

```
1) void rimuovi (PNode u) {
    if (u != nullptr) {
        rimuovi (u->left);
        rimuovi (u->right);
        delete u
    }
}
```

```
void eliminaMaggioriK (PTree T, int k) {
```

```
    PNode iter, temp;
    iter = T->root;
    while (iter->key != k) {
        if (iter->key > k) {
            rimuovi (iter->right);
            iter->right = nullptr;
            temp = iter;
            transplant (T, iter, iter->left);
            iter = iter->left;
        }
        else
            iter = iter->right;
    }
    rimuovi (iter->right);
    iter->right = nullptr;
}
```

$$T(n) = \Theta(n)$$

```
typedef struct Node {
    int key;
    Node* left;
    Node* right;
    Node* p;
}* PNode;
```

```
typedef struct Tree {
    PNode root;
}* PTree;
```

```
2) void ordina (vector<int>& arr) {
```

```
    int m = arr.size();
```

```
    if (m > 1) {
```

// Poiché i numeri vanno da 0 a $m^4 - 1$ abbiamo bisogno di 4 passi perché $m^4 - 1$

// può essere rappresentato con 4 cifre in base m

```
    for (int exp = 1; exp <= pow(m, 3); exp *= m) {
```

```
        countingSort (arr, exp, m)
```

```
}
```

```
}
```

```
}
```

```
void countingSort (vector<int>& A, int exp, int m) {
```

```
    vector<int> output (m);
```

```
    vector<int> count (m, 0);
```

// Conta le occorrenze di ogni cifra

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

```
        int index = (arr[i] / exp) % m;
```

```
        count[index]++;
```

```
}
```

```
    for (int i = 1; i < m; i++) {
```

```
        count[i] += count[i + 1];
```

```
}
```

```
    for (int i = m - 1; i >= 0; i--) {
```

```
        int index = (arr[i] / exp) % m;
```

```
        output[count[index] - 1] = arr[i];
```

```
        count[index]--;
```

```
}
```

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

```
        arr[i] = output[i];
```

```
}
```

```
}
```

SPIEGAZIONE DELL'ALGORITMO

Radix Sort adattato:

- Usiamo una versione modificata di Radix Sort che opera in base m invece che in base 10
- Questo perché i numeri sono compresi tra 0 e $m^4 - 1$, quindi possono essere rappresentati con al massimo 4 cifre in base m

Counting Sort:

- Per ogni "cifra" (in base m) eseguiamo un Counting Sort
- Questo garantisce la stabilità dell'ordinamento

COMPLESSITÀ

Counting Sort: $O(m)$ lo eseguiamo 4 volte

$$T(m) = O(m)$$

3) CORRETTEZZA

My Function:

- Calcola i cammini minimi da x a tutti gli altri vertici (simile a Dijkstra)
- Cerca l'arco (u, x) che minimizza $d[u] + w(u, x)$, che rappresenta un ciclo passante per x
- Se tale ciclo esiste, restituisce il suo peso, altrimenti $+\infty$

My Algorithm:

- Considera ogni vertice come possibile punto di partenza/fine di un ciclo
- Mantiene il minimo tra tutti i cicli trovati
- Se nessun ciclo esiste, restituisce $+\infty$

COMPLESSITÀ COMPUTAZIONALE

My Function:

- Implementazione modificata di Dijkstra: $O(n^2)$
- Si potrebbe ottimizzare come heap a $O(m + m \log m)$

My Algorithm:

- Chiama MyFunction m volte
- Complessità totale $O(m^3)$
- Con heap binario $O(mm + m \log m)$

VERTICE A

Cammini minimi da A:

- A: 0
- B: 2 ($A \rightarrow B$)
- C: 7 ($A \rightarrow B \rightarrow C$)
- E: 5 ($A \rightarrow B \rightarrow D \rightarrow E$)
- D: 7 ($A \rightarrow B \rightarrow C \rightarrow D$ oppure $A \rightarrow B \rightarrow E \rightarrow D$)

Archi entranti in A

- E \rightarrow A (peso 2)
- C \rightarrow A (peso 2)

Calcolo del ciclo minimo:

$$\cdot d[E] + \omega(E \rightarrow A) = 5 + 2 = 7$$

$$\cdot d[C] + \omega(C \rightarrow A) = 7 + 2 = 9$$

Minimo ciclo che termina in A: $\min(7, 9) = 7$ ($A \rightarrow B \rightarrow D \rightarrow E \rightarrow A$)

VERTICE B

Cammini minimi da B:

- B: 0
- C: 5 ($B \rightarrow C$)
- E: 3 ($B \rightarrow E$)
- A: 5 ($B \rightarrow E \rightarrow A$ oppure $B \rightarrow C \rightarrow A$)
- D: 4 ($B \rightarrow E \rightarrow D$ oppure $B \rightarrow C \rightarrow D$)

Archi entranti in B:

- A \rightarrow B (peso 2)

Calcolo del ciclo minimo:

$$d[A] + \omega(A \rightarrow B) = 5 + 2 = 7$$

Minimo ciclo che termina in B: 7 ($B \rightarrow E \rightarrow A \rightarrow B$)

