

Algoritmi e Strutture Dati

a.a. 2023/24

Compito del 9/9/2024

Cognome: _____

Nome: _____

Matricola: _____

E-mail: _____

15 minuti

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.

- Si consideri una tabella Hash di dimensione $m = 13$, e indirizzamento aperto con doppio Hashing basato sulle funzioni $h_1(k) = k \bmod m$ e $h_2(k) = 1 + (k \bmod (m - 2))$. Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 42, 16, 7, 85.

- Si stabilisca se la seguente affermazione è vera o falsa:

“Se $G = (V, E)$ è un grafo non orientato con k componenti connesse, allora $|E| \geq |V| - k$.”

In caso affermativo si fornisca una dimostrazione, altrimenti un controesempio.

- Un algoritmo per determinare i cammini minimi tra tutte le coppie di vertici di un grafo pesato ha complessità

$$T(n) = 9T\left(\frac{n}{2}\right) + 9n^2 + 2\log n$$

dove n rappresenta il numero di vertici del grafo in ingresso. Si dica se l'algoritmo in questione è preferibile o meno all'algoritmo di Floyd-Warshall e perché.

Algoritmi e Strutture Dati

a.a. 2023/24

Compito del 9/9/2024

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. Siano T_1 e T_2 due alberi binari completi aventi lo stesso numero di nodi che contengono interi e con i seguenti campi: key, left e right.

a. Si scriva una procedura **efficiente** che vada a sottrarre ai nodi di T_1 i valori dei nodi di T_2 . Per esempio:

$$\begin{array}{r} 3 \quad 4 \\ 5 \quad 6 \quad 7 \quad 8 \end{array} - \begin{array}{r} 7 \quad 10 \\ 2 \quad 3 \quad 5 \quad 4 \end{array} = \begin{array}{r} -4 \quad -6 \\ 3 \quad 3 \quad 2 \quad 4 \end{array}$$

Il prototipo della procedura è:

```
void sottrazione_alberi(PNode r1, PNode r2)
```

dove r_1 è la radice di T_1 e r_2 è la radice di T_2 .

- b. Valutare e giustificare la complessità della procedura, indicando eventuali relazioni di ricorrenza e mostrando la loro risoluzione.
- c. Specificare il linguaggio di programmazione scelto e la **definizione** di PNode.
2. Sia A un vettore di n interi e si consideri il problema di determinare se esistono due interi che occorrono in A lo stesso numero di volte.
- Si scriva un algoritmo **efficiente** per il problema proposto.
 - Si scriva un algoritmo **efficiente** per il problema proposto nel caso in cui in A occorrono **c valori distinti**, dove c è una costante intera positiva.
 - Si determini e giustifichi la complessità degli algoritmi proposti.

Si devono scrivere eventuali funzioni/procedure ausiliarie utilizzate.

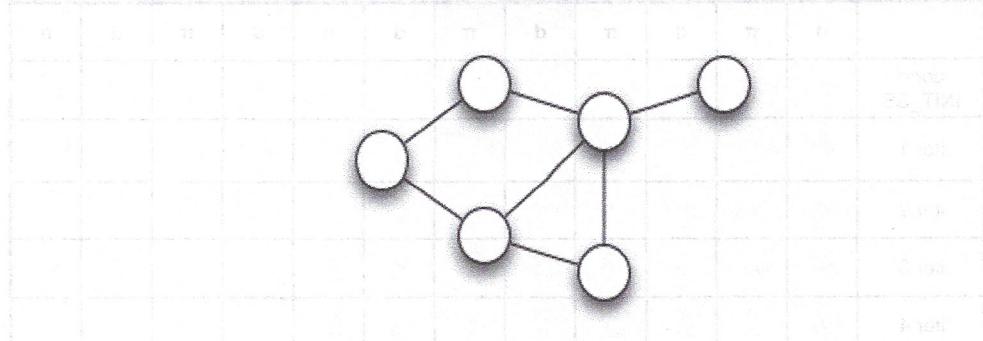
3. Si stabilisca quale problema risolve il seguente algoritmo, dove A rappresenta la matrice di adiacenza di un grafo non orientato (si assuma che i vertici siano numerati da 1 a n):

```
MyAlgorithm(A)
1. n = numero di righe di A
2. S = ∅
3. for i = 1 to n
4.   if MyFunction(S,i) then
5.     S = S ∪ {i}
6. return S

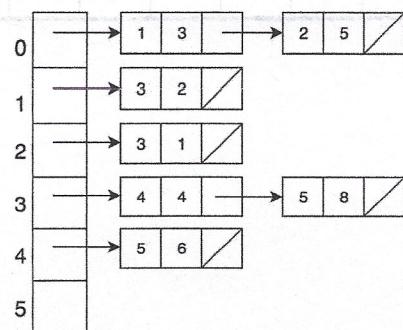
MyFunction(S,i)
1. for each j ∈ S
2.   if a[i,j] = 1 then
3.     return FALSE
4. return TRUE
```

Si dimostri la correttezza dell'algoritmo e si determini la sua complessità.

