

# Algoritmi e Strutture Dati

a.a. 2018/19

## Compito del 13/06/2019

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

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

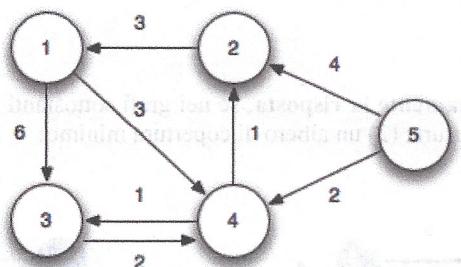
Matricola: \_\_\_\_\_

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

### Parte II

*(2.5 ore; ogni esercizio vale 6 punti)*

1. Si disegni l'albero binario di ricerca la cui visita in post-ordine ha come risultato 4, 7, 6, 15, 23, 21, 18, 12. Si effettuino poi le seguenti operazioni nell'ordine dato e disegnando l'albero risultante dopo ogni singola operazione di inserimento e ogni singola operazione di cancellazione:
  - a. inserimenti: 19, 20, 24;
  - b. cancellazioni: 18, 15, 19, 21. Ove necessario, si utilizzi il successore.
2. Sia  $A$  un array di  $n$  numeri interi. Si consideri il problema di decidere se esistono 3 posizioni distinte  $x, y, z$  in  $A$  tali che  $A[x] + A[y] + A[z] = 0$ . Scrivere un algoritmo di complessità  $O(n^2)$  per risolvere il problema.  
(Suggerimento: si ordini  $A$ , poi si utilizzino 3 contatori  $i, j, k$ :  $i$  assume tutti i valori da 1 a  $n - 2$ ; poi per ogni valore di  $i$ ,  $j$  viene inizializzato a  $i + 1$ , e  $k$  ad  $n \dots$ ).  
Dimostrare che la complessità della soluzione proposta sia  $O(n^2)$ .
3. 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 (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 INIT_SS	0	NIL	$\infty$	NIL	$\infty$	NIL	$\infty$	NIL	0	NIL
1 iterazione 1	0	NIL	$\infty$	NIL	6	1	3	1	$\infty$	NIL
4 iterazione 2	0	NIL	4	4	4	4	3	1	$\infty$	NIL
3 iterazione 3	0	NIL	4	4	4	4	3	1	$\infty$	NIL
2 iterazione 4	0	NIL	4	4	4	4	3	1	$\infty$	NIL
5 iterazione 5	0	NIL	4	4	4	4	3	1	$\infty$	NIL

# Algoritmi e Strutture Dati

a.a. 2018/19

Compito del 13/06/2019

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

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

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

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

## Parte I

(30 minuti; ogni esercizio vale 2 punti)

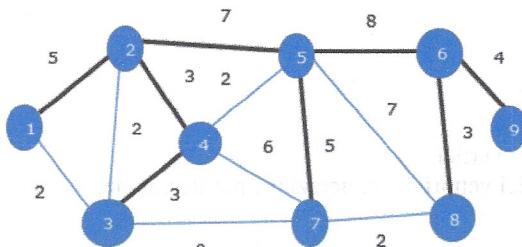
- Si consideri una tabella Hash di dimensione  $m = 8$ , e indirizzamento aperto con doppio Hashing basato sulle funzioni  $h_1(k) = k \bmod m$  e  $h_2(k) = 1 + 2 * (k \bmod (m - 3))$ . Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 34, 12, 18, 9.

- Il Prof. C. Lick sostiene di aver sviluppato un algoritmo di complessità

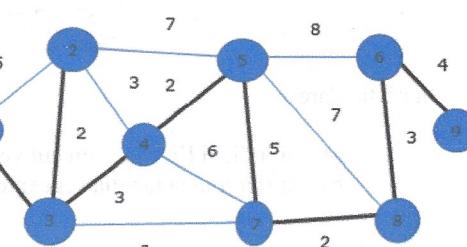
$$T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{2}$$

che riceve in ingresso un numero intero  $k$  e un grafo non orientato  $G = (V, E)$  (con  $n$  vertici) e risponde TRUE se è possibile estrarre da  $G$  un sottoinsieme  $C$  di  $k$  vertici mutuamente adiacenti (ovvero:  $\forall u, v \in C : u \neq v \Rightarrow (u, v) \in E$ ). Si dica, giustificando tecnicamente la risposta, se l'affermazione è verosimile.

