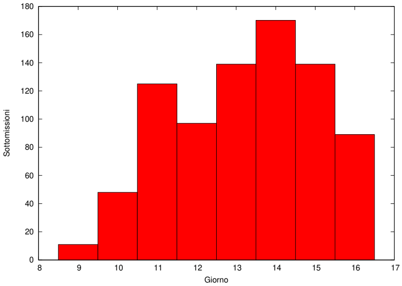
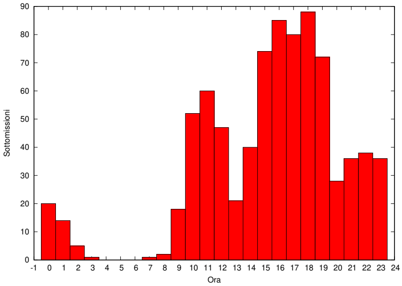
Soluzione Primo Progetto ASD 2020/2021

Lettere da Powarts

STATISTICHE

Numero sottoposizioni: `819`





- 73 gruppi partecipanti, di cui 69 gruppi hanno fatto almeno una sottoposizione;
- 178 studenti iscritti, di cui 174 appartenenti a gruppi che hanno fatto almeno una sottoposizione;

Risultati

Punteggi

- $P < 30$ $\longrightarrow$ progetto non passato
- $30 \leq P < 75$ $\longrightarrow$ 1 punti bonus (18 gruppi)
- $75 \leq P < 100$ $\longrightarrow$ 2 punti bonus (19 gruppi)
- $P = 100$ $\longrightarrow$ 3 punti bonus (31 gruppi)

Classifiche e sorgenti sul sito (controllate i numeri di matricola):
`https://judge.science.unitn.it/slides/asd20/classifica_prog1.pdf`

IL PROBLEMA I

Consideriamo un grafo connesso e non orientato $G = (V, E)$ con N nodi ed M archi e un nodo P rappresentante la città di Powarts. Partendo dal nodo P, scegliendo sempre **il percorso più breve**, verranno raggiunti gli altri nodi, per la consegna delle lettere per Powarts.

Il problema II

Colui Che Non Deve Essere Nominato attaccherà una città e costringerà i gufi a non passare per essa durante le consegne. Questo corrisponde al rimuovere un nodo e tutti gli archi ad esso collegati.

Il nodo da rimuovere R è scelto in modo da **massimizzare** il numero K di nodi n per cui il cammino minimo tra P ed n è più lungo del cammino minimo prima della rimozione di R.

Problema

Trovare K, ovvero il massimo numero di studenti che non riceverà la lettera entro il tempo minimo previsto inizialmente.

Sottoproblema: alberi

Se il grafo di input è un albero e scegliamo Powarts come radice, la scelta migliore per Colui Che Non Deve Essere Nominato sarà attaccare il figlio u di P tale che il numero di nodi nel sottoalbero radicato in u sia massimo. Quindi:

- contiamo il numero di nodi in ogni sottoalbero radicato in un figlio di P (con delle visite);
- dopo aver individuato il nodo attaccato u, usiamo un'ulteriore visita per stampare l'elenco dei nodi presenti nel sottoalbero radicato in u.

- ⇒ soluzione: `tree.cpp` (49 SLOC, Source Lines Of Code[1])
- ⇒ complessità: $O(N + M)$
- ⇒ 45 punti

[1] Ottenuto con SLOCCount di David A. Wheeler

Sottoproblema: cammini minimi unici

Se il grafo in input è tale che esiste **un unico percorso di lunghezza minima** tra Powarts e ogni altro nodo, è possibile ricondursi al caso degli alberi:

- calcoliamo l'albero dei cammini minimi;
- applichiamo la soluzione precedente a questo albero.

