

Algoritmi e Strutture Dati

a.a. 2021/22

Compito del 6/6/2022

Cognome: _____

Nome: _____

Matricola: _____

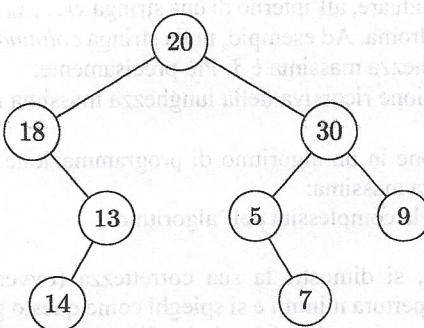
E-mail: _____

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.

- Dato il seguente albero, redigere la sequenza dei nodi visitati in preordine, in ordine simmetrico, in postordine e in ampiezza.



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.

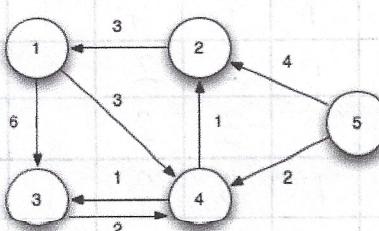
- Un algoritmo ricorsivo \mathcal{A} accetta in ingresso un grafo orientato e pesato $G = (V, E, w)$, due vertici $u, v \in V$ e una costante k , e restituisce TRUE se esiste in G un cammino tra u e v di lunghezza minore o uguale a k e FALSE in caso contrario. La sua complessità è data da

$$T(n) = 4T\left(\frac{n}{2}\right) + 3n^2$$

dove n rappresenta il numero di vertici del grafo. Esistono algoritmi più efficienti di \mathcal{A} per risolvere il problema dato? Si assuma che $w(u, v) \geq 0$ per ogni $(u, v) \in E$.

- Si indichi l'ordine in cui l'algoritmo di Dijkstra estrae i vertici dal seguente grafo, considerando i seguenti due casi:

- vertice sorgente $s = 1$
- vertice sorgente $s = 5$



Algoritmi e Strutture Dati

a.a. 2021/22

Compito del 6/6/2022

Cognome: _____

Nome: _____

Matricola: _____

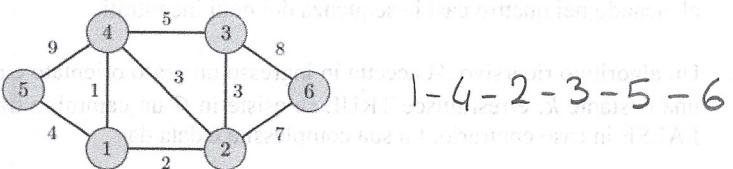
E-mail: _____

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 albero binario. Si vogliono stampare le chiavi di T memorizzate in nodi il cui sottoalbero radicato nel figlio sinistro contiene più chiavi del sottoalbero radicato nel figlio destro.
 - a) Si rappresenti un **albero binario di ricerca** la cui visita in preordine ha come risultato 30, 25, 21, 40, 35, 45. Si mostri quali chiavi verrebbero stampate in base alla condizione sopra descritta.
 - b) Scrivere una procedura **efficiente** in C o C++ per risolvere il problema proposto.
 - c) Valutarne la complessità.
2. Scrivere un algoritmo per individuare, all'interno di una stringa $x_1 \dots x_n$ la lunghezza massima di una sottostringa (di caratteri consecutivi) palindroma. Ad esempio, nella stringa *colonna* la sottostringa palindroma di lunghezza massima è *olo*, dunque la lunghezza massima è 3. Più precisamente:
 - a) dare una caratterizzazione ricorsiva della lunghezza massima $lung[i, j]$ di una sottostringa palindroma di $x_i \dots x_j$;
 - b) tradurre tale definizione in un algoritmo di programmazione dinamica con il metodo **top-down** che determina la lunghezza massima;
 - c) valutare e giustificare la complessità dell'algoritmo
3. Si scriva l'algoritmo di Prim, si dimostri la sua correttezza (ovvero: si enunci e si dimostri il teorema fondamentale degli alberi di copertura minimi e si spieghi come questo garantisca la correttezza dell'algoritmo), si fornisca la sua complessità computazionale e si simuli accuratamente la sua esecuzione sul seguente grafo utilizzando il vertice 1 come "sorgente":