Si simuli infine accuratamente la sua esecuzione sul seguente grafo e si mostri (con alcuni esempi) che il risultato finale può dipendere dal modo in cui vengono numerati i vertici del grafo. In particolare, si determini una numerazione che consentirebbe all'algoritmo di restituire in uscita il massimo numero di vertici.



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:



In particolare:

- come sorgenti, si utilizzi sia il vertice 0 che il vertice 5
- in entrambi i casi si indichi l'ordine con cui vengono estratti i vertici
- si riempiano le tabelle seguenti con i valori dei vettori d e π , iterazione per iterazione

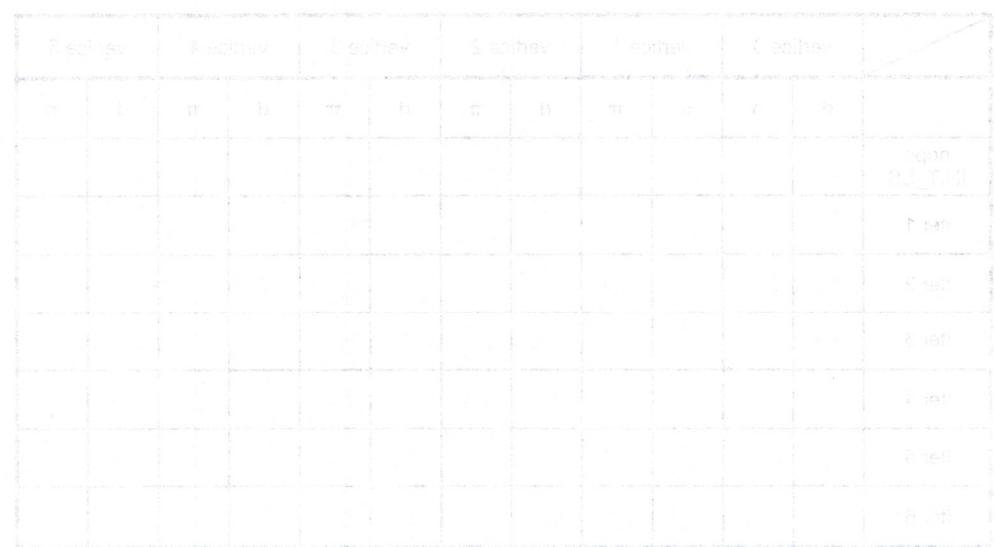
Vertice sorgente: 0

		vertice 0		vertice 1		vertice 2		vertice 3		vertice 4		vertice 5	
		d	π										
	dopo INIT_SS	0	NIL	∞	NIL								
0	iter 1	0	NIL	3	0	5	0	∞	NIL	∞	NIL	∞	NIL
1	iter 2	0	NIL	3	0	5	0	5	1	∞	NIL	∞	NIL
2	iter 3	0	NIL	3	0	5	0	5	1	∞	NIL	∞	NIL
3	iter 4	0	NIL	3	0	5	0	5	1	9	3	13	3
4	iter 5	0	NIL	3	0	5	0	5	1	9	3	13	3
5	iter 6	0	NIL	3	0	5	0	5	1	9	3	13	3

Vertice sorgente: 5

		vertice 0		vertice 1		vertice 2		vertice 3		vertice 4		vertice 5	
		d	π										
dopo	INIT_SS	∞	Nil	0	Nil								
5	iter 1	∞	Nil	∞	Nil	∞	Nil	8	5	6	5	0	Nil
4	iter 2	∞	Nil	∞	Nil	∞	Nil	8	5	6	5	0	Nil
3	iter 3	∞	Nil	10	3	9	3	8	5	6	5	0	Nil
2	iter 4	14	2	10	3	9	3	8	5	6	5	0	Nil
1	iter 5	13	1	10	3	9	3	8	5	6	5	0	Nil
0	iter 6	13	1	10	3	9	3	8	5	6	5	0	Nil

Figura 10: Distanze e percorsi



PART I

1)

