

# Algoritmi e Strutture Dati

a.a. 2022/23

## Compito del 5/6/2023

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 BST di dimensione  $n$  e sia  $k$  una chiave di  $T$  tale che  $T \rightarrow \text{root} \rightarrow \text{key} < k$ . Si vuole costruire un **nuovo** BST  $T'$  contenente tutte e sole le chiavi di  $T$  appartenenti all'intervallo  $[T \rightarrow \text{root} \rightarrow \text{key}, k]$ .
  - a. Si scriva una funzione **efficiente** in C o C++ per risolvere il problema. Il prototipo della funzione è:  
PTree creaBSTInterval(PTree t, int k)
  - b. Valutare e giustificare la complessità della funzione.
  - c. Specificare il linguaggio di programmazione scelto e la definizione di PTree.
2. Scrivere un algoritmo che date due sequenze  $X$  e  $Y$ , rispettivamente di  $m$  e  $n$  caratteri, restituisca la lunghezza di una sottosequenza di lunghezza massima comune a  $X$  e  $Y$  (LCS). Più precisamente:
  - a. dare una caratterizzazione ricorsiva della lunghezza di una LCS, definendo  $\text{lung}[i, j]$  come la lunghezza di una LCS delle sequenze  $X^i$  e  $Y^j$  con  $0 \leq i \leq m$  e  $0 \leq j \leq n$ . Si ricordi che  $X^i$  è il prefisso di  $X$  di lunghezza  $i$  e  $Y^j$  è il prefisso di  $Y$  di lunghezza  $j$ ;
  - b. tradurre tale definizione in un algoritmo di programmazione dinamica con il metodo **top-down** che determina la lunghezza di una LCS;
  - c. valutare e giustificare la complessità dell'algoritmo.
3. Il **diametro** di un grafo pesato è definito come la *massima distanza tra due vertici* del grafo, dove, come è noto, per “distanza tra due vertici”  $u$  e  $v$  si intende la lunghezza del cammino minimo tra  $u$  e  $v$ . Si sviluppi un algoritmo efficiente (ovvero polinomiale) che accetti in ingresso un grafo  $G$  e ne calcoli il suo diametro (si assuma che  $G$  non contenga cicli negativi). Si dimostri la correttezza dell'algoritmo proposto e si determini la sua complessità computazionale.  
**Nota.** Nel caso in cui si faccia uso di algoritmi noti si dimostri anche la loro correttezza.
4. Si scriva l'algoritmo di Dijkstra, si dimostri la sua correttezza, si fornisca la sua complessità computazionale e si simuli accuratamente la sua esecuzione sul seguente grafo rappresentato mediante lista di adiacenza (utilizzando il vertice 0 come sorgente):

# Algoritmi e Strutture Dati

a.a. 2022/23

## Compito del 5/6/2023

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. Sia  $H$  un vettore i cui elementi costituiscono un **Max-heap** contenente  $n$  interi. Si indichi quali tra le seguenti affermazioni sono corrette, **motivando brevemente le risposte**:

- a. la ricerca di un intero  $k$  in  $H$  ha costo  $O(\log n)$ ;
- b.  $H$  può contenere chiavi ripetute;
- c. la chiave minima si trova sicuramente in una foglia;
- d. la chiave minima si trova sicuramente in  $H[\text{heapsize}[H]]$ .

2. Un algoritmo ricorsivo  $\mathcal{A}$  accetta in ingresso una costante  $k$  e un grafo  $G$  connesso non orientato e pesato, e restituisce TRUE se esiste in  $G$  un albero di copertura di peso minore o uguale a  $k$  e FALSE in caso contrario. La sua complessità è data da

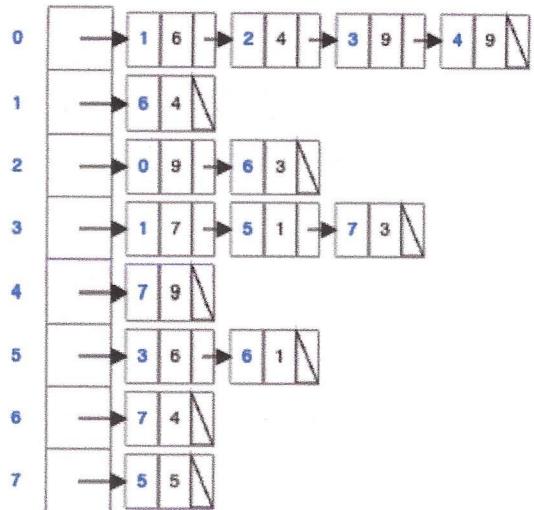
$$T(m) = 16T\left(\frac{m}{4}\right) + 5m + 7$$

dove  $m$  rappresenta il numero di archi di  $G$ . Esistono algoritmi più efficienti di  $\mathcal{A}$  per risolvere il problema dato?