- Si dica, giustificando tecnicamente la risposta, se nei grafi sottostanti gli archi indicati in grassetto formano o meno (1) un albero di copertura; (2) un albero di copertura minimo:



(a)



(b)

**Nota:** si giustifichino tecnicamente tutte le risposte. In caso di discussioni poco formali o approssimative l'esercizio non verrà valutato pienamente.

4. Dato un grafo orientato e pesato  $G=(V, E)$  con pesi positivi, cioè  $w(u,v) > 0$  per ogni  $(u,v) \in E$ , si vuole determinare se esiste in  $G$  un ciclo  $c = \langle x_0, x_1, \dots, x_q \rangle$  (con  $x_0 = x_q$ ) per cui sia soddisfatta la seguente condizione:

$$\prod_{i=1}^q 10w(x_{i-1}, x_i)^2 > 10^q$$

Si sviluppi un algoritmo per risolvere questo problema, se ne discuta la correttezza e si determini la sua complessità computazionale. (Suggerimento: si cerchi di ricondurre il problema dato ad uno noto.)

**Nota:** si giustifichino tecnicamente tutte le risposte. In caso di discussioni poco formali o approssimative l'esercizio non verrà valutato pienamente.

## PART I

1)  $m = 8$

$h_1(k) = k \bmod m$

$h_2(k) = 1 + 2(k \bmod (m-3))$

$((k \bmod m) + i(1 + 2(k \bmod (m-3)))) \bmod m$

$((k \bmod 8) + i(1 + 2(k \bmod 5))) \bmod 8$

$i=0$

$(34 \bmod 8) \bmod 8 = 2 \bmod 8 = 2$

$(12 \bmod 8) \bmod 8 = 4 \bmod 8 = 4$

$(18 \bmod 8) \bmod 8 = 2 \bmod 8 = 2 \text{ collisione}$

$i=1$

$(2 + 1(1 + 2(18 \bmod 5))) \bmod 8 = (2 + 1(1+6)) \bmod 8 = (2 + 1 + 6) \bmod 8 = 9 \bmod 8 = 1$

$i=0$

$(9 \bmod 8) \bmod 8 = 1 \bmod 8 = 1 \text{ collisione}$

$i=1$

$((9 \bmod 8) + 1(1 + 2(9 \bmod 5))) \bmod 8 = (1 + 1(1+2\cdot4)) \bmod 8 = (1 + 1(1+8)) \bmod 8 = (1 + 1 + 8) \bmod 8 = 10 \bmod 8 = 2 \text{ collisione}$

$i=2$

$(1 + 2(1+8)) \bmod 8 = (1 + 2 + 16) \bmod 8 = 19 \bmod 8 = 3$

0
18
34
9
12
5
6
7

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

$$a=3 \quad b=3 \quad f(m) = \frac{m}{2} \quad d = \log_2 3 = 1 \Rightarrow g(m) = m$$

$$f(m) \approx g(m)$$

2° CASO DEL TEOREMA MASTER

$$m = \Theta(m)$$

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

Il problema della clique è un noto problema NP-Completo. Se esistesse un algoritmo che lo risolve in tempo  $\Theta(m \log m)$ , questo implicherebbe che  $P=NP$ . I problemi NPC non sono attualmente risolvibili in tempo polinomiale, tanto meno in tempo quasi-lineare.

3) ENTRAMBI SONO ALBERI DI COPERTURA

L'albero A non è un MST poiché ci sono archi con peso minore.

