

Algoritmi e Strutture Dati

a.a. 2020/21

Compito del 30/06/2021

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. 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: 25, 41, 68, 19.

2. Un algoritmo ricorsivo MyMST è in grado di determinare un albero di copertura minimo all'interno di un grafo pesato G . La sua complessità è pari a:

$$T(m) = 4T(m/2) + 7(m^2 + 1) + 5\log m$$

dove m rappresenta il numero di archi del grafo. Esistono algoritmi più efficienti di MyMST per risolvere il problema dato?

$$\left. \begin{array}{l} i = 1 \text{ se } 0 \\ \text{ se } (i,j) \in E \text{ e } i < j \\ \text{ se } (i,j) \in E \text{ e } i > j \end{array} \right\} = R_{ij}W$$

3. Siano \mathcal{P} e \mathcal{Q} due problemi in NP e si supponga $\mathcal{P} \leq_P \mathcal{Q}$. Si stabilisca se le seguenti affermazioni sono vere o false:

- (a) Se \mathcal{Q} è risolvibile in tempo quadratico, allora \mathcal{P} è risolvibile in tempo quadratico
(b) Se \mathcal{P} è risolvibile in tempo polinomiale, allora \mathcal{Q} è risolvibile in tempo polinomiale
(c) Se \mathcal{P} è un problema NP-completo, allora \mathcal{Q} è NP-completo

Algoritmi e Strutture Dati

a.a. 2020/21

Compito del 30/06/2021

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. Dato un albero binario i cui nodi contengono interi, si vuole aggiungere ad ogni foglia un figlio contenente la somma dei valori che appaiono nel cammino dalla radice a tale foglia. Se la somma di tali valori è positiva sarà aggiunto come figlio sinistro, altrimenti come figlio destro.
 - i) Scrivere una procedura efficiente in C *aggiungiFigli(Node u)* che dato in input la radice u di un albero T aggiunge alle foglie i figli come sopra specificato.
 - ii) Valutarne la complessità, indicando eventuali relazioni di ricorrenza.
 - iii) Scrivere il tipo *Node* in C.
2. Sia A un array di lunghezza $n - k$ con $k \geq 2$, privo di ripetizioni e contenente interi nell'intervallo $[n^2 + 1, n^2 + n]$. Si consideri il problema di determinare i k numeri interi appartenenti all'intervallo $[n^2 + 1, n^2 + n]$ che non compaiono in A . Si scriva una procedura efficiente che, dati A , n e k , risolva il problema proposto stampando gli interi che non compaiono in A . Calcolarne la complessità.
3. Sia $G = (V, E)$ un grafo orientato e pesato, con funzione peso $w : E \rightarrow \mathbb{R}$, che non contenga cicli negativi e cappi. Si assuma che i vertici siano numerati da 1 a n , ovvero $V = \{1, \dots, n\}$, e sia W la matrice di dimensione $n \times n$ definita come segue:

$$W[i,j] = \begin{cases} 0 & \text{se } i = j \\ w(i,j) & \text{se } i \neq j \text{ e } (i,j) \in E \\ +\infty & \text{se } i \neq j \text{ e } (i,j) \notin E \end{cases}$$

Si stabilisca quali problemi risolvono i due algoritmi riportati di seguito (che prendono in ingresso la matrice W associata al grafo), se ne dimostri la correttezza e se ne calcoli la complessità.

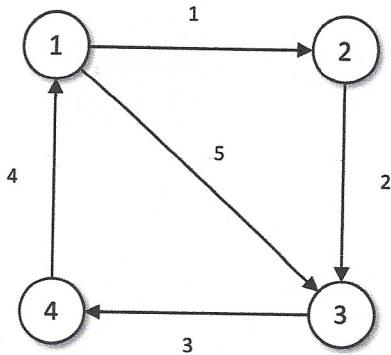
MyAlgorithm1(W)

```
1. n = n.ro di righe di W
2. alloca spazio per un vettore D
di dimensione n
3. D[1] = 0
4. for i = 2 to n
5.   D[i] = +∞
6. for k = 1 to n
7.   for i = 1 to n
8.     for j = 1 to n
9.       D[j] = min{ D[j], D[i] + W[i,j] }
10. return D
```

MyAlgorithm2(W)

```
1. n = n.ro di righe di W
2. alloca spazio per una matrice D
di dimensione n x n
3. for i = 1 to n
4.   for j = 1 to n
5.     D[i,j] = W[i,j]
6.   for k = 1 to n
7.     for i = 1 to n
8.       for j = 1 to n
9.         D[i,j] = min{ D[i,j], D[i,k] + D[k,j] }
10. return D
```

Cosa restituiscono in uscita i due algoritmi in presenza del grafo seguente? (Scrivere esplicitamente il vettore D nel primo caso e la matrice D nel secondo. Non è necessario riportare la simulazione degli algoritmi.)



