, s)$ , e  $S(s, s)$  non può valere  $-\infty$  perché per ipotesi non ci sono archi con pesi negativi, quindi neanche cicli negativi.

2) Al momento dell'estrazione di  $u$ ,  $S \neq \emptyset$ , perché in  $S$  ci sarà almeno la sorgente  $s$ .

3)  $u$  non è raggiungibile dalla sorgente: se così fosse,  $S(s, u) = +\infty = d[u]$ , che sarebbe corretto e non violerebbe la proprietà che abbiamo voluto violare per assurdo; quindi,  $S(s, u) \neq +\infty$

4) Ci poniamo nell'istante in cui  $S$  è già stato esteso ( $S \subseteq S$ ) ma  $u$  no ( $u \in Q = V \setminus S$ ). Per il punto precedente, esiste un cammino minimo  $p$  tra  $s$  e  $u$ : sia  $(x, y)$  un arco appartenente a  $p$  che attraversa il leglio, cioè tale che  $x \in S$  e  $y \in Q$ .



5) Per ipotesi, vale che  $d[x] = \delta(s, x)$ : infatti,  $u$  è il primo vertice per cui questa proprietà non vale.

6) Poiché siamo su un cammino minimo, possiamo applicare la proprietà della convergenza: al momento dell'estrazione di  $x$ , applichiamo la RELAX su  $y$  e otteniamo che  $d[y] = \delta(s, x) + \omega(x, y) = \delta(s, y)$

7) Dal momento che stiamo per estrarre il nodo  $u$ , siccome l'algoritmo di Dijkstra estrae il vertice avente campo  $d$  più piccolo, dovrà valere che  $d[u] \leq d[y]$

8) Non può accadere che  $\delta(s, y) > \delta(s, u)$ : in virtù del fatto che ci troviamo in un cammino minimo e che i pesi sono tutti maggiori o uguali a zero, allora  $\delta(s, y) \leq \delta(s, u)$

9) Per la proprietà del limite inferiore, vale sicuramente che  $\delta(s, u) \leq d[u]$

Ricostuiamo l'assurdo sulla base delle precedenti osservazioni:

$$\begin{aligned} \delta(s, u) &\leq d[u] && \text{PER OSS. 9} \\ &\leq d[y] && \text{PER OSS. 7} \\ &= \delta(s, y) && \text{PER OSS. 6} \\ &\leq \delta(s, u) && \text{PER. OSS. 8} \end{aligned}$$

Avvalendoci delle osservazioni, siamo riusciti a "rinchiedere" il valore  $d[u]$  tra  $\delta(s, u)$  e  $\delta(s, u)$ :  $\delta(s, u) \leq d[u] \leq \delta(s, u) \Leftrightarrow d[u] = \delta(s, u)$

che è assurdo in quanto andiamo ad invalidare l'ipotesi che  $d[u] \neq \delta(s, u)$

4)  $P = \{\Theta \mid \Theta \text{ è un problema decisionale per il quale } \exists \text{ algoritmo POLINOMIALE che risolve il problema}\}$

$NP = \{\Theta \mid \Theta \text{ è un problema decisionale per cui } \exists \text{ algoritmo POLINOMIALE DI VERIFICA per } \Theta\}$

$NPC = \{\Theta \mid \Theta \in NPC \text{ è un problema decisionale tale che:}$

- 1)  $\Theta \in NP$
- 2)  $\forall \Theta' \in NP : \Theta' \leq_p \Theta\}$

### TEOREMA FONDAMENTALE DELLA NP COMPLETITÀ

$$P \cap NPC \neq \emptyset \Rightarrow P = NP$$

#### DIMOSTRAZIONE

Per ipotesi:  $\exists \Theta \in NPC \wedge P$ , cioè  $\Theta \in NPC \wedge \Theta \in P$ . Bisogna dimostrare che:

1)  $P \subseteq NP$ , ma questo è banalmente vero: un problema risolvibile polinomialmente è anche verificabile polinomialmente

2)  $NP \subseteq P$

Sia  $Q$  un problema in  $NP$ . Allora, dovrà valere che  $\forall Q \in NP : Q \in P$ .

Essendo  $Q \in NP$  e  $\Theta \in NPC$  per ipotesi, allora:  $\underbrace{Q \leq_p \Theta}_{\substack{\text{PER IPOTESI} \\ \text{DEF. NPC}} \in P}$

Il fatto che, per definizione di  $NPC$ ,  $Q$  è riducibile a  $\Theta$ , che per ipotesi appartiene a  $P$ , vuol dire che esiste un algoritmo polinomiale che traduce le istanze di  $Q$  in istanze di  $\Theta$ , che può essere risolto polinomialmente. Significherebbe che  $Q$  si può risolvere in tempo polinomiale, cioè  $Q \in P$ , come volevasi dimostrare.