3. Sia dato un grafo non orientato connesso  $G = (V, E, w)$  con pesi positivi e si supponga di eseguire l'algoritmo di Kruskal su  $G$  sostituendo i pesi originali come segue:

- a)  $w(u,v) \rightarrow 2 + w(u,v)$
- b)  $w(u,v) \rightarrow 2w(u,v)$
- c)  $w(u,v) \rightarrow 1 / w(u,v)$
- d)  $w(u,v) \rightarrow \log w(u,v)$
- e)  $w(u,v) \rightarrow w(u,v)^2$

In quali casi si otterranno gli stessi risultati del grafo originale? Perché?



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 0		vertice 1		vertice 2		vertice 3		vertice 4		vertice 5		vertice 6		vertice 7		
	$d$	$\pi$															
dopo INIT SS	0	NIL	00	NIL													
0 iter 1	0	NIL	6	0	4	0	9	0	9	0	0	00	NIL	00	NIL	00	NIL
2 iter 2	0	NIL	6	0	4	0	9	0	9	0	0	00	NIL	7	2	00	NIL
1 iter 3	0	NIL	6	0	4	0	9	0	9	0	0	00	NIL	7	2	00	NIL
6 iter 4	0	NIL	6	0	4	0	9	0	9	0	0	00	NIL	7	2	11	6
3 iter 5	0	NIL	6	0	4	0	9	0	9	0	10	3	7	2	11	6	
4 iter 6	0	NIL	6	0	4	0	9	0	9	0	10	3	7	2	11	6	
5 iter 7	0	NIL	6	0	4	0	9	0	9	0	10	3	7	2	11	6	
7 iter 8	0	NIL	6	0	4	0	9	0	9	0	10	3	7	2	11	6	

## PART I

- 1) a) La ricerca di un intero  $k$  in  $H$  non ha costo  $O(\log n)$  ma  $O(n)$  perché nel caso peggiore dobbiamo scorrere tutto l'array per trovare l'elemento.
- b) Si  $H$  può contenere chiavi ripetute
- c) La chiave minima è sicuramente una foglia
- d) Il valore minimo non si trova sicuramente in  $H[\text{heapsize}[H]]$  perché un max heap non è necessariamente ordinato in modo decrescente

$$2) T(m) = 16 T\left(\frac{m}{4}\right) + 5m + 7$$

$$\log_4 16 = 2 \rightarrow g(m) = m^2 \quad f(m) = 5m + 7 = 5m$$

$$f(m) \leq g(m)$$

1° CASO DEL TEOREMA MASTER

$$f(m) = O(m^{d-\varepsilon}) \quad \exists \varepsilon > 0$$

$$5m = m^{2-\varepsilon} \rightarrow 1 = 2 - \varepsilon \rightarrow \varepsilon = 1 > 0 \vee$$

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

Esistono algoritmi più efficienti per questo problema?

Il problema chiede se esiste un albero di copertura di peso  $\leq k$ .

Questo è equivalente a verificare il peso dell'MST di  $G$  è  $\leq k$ .

Algoritmi noti per MST

KRUSCAL:  $O(m \log m)$

PRIM:  $O(m \log m)$

ENTRAMBI SONO ASINTOTICAMENTE PIÙ EFFICIENTI DELL'ALGORITMO PROPOSTO

3)

a) Se  $w(e_1) < w(e_2)$ , allora  $2+w(e_1) < 2+w(e_2)$ 

L'ordinamento è preservato, MST INVARIATO

b) Se  $w(e_1) < w(e_2)$ , allora  $2w(e_1) < 2w(e_2)$ 

L'ordinamento è preservato, MST INVARIATO

c) Se  $w(e_1) < w(e_2)$ , allora  $\frac{1}{w(e_1)} > \frac{1}{w(e_2)}$ 

L'ordine è INVERTITO, MST DIVERSO (kruskal selezionerebbe gli archi con peso più alto)

d) Se  $w(e_1) < w(e_2)$ , allora  $\log w(e_1) < \log w(e_2)$ 

L'ordinamento è preservato, MST INVARIATO

e) Se  $w(e_1) < w(e_2)$ , allora  $w(e_1)^2 < w(e_2)^2$  (perché  $w(u,v) > 0$  e  $x^2$  è crescente per  $x > 0$ )

