nuova spiegazione del problema dello zaino (Knapsack)

*NOTA: il problema descritto è quello dello **zaino 0-1** (ogni oggetto può essere inserito al massimo **1** sola volta)*

*ATTENZIONE: alle pagine 9-11 l'indice di colonna è indicato con **p** mentre nella implementazione (pag.13-14) con **w***

Condizioni iniziali:

riga 0

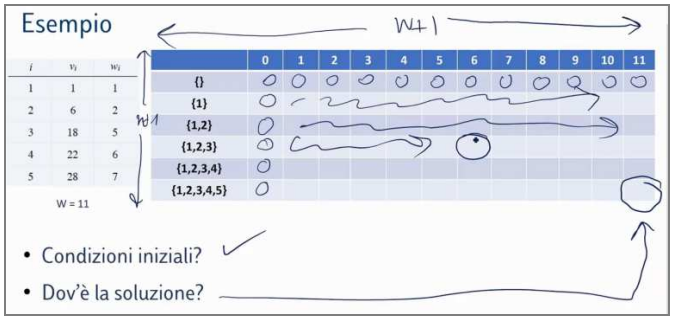
```
FOR w = 0 TO W
    M[0, w] ← 0.
```

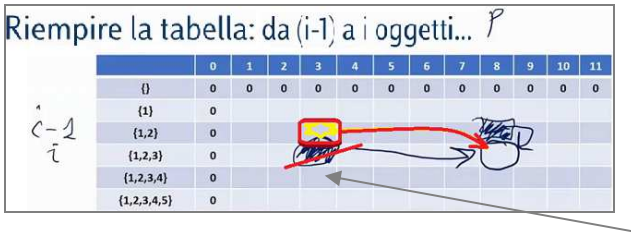
anche colonna 0!!!

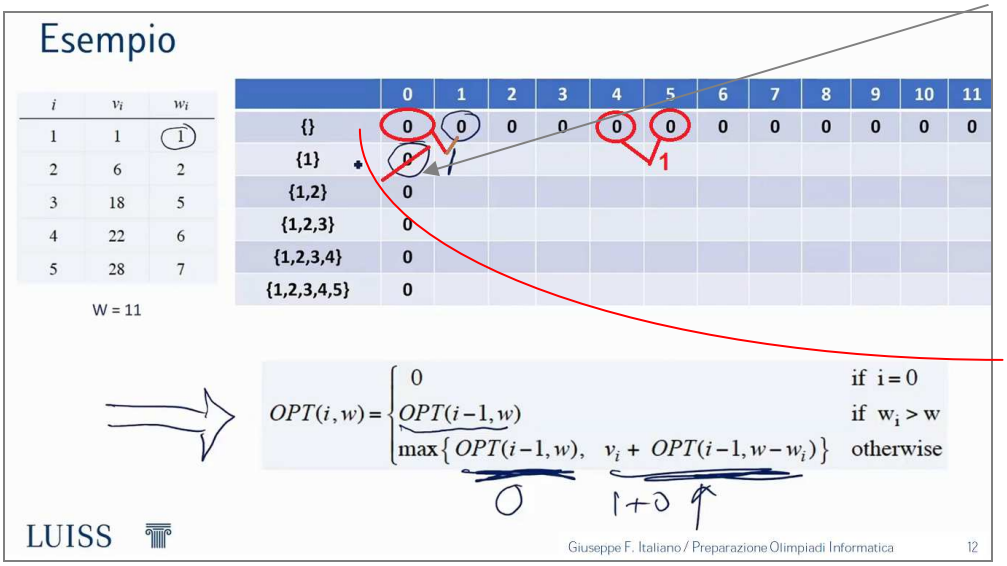
```
FOR i = 1 TO n
    M[i, 0] ← 0.
```

soluzione in $M[n, W]$.

matrice **OPT** ovvero **M** di **n+1 righe** e **w+1 colonne**







riempimento della tabella:

a pag.14 va corretto il ciclo su **w**

```
FOR i = 1 TO n
    FOR w = 1 TO W
```

riempimento per righe:

per la riga i, per ogni colonna p <= W (zaino di peso p) l'elemento di posto (i, p) per p >= $\mathbf{w_i}$ viene impostato in base al miglior valore (max) tra l'elemento (i-1, p) e l'elemento (**i-1, p - w_i**).