4. La seguente tabella fornisce le distanze (in unità di 100 miglia) tra gli aeroporti delle città di Londra, Città del Messico, New York, Parigi, Pechino e Tokyo:

	<i>L</i>	<i>CM</i>	<i>NY</i>	<i>Pa</i>	<i>Pe</i>	<i>T</i>
<i>L</i>	—	56	35	2	50	60
<i>CM</i>	56	—	21	57	78	70
<i>NY</i>	35	21	—	36	68	67
<i>Pa</i>	2	57	36	—	51	61
<i>Pe</i>	50	78	68	51	—	13
<i>T</i>	60	70	67	61	13	—

Utilizzando l'algoritmo di Prim, si determini un albero di copertura minimo per il grafo corrispondente.

In particolare, utilizzando il vertice L come sorgente:

- a) si indichi l'ordine con cui vengono estratti i vertici
 b) si riempia la tabella seguente, con i valori dei campi key e π , per ogni singola iterazione

È possibile, utilizzando un altro vertice sorgente, determinare un albero di copertura minimo diverso da quello trovato? In caso affermativo lo si determini (nel qual caso, non è necessario riportare la simulazione dettagliata dell'algoritmo), altrimenti si fornisca una motivazione formale.

	vertice <i>L</i>		vertice <i>CM</i>		vertice <i>NY</i>		vertice <i>Pa</i>		vertice <i>Pe</i>		vertice <i>T</i>	
	key	π	key	π	key	π	key	π	key	π	key	π
dopo inizializzazione	0	NIL	00	NIL	00	NIL	00	NIL	00	NIL	00	NIL
iterazione 1	0	NIL	56	L	35	L	2	L	50	L	60	L
iterazione 2	0	NIL	56	L	35	L	2	L	50	L	60	L
iterazione 3	0	NIL	21	NY	35	L	2	L	50	L	60	L
iterazione 4	0	NIL	21	NY	35	L	2	L	50	L	60	L
iterazione 5	0	NIL	21	NY	35	L	2	L	50	L	13	PE
iterazione 6	0	NIL	21	NY	35	L	2	L	50	L	13	PE



Fig. 39. A schematic drawing of a rectangular room with vertices labeled A, B, C, and D. A diagonal line segment connects vertex A to vertex C. A vertical line segment connects vertex A to vertex D. A horizontal line segment connects vertex D to vertex C.

	A	B	C	D	E	F	G	H
A	0	62	124	186	248	310	372	434
B	62	0	62	124	186	248	310	372
C	124	62	0	62	124	186	248	310
D	186	124	62	0	62	124	186	248
E	248	186	124	62	0	62	124	186
F	310	248	186	124	62	0	62	124
G	372	310	248	186	124	62	0	62
H	434	372	310	248	186	124	62	0

and conditions of Fig. 39. It is clear that the angle θ is the angle between the lines AB and AC , or BC and AC , or AB and BC .

It follows from the above that the angle θ is the angle between the lines AB and AC , or BC and AC , or AB and BC .

It follows from the above that the angle θ is the angle between the lines AB and AC , or BC and AC , or AB and BC .

| Condition |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| n | n+1 | n+2 | n+3 | n+4 | n+5 | n+6 |
| 0 | 62 | 124 | 186 | 248 | 310 | 372 |
| 62 | 0 | 62 | 124 | 186 | 248 | 310 |
| 124 | 62 | 0 | 62 | 124 | 186 | 248 |
| 186 | 124 | 62 | 0 | 62 | 124 | 186 |
| 248 | 186 | 124 | 62 | 0 | 62 | 124 |
| 310 | 248 | 186 | 124 | 62 | 0 | 62 |
| 372 | 310 | 248 | 186 | 124 | 62 | 0 |
| 434 | 372 | 310 | 248 | 186 | 124 | 62 |

PART I

1)

