

# Algoritmi e Strutture Dati

a.a. 2021/22

## Compito del 10/01/2023

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

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

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

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

### Algoritmi

#### Parte I

(30 minuti; ogni esercizio vale 2 punti)

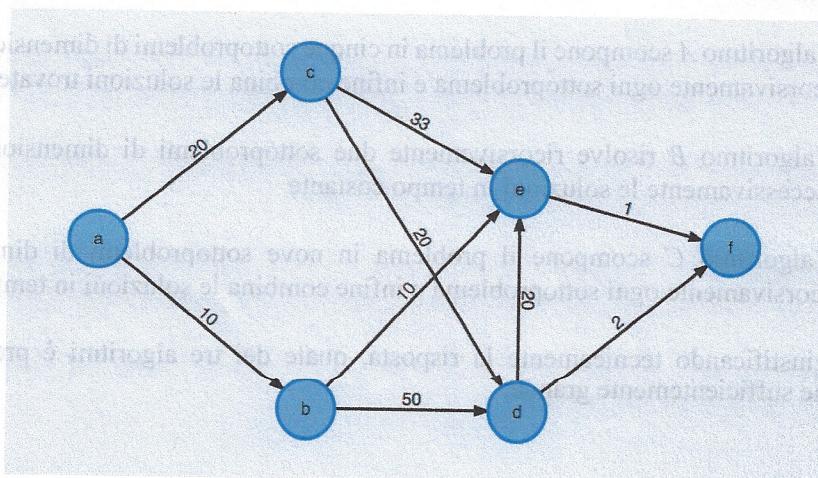
**Avvertenza:** Si giustifichino tecnicamente tutte le risposte. In caso di discussioni poco formali o approssimative gli esercizi non verranno valutati pienamente.

- Si consideri una tabella Hash di dimensione  $m = 8$ , e indirizzamento aperto con doppio Hashing basato sulle funzioni  $h_1(k) = k \bmod m$  e  $h_2(k) = 1 + 2 * (k \bmod (m - 1))$ . Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 10, 66, 71, 157.

- In un grafo non orientato e pesato  $G$ , costituito da 10 vertici e 8 archi, risulta che i pesi degli archi sono uguali tra di loro. Stabilire (giustificando la risposta) quale delle seguenti affermazione è corretta:

- a)  $G$  non possiede un albero di copertura minima
- b)  $G$  possiede esattamente un albero di copertura minima
- c)  $G$  possiede più di un albero di copertura minima

- Si supponga di eseguire l'algoritmo di Dijkstra sul seguente grafo:



In quale ordine verranno estratti i vertici del grafo e quante operazioni di rilassamento ("relax") realizzerà l'algoritmo?

- a) nel caso in cui si utilizzi "a" come vertice sorgente? A - B - C - E - F - D 9
- b) nel caso in cui si utilizzi "f" come vertice sorgente? NON HO ARCHI USCENTI, QUINDI ESTRAGGO SOLO F E NON APPLICO RELAX

# Algoritmi e Strutture Dati

a.a. 2021/22

## Compito del 10/01/2023

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 i cui nodi  $x$  hanno i campi **left**, **right** e **key**, dove **key** è un numero intero. L'albero si dice  **$k$ -compreso**, per un certo numero naturale  $k$ , se per ogni nodo  $x$  la somma delle chiavi dei nodi dell'albero radicato in  $x$  è compresa tra  $-k$  e  $k$ .

a. Scrivere una funzione **efficiente** in C o C++  **$k\_compreso(u, k)$**  che dato in input la radice  $u$  di un albero  $T$  e un valore  $k$  verifica se  $T$  è  **$k$ -compreso** e ritorna 1 se  $T$  è  **$k$ -compreso**, 0 altrimenti.

b. Valutare la complessità della funzione, indicando eventuali relazioni di ricorrenza e la loro **risoluzione tramite il metodo di sostituzione**.

c. Specificare il linguaggio di programmazione scelto.