*NOTA: il prof ha indicato per errore l'elemento (**i**, p - $\mathbf{w_i}$) anche nell'Esempio per l'elem. (1, 1)*

Per l'**oggetto 1** di peso 1 e valore 1, per uno zaino di peso w=1 il valore ottimo è 1 cioè OPT(1,1)=1 perchè:

$1=w_1 > w=1$ è FALSA quindi

va considerato il **massimo** tra:

OPT(i-1, 1) = OPT(0,1) = **0** **e**

$\mathbf{v_1}$+OPT(**i-1**,w-w_1)= 1+OPT(0,1-1)

=**OPT(0,0)** = 1+**0** = **1** **non** OPT(1,0)

Analogam. per w=5: OPT(1,5)=**1**

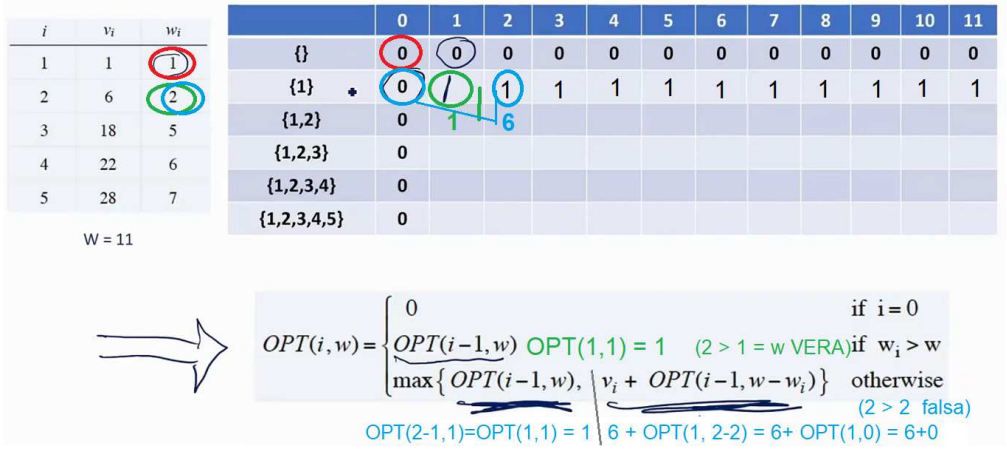
Infatti $w_1=1 > 5$ è FALSA, quindi

va ricercato il massimo tra:

OPT(0, 5) = **0** **e**

v_1+OPT(0,5-1)= 1+**OPT(0,4)**= 1+0 =**1**

Va considerato l'OPT(i-1, **p-w_i**) perchè prendendo l'oggetto **i** di peso w_i va recuperata la soluzione ottima per l'oggetto **i-1** senza il peso di i (cioè w_i) che si sta per aggiungere.



Per l'**oggetto 2** di peso 2 e valore 6, per uno zaino di peso **w=1** la condizione **2**=w_2 >**w=1** è **VERA** quindi il valore ottimo è 1 cioè **OPT(2,1) =1** =OPT(1,1)=OPT(2-1,1)

per uno zaino di peso **w=2** la condizione **2**=w_2 >**w=2** è **FALSA** quindi va considerate il **max** tra:

OPT(i-1, **2**) = OPT(1,2) = **1** **e**

$\mathbf{v_2}$+OPT(i-1,**w-w_2**)= 6 + OPT(1,**2-2**)

= 6 + **OPT(1,0)** = 6+**0** = **6**

L'implementazione corretta per pag. 14 è la seguente

```
for w = 0 to W                                  // Inizializzazione riga 0
        M[0,w] ← 0
for i = 1 to n                                  // Inizializzazione colonna 0
        M[i,0]  ← 0
for  i = 1 to n
 for w = 1 to W
   if (wi > w) {                                //elemento i non fa parte della soluzione per peso w
        M[i,w] ← M[i-1,w]
   else                                         //elemento i potrebbe essere parte della soluzione
        M[i,w] ← max ( M[i-1,w] , vi + M[i-1,w - wi] )   //se il massimo fosse raggiunto dal 2^ termine
RETURN M[n,W]
```

