

# Algoritmi e Strutture Dati

a.a. 2018/19

## Compito del 03/06/2019

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

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

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

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

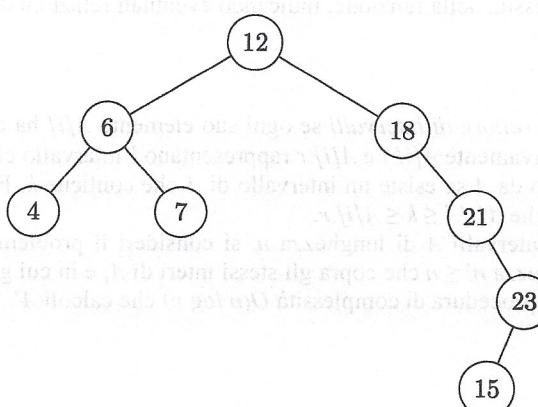
### Parte I

(30 minuti; ogni esercizio vale 2 punti)

Si consideri l'albero di alberi da frutta rappresentato nell'immagine. I nodi sono numerati da 1 a 8. I nodi 12, 18 e 21 sono rispettivamente albero di ciliegia, di pesca e di mandorla.

#### 1. Dato il seguente albero

Si eseguire una visita in preordine, una visita in ordine simmetrico, una visita in postordine e una visita in ampiezza elencando nei quattro casi la sequenza dei nodi incontrati.



Individuare gli alberi da ciliegia, da pesca e da mandorla.

Eseguire una visita in preordine, una visita in ordine simmetrico, una visita in postordine e una visita in ampiezza elencando nei quattro casi la sequenza dei nodi incontrati.

#### 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:

$$T(n) = 4T(n/2) + n^2$$

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

#### 3. Siano $\mathcal{P}$ e $\mathcal{Q}$ due problemi in NP e si supponga $\mathcal{P} \leq \mathcal{Q}$ . Si stabilisca, giustificando tecnicamente la risposta, se le seguenti affermazioni sono vere o false:

(a) Se  $\mathcal{Q}$  è risolvibile in tempo polinomiale, allora  $\mathcal{P}$  è risolvibile in tempo polinomiale

(b) Se  $\mathcal{Q}$  è un problema NP-completo, allora  $\mathcal{P}$  è NP-completo

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

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

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

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

## Parte II

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

- Sia dato un albero binario i cui nodi contengono una chiave intera  $x.key$ , oltre ai campi  $x.left$ ,  $x.right$  che rappresentano rispettivamente il figlio sinistro e il figlio destro. Si definisce *grado di squilibrio* di un nodo il valore assoluto della differenza tra la somma delle chiavi nei nodi foglia del sottoalbero sinistro e la somma delle chiavi dei nodi foglia del sottoalbero destro. Il grado di squilibrio di un albero è il massimo grado di squilibrio dei suoi nodi.  
Scrivere una funzione efficiente in C, di nome `gradosquil(u)`, che data la radice di un albero binario, calcola il grado di squilibrio dell'albero.

Analizzare la complessità della funzione, indicando eventuali relazioni di ricorrenza.

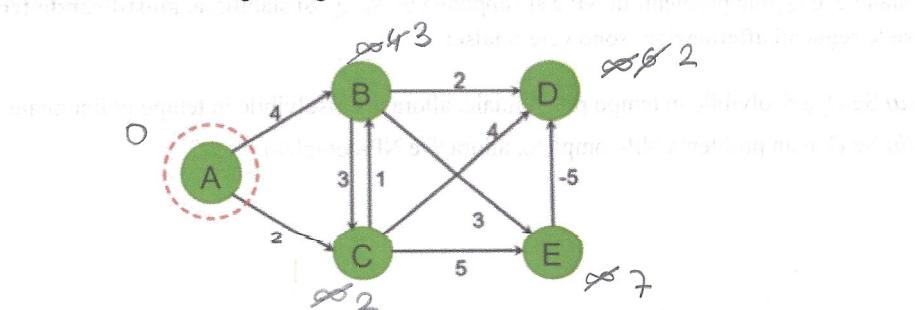
- Un vettore  $A$  è detto *vettore di intervalli* se ogni suo elemento  $A[i]$  ha due campi interi  $A[i].l$  e  $A[i].r$  tali che  $A[i].l \leq A[i].r$ . Intuitivamente  $A[i].l$  e  $A[i].r$  rappresentano l'intervallo chiuso di interi  $[A[i].l, A[i].r]$ . Un intero  $k$  è coperto da  $A$  se esiste un intervallo di  $A$  che contiene  $k$ . Formalmente  $k$  è coperto da  $A$  se esiste un indice  $i$  di  $A$  tale che  $A[i].l \leq k \leq A[i].r$ .  
Dato un vettore di intervalli  $A$  di lunghezza  $n$ , si consideri il problema di determinare un nuovo vettore di intervalli  $A'$  di lunghezza  $n' \leq n$  che copre gli stessi interi di  $A$ , e in cui gli intervalli siano disgiunti. Scrivere lo pseudocodice di una procedura di complessità  $O(n \log n)$  che calcoli  $A'$ .
- Si consideri il seguente algoritmo, che accetta in ingresso un grafo orientato e pesato  $G = (V, E)$ , con funzione peso  $w : E \rightarrow \mathbb{R}$ , e un vertice "sorgente"  $s \in V$ :

```

MyAlgorithm( G, w, s )
1.  $n = |V[G]|$ 
2. for each  $u \in V[G]$  do
3.    $d[u] = \infty$ 
4.    $d[s] = 0$ 
5. for  $i = 1$  to  $n$  do
6.   for each  $u \in V[G]$ 
7.     for each  $v \in \text{Adj}[u]$  /*  $\text{Adj}[u] = \text{insieme dei vertici adiacenti a } u$  */
8.        $d[v] = \min \{ d[v], d[u] + w(u,v) \}$ 
9. return  $d$ 