2. Dato un vettore  $A$  di  $n$  numeri naturali, scrivere una procedura **efficiente** che ordini  $A$  in modo tale che nel vettore risultante, dati  $i$  e  $j$  con  $1 \leq i \leq j \leq n$ , vale  $\text{mod}(A[i], 3) \leq \text{mod}(A[j], 3)$ , dove  $\text{mod}(x, 3)$  è il resto della divisione di  $x$  per 3.

a. Dire se la soluzione proposta è in loco e se è stabile.

b. Valutare e giustificare la complessità della procedura proposta.

3. Per un certo problema sono stati sviluppati tre algoritmi risolutivi (dove  $n$  rappresenta la dimensione dei dati di ingresso):

- L'algoritmo  $A$  scomponete il problema in cinque sottoproblemi di dimensioni dimezzate, risolve ricorsivamente ogni sottoproblema e infine combina le soluzioni trovate in tempo lineare
- L'algoritmo  $B$  risolve ricorsivamente due sottoproblemi di dimensione  $n - 1$  e combina successivamente le soluzioni in tempo costante
- L'algoritmo  $C$  scomponete il problema in nove sottoproblemi di dimensione  $n/3$ , risolve ricorsivamente ogni sottoproblema e infine combina le soluzioni in tempo quadratico

Si dica, giustificando tecnicamente la risposta, quale dei tre algoritmi è preferibile per input di dimensione sufficientemente grande.

4. Si definiscano formalmente le classi P, NP ed NPC e si enunci e dimostri il teorema fondamentale della NP-completezza. Siano inoltre  $P$  e  $Q$  due problemi in NP e si supponga  $P \leq_p Q$ . Si stabilisca se la seguente affermazione è vera o falsa: "Se  $P$  è un problema NP-completo, allora  $Q$  è NP-completo". Potremmo dire la stessa cosa se  $Q \leq_p P$ ? Perché?

## PART I

1)

0
1
2
3
4
5
6
7
10
157
66
71

$$m=8 \quad k = 10, 66, 71, 157$$

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

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

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

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

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

$$h(0, 10) = (10 \bmod 8 + 0(1+2(10 \bmod 5))) \bmod 8 = (2+0) \bmod 8 = 2$$

$$h(0, 66) = (66 \bmod 8 + 0(1+2(66 \bmod 5))) \bmod 8 = (2+0) \bmod 8 = 2 \text{ collisione}$$

$$h(1, 66) = (2+1(1+2(1))) \bmod 8 = (2+1(1+2)) \bmod 8 = 5 \bmod 8 = 5$$

$$h(0, 71) = (71 \bmod 8 + 0(1+2(71 \bmod 5))) \bmod 8 = (7+0) \bmod 8 = 7$$

$$h(0, 157) = (157 \bmod 8 + 0(1+2(157 \bmod 5))) \bmod 8 = (5+0) \bmod 8 = 5 \text{ collisione}$$

$$\begin{aligned} h(1, 157) &= (157 \bmod 8 + 1(1+2(157 \bmod 5))) \bmod 8 = (5+1(1+4)) \bmod 8 = \\ &= (5+1+4) \bmod 8 = 10 \bmod 8 = 2 \text{ collisione} \end{aligned}$$

$$\begin{aligned} h(2, 157) &= (157 \bmod 8 + 2(1+2(157 \bmod 5))) \bmod 8 = (5+2(1+4)) \bmod 8 = \\ &= (5+2+8) \bmod 8 = 15 \bmod 8 = 7 \text{ collisione} \end{aligned}$$

$$\begin{aligned} h(3, 157) &= (157 \bmod 8 + 3(1+2(157 \bmod 5))) \bmod 8 = (5+3(1+4)) \bmod 8 = \\ &= (5+3+12) \bmod 8 = 20 \bmod 8 = 4 \end{aligned}$$

2) Grafo G: 10 vertici, 8 archi, tutti i pesi uguali;

ANALISI:

- Un grafo con  $|V|=10$  e  $|E|=8$  non è连通的, poiché servono almeno  $|V|-1=9$  archi per connettere 10 vertici
- Un albero di copertura minima richiede che il grafo sia连通的