Il programma e la tabella di traccia sono riportati a pag.3 del documento *Problema dello zaino_knapsack.pdf*

L'algoritmo per **individuare gli elementi che determinano la soluzione ottima** è il seguente:

partendo dalla posizione della soluzione M[n,W] fino al raggiungimento dell'elemento iniziale M[0,0]
per ogni elemento di indici i e k:
if (M[**i**, k] <> M[**i-1**, k])
    marcare l'elemento **i-esimo** come parte dello zaino
    k ← k - wi
    i ← i - 1
else
    i ← i - 1 // L'elemento i-esimo non è nello zaino

pag 49 http://didawiki.cli.di.unipi.it/lib/exe/fetch.php/informatica/all-a/problemazaino_pubblicato.pdf

i = 5, k = 11

M(5,11) = M(4,11) = 40 VERA

i = 4 = i-1 = 5-1

M(**4,11**) <> M(3,11) = 25 **falsa** ➔**oggetto 4: val.22 p.6**

k = 5 =k-w4=11-6 **i = 3** =i-1=4-1

M(**3,5**) = 18 <> M(2,5) =7 **falsa** ➔**oggetto 3: val.18 p.5**

k = 0 =k-w3= 5-5 **i = 2** =i-1=3-1

M(2,0) = M(1,0) = 0 VERA

i = 1 =i-1 = 2-1

M(1,0) = M(0,0) = 0 VERA

i = 0 =i-1 = 1-1

Soluzioni proposte dagli alunni per **Numeri di Figonacci** https://training.olinfo.it/#/task/figonacci/statement

```cpp
using namespace std;


int figonacci(int n, int m)
{
    vector<int> fig(n);
    fig[0] = -1;
    fig[1] = 0;
    fig[2] = 1;

    for(int i=3;i<=n;i++)
    {
        //fig[3] = (fig[2]-fig[1]) + (fig[2]-fig[0]) -->
        fig[i] = (fig[i-1]*(i-1))-(fig[i-3]*(i-2));

        fig[i] = (fig[i] % m +m)%m;
    }

    return fig[n];
}

int main()
```

Soluzione parziale (*int* non adatto, meglio *long long*)

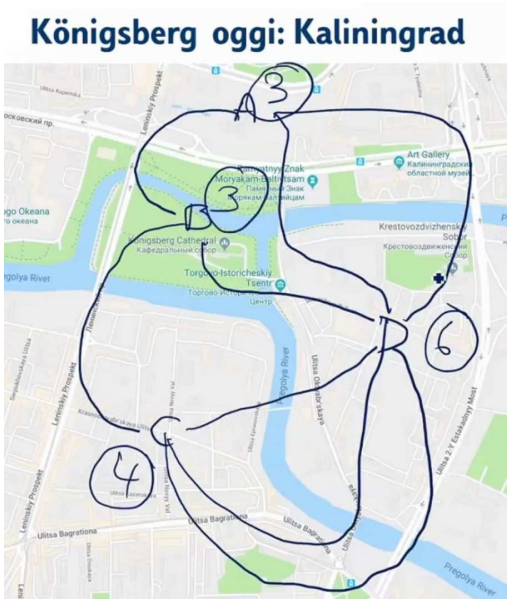
```cpp
#include <iostream>
#include <vector>
#include <cmath>

struct figonacci{
    Long Long num, sott;
};

int main(){
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);

    int n, mod;
    std::cin >> n >> mod;
    std::vector <figonacci>vet (n+1);
    vet[0].num = (-1 % mod + mod) % mod;
    vet[0].sott = (1 % mod + mod) % mod;
    vet[1].num = (0 % mod + mod) % mod;
    vet[1].sott = (1 % mod + mod) % mod;
    for(int i = 2; i < n+1; i++){
        vet[i].num = ((vet[i-1].sott + vet[i-1].num*i) % mod + mod) % mod;
        vet[i].sott = ((vet[i-1].sott - vet[i].num) % mod + mod) % mod;
    }

    std::cout << vet[n].num;
}
```

soluzione completa per i casi di test del Portale

GRAFI

La teoria dei grafi nasce con la pubblicazione di Eulero sui "Sette ponti di Königsberg".