```

- Qual è la sua complessità?
- Quale problema risolve?
- L'algoritmo continua ad essere corretto se esegue un'iterazione in meno? Perché?
- Si simuli la sua esecuzione sul seguente grafo, utilizzando il vertice A come sorgente.



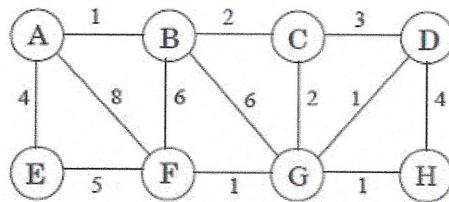
In particolare, si riempia la tabella seguente con i valori del vettore  $d$ , iterazione per iterazione:

	A	B	C	D	E
dopo istr. 4	0	$\infty$	$\infty$	$\infty$	$\infty$
$i = 1$	0	3	2	2	7
$i = 2$	0	3	2	2	6
$i = 3$	0	3	2	2	6
$i = 4$	0	3	2	2	6
$i = 5$	0	3	2	2	6

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

4. Si enunci e si dimostri il teorema fondamentale degli alberi di copertura minimi.

Si consideri inoltre il grafo  $G$  riportato di seguito



e i tagli

$$T_1 = (\{A, B, C\}, \{D, E, F, G, H\}) \quad \text{e} \quad T_2 = (\{A, D\}, \{B, C, E, F, G, H\}).$$

- (a) Quali degli archi che attraversano  $T_1$  appartengono ad **almeno** un albero di copertura minima di  $G$ ?
- (b) Quali degli archi che attraversano  $T_1$  appartengono a **tutti** gli alberi di copertura minimi di  $G$ ?
- (c) Quali degli archi che attraversano  $T_2$  appartengono ad **almeno** un albero di copertura minima di  $G$ ?
- (d) Quali degli archi che attraversano  $T_2$  appartengono a **tutti** gli alberi di copertura minimi di  $G$ ?

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

## PART I

1) PREORDINE: 12 - 6 - 4 - 7 - 18 - 21 - 23 - 15

SIMMETRICO: 4 - 6 - 7 - 12 - 18 - 21 - 15 - 23

POSTORDINE: 4 - 7 - 6 - 15 - 23 - 21 - 18 - 12

AMPIZZA: 12 - 6 - 18 - 4 - 7 - 21 - 23 - 15

$$2) T(n) = 4 T\left(\frac{n}{2}\right) + n^2$$

$$a=4 \quad b=2 \quad d=\log_2 4 = 2 \quad g(n)=n^2 \quad f(n)=n^2$$

$$f(n)=g(n)$$

## SECONDO CASO DEL TEOREMA MASTER

$$f(n)=\Theta(n^d) \Rightarrow n^2=\Theta(n^2)$$

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

## CONFRONTO CON FLOYD-WARSHALL

L'algoritmo ricorsivo A con complessità  $\Theta(n^2 \log n)$  è asintoticamente più efficiente dell'algoritmo di Floyd-Warshall, che ha complessità  $\Theta(n^3)$ , poiché  $n^2 \log n$  cresce più lentamente di  $n^3$ .

3)

a) Poiché  $P \leq_p Q$ , se  $Q$  è risolvibile in tempo polinomiale cioè ( $Q \in P$ ), allora anche  $P$  è risolvibile in tempo polinomiale.