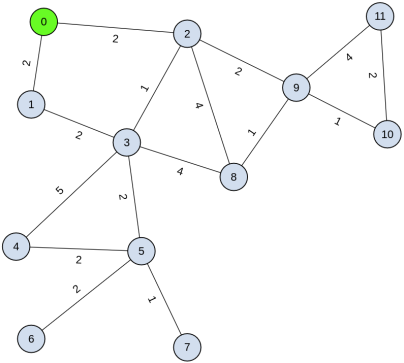
DEFINIZIONE

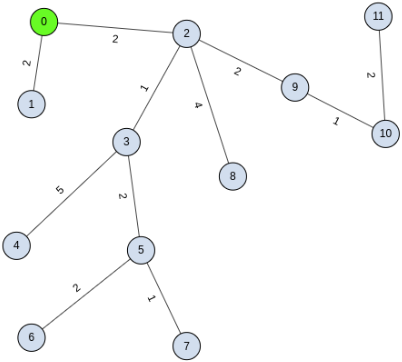
L'albero dei cammini minimi T di un grafo pesato G, rispetto ad un vertice P, è un sottografo di G che:

- contiene tutti i vertici di G;
- ha come radice P;
- è tale che la distanza tra P e un qualsiasi altro vertice u in T è pari alla lunghezza del percorso più breve da P a u in G.

Nota: in generale l'albero dei cammini minimi non è unico.

ESEMPIO


0
2
11
2
2
4
1
2
2
1
9
4
1
2
3
10
4
8
5
2
4
5
2
2
1
6
7


0
2
11
2
2
2
1
9
4
1
2
1
3
10
8
5
2
4
5
2
1
6
7

DIJKSTRA

Con l'algoritmo di Dijkstra è possibile calcolare in modo efficiente la distanza minima tra il nodo *P* ed ogni altro nodo nel grafo.

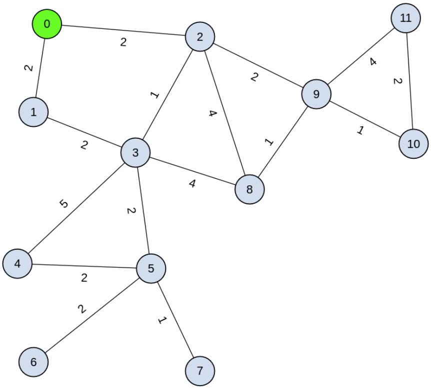
```
struct node {
    int id;
    int dist;
};

bool operator < (const node a, const node b){
    return a.dist > b.dist;
}

void dijkstra(int source, vector<bool>& visited){
    priority_queue<node> q;
    q.push({source,0});
    dist[source] = 0;
    ...
```

```
    while (!q.empty()){
        int u = q.top().id;
        int d = q.top().dist;
        q.pop();
        if(!visited[u]){
            visited[u] = true;
            for(pair<int,int> p : graph[u]) {
                int v = p.first;
                int weight = p.second;
                if(!visited[v]){
                    if (dist[v] > dist[u] + weight){
                        dist[v] = dist[u] + weight;
                        q.push({v,dist[v]});
                    }
                }
            }
        }
    }
}
```

ESEMPIO



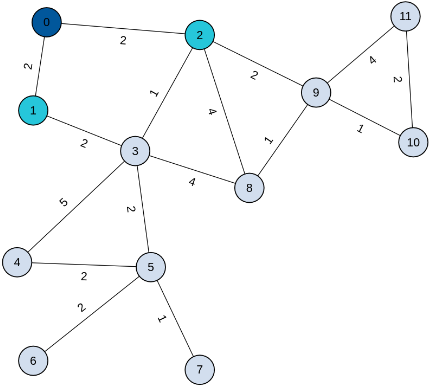
- Nodo di partenza: 0
- Distanza: 0

dist =

0	1	2	3	4	5	6	7	8	9	10	11
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

```
priority_queue = {[0,0]}
```

Esempio



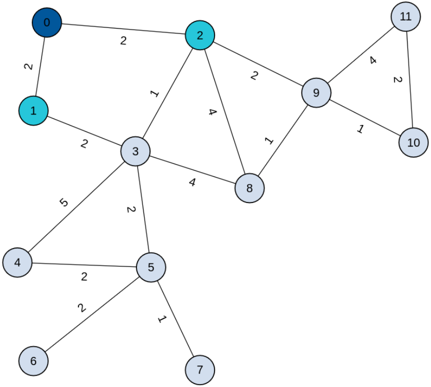
- Nodo: 0
- Distanza: 0

dist =

0	1	2	3	4	5	6	7	8	9	10	11
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

`priority_queue = { }`

ESEMPIO



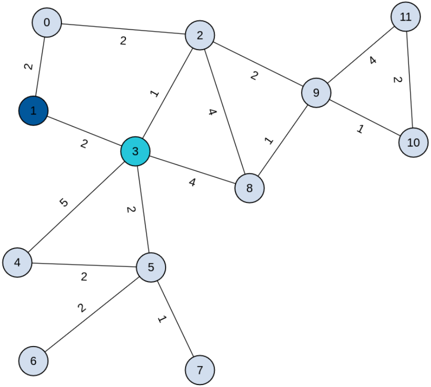
- Nodo: 0
- Distanza: 0
- Nodo adiacente: 1
- Distanza: $0 + 2 = 2$
- Nodo adiacente: 2
- Distanza: $0 + 2 = 2$