In particolare: si indichi l'ordine con cui vengono estratti i vertici e si riempia la tabella seguente con i valori dei vettori key e π , iterazione per iterazione:

	vertice 1		vertice 2		vertice 3		vertice 4		vertice 5		vertice 6	
	key[1]	$\pi[1]$	key[2]	$\pi[2]$	key[3]	$\pi[3]$	key[4]	$\pi[4]$	key[5]	$\pi[5]$	key[6]	$\pi[6]$
dopo inizializzazione	0	NIL	∞	NIL								
iterazione 1	0	NIL	2	1	∞	NIL	1	1	4	1	∞	NIL
iterazione 2	0	NIL	2	1	5	4	1	1	4	1	∞	NIL
iterazione 3	0	NIL	2	1	3	2	1	1	4	1	7	2
iterazione 4	0	NIL	2	1	3	2	1	1	4	1	7	2
iterazione 5	0	NIL	2	1	3	2	1	1	4	1	7	2
iterazione 6	0	NIL	2	1	3	2	1	1	4	1	7	2

4. Sia $A = (a_{ij})$ la matrice di adiacenza di un grafo orientato $G = (V, E)$.

4.1. Cosa rappresentano gli elementi della matrice

$$A^2 = A \cdot A$$

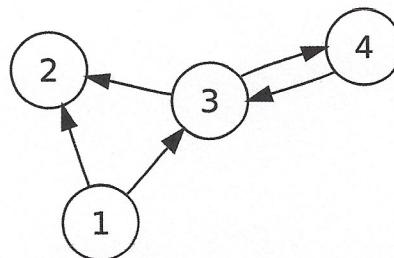
dove “ \cdot ” denota l’operazione di moltiplicazione tra matrici?

4.2. Più in generale, dato un intero $k \geq 2$, cosa rappresentano gli elementi della matrice

$$A^k = \underbrace{A \cdot A \cdot \dots \cdot A}_k ?$$

(Si forniscano adeguate dimostrazioni formali a supporto delle affermazioni fatte.)

4.3. Si determinino le matrici A , A^2 e A^3 per il grafo riportato di seguito:



4.4. Il seguente algoritmo accetta in ingresso la matrice di adiacenza di un grafo orientato e restituisce TRUE o FALSE. Qual è la funziona svolta e qual è la sua complessità? Giustificare le risposte.

MyAlgorithm(A)

```

1. n = numero di righe di A
2. crea due matrici n x n B e C ] C
3. for i = 1 to n
4.   for j = 1 to n
5.     B[i,j] = 0
6.     for k = 1 to n
7.       B[i,j] = B[i,j] + A[i,k]*A[k,j] ] Θ(n³)
8. for i = 1 to n
9.   for j = 1 to n
10.    C[i,j] = 0
11.    for k = 1 to n
12.      C[i,j] = C[i,j] + B[i,k]*A[k,j] ] Θ(n³)
13. for i = 1 to n
14.   if C[i,i] = 0 then ] O(n)
15.     return TRUE
16. return FALSE
  