Esiste un percorso che attraversi tutti i ponti ma una sola volta?



Nel 1736 Eulero dimostrò che il percorso non era possibile enunciando il teorema:

> *Un qualsiasi grafo è percorribile se e solo se ha **tutti i nodi di grado pari**, o **due di essi sono di grado dispari**; per percorrere un grafo "possibile" con due nodi di grado dispari, è necessario partire da uno di essi, e si terminerà sull'altro nodo dispari.*
>
> https://it.wikipedia.org/wiki/Problema_dei_ponti_di_K%C3%B6nigsberg

In questo caso il Grafo è il disegno che si ottiene indicando le zone della città come **NODI** (A, B, C, D), detti anche **Vertici**, e i ponti come **ARCHI** che collegano i nodi (le zone).

Il **grado di un nodo** è il numero di archi che partono dal nodo:

- A , C e D sono nodi di grado 3,
- B è di grado 5.

Gli ARCHI possono anche essere orientati (cioè percorribili solo in un verso) e rappresentano relazioni tra i NODI:

grafo non orientato:	**relazione simmetrica**	**A —— B**
grafo orientato:	**relazione non simmetrica**	**A ⟶ B**

Il grafo a pag.21 rappresenta la relazione non simmetrica "B è una lingua incomprensibile per A". In questo caso nodi e archi non rappresentano oggetti fisici ma concetti; la relazione è rappresentata con A ⟶ B

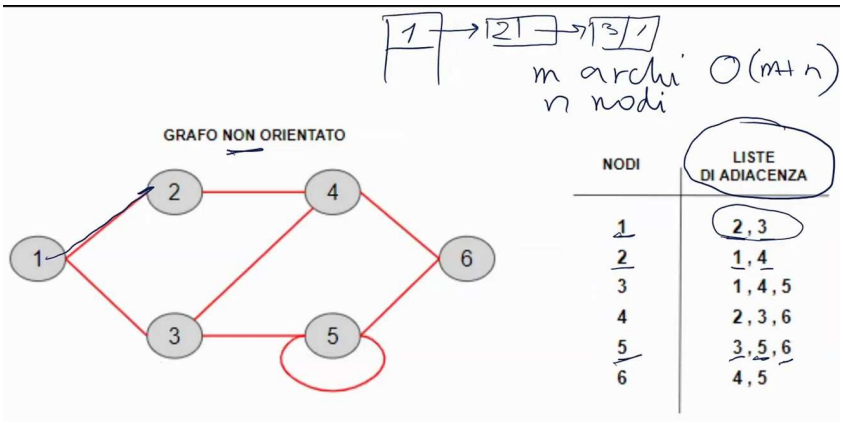
Esempi:

• in Italia si dice "parli arabo": per gli italiani l'arabo è incomprensibile;	Italian ⟶ Arabic
• in Grecia si dice "parli come un cinese"	Greek ⟶ Chinese
• gli anglofoni dicono "parli greco"	English ⟶ Greek
• in Cina si dice che l'inglese è come "chicken instestines"	Chinese ⟶ English

Nel grafo sono presenti alcuni **cicli** (**circuiti**) da alcuni nodi del grafo si può tornare ad essi percorrendo archi del grafo); un esempio: il circuito costituito dai nodi Greek, Chinese, English.

https://www.quora.com/Why-is-it-that-when-people-dont-understand-something-they-say-its-Greek-to-me

La rappresentazione di un grafo può essere fatta con **liste di adiacenza** oppure con **matrice di adiacenza.**



liste di adiacenza per grafo NON orientato:

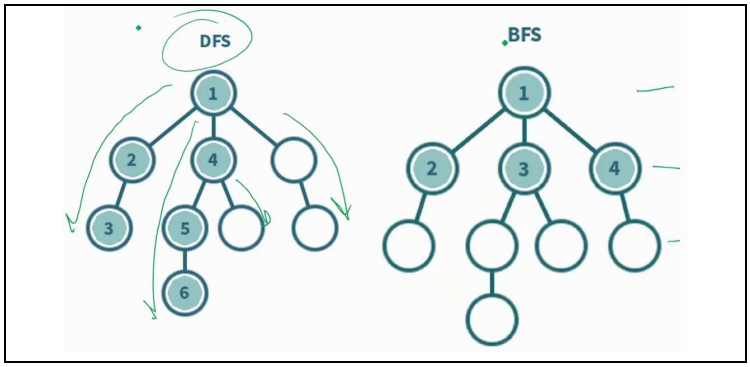
- il nodo 1 ha una lista che rappresenta il collegamento ai nodi 2 e 3
- il nodo 2 ha una lista che rappresenta il collegamento ai nodi 1 e 4
- il nodo 5 ha una lista che rappresenta il collegamento ai nodi 3 e 6 e a sé stesso

La **matrice di adiacenza** si costruisce indicando in posizione (i,j) **1** se c'è un arco che collega il nodo i a j, **0** altrimenti.

Per un Grafo NON orientato la matrice di adiacenze è simmetrica rispetto alla diagonale principale (i,i).

Per il grafo orientato:

Le matrici di adiacenza, soprattutto per i grafi orientati, sono sparse (molti zero), quindi sono meno efficienti delle liste che ottimizzano spazi e tempi di esecuzione.



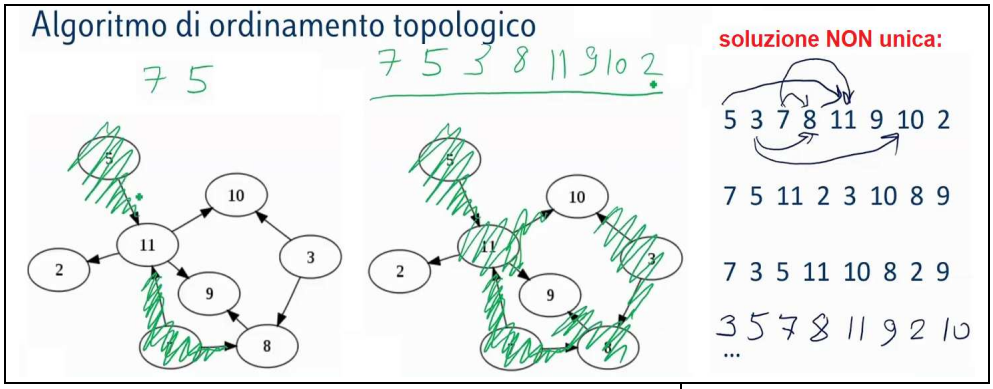
Gli algoritmi sui grafi utili per le OII sono 3:

1. Ordinamento topologico
2. Visita in profondità (DFS) e in ampiezza (BFS)
3. Cammini minimi (Dijkstra)

DFS = Depth First Search

BFS= Breadth First Search

Nella lezione 6 viene illustrato solo l' **Ordinamento topologico** (per **grafi orientati senza cicli - DAG**)



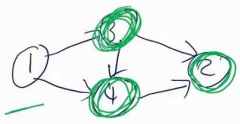
L'ordinamento topologico non è un ordinamento totale, perché la soluzione può non essere unica.

Un DAG è un Grafo orientato che non ha cicli (circuiti): da un qualsiasi vertice del grafo non si può tornare ad esso percorrendo gli archi del grafo.

I nodi di un DAG (Directed Acyclic Graph) si dicono **ordinati topologicamente** se i nodi sono elencati in modo che ogni nodo sia esposto prima di tutti i nodi collegati ai suoi archi uscenti. Questo Algoritmo ha **complessità lineare** O(n+m).

Assunzioni

- $1 < N < 100000$.
- $1 < M < 100000$.
- $1 \le Q \le N$.