VERTICE C

Cammini minimi da C:

- C: 0
- D: 1 ($C \rightarrow D$)
- A: 2 ($C \rightarrow A$)
- B: 4 ($C \rightarrow A \rightarrow B$)
- E: 6 ($C \rightarrow A \rightarrow B \rightarrow E$)

Archi entranti in C:

- $B \rightarrow C$ (peso 5)

Calcolo del ciclo minimo:

$$d[B] + w(B \rightarrow C) = 4 + 5 = 9$$

Minimo ciclo che termina in C: 9 ($C \rightarrow A \rightarrow B \rightarrow C$)

VERTICE D

Cammini minimi da D

- D: 0
- Nessun arco uscente \rightarrow Tutti gli altri nodi rimangono a $+\infty$

Archi entranti in D

- $C \rightarrow D$ (peso 1)
- $E \rightarrow D$ (peso 1)

Calcolo del ciclo minimo

- $d[C] = +\infty, d[E] = +\infty \rightarrow$ nessun ciclo valido

Restituisce $+\infty$

VERTICE E

Cammini minimi da E

- E: 0
- D: 1 ($E \rightarrow D$)
- A: 2 ($E \rightarrow A$)
- B: 4 ($E \rightarrow A \rightarrow B$)
- C: 9 ($E \rightarrow A \rightarrow B \rightarrow C$)

Archi entranti in E

- B \rightarrow E (peso 3)

Calcolo del ciclo minimo

$$d[B] + w(B \rightarrow E) = 4 + 3 = 7$$

Minimo ciclo che termina in E: 7 ($E \rightarrow A \rightarrow B \rightarrow E$)

RISULTATO DI MYALGORITHM

Valori restituiti da MyFunction

- A: 7
- B: 7
- C: 9
- D: + ∞
- E: 7

Minimo tra tutti i cicli: $\min(7, 7, 9, +\infty, 7) = 7$

Peso del ciclo minimo nel grafo è 7

Cicli corrispondenti:

- 1) $A \rightarrow B \rightarrow E \rightarrow A$ ($2 + 3 + 2 = 7$)
- 2) $B \rightarrow E \rightarrow A \rightarrow B$ ($3 + 2 + 2 = 7$)
- 3) $E \rightarrow A \rightarrow B \rightarrow E$ ($2 + 2 + 3 = 7$)

4) L'algoritmo restituisce il numero di componenti connesse nel grafo non orientato G.

Ecco perché:

1) INIZIALIZZAZIONE: L'algoritmo inizia creando un insieme separato (MAKE-SET) per ogni vertice, incrementando un contatore m per ogni vertice. All'inizio m è uguale al numero totale di vertici.

2) ELABORAZIONE DEGLI ARCHI: Per ogni arco (u, v) , se u e v appartengono a insiemi diversi ($\text{FIND-SET}(u) \neq \text{FIND-SET}(v)$) vengono uniti (UNION) e il contatore m viene decrementato.

3) LOGICA: Ogni UNION riduce m di 1 perché due componenti separate vengono unite in una. Alla fine, m rappresenterà il numero di insiemi disgiunti rimanenti, che corrisponde al numero di componenti connesse nel grafo.

$\{a\} \{b\} \{c\} \{d\} \{e\} \{f\} \{g\} \{h\} \{i\} \{j\}$

ARCO A-B

- $\text{FIND-SET}(a) \neq \text{FIND-SET}(b)$

$\text{UNION}(a, b) \rightarrow \{a, b\} \{c\} \{d\} \{e\} \{f\} \{g\} \{h\} \{i\} \{j\}$

$$m = 10 - 1 = 9$$

ARCO A-C

- $\text{FIND-SET}(a) \neq \text{FIND-SET}(c)$

$\text{UNION}(a, c) \rightarrow \{a, b, c\} \{d\} \{e\} \{f\} \{g\} \{h\} \{i\} \{j\}$

$$m = 9 - 1 = 8$$

ARCO B-C

- $\text{FIND-SET}(b) = \text{FIND-SET}(c)$

ARCO B-D

- $\text{FIND-SET}(b) \neq \text{FIND-SET}(d)$

$\text{UNION}(b, d) \rightarrow \{a, b, c, d\} \{e\} \{f\} \{g\} \{h\} \{i\} \{j\}$

$$m = 8 - 1 = 7$$

ARCO E-F

- $\text{FIND-SET}(e) \neq \text{FIND-SET}(f)$

$\text{UNION}(e, f) \rightarrow \{a, b, c, d\} \{e, f\} \{g\} \{h\} \{i\} \{j\}$

$$m = 7 - 1 = 6$$

ARCO E-G

• FIND-SET(e) ≠ FIND-SET(g)

UNION(e,g) → {a,b,c,d}{e,f,g}{h}{i}{j}

$$m=6-1=5$$

ARCO H-I

• FIND-SET(h) ≠ FIND-SET(i)

UNION(h,i) → {a,b,c,d}{e,f,g}{h,i}{j}

$$m=5-1=4$$

L'algoritmo restituisce $m=4$, che corrisponde al numero di componenti connesse nel grafo.