PART II

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  $(d, G\pi)$

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

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

$\pi[v] = u$

$T(\text{DIJKSTRA}) :$

- HEAP  $O(m \log m)$

- GRAFO SPARSO:  $m \approx n = O(n \log m)$

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

- ARRAY  $O(n^2)$

- Sia  $G = (V, E)$  un grafo orientato pesato sugli archi, cioè con  $\omega : E \rightarrow \mathbb{R}$  tale che  $\forall (u, v) \in E : \omega(u, v) \geq 0$ . Allora, alla fine dell'algorithmo di Dijkstra, si ha:
- 1)  $\forall v \in V : d[v] = \delta(s, v)$
  - 2)  $G_T$  è un albero di cammini minimi

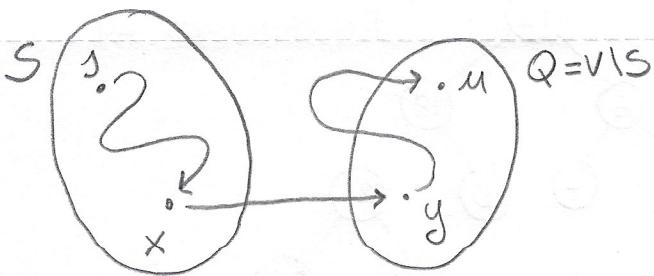
Dimosberemo la correttezza del primo punto attraverso la dimostrazione di un'affermazione più forte ma equivalente, cioè che per ogni  $u \in V$ , al momento dell'estrazione di  $u$ , risulta  $d[u] = \delta(s, u)$ . La dimostrazione avviene per assurdo.

### DIMOSTRAZIONE

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

### OSSERVAZIONI

- 1)  $u$  non può essere la sorgente: dopo la INIT-ss,  $d[s] = 0 = \delta(s, s)$ , e  $\delta(s, s)$  non può valere  $-\infty$  perché per ipotesi non ci sono archi con pesi negativi, quindi neanche ciclinegativi
- 2) Al momento dell'estrazione di  $u$ ,  $S \neq \emptyset$ , perché in  $S$  ci sarà almeno la sorgente  $s$
- 3)  $u$  non è irraggiungibile dalla sorgente: se così fosse,  $\delta(s, u) = +\infty = d[u]$ , che sarebbe corretto e non violerebbe la proprietà che abbiamo voluto violare per assurdo; quindi  $\delta(s, u) \neq +\infty$
- 4) Ci poniamo nell'istante in cui  $s$  è già stato estratto ( $s \in S$ ) 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 apparteneente a  $p$  che attraversa il taglio, 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  $d[y] = \delta(s, x) + \omega(x, y) = \delta(s, y)$

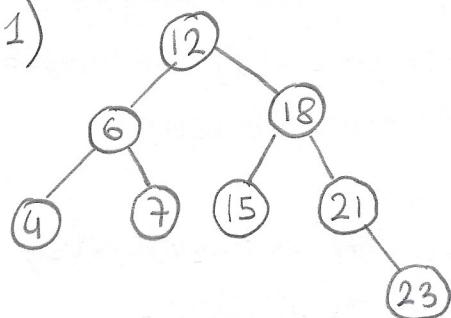
- 7) Dal momento che stiamo per estrarre il nodo  $u$ , siccome l'algoritmo di Dijkstra estrae il vertice avente campo di più piccolo, dovrà valere che  $d[u] \leq d[y]$
- 8) Non può accadere che  $\delta(s, y) > \delta(s, u)$ : in virtù del fatto che ci troviamo in un cammino minimo e che i pesi sono 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 in base alle precedenti osservazioni:

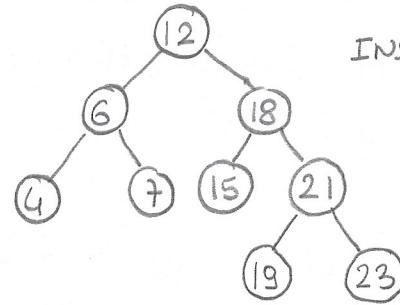
$$\begin{aligned} \delta(s, u) &\leq d[u] && \text{PER OSS. 9} \\ &\leq d[y] && \text{PER OSS. 7} \\ &= \delta(s, y) && \text{PER. OSS 6} \\ &\leq \delta(s, u) && \text{PER. OSS. 8} \end{aligned}$$