```

ESAME 6/6/22

①

PART I

1) PREORDINE: 20-18-13-14-30-5-7-9

SIMMETRICO: 18-14-13-20-5-7-30-9

POSTORDINE: 14-13-18-7-5-9-30-20

AMPIETTA: 20-18-30-13-5-9-14-7

$$2) \alpha = 4 \quad d = \log_b \alpha = \log_2 4 = 2 \quad g(n) = n^d = n^2$$

$$b = 2 \quad f(n) = 3n^2$$

$f(n) \approx g(n)$ 2° CASO

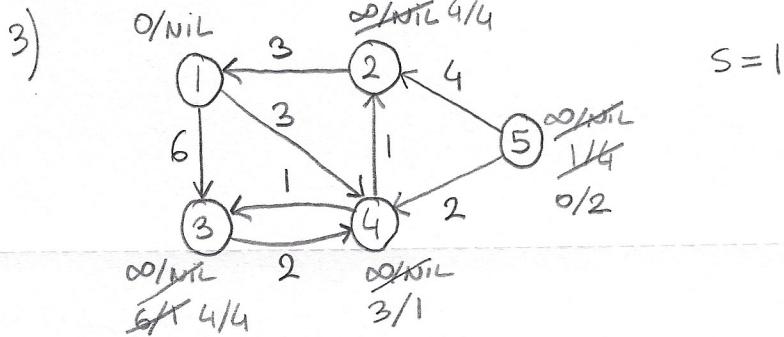
$f(n) = \Theta(g(n)) \Rightarrow f(n) \leq g(n)$

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

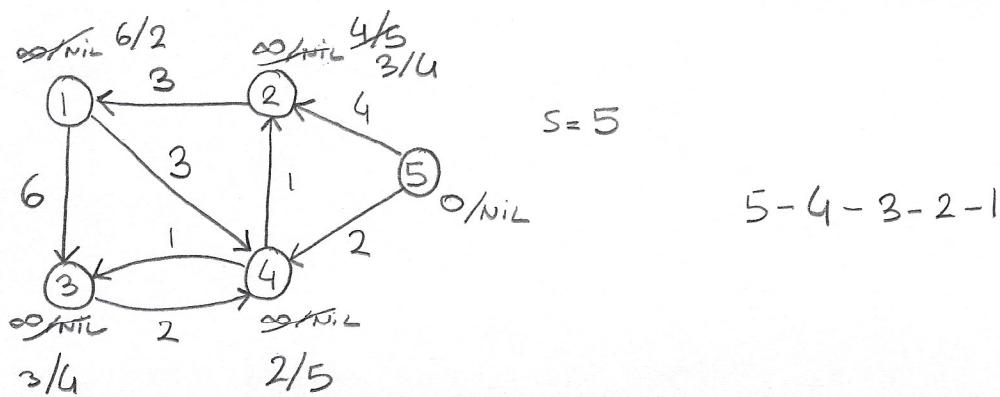
FLOYD-WARSHALL $\rightarrow \Theta(n^3)$

$$\lim_{n \rightarrow +\infty} \frac{\alpha^2 \log n}{n^3} = \lim_{n \rightarrow +\infty} \frac{\log n}{n} = 0$$

L'ALGORITMO A E' ASINTOTICAMENTE PIU' EFFICIENTE

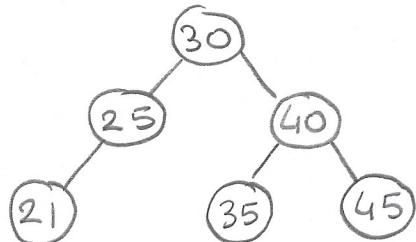


1-4-3-2-5



PART II

1)



30-25-21-40-35-45 IN PREORDINE

STAMPA: 25

```
void stampaKey (Node u) {
    stampaAux(u);
}
```

```
int stampaAux (Node u) {
```

```
    if (u == NULL)
```

```
        return 0;
```

```
    else {
```

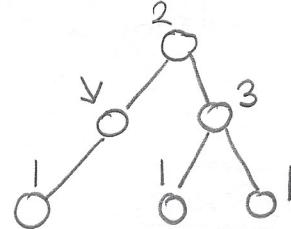
```
        int fsx = stampaAux(u->left);
```

```
        int fdx = stampaAux(u->right);
```

```
        if (fsx > fdx)
```

```
            std::cout << u->key;
```

```
        return fsx+fdx+1;
    }
}
```



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

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

2)

$$\text{lun}[i, j] = \begin{cases} 0 & \text{se } i > j \\ 1 & \text{se } i = j \\ 2 + \text{lun}[i+1, j-1] & \text{se } x[i] = x[j] \\ \text{lun}[i, j-1] + \text{lun}[i+1, j] & \text{se } x[i] \neq x[j] \end{cases}$$

```

int lpol (std::vector<char> s) {
    int maxlung = s.size();
    std::vector<int> mat [maxlung][maxlung];
    for (i=0; i < s.size(); i++)
        for(j=0; j < s.size(); j++)
            mat[i][j] = -1;
    return lpeux (s, mat, 0, s.size());
}