→ INSIEME PROBLEMI POLINOMIALI

Questo deriva direttamente dalla definizione di riducibilità polinomiale: se possiamo risolvere  $Q$  in tempo polinomiale e trasformare istanze di  $P$  in istanze di  $Q$  in tempo polinomiale, allora possiamo risolvere  $P$  componendo le due trasformazioni.

L'affermazione è VERA.

b) Per definizione, un problema è NP-completo se:

1) Appartiene a NP

2) Tutti i problemi in NP sono riducibili a esso in tempo polinomiale

Seppiamo che  $Q$  è NP-completo e  $P \leq_p Q$ . Tuttavia questo non implica automaticamente che  $P$  sia NP-completo.

Perché  $P$  sia NP-completo, dovremmo dimostrare che tutti i problemi in NP si riducono a  $P$ . Ma la riduzione  $P \leq_p Q$  va nella direzione opposta (riduciamo  $P$  a  $Q$  NON  $Q$  a  $P$ ).

Inoltre  $P$  potrebbe essere un problema "facile" in NP (ad esempio un problema in P), mentre  $Q$  è NP-completo. La riduzione  $P \leq_p Q$  non ci dà informazioni sufficienti per concludere che  $P$  sia NP-Hard.

L'affermazione è FALSA. La NP-completatezza di  $Q$  non implica quella di  $P$ .

## PARTE II

3) L'algoritmo presentato è una variente di Bellman-Ford, che calcola i cammini minimi da un vertice sorgente  $s$  a tutti gli altri vertici in un grafo orientato e pesato  $G = (V, E)$ . Ecco il suo funzionamento:

- 1) Inizializza le distanze  $d[u] \rightarrow \infty$  per tutti i vertici  $u$ , tranne  $d[s] = 0$
- 2) Esegue  $m$  iterazioni (dove  $m = |V|$ ), e in ogni iterazione rilessa tutti gli archi del grafo
- 3) Restituisce l'array  $d$  delle distanze minime

La complessità è  $O(m^3)$  nel caso peggiore.

L'algoritmo calcola i cammini minimi da una sorgente  $s$  a tutti gli altri vertici in un grafo orientato e pesato, anche con pesi negativi (a meno di cicli negativi raggiungibili da  $s$ ).

- Se il grafo contiene cicli negativi raggiungibili da  $s$ , l'algoritmo non è in grado di rilevarli (è differente di Bellman-Ford standard, che può farlo con una verifica aggiuntiva)
- Se non ci sono cicli negativi, restituisce le distanze minime corrette.

Risolve il problema dei cammini minimi da sorgente singola in grafi orientati e pesati (senza rilevamento di cicli negativi)

L'algoritmo esegue  $m$  iterazioni, ma Bellman-Ford standard ne richiede  $m-1$  per garantire la correttezza (dove  $m = |V|$ )

- Se il grafo non ha cicli negativi:
  - \* Dopo  $m-1$  iterazioni, le distanze sono già corrette. Un'iterazione aggiuntiva non cambia nulla.
  - \* Quindi, rimuovere un'iterazione ( $m-1$  invece di  $m$ ) PRESERVA LA CORRETTEZZA.

- Se il grafo ha cicli negativi:
  - \* L'algoritmo dato non li rileva comunque, quindi la riduzione a  $m-1$  iterazioni non influenza

Perché  $m-1$  iterazioni sono sufficienti?

In un grafo senza cicli negativi, il cammino minimo da  $s$  a qualsiasi  $u$  contiene al più  $m-1$  archi.

In un grafo senza cicli negativi, il cammino minimo da  $s$  a qualsiasi  $u$  contiene al più  $m-1$  archi.

Dopo  $m-1$  iterazioni, tutti i cammini sono stati correttamente rilassati.

PART II

## 4) TEOREMA FONDAMENTALE DEGLI MST

Sia  $G = (V, E, \omega)$  un grafo non orientato e连通的, dove  $\omega: E \rightarrow \mathbb{R}$  è una funzione peso sugli archi. Vengono le seguenti ipotesi:

- Sia  $A \subseteq E$  contenuto in qualche MST
- Sia  $(S, V \setminus S)$  un taglio che "rispetta"  $A$ , cioè tale che nessun arco di  $A$  attraversa il taglio
- Sia  $(u, v)$  un arco leggero che attraversa il taglio

Allora il teorema fondamentale degli MST afferma che  $A \cup \{(u, v)\}$  è contenuto in qualche MST. Si può anche dire che  $(u, v)$  è sicuro per  $A$ .