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

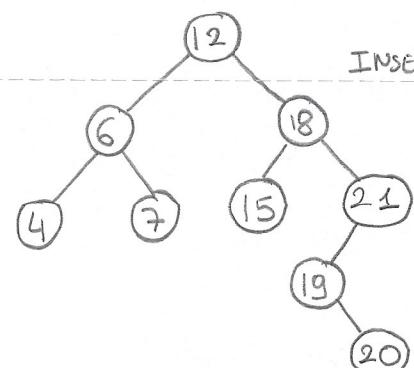
1)



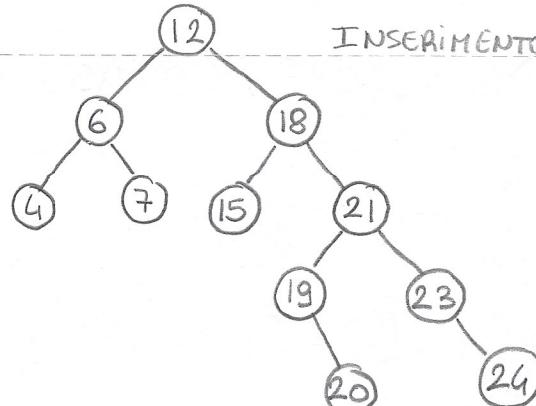
INSEGNAMENTO 19

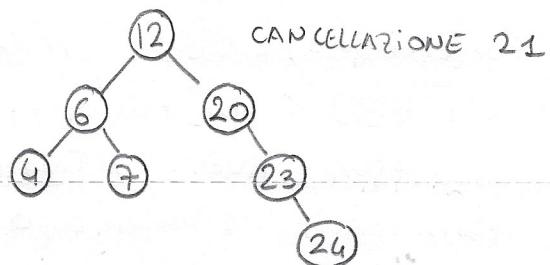
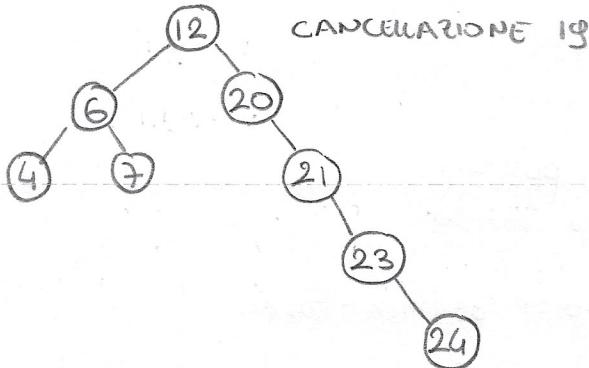
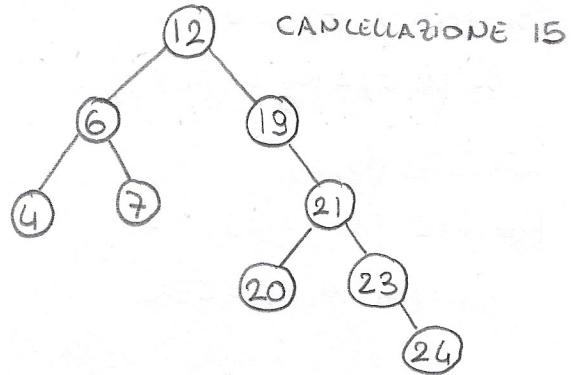
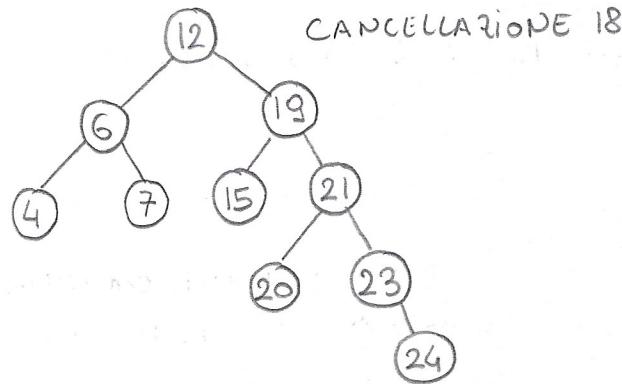