0
1
2
3
4
5
6
7
8
9
10
11
12
42
7
16
85

$$m=13 \quad k = 42, 16, 7, 85$$

$$h_1(k) = k \bmod m$$

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

$$h(i,k) = (h_1(k) + i \cdot h_2(k)) \bmod m$$

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

$$= ((k \bmod 13) + i(1 + k \bmod 11)) \bmod 13$$

$$h(0, 42) = ((42 \bmod 13) + 0(1 + 42 \bmod 11)) \bmod 13 = (3 + 0) \bmod 13 = 3 \bmod 13 = 3$$

$$h(0, 16) = ((16 \bmod 13) + 0(1 + 16 \bmod 11)) \bmod 13 = (3 + 0) \bmod 13 = 3 \text{ collisione}$$

$$h(1, 16) = ((16 \bmod 13) + 1(1 + 16 \bmod 11)) \bmod 13 = (3 + 1(1+5)) \bmod 13 = \\ = (3 + 1 + 5) \bmod 13 = 9$$

$$h(0, 7) = ((7 \bmod 13) + 0(1 + 7 \bmod 11)) \bmod 13 = (7 + 0) \bmod 13 = 7$$

$$h(0, 85) = ((85 \bmod 13) + 0(1 + 85 \bmod 11)) \bmod 13 = (7 + 0) \bmod 13 = 7 \text{ collisione}$$

$$h(1, 85) = ((85 \bmod 13) + 1(1 + 85 \bmod 11)) \bmod 13 = (7 + 1(1+8)) \bmod 13 = \\ = (7 + 1 + 8) \bmod 13 = 16 \bmod 13 = 3 \text{ collisione}$$

$$h(2, 85) = ((85 \bmod 13) + 2(1 + 85 \bmod 11)) \bmod 13 = (7 + 2(1+8)) \bmod 13 =$$

$$= (7 + 2 + 16) \bmod 13 = 25 \% 13 = 12$$

2) DEFINIZIONE DI COMPONENTE CONNESSA: Una componente connessa è un sottoinsieme di G in cui ogni coppia di vertici è collegata tramite un cammino e che non è collegato a nessun altro vertice di G

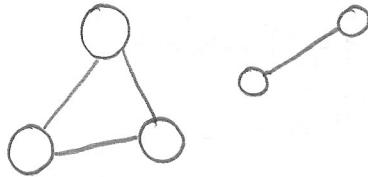
NUMERO MINIMO DI ARCHI IN UNA COMPONENTE CONNESSA: Una componente connessa con $|V_G|$ vertici ha almeno $|V_G| - 1$ archi per rimanere connessa

GRAFO CON K COMPONENTI CONNESSE:

- Dividiamo G in K componenti connesse
- Il numero totale di vertici è $|V| = \sum |V_i|$
- Il numero totale di archi è: $|E| \geq \sum (|V_i|-1) = |V| - K$

L'AFFERMAZIONE È VERA

(es.)



$$\begin{aligned}|E| &= 4 \\ |V| &= 5 \\ K &= 2\end{aligned}$$

$$|E| \geq |V| - K \Rightarrow 4 \geq 5 - 2 \Rightarrow 4 \geq 3 //$$

$$3) T(n) = 9T\left(\frac{n}{2}\right) + 9n^2 + 2\log n$$

$$a=9 \quad b=2 \quad f(n) = 9n^2 + 2\log n \rightarrow n^2 \quad d = \log_2 9 \rightarrow g(n) = n^d = n^{\log_2 9}$$

$$f(n) \leq g(n) \quad 1^\circ \text{ CASO}$$

$$f(n) = O(n^{d-\varepsilon})$$

$$n^2 = O(n^{\log_2 9 - \varepsilon}) \Rightarrow \varepsilon = \log_2 9 - n^2$$

$$T(n) = \Theta(n^d) = \Theta(n^{\log_2 9})$$

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

$3 < \log_2 9 \Rightarrow$ FLOYD-WARSHALL È PREFERIBILE ALL'ALGORITMO PROPOSTO

PART II

1) struct Node {

```
int key;
Node* left;
Node* right;
```

};

void soluzioneAlberi(Node* r1, Node* r2) {

if (r1 && r2) {

r1->key == r2->key;

soluzioneAlberi(r1->left, r2->left);

soluzioneAlberi(r1->right, r2->right);

}

}

 $T(n) = O(n)$ dove n è il numero di nodi (visita completa dell'albero)

2) bool stesseOccorrenze(vector<int> &arr) {

int i=0, j=0;

bool ok=false;

vector<int> occ(arr.size()+1);

if (arr.size() < 2)

return false;

mergesort(&arr, &occ, 0, arr.size()-1);

while (i < arr.size() && !ok) {

j = i+1;

while (j < arr.size() && arr[i] == arr[j])

j++;

if (occ[j-i] == 0) {

occ[j-i] = 1;

i=j;

}

else {

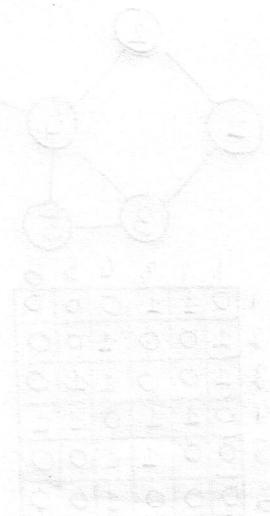
ok=true;

}

}

return ok;

}



3) L'algoritmo identifica un insieme indipendente massimo in un grafo.