L'ordinamento è preservato, MST INVARIATO

## PART II

3) Diametro (G)

 $m \leftarrow \# \text{ vertici } G$  $D \leftarrow \text{pesi di } G$ for  $k=1$  TO  $m$     for  $i=1$  TO  $m$         for  $j=1$  TO  $m$ 

$$d_{ij} = \min(d_{ij}, d_{ik} + d_{kj})$$

massimo  $\leftarrow 0$ for  $i=1$  TO  $m$     for  $j=1$  TO  $m$         if ( $d_{ij} \neq +\infty$ )

$$\text{massimo} = \max(\text{massimo}, d_{ij})$$

return massimo

MODIFICA DI FLOYD-WARSHALL

## CORRETTEZZA

Sia  $W$  matrice di adiacenza  $n \times n$  di un grafo orientato e pesato sugli archi.

Sia  $d_{ij}^{(k)} = \min_{p \in D_{ij}} w(p)$

Dato un vertice  $k=1, \dots, n$  quando  $k=n$ :  $d_{ij}^{(n)} = s(i, j)$

## DIMOSTRAZIONE

Scompongo l'insieme dei cammini da  $i$  a  $j$ ,  $D_{ij}$ , in quelli che passano per il vertice  $k$  ( $D_{ij}^{(k)}$ ) e in quelli che non passano per  $k$  ( $D_{ij}^{(k-1)}$ )

$$D_{ij}^{(k)} = D_{ij}^{(k-1)} \cup (D_{ik}^{(k-1)} + D_{kj}^{(k-1)})$$



$$\text{quindi } d_{ij}^{(k)} = \min_{p \in D_{ij}^{(k)}} w(p)$$

$$\begin{aligned} &= \min \left\{ \begin{array}{l} \min_{p \in D_{ij}^{(k-1)}} w(p) \\ \min_{p \in D_{ik}^{(k-1)}} w(p) + \min_{p \in D_{kj}^{(k-1)}} w(p) \end{array} \right\} \\ &= \min \left\{ d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right\} \end{aligned}$$

4)  $D_{i,j,k,s,b,a}(G, \omega, s)$

INIT-SS ( $G, s$ )

$Q \leftarrow V[G]$

$S \leftarrow \emptyset$

while  $Q \neq \emptyset$  do

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

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

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

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

return ( $d, G_T$ )

INIT-SS ( $G, s$ )

For each  $v \in V[G]$  do

$d[v] \leftarrow +\infty$

$\pi[v] \leftarrow \text{NIL}$

$d[s] \leftarrow 0$

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

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

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

$\pi[v] = u$

## CORRETTEZZA

Sia  $G = (V, E)$  un grafo orientato e 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

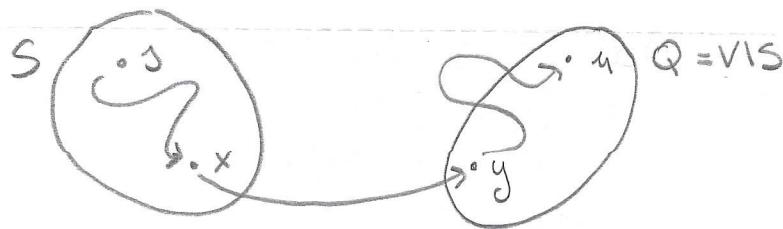
Dimostreremo la correttezza del primo punto attraverso la dimostrazione di un'affermazione più forte ma equivalente, cioè che per ogni  $u \in V$ , al momento dell'estrazione di  $u$ , risulta  $d[u] = S(s, u)$ . La dimostrazione avviene per assurdo e si avvale di 9 osservazioni.

## DIMOSTRAZIONE

Supponiamo per assurdo che esiste 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 la INIT-ss,  $d[s] = 0 = S(s, s)$ , e  $S(s, s)$  non può valere  $-\infty$  perché per ipotesi non ci sono archi con peso negativo, quindi neanche cicli negativi.
- 2) Al momento dell'estrazione  $S \neq \emptyset$ , 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 controlo 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 eletto (SES) 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 taglio, cioè tale che  $x \in S$  e  $y \in Q$ .