G non possiede un albero di copertura minima. La risposta corretta è A

## PART II

1) bool KCompreso (PNode u, int k) {

    int sum = 0;

    return KCompresoAux (u, k, sum);

}

bool KCompresoAux (PNode u, int k, int & sum) {

    if (u == nullptr)

        return true;

    else {

        int sommaSx = 0;

        int sommaDx = 0;

        bool leftKContained = KCompresoAux (u->left, k, sommaSx);

        bool rightKContained = KCompresoAux (u->right, k, sommaDx);

        sum = sommaSx + sommaDx + u->key;

        bool ukContained = sum < k && sum > -k;

        return leftKContained && rightKContained && ukContained;

}

}

$T(n) = O(h)$

2) void swap (vector<int>& A, int i, int j) {

    int temp = A[i];

    A[i] = A[j];

    A[j] = temp;

```

void OrdineModulo3(vector<int>& A) {
    int m = A.size();
    // Prima partition: separa gli elementi con resto 0
    int k = 0; // Indice per elementi con resto 0
    for (int i = 0; i < m; i++) {
        if (A[i] % 3 == 0) {
            swap(A, k, i);
            k++;
        }
    }
}

```

```

// Seconda partition: separa gli elementi con resto 1 da quelli con resto 2
int j = k;
for (int i = k; i < m; i++) {
    if (A[i] % 3 == 1) {
        swap(A, j, i);
        j++;
    }
}
}

```

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

ALGORITMO IN COCO E STABILE

3) ALGORITMO A:

$$T(n) = 5T\left(\frac{n}{2}\right) + n$$

$$a = 5 \quad b = 2 \quad f(n) = n \quad g(n) = n^d = n^{\log_2 5}$$

$$f(n) \leq g(n)$$

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

$$n = O(n^{\log_2 5 - \varepsilon}) \rightarrow \varepsilon = \log_2 5 - 1 > 0$$

$$T(n) = \Theta(n^d) = \Theta(n^{\log_2 5})$$

ALGORITMO B:  $T(n) = 2T(n-1) + c$

$$2 \underbrace{T(n-1)}_{\text{c}} + c$$

$$2(2T(n-1) - 1) + c + c$$

$$2^2 T(n-2) + 2c$$

$$2^k T(n-k) + \sum_{i=0}^{k-1} 2^i \cdot c$$

CONDIZIONE DI ARRESTO

Quando  $k=n$ ,  $T(n-k) = T(0) = O(1)$ . Sostituendo:  $T(n) = 2^n \cdot O(1) + \sum_{i=0}^{n-1} 2^i \cdot O(1)$

Il secondo termine è una sommatoria geometrica

$$\sum_{i=0}^k q^i = \frac{q^{k+1}-1}{q-1} \Rightarrow \sum_{i=0}^{n-1} 2^i = \frac{2^{n-1+1}-1}{2-1} = 2^n - 1$$

Quindi  $T(n) = O(2^n) + O(2^n) = O(2^n)$

ALGORITMO C:  $T(n) = g T\left(\frac{n}{3}\right) + O(n^2)$

$$a=g \quad b=3 \quad f(n)=n^2 \quad g(n)=n^d = n^{\log_3 g} = n^2$$

$$f(n) \approx g(n)$$

2° CASO

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

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

In conclusione abbiamo

ALGORITMO A:  $\Theta(n \log_2 5) \rightarrow$  PEGGIO DI  $O(n^2)$

ALGORITMO B:  $\Theta(2^n) \rightarrow$  ESPONENZIALE, PEGGIORÉ DEI TRE

ALGORITMO C:  $\Theta(n^2 \log n) \rightarrow$  PREFERIBILE

4)  $P = \{ \mathcal{P} \mid \mathcal{P} \text{ è un problema decisionale per il quale } \exists \text{ algoritmo polinomiale che risolve il problema} \}$

$NP = \{ \mathcal{P} \mid \mathcal{P} \text{ è un problema decisionale per il quale } \exists \text{ algoritmo polinomiale di verifica per } \mathcal{P} \}$

$NPC = \{ \mathcal{P} \mid \mathcal{P} \in NPC \text{ è un problema decisionale tale che:}$