Esempi di input/output

1 3 4 2

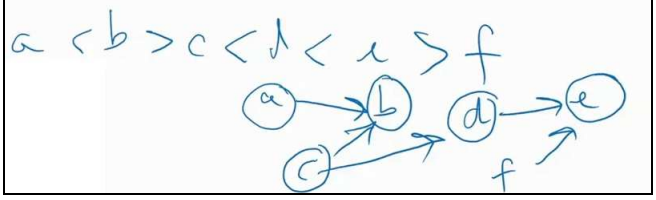
File input.txt	File output.txt
4 5 3 1 3 1 4 3 2 3 4 4 2	1

Esempio: problema **Torero Escamillo (torero)**

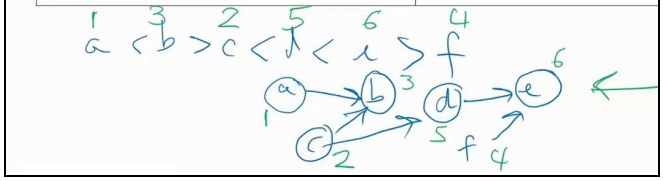
Eliminato il nodo 3 (=Q) ancora non disponibile vanno eliminati tutti i nodi (abiti) che dipendono (partono) da 3; vanno quindi eliminati anche i nodi 2 e 3.

Resta solo il nodo 1, la risposta (è indossabile solo un abito)

input.txt	output.txt
6 <><<>	2 5 1 3 6 4
5 >><	5 3 1 2 4
8 >><>><	6 5 4 7 3 2 8 1



input.txt	output.txt
6 <><<>	2 5 1 3 6 4
5 >><	5 3 1 2 4
8 >><>><	6 5 4 7 3 2 8 1



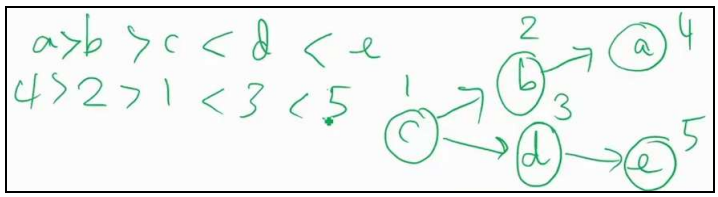
Esempio: problema **Rispetta i versi (disuguaglianze)**

Il problema può essere risolto anche con il grafo della relazione "X è minore di Y" per X e Y <= N:

- si indicano prima i segni: **< > < < >**
- poi si inseriscono le incognite tra i segni (a, b, c, d, e, f)
- si costruisce il grafo (con archi dal minore al maggiore)
- si abbinano i numeri alle lettere

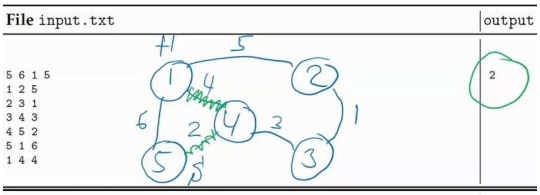
partendo dal numero più alto (**6**) che corrisponde all'unica lettera da cui non partono archi; la lettera **e** viene quindi abbinata al **6** e **1** al nodo **a** senza archi entranti.

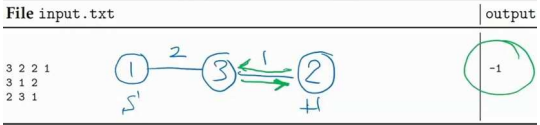
caso:2

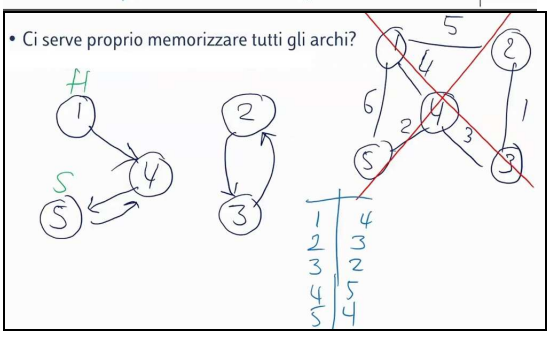


esercizio problema **Sunnydale (sunny)**

Suggerimenti:







Per ogni nodo basta **memorizzare solo l'arco a luminosità minore**, quello che Harmony sceglierebbe.