INSEGNAMENTO 20



INSEGNAMENTO 24





$$4) \prod_{i=1}^q 10 \omega(x_{i-1}, x_i)^2 > 10^q$$

$$\log\left(\prod_{i=1}^q 10 \omega(x_{i-1}, x_i)^2\right) > \log(10^q) \Rightarrow \sum_{i=1}^q \log(10) + \sum_{i=1}^q \log(\omega(x_{i-1}, x_i)^2) > q \log(10)$$

$$q \log(10) + 2 \sum_{i=1}^q \log(\omega(x_{i-1}, x_i)) > q \log(10)$$

$$\cancel{2} \sum_{i=1}^q \log(\omega(x_{i-1}, x_i)) > 0$$

$$\sum_{i=1}^q \log(\omega(x_{i-1}, x_i)) > 0$$

La condizione si traduce quindi nella ricerca di un ciclo nel grafo dove la somma dei logaritmi dei pesi degli archi sia positiva.

Possiamo definire un nuovo grafo  $G'$  con gli stessi nodi e archi di  $G$ , ma con pesi modificati:  
 $w'(u, v) = \log(w(u, v)).$

Il problema originale è equivalente a trovare un ciclo in  $G'$  con somma pesi positiva.

## BELLMAN - FORD MODIFICATO ( $G, w, s$ )

for each  $(u, v)$  in  $E$ : // Trasformazione dei pesi

$$w'(u, v) = \log(w(u, v))$$

INIT-SS ( $G, s$ )

For  $i = 1$  TO  $|V[G]| - 1$

for each  $(u, v) \in E[G]$

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

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

for each  $(u, v) \in E[G]$  // Controllo cicli positivi

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

return TRUE // Trovato ciclo con somma positiva

return FALSE // Nessun ciclo con somma positiva trovato

La complessità computazionale è dominata dall'algorithmo di Bellman-Ford:

$$T(\text{BELLMAN-FORD}) = \Theta(m \cdot n)$$

L'algorithmo è corretto perché:

- La trasformazione logaritmica preserva l'ordine delle diseguaglianze
- L'algorithmo di Bellman-Ford è in grado di rilevare cicli raggiungibili con peso positivo
- Un ciclo con somma dei logaritmi positiva corrisponde esattamente a un ciclo che soddisfa la condizione originale.

```

2) #include <vector>
#include <algorithm>
using namespace std;
bool hasThreeSumZero(vector<int>& nums) {
    int n = nums.size();
    if (n < 3)
        return false;
    sort(nums.begin(), nums.end());
    for (int i=0; i < n-2; i++) {
        // Esegui il codice solo se è il primo elemento o se non è un duplicato
        if (i == 0 || nums[i] != nums[i-1]) {
            int target = -nums[i];
            int j = i+1;
            int k = n-1;
            while (j < k) {
                int sum = nums[j] + nums[k];
                if (sum == target)
                    return true;
                else if (sum < target)
                    j++;
                else
                    k--;
            }
        }
    }
    // Se è un duplicato, il for incrementerà i automaticamente
}
return false;
}

```

Complessità

- Ordinamento:  $O(n \log n)$
- Ciclo esterno:  $O(n)$
- Ciclo while di ricerca di  $k$ :  $O(n)$  per ogni iterazione del ciclo esterno

$$\text{Totale} = O(n \log n) + O(n^2) = O(n^2)$$

L'algoritmo è ottimale per questo problema, poiché qualsiasi soluzione deve almeno esaminare tutte le possibili coppie di elementi, che sono  $O(n^2)$ .