0	
1	25
2	
3	68
4	41
5	19
6	
7	

$$m=8 \quad k=25, 41, 68, 19$$

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

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

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

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

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

$$h(0, 25) = ((25 \bmod 8) + 0(1+2(25 \bmod 5))) \bmod 8 = (1+0) \bmod 8 = 1$$

$$h(0, 41) = ((41 \bmod 8) + 0(1+2(41 \bmod 5))) \bmod 8 = (1+0) \bmod 8 = 1 \text{ collisione}$$

$$\begin{aligned} h(1, 41) &= ((41 \bmod 8) + 1(1+2(41 \bmod 5))) \bmod 8 = (1+1(1+2(1))) \bmod 8 = \\ &= (1+1(1+2)) \bmod 8 = (1+1+2) \bmod 8 = 4 \bmod 8 = 4 \end{aligned}$$

$$h(0, 68) = ((68 \bmod 8) + 0(1+2(68 \bmod 5))) \bmod 8 = (4+0) \bmod 8 \text{ collisione}$$

$$\begin{aligned} h(1, 68) &= ((68 \bmod 8) + 1(1+2(68 \bmod 5))) \bmod 8 = (4+1(1+2(3))) \bmod 8 = \\ &= (4+1(1+6)) \bmod 8 = (4+1+6) \bmod 8 = 11 \bmod 8 = 3 \end{aligned}$$

$$h(0, 19) = ((19 \bmod 8) + 0(1+2(19 \bmod 5))) \bmod 8 = (3+0) \bmod 8 = 3 \text{ collisione}$$

$$\begin{aligned} h(1, 19) &= ((19 \bmod 8) + 1(1+2(19 \bmod 5))) \bmod 8 = (3+1(1+2(4))) \bmod 8 = \\ &= (3+1(1+8)) \bmod 8 = (3+1+8) \bmod 8 = 12 \bmod 8 = 4 \text{ collisione} \end{aligned}$$

$$\begin{aligned} h(2, 19) &= ((19 \bmod 8) + 2(1+2(19 \bmod 5))) \bmod 8 = (3+2(1+8)) \bmod 8 = \\ &= (3+2+16) \bmod 8 = 21 \bmod 8 = 5 \end{aligned}$$

$$2) T(m) = 4T\left(\frac{m}{2}\right) + 7(m^2+1) + 5\log m$$

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

$$f(m) = m^2$$

$$f(m) \approx g(m) \quad 2^{\text{o}} \text{ CASO}$$

$$f(m) = \Theta(g(m)) \Rightarrow m^2 = \Theta(m^2)$$

$$T(m) = \Theta(m^d \log m) = \Theta(m^2 \log m) \quad m \geq m-1$$

$$\text{KRUSCAL} = O(m \log m)$$

$$\text{PRIM} = O(m \log m) \rightarrow \text{PIÙ EFFICIENTE}$$

3) a) Se Q è risolvibile in tempo quadratico, allora P è risolvibile in tempo quadratico

FALSO, la riduzione $P \leq_p Q$ garantisce che P non sia più difficile di Q , ma non implica che P erediti la stessa complessità

b) Se P è risolvibile in tempo polinomiale, allora Q è risolvibile in tempo polinomiale

FALSO, la riduzione non implica che Q diventi più semplice se P è polinomiale

c) Se P è NP-completo, allora Q è NP-completo

FALSO, non vi è garanzia che Q sia NP-completo solo perché $P \leq_p Q$

PART II

```
1) typedef struct Node {
    int val;
    struct Node* left;
    struct Node* right;
} Node;
```

```
Node* creaNodo(int val) {
    Node* nuovo = (Node*) malloc(sizeof(Node));
    nuovo->val = val;
    nuovo->left = NULL;
    nuovo->right = NULL;
    return nuovo;
}
```

(2)

```
void aggiungiFigli (Node* root) {
    aggiungiFigliAux (root, 0);
}
```

```
void aggiungiFigliAux (Node* root, int somma) {
    if (!root)
        return;
    somma += root->val;
    // Se è una foglia, aggiungi il nuovo nodo
    if (!root->left && !root->right) {
        if (somma > 0)
            root->left = crea Nodo (somma);
        else
            root->right = crea Nodo (somma);
    }
    else {
        aggiungiFigliAux (root->left, somma);
        aggiungiFigliAux (root->right, somma);
    }
}
```

$$T(n) = O(n)$$

2) void numeriMancanti (int* A, int m, int k) {

 int start = m * m + 1;

 int end = m * m + m;