```

```

int lpeux (std::vector<char> s, std::vector<int> mat, int i, int j) {
    if (mat[i][j] == -1) {
        if (i > j)
            mat[i][j] = 0;
        else {
            if (i == j)
                mat[i][j] = 1;
            else {
                int val;
                if (s[i] == s[j] && (val = lpeux (s, mat, i+1, j-1)) == j-i-1)
                    mat[i][j] = val + 2;
                else {
                    int v1 = lpeux (s, mat, i+1, j);
                    int v2 = lpeux (s, mat, i, j-1);
                    mat[i][j] = (v1 >= v2 ? v1 : v2)
                }
            }
        }
    }
}

```

$$T(n) = \Theta(n^2) \text{ con } n \text{ LUNGHEZZA STRINGA}$$

3) MST-PRIM (G, ω, r)

1. $Q \leftarrow V[G]$
2. For each $u \in V[G]$
3. $\text{key}[u] \leftarrow \infty$
4. $\pi[u] \leftarrow \text{NIL}$
5. $\text{key}[r] \leftarrow 0$
6. while $Q \neq \emptyset$ do
7. $u \leftarrow \text{ExtractMin}(Q)$
8. For each $v \in \text{Adj}[u]$
9. if $v \in Q \wedge \omega(u, v) < \text{key}[v]$
10. $\text{key}[v] \leftarrow \omega(u, v)$
11. $\pi[v] \leftarrow u$
12. return $A = \{(u, \pi[u]) \in E \mid u \in V \setminus \{r\}\}$

Prim è conseguenza del teorema fondamentale degli MST: in ogni istante esiste una foresta contenuta in qualche MST. Ad ogni iterazione viene creato un taglio $(Q, V[\alpha] \setminus Q)$ in cui Q rappresenta i nodi già estratti e $V[\alpha] \setminus Q$ i nodi da estrarre. La garanzia che gli archi considerati (deducibili dai campi key e π dei nodi) siano sicuri lo si intuisce dalla riga 9. Alla fine dell'algoritmo non ho una struttura che contiene l'MST risultante, ma posso dedurlo dai campi aggiornati dei nodi del grafo.

TEOREMA FONDAMENTALE DEGLI MST

Sia $G(V, E, \omega)$ un grafo NON ORIENTATO CONNESSO

Siano

- 1) $A \subseteq E$ contenuto in qualche MST
- 2) $(S, V \setminus S)$ che rispetta A
- 3) (u, v) un arco leggero che attraversa $(S, V \setminus S)$

Allora (u, v) è sicuro per A

$A \cup \{(u, v)\}$ contenuto in qualche MST (potrebbe essere diverso da quello dell'ipotesi)

$(u, v) \in E \quad \forall (x, y) \in E: \omega(u, v) \leq \omega(x, y)$

Allora $\exists \text{ MST } T \ni (u, v) \in T$

4) $A^2 =$ Distanza tra due vertici con un cammino di lunghezza $k=2$

$A^K = \underbrace{A \cdot A \cdot \dots \cdot A}_K \rightarrow$ Rappresentano i "passi" che vengono fatti durante il percorso per arrivare da un nodo all'altro

	1	2	3	4
1	0	1	1	0
2	0	0	0	0
3	0	1	0	1
4	0	0	1	0

A

	1	2	3	4
1	0	1	0	1
2	0	0	0	0
3	0	0	1	0
4	0	1	0	1

 A^2

	1	2	3	4
1	0	0	1	0
2	0	0	0	0
3	0	1	0	1
4	0	0	1	0

 A^3

4.4) Ritorna FALSE se per ogni nodo esiste un cammino di $\ell=3$ che torna al nodo stesso.

Ritorna TRUE al primo nodo che NON presenta la proprietà precedente

$$T(n) = C + \Theta(2m^3) + O(n) = m^3$$