## DIMOSTRAZIONE

Chiamiamo  $T$  l'albero di copertura minima che contiene  $A$ , per ipotesi.

Distinguiamo due casi:

- $(u, v) \in T \Rightarrow A \cup \{(u, v)\} \in T$ . Allora la condizione è banalmente verificata.
- $(u, v) \notin T$ . Notiamo che l'aggiunta di  $(u, v)$  a  $T$  provoca la formazione di un ciclo. Indichiamo quindi con  $(x, y)$  un arco che si trova nel ciclo e che attraversa il taglio, e decidiamo di toglierlo.

Sia  $T' = T \cup \{(u, v)\} \setminus \{(x, y)\}$  un nuovo albero. Allora, per ipotesi vale che  $W(T) \leq W(T')$  (essendo  $T$  un MST), ma vale anche che  $W(T) \geq W(T')$ :

$$W(T') = W(T) + \underbrace{\omega(u, v) - \omega(x, y)}_{\leq 0} \leq W(T)$$

Quindi  $W(T) = W(T')$  e dunque  $T'$  è un MST.

Bisogna ancora dimostrare che  $A \cup \{(u, v)\} \subseteq T'$ . Questo è sempre vero, perché:

- $A \subseteq T'$  perché  $A$  rispetta il taglio per ipotesi (quindi nemmeno  $(x, y)$  apparteneva ad  $A$ )
- $(u, v) \in T'$  per costruzione

Ciò significa che se togliendo e aggiungendo un arco la somma dei pesi non cambia, allora vuol dire che  $\omega(u, v) = \omega(x, y)$ . Ma come è possibile che questo accada nel caso in cui gli archi che attraversano un taglio hanno tutti peso diverso? Tale situazione si risolve nel primo caso: l'arco leggero appartiene già all'albero di copertura minima, per cui non c'è bisogno di cercare altri MST attraverso la strategia del "cuci-e-taglia".

a) Archi che attraversano  $T_1$ :  $\{A-E\}, \{A-F\}, \{B-F\}, \{B-G\}, \{C-D\}, \{C-G\}$

Arco di peso minimo:  $\{C-G\}$

Per il teorema dell'arco sicuro  $\{C-G\}$  deve appartenere ad almeno un MST

Nessun altro arco è garantito, ma potrebbero apparire in alcuni MST a seconda delle scelte (es.  $\{C-D\}$  o  $\{B-G\}$  se non formano cicli)

$\{C-G\}$  appartiene ad almeno un MST

b) Unicità dell'arco minimo:  $\{C-G\}$  è l'unico arco con peso minimo che attraversa  $T_1$ .

Quindi  $\{C-G\}$  deve apparire in tutti gli MST.

c) Archi di peso minimo:  $\{A-B\}$  e  $\{D-G\}$

Per il teorema dell'arco sicuro, almeno uno di questi due deve apparire in ogni MST

Possono esistere MST che includono  $\{A-B\}$  ma non  $\{D-G\}$  e viceversa.

$\{A-B, D-G\}$  appartengono ad almeno un MST.

d) Non c'è un unico arco minimo: Sia  $\{A-B\}$  che  $\{D-G\}$  hanno peso 1

Nessuno dei due è obbligatorio per tutti gli MST, poiché l'algoritmo può scegliere alternativamente tra loro.

Nessun arco di  $T_2$  appartiene

Node

x.key

x.left

x.right

(3)


 $| \text{Somma chiavi left} - \text{somma chiavi right} | = \text{grado di squilibrio}$ 


ritorno il valore delle foglie



Faccio una somma dei nodi sinistro e destro e la ritorno



struct Node {

int key;

Node\* left;

Node\* right;

}

int gradosquil (Node\* u) {

int max\_squilibrio = 0;

somma Foglie E Grado (u, max\_squilibrio);

return max\_squilibrio;

}

```

int sommaFoglieEGradi (Node* u, int & max_squilibrio) {
    if (u == nullptr)
        return 0;
    if (u->left == nullptr && u->right == nullptr) // Se è una foglia restituisce il suo
        return u->key;                                valore
    // Calcola ricorsivamente la somma delle foglie dei sottoalberi
    int somma_sx = sommaFoglieEGradi (u->left, max_squilibrio);
    int somma_dx = sommaFoglieEGradi (u->right, max_squilibrio);

    // Calcola lo squilibrio del nodo corrente
    int squilibrio = abs(somma_sx - somma_dx);
    max_squilibrio = std::max(max_squilibrio, squilibrio);

    // Restituisce la somma delle foglie del sottoalbero
    return somma_sx + somma_dx;
}

```

$T(n) = O(n)$  // Visita ogni nodo una sola volta