- 1)  $\mathcal{P} \in NP$
- 2)  $\forall \mathcal{P}' \in NP : \mathcal{P}' \leq_p \mathcal{P}$

### TEOREMA FONDAMENTALE DELLA NP COMPLETITÀ

$$P \cap NPC \neq \emptyset \Rightarrow P = NP$$

Per ipotesi:  $\exists \mathcal{P} \in NPC \cap P$ , cioè  $\mathcal{P} \in NPC$  e  $\mathcal{P} \in P$

Dimostrazione:

-  $P \subseteq NP \Rightarrow$  Banalmente vero. Tutti i problemi risolvibili polinomialmente sono anche verificabili polinomialmente.

-  $NP \subseteq P \Rightarrow$  Sia  $\mathcal{P}' \in NP$ . Allora dovrà valere che  $\nexists Q \in NP : Q \leq_p \mathcal{P}'$

Per ipotesi  $\mathcal{P} \in NPC$

Per definizione di  $NPC$ :  $\mathcal{P} \in NPC$  se  $\begin{cases} \mathcal{P} \in NP \\ \forall \mathcal{P}' \in NP : \mathcal{P}' \leq_p \mathcal{P} \end{cases}$

Applico l'ipotesi:  $\mathcal{P}' \leq_p \mathcal{P} \in P$

Dunque  $\mathcal{P}' \in NP$  ma  $\mathcal{P}' \in P \Rightarrow$  Se  $P \cap NPC \neq \emptyset \Rightarrow P = NP$

Supponendo  $P \neq Q$  due problemi in  $NP$  e  $P \leq_p Q$ :

- "Se  $P$  è un problema  $NP$ -completo, allora  $Q$  è  $NP$ -completo"

VERO. Se  $P \leq_p Q$  e  $P \in NPC$ , allora  $Q \in NPC$ . La riducibilità polinomiale e l'appartenenza a  $NP$  garantiscono che  $Q$  eredita la proprietà di  $NPC$ .

### DIMOSTRAZIONE

-  $P \in NPC$  implica che  $P \in NP$  e ogni problema in  $NP$  è riducibile in tempo polinomiale  $\rightarrow P$

- Poiché  $P \leq_p Q$ , ogni problema in  $NP$  che è riducibile a  $P$  può essere ridotto in tempo polinomiale a  $Q$  (componendo le riduzioni polinomiali)

- Quindi, Q è almeno difficile quanto P e può risolvere tutti i problemi in NP tramite riduzione polinomiale
- Poiché Q ∈ NP (dato iniziale), e tutti i problemi in NP sono riducibili a Q, segue che Q è NPC. Ovvvero,  $Q \in \text{NP} \Leftrightarrow Q \in \text{NPC}$  ( $Q \in \text{NP-HARD}$ )
- "Se  $Q \leq_p P$ , allora se P è un problema NP-completo, allora Q è NP-completo" FALSO. La riducibilità  $Q \leq_p P$  implica che Q è almeno tanto facile quanto P, ma non garantisce che ogni problema in NP sia riducibile in tempo polinomiale a Q. Anche se P è NPC, questa condizione non dice nulla su quanto Q sia difficile rispetto agli altri problemi in NP. Q potrebbe addirittura essere un problema "più facile" o addirittura in P, senza avere la proprietà che tutti i problemi in NP siano riducibili a Q.

## PART II ES.2 ALTERNATIVA

```

void sortByMod3 (std::vector<int> & A) {
    // Costruiamo il vettore ordinato
    std::vector<int> sorted_A;
    for (int remainder = 0; remainder < 3; remainder++) {
        for (int num : A) {
            if (num % 3 == remainder)
                sorted_A.push_back(num);
        }
    }
    // Copiamo il vettore ordinato nel vettore originale
    for (size_t i = 0; i < A.size(); i++) {
        A[i] = sorted_A[i];
    }
}

int main() {
    std::vector<int> A = {4, 7, 2, 5, 8, 1, 6, 3, 9, 11, 16};
    sortByMod3(A);
    std::cout << "Vettore ordinato: ";
    for (int num : A) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
    return 0;
}

```