`dist` =

0	1	2	3	4	5	6	7	8	9	10	11
0	2	2	∞	∞	∞	∞	∞	∞	∞	∞	∞

`priority_queue = { [1,2], [2,2] }`

ESEMPIO

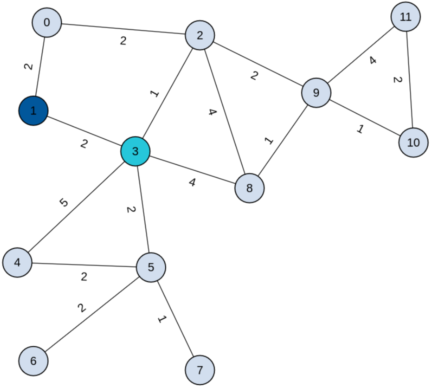


- Nodo: 1
- Distanza: 2

`dist =`

0	1	2	3	4	5	6	7	8	9	10	11
0	2	2	∞	∞	∞	∞	∞	∞	∞	∞	∞

`priority_queue = { [2, 2] }`

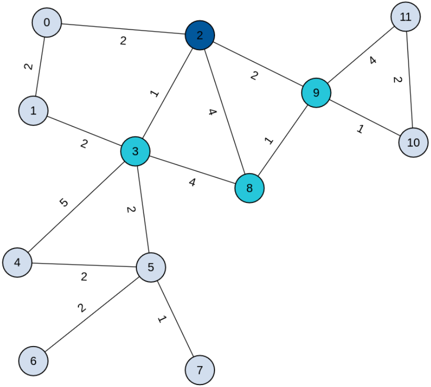


- Nodo: 1
- Distanza: 2
- Nodo adiacente: 3
- Distanza: $2 + 2 = 4$

`dist =`

0	1	2	3	4	5	6	7	8	9	10	11
0	2	2	4	∞	∞	∞	∞	∞	∞	∞	∞

`priority_queue = { [2, 2], [3, 4] }`



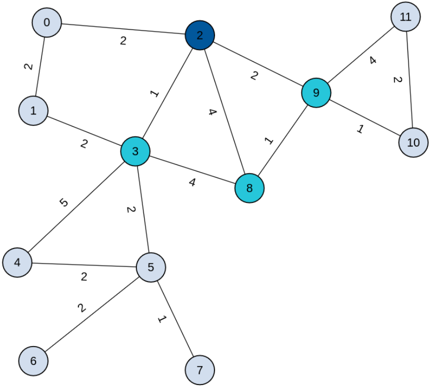
- Nodo: 2
- Distanza: 2

dist =

0	1	2	3	4	5	6	7	8	9	10	11
0	2	2	4	∞	∞	∞	∞	∞	∞	∞	∞

```
priority_queue = { [3, 4] }
```

ESEMPIO

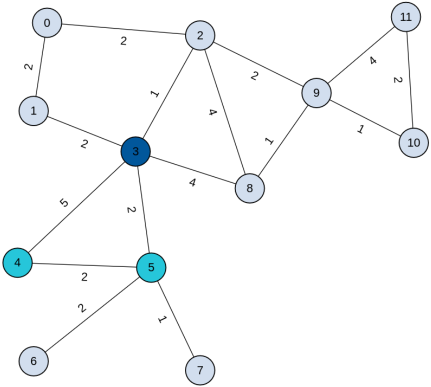


- Nodo: 2
- Distanza: 2
- Nodo adiacente: 3
- Distanza: $2 + 1 = 3$
- Nodo adiacente: 8
- Distanza: $2 + 4 = 6$
- Nodo adiacente: 9
- Distanza: $2 + 2 = 4$