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

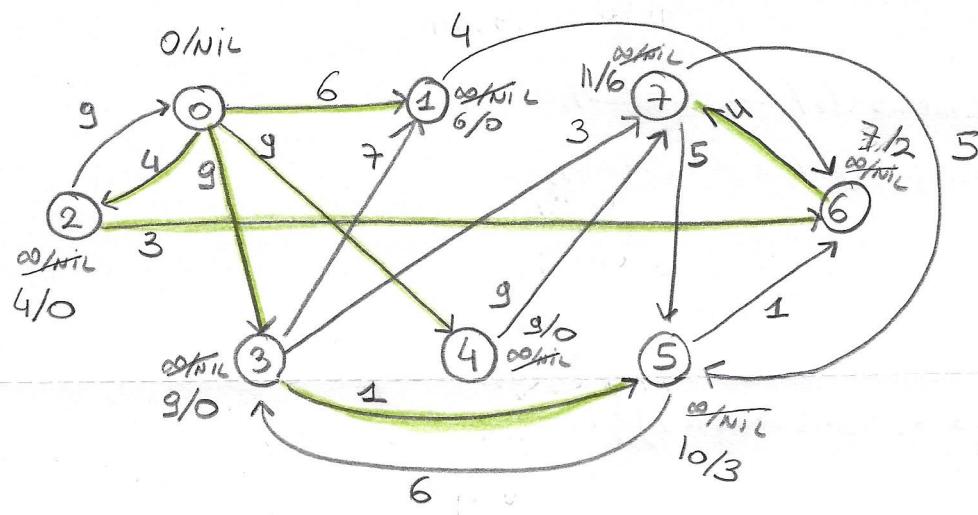
6) Poiché siamo su un cammino minimo, possiamo applicare la proprietà della convergenza: al momento dell'estrazione di  $x$ , applichiamo la RELAX su  $y$  e otteniamo che  $d[y] = S(s, x) + w(x, y) = S(s, y)$

- 7) Dal momento che stiamo per estrarre il nodo  $u$ , siccome l'algorithm di Diskstra estrae il vertice svente campo  $d$  più piccolo, dovrà valere che  $d[u] \leq d[y]$
- 8) Non può accadere che  $S(s, y) > S(s, u)$ : in virtù del fatto che ci troviamo in un cammino minimo e che i pesi sono tutti maggiori o uguali a zero, allora  $S(s, y) \leq S(s, u)$
- 9) Per la proprietà del limite inferiore, vale sicuramente  $S(s, u) \leq d[u]$

Ricostriamo l'assurdo sulla base delle precedenti operazioni:

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

Avalendoci delle osservazioni, siamo riusciti a "inchiodare" il valore  $d[u]$  tra  $S(s, u)$  e  $S(s, u)$ :  $S(s, u) \leq d[u] \leq S(s, u) \Rightarrow d[u] = S(s, u)$ , che è assurdo in quanto andiamo ad invalidare l'ipotesi che  $d[u] \neq S(s, u)$ .



0-2-1-6-3-4-5-7

2) a) Definiamo  $\text{lung}[i][j]$  come la lunghezza della LCS tra i prefissi  $X[0..i-1]$  e  $Y[0..j-1]$ :

• CASO BASE

Se  $i == 0$  e  $j == 0$ :  $\text{lung}[i][j] = 0$  (una sequenza è vuota)

• CASO RICORSIVO

Se  $X[i-1] == Y[j-1]$ :  $\text{lung}[i][j] = 1 + \text{lung}[i-1][j-1]$

Altrimenti:  $\text{lung}[i][j] = \max(\text{lung}[i-1][j], \text{lung}[i][j-1])$

using namespace std;

```
int lcs_memo(const string &X, const string &Y, int i, int j, vector<vector<int>> &memo){
```

// Caso base

```
if(i == 0 || j == 0)
```

```
    return 0;
```

// Se già calcolato, ritorna il valore memoizzato

```
if(memo[i][j] != -1)
```

```
    return memo[i][j];
```