 // Crea un array booleano per tracciare i numeri presenti

 int* presenti = (int*) malloc((end - start + 1), sizeof(int));

 for (int i = 0; i < m - k; i++) {

 presenti[A[i] - start] = 1;

}

// Stampa i numeri mancanti

 for (int i = 0; i <= end - start; i++) {

 if (!presenti[i]) {

 printf("%d", i + start);

}

}

 free(presenti);

}

$T(m) = O(m)$

3) My Algorithm 1

1	0	1	3	6
---	---	---	---	---

Calcola le distanze minime da una singola sorgente a tutti gli altri nodi $T(m) = O(m^3)$

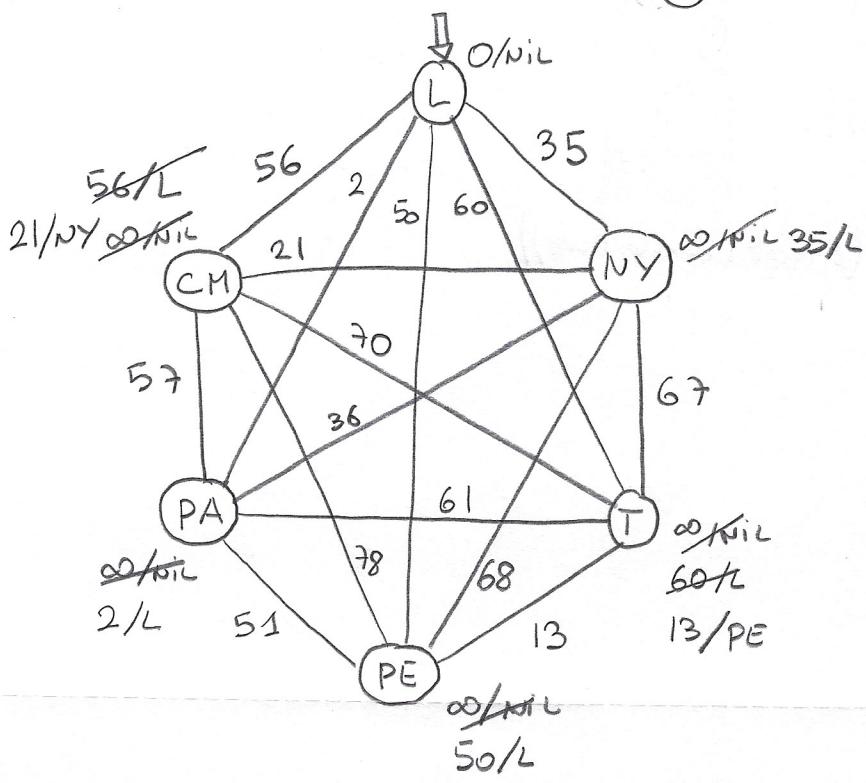
My Algorithm 2

0	1	3	6
9	0	2	5
7	8	0	3
4	5	7	0

Calcola le distanze minime tra tutte le coppie di nodi.

$T(m) = O(m^3)$

4)



ANCHE SE CAMBIAMO SORGENTI VERRANNO SEMPRE PRESI GLI ARCHI CON PESO MINORE

PART II - SOLUZIONI ALTERNATIVE

```
1) typedef struct node {
    int key;
    struct node* left;
    struct node* right;
    struct node* p;
}* Node;
```

```
void aggiungiFigli (Node u) {
    int path = 0;
    addAux(u, path);
}
```

```

void addAux (Node u, int path) {
    if (u) {
        if (!u->left && !u->right) {
            int val = path + u->key;
            Node sum = (Node) malloc (sizeof (struct node)),
            sum->key = val;
            sum->left = NULL;
            sum->right = NULL;
            sum->p = u;
            if (val >= 0)
                u->left = sum;
            else
                u->right = sum;
        }
        else {
            path = u->key;
            addAux (u->left, path);
            addAux (u->right, path);
        }
    }
}

```

2) stampaNum (Array A, int m, int k) {

//Crea vettore occorrenze [m]

dim \leftarrow m - k

For $i=1$ to m

$C[i] = 0$

For $i=1$ to dim

$C[A[i]-m^2]++$

For $i=1$ to m {

If ($C[i] == 0$)

print ($C[i]+m^2$)

}

}