A questo punto ogni nodo ha 1 solo vicino, quindi si può utilizzare un array.

Una soluzione, con relativa descrizione, è disponibile a pag.81 della Guida Bugatti nel cap. 8 I GRAFI

soluzione parziale

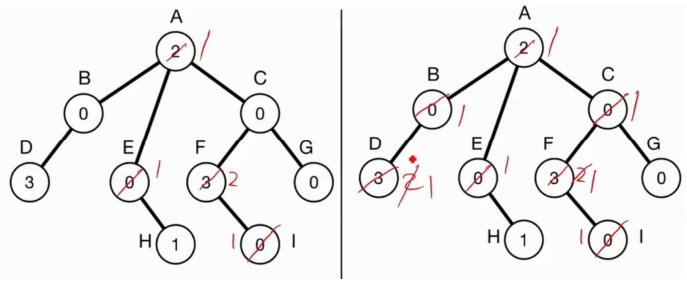
```
7   freopen("input.txt", "r", stdin);
8   freopen("output.txt", "w", stdout);
9
10  int i,j,k,t,t2,l;
11  bool controllo=0;
12  unsigned n,m,h,s;
13  cin >> n >> m >> h >> s;
14  unsigned c=0;
15  bool fg[50000]={0};
16  int g[50000];
17  unsigned min[50000]={50001};
18  for(i=0;i<m;i++){
19      cin >> t >> t2 >> l;
20      if(l<min[t]){
21          g[t]=t2;
22          min[t]=l;
23      }
24      if(l<min[t2]){
25          g[t2]=t;
26          min[t2]=l;
27      }
28  }
29  while(!fg[h]){
30      fg[h]=1;
31      if(h==s){
32          controllo=1;
33          break;
34      }
35      h=g[h];
36      c++;
37  }
38  if(controllo==1)
39      cout << c;
40  else
```

esercizio problema **Monete a posto (monete)**

Ad esempio, nella situazione mostrata qui sopra, William impiega:

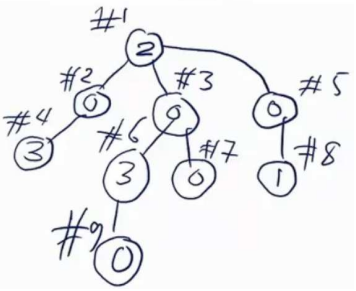
- Una operazione per spostare una moneta da A a E.
- Una operazione per spostare una moneta da F a I.
- Una operazione per spostare una moneta da F a C.
- Una operazione per spostare una moneta da D a B.
- Quattro operazioni per spostare una moneta da D a G (passando per B, A e infine C).

} operazione



Costruzione del grafo dai dati di input: #1 è la radice

input.txt	output.txt
9 2 0 0 3 0 3 0 1 0 1 4 2 1 3 3 5 6 a1 a2 a3 a4 a5 a6 a7 a8 a9 <<monete b2 b3 b4 b5 b6 b7 b8 b9 <<bj agganciato a j	8
10 0 0 1 2 1 1 2 0 3 0 1 2 2 2 4 3 5 1 8	11



b_2=**1** il contenitore **#2** è agganciato a **#1**

b_3=**1** il contenitore **#3** è agganciato a **#1**

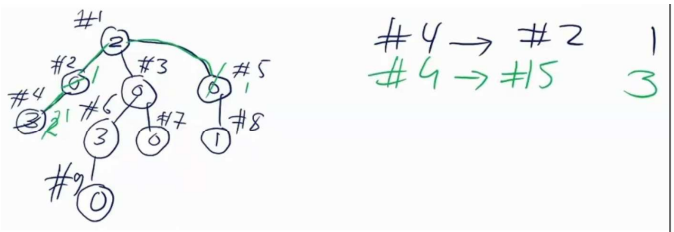
b_4=**2** il contenitore **#4** è agganciato a **#2**