`dist =`

0	1	2	3	4	5	6	7	8	9	10	11
0	2	2	3	∞	∞	∞	∞	6	4	∞	∞

`priority_queue = { [3, 3], [3, 4], [9, 4], [8, 6]}`



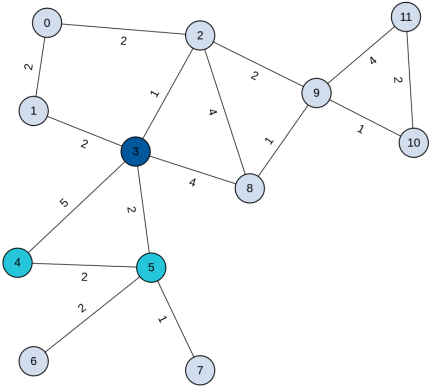
- Nodo: 3
- Distanza: 3

dist =

0	1	2	3	4	5	6	7	8	9	10	11
0	2	2	3	∞	∞	∞	∞	6	4	∞	∞

```
priority_queue = { [3, 4], [9, 4], [8, 6] }
```

ESEMPIO

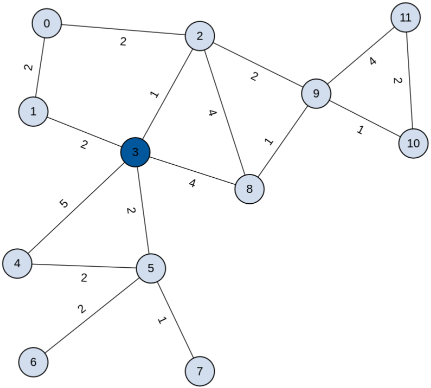


- Nodo: 3
- Distanza: 3
- Nodo adiacente: 4
- Distanza: $3 + 5 = 8$
- Nodo adiacente: 5
- Distanza: $3 + 2 = 5$

`dist =`

0	1	2	3	4	5	6	7	8	9	10	11
0	2	2	3	8	5	∞	∞	6	4	∞	∞

```
priority_queue = { [3, 4], [9, 4], [5, 5], [8, 6],
[4, 8] }
```



- Nodo: 3
- Distanza: 4
- Nodo già visitato, lo ignoro.

dist =

0	1	2	3	4	5	6	7	8	9	10	11
0	2	2	3	8	5	∞	∞	6	4	∞	∞

```
priority_queue = { [9, 4], [5, 5], [8, 6], [4, 8] }
```

Nota: per la costruzione dell'albero dei cammini minimi, ogni volta che aggiorniamo il vettore delle distanze, memorizziamo anche il nodo precedente da cui siamo arrivati.

- $\Rightarrow$ soluzione: `geodetics.cpp` (93 SLOC)
- $\Rightarrow$ complessità: $O((M+N)\log N)$ [2]
- $\Rightarrow$ 70 punti

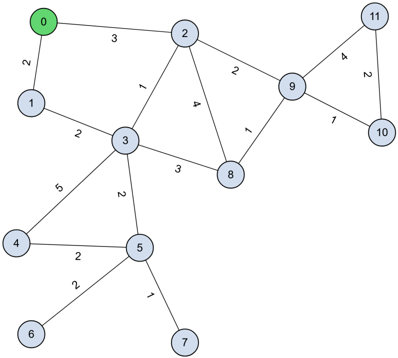
[2] Analisi di complessità per varie implementazioni di una coda con priorità: `https://judge.science.unitn.it/slides/asd17/sol_prog1.pdf`

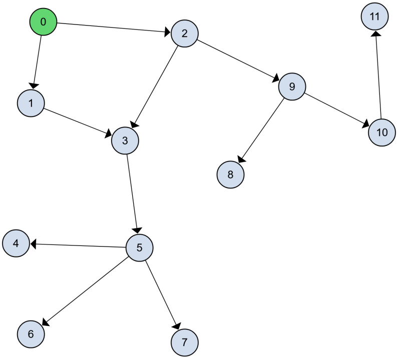
Caso generale

Se il grafo in input è tale che possa esistere **più di un percorso di lunghezza minima** tra Powarts e gli altri nodi, il problema è più difficile da risolvere:

- se esiste più di un percorso di lunghezza minima tra Powarts e un altro nodo, l'albero dei cammini minimi radicato in Powarts non è unico.
- l'unione di tutti gli alberi dei cammini minimi radicati in Powarts dà origine a un DAG[3] dei cammini minimi.

[3] Directed Acyclic Graph

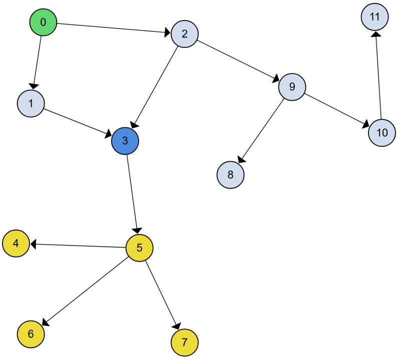




L'unione di tutti gli alberi dei cammini minimi radicati in Powarts dà origine a un DAG dei cammini minimi.

DEFINIZIONE

Dato un nodo sorgente *S*, un nodo *D* è **dominator** di un nodo *N* se **ogni percorso** da *S* a *N* deve passare per *D*.



- il nodo 3 è dominator dei nodi {3, 4, 5, 6, 7}.
- il nodo 5 è dominator dei nodi {5, 4, 6, 7}.

Soluzione

Se il grafo in input è tale che possa esistere **più di un percorso di lunghezza minima** tra Powarts e gli altri nodi:

- calcoliamo il DAG G dei cammini minimi;
- data la radice P di G, troviamo la città U tale che $U \neq P$ e che U sia dominator di più nodi di qualsiasi altro nodo $V \neq P$.

Soluzione

Per trovare i nodi dominati da un certo nodo U, procediamo in questo modo:

- inizializziamo un vettore di interi *dominator* di $|V|$ elementi, inizialmente impostati a un valore impossibile $NONE$;
- dato che non vogliamo considerare P come nodo da attaccare, per ogni vicino U di P diciamo che $dominator[U] = U$;
- visitiamo il grafo G **in ordine topologico**.

Soluzione

Sia S uno stack di nodi in ordine topologico per il grafo G. Per ogni nodo U estratto dallo stack S, vengono visitati i suoi nodi adiacenti. Per ogni nodo adiacente V si distinguono due casi:

- V non è mai stato visto prima. In questo caso, impostiamo $dominator[V] = dominator[U]$;
- V è già stato visto. Questo significa che $dominator[V] \neq NONE$. Si distinguono due ulteriori casi:
  - se $dominator[V] \neq dominator[U]$, significa che il nodo V è raggiungibile da **due cammini minimi aventi come Lowest Common Ancestor[4] la radice P**, e quindi impostiamo $dominator[V] = V$.
  - altrimenti, se $dominator[V] = dominator[U]$ significa che i due cammini minimi che raggiungono V hanno come Lowest Common Ancestor un nodo che non è la radice P, e quindi dipendono da esso.

[4] https://en.wikipedia.org/wiki/Lowest_common_ancestor

Codice

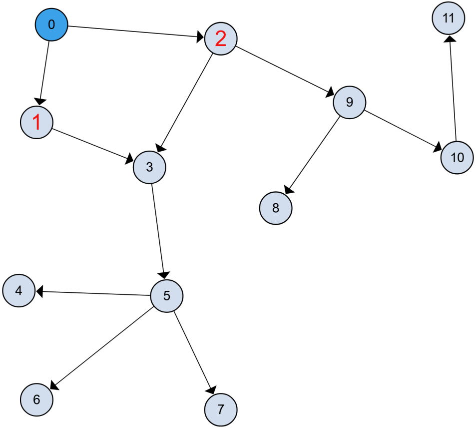
```
int powarts(vector<vector<int>> &dag, stack<int> &top_sort)
    vector<int> dominated_nodes(dag.size(), 0);
    vector<int> dominator(dag.size(), NONE);

    for(int adj : dag[top_sort.top()]){
        dominator[adj] = adj;
        dominated_nodes[adj]++;
    }
    top_sort.pop();
```

CODICE

```
while(!top_sort.empty()){
    int u = top_sort.top();
    top_sort.pop();
    for(int v : dag[u]){
        if(dominator[v] == NONE){
            dominator[v] = dominator[u];
            dominated_nodes[dominator[u]]++;
        }else if(dominator[v] != dominator[u] &&
            dominator[v] != v){
            dominated_nodes[dominator[v]]--;
            dominator[v] = v;
            dominated_nodes[v]++;
        }
    }
}
```

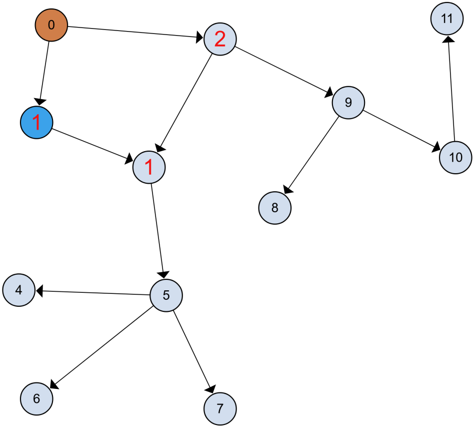
Esempio



- Nodo corrente: 0

```
top_sort = {0, 1, 2, 9, 8, 10, 11, 3, 5, 4, 6, 7}
```

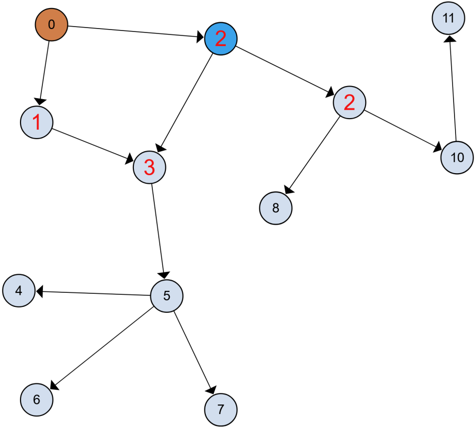
Esempio



- Nodo corrente: 1

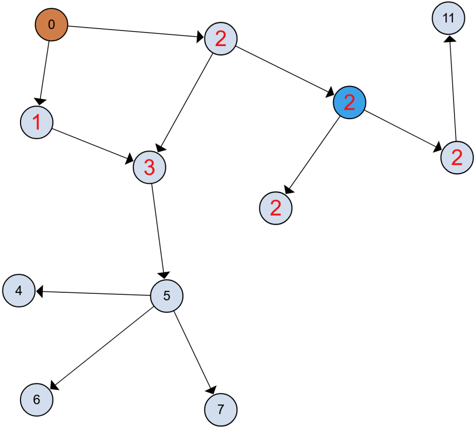
`top_sort = {1, 2, 9, 8, 10, 11, 3, 5, 4, 6, 7}`

ESEMPIO



- Nodo corrente: 2

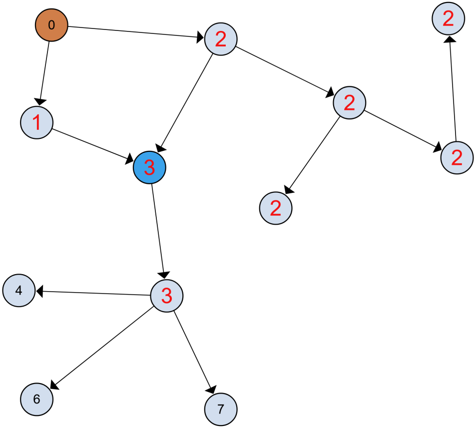
`top_sort = {2, 9, 8, 10, 11, 3, 5, 4, 6, 7}`



- Nodo corrente: 9

`top_sort = {9, 8, 10, 11, 3, 5, 4, 6, 7}`

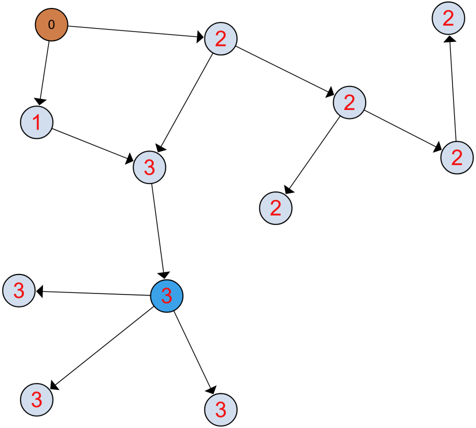
Esempio



- Nodo corrente: 3

`top_sort = {3, 5, 4, 6, 7}`

ESEMPIO



- Nodo corrente: 5

`top_sort = {5, 4, 6, 7}`

- $\Rightarrow$ soluzione: `powarts.cpp` (122 SLOC)
- $\Rightarrow$ complessità: $O((M + N) \log N)$
- $\Rightarrow$ 100 punti