FUNZIONAMENTO:

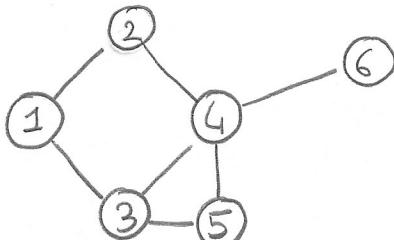
- Mantiene un insieme S di vertici che non condividono archi
- Aggiunge un vertice $i \in S$ solo se nessun vertice in S è connesso a i

CORRETTEZZA:

- Un vertice è aggiunto a S solo se non è connesso a nessuno degli elementi già in S , garantendo che S sia indipendente
- L'algoritmo non esplora tutte le combinazioni, quindi il risultato potrebbe non essere massimo

COMPLESSITÀ:

- $O(m^3)$: ciclo su m vertici e verifica di connettività per ciascuna coppia ($O(n^2)$)



	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	0	1	0	0
3	1	0	0	1	1	0
4	0	1	1	0	1	1
5	0	0	1	1	0	0
6	0	0	0	1	0	0

$$S = \emptyset$$

$i=1$ TRUE

$$S = \{1\}$$

$i=2$ FALSE

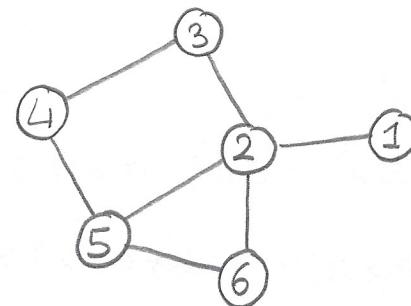
$i=3$ FALSE

$i=4$ TRUE

$$S = \{1, 4\}$$

$i=5$ FALSE

$i=6$ FALSE



	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	0	1	0	1	1
3	0	1	0	1	0	0
4	0	0	1	0	1	0
5	0	1	0	1	0	1
6	0	1	0	0	1	0

$$S = \emptyset$$

$i=1$ TRUE

$$S = \{1\}$$

$i=2$ FALSE

$i=3$ TRUE

$$S = \{1, 3\}$$

$i=4$ FALSE

$i=5$ TRUE

$$S = \{1, 3, 5\}$$

$i=6$ FALSE

Se numeriamo come primo vertice il vertice più isolato dagli altri avremmo sicuramente un numero di vertici maggiore rispetto ad altre scelte di numerazione.

ESAME 09/09/24

③

4) DIJKSTRA (G, ω, 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 $v \in \text{Adj}[u]$

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

return (d, π)

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

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

$\pi[v] = u$

DIMOSTRAZIONE

COSA SI INTENDE DIMOSTRARE

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

π è un albero di cammini minimi

PROCEDIMENTO

Dimostriamo solamente la prima parte, per farlo utilizzeremo le proprietà di

Disegualanza Triangolare: $\delta(s, v) \leq \delta(s, u) + \omega(u, v)$

Limite Inferiore: $\delta(s, u) \leq d[u]$ in ogni momento dell'esecuzione

Assenza di cammino: $\forall v \in V$ non raggiunto da s , $\delta(s, v) = \infty$

Convergenza: In breve, se si ha un cammino minimo da s a u , dopo la prima
RELAX ($u, v, \omega(u, v)$) avrà che $d[v] = \delta(s, v)$

Procedendo con la dimostrazione:

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

1) $u \neq s$ perché dopo INIT-SS sicuramente $d[s] = \delta(s, s)$

2) Quindi, al momento di estrarre u , l'insieme S dei vertici già estratti 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 $\delta(s, u) = \infty = d[u]$ dalla INIT

b) CASO 2: cammino minimo tra s, u

- 4) Al momento di estrarre x , per ipotesi $d[x] = S(s, x)$
 5) Dopo la RELAX($u, v, w(u, v)$), $d[y] = S(s, y)$
 6) Ora però poniamo che Dijkstra estraiga il vertice u , sarà quindi vero che
 $d[u] \leq d[v]$
 7) $S(y, y) \leq S(s, u)$. Per ipotesi s, \dots, y era un cammino minimo. Essendo da y a u
 pesi ≥ 0 allora per certo $S(s, y) \leq S(s, u)$
 8) $S(s, u) \leq d[u]$ LIMITE INFERIORE