// Caso ricorsivo

```
if(X[i-1] == Y[j-1])
```

```
    memo[i][j] = 1 + lcs_memo(X, Y, i-1, j-1, memo);
```

```
else {
```

```
    memo[i][j] = max(lcs_memo(X, Y, i-1, j, memo),  
                      lcs_memo(X, Y, i, j-1, memo));
```

```
}
```

```
return memo[i][j];
```

```
}
```

```
int lcs_top-down(const string &X, const string &Y) {
```

```
    int m = X.size();
```

```
    int n = Y.size();
```

// Inizializza la tabella di memoization con -1

```
vector<vector<int>> memo(m+1, vector<int>(n+1, -1));
```

```
return lcs_memo(X, Y, m, n, memo);
```

```
}
```

$T(n) = O(m \times n)$

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;`

`PNode buildBST (std::vector<int> v, int inf, int sup, PNode padre){`

`if(inf > sup)`

`return nullptr;`

`int medi;`

`PNode root;`

`medi = (inf + sup) / 2;`

`root = new Node (v[medi], padre);`

`root -> left = buildBST (v, inf, sup, root);`

`root -> right = buildBST (v, inf, sup, root);`

`return root;`

`}`

// Aggiungo le chiavi dell'intervallo all'interno di un vettore.

```
void collectKeys (PNode u, int lower, int upper, std::vector<int>& keys) {
    if (u) {
        if (u->key > upper)
            collectKeys (u->left, lower, upper, keys);
        else if (u->key < lower)
            collectKeys (u->right, lower, upper, keys);
        else {
            collectKeys (u->left, lower, upper, keys);
            keys.push_back (u->key);
            collectKeys (u->right, lower, upper, keys);
        }
    }
}
```

PTree creaBST Interval (PTree t, int k) {

```
if (!t || !t->root)
    return new Tree();
std::vector<int> keys;
int lower;
lower = t->root->key; // Estremo inferiore: chiave della radice di t
collectKeys (t->root, lower, k, keys);
```

```
PTree ris = new Tree();
if (!keys.empty())
    ris->root = buildBST (keys, 0, keys.size() - 1, nullptr);
return ris;
```

$$T(n) = \begin{cases} c & n=0 \\ T(k) + T(n-k-1) & n>0 \end{cases}$$

$$\Rightarrow T(n) = (c+d)n + c \Rightarrow T(n) = O(n)$$

m = numero di nodi dell'albero

k = numero di nodi nel sottoalbero sinistro della radice

Non vengono necessariamente visitati tutti i nodi di t.

CREA BST

$$T(m) = \begin{cases} c & m=0 \\ 2T\left(\frac{m}{2}\right) + d & m>0 \end{cases}$$

$m$  = numero di chiavi contenute nel vettore  $v$

APPLICO TEOREMA MASTER

$$a=2 \quad b=2 \quad f(m)=d \quad g(m)=m$$

1° CASO DEL TEOREMA MASTER

$$f(m) = O(m^{1-\varepsilon}) \rightarrow \varepsilon = 1 \quad \exists \varepsilon > 0$$

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

CREA BST INTERVAL

$$O(m) + \Theta(m) = O(m) \text{ perche' la dimensione di ris e' } m \leq m.$$

SOLUZIONE ALTERNATIVA CON BST SBILANCIATO

/\* Funzione ausiliaria per creare un BST contenente le chiavi con valori compresi nell'intervallo \*/

PNode copyInRange(PNode u, int min, int max, PNode padre) {

if (!u)

return nullptr;

if (u-&gt;key &gt; max)

return copyInRange(u-&gt;left, min, max, padre);

if (u-&gt;key &lt; min)

return copyInRange(u-&gt;right, min, max, padre);

PNode res = new Node{u-&gt;key, padre, nullptr, nullptr};

res-&gt;left = copyInRange(u-&gt;left, min, max, res);

res-&gt;right = copyInRange(u-&gt;right, min, max, res);

return res;

}

PTree crea BST Interval (PTree t, int k) {

PTree res = new Tree{nullptr};

if (t->root)

res->root = copyInRange (t->root, t->root->key, k, nullptr);

return res;

}

$$T(n) \begin{cases} c & n=0 \\ T(k) + T(n-k-1) + d & n>0 \end{cases}$$

n = numero di nodi dell'albero

k = numero dei nodi del sottosalbero sinistro della radice

$$T(n) = (c+d)n + c \Rightarrow T(n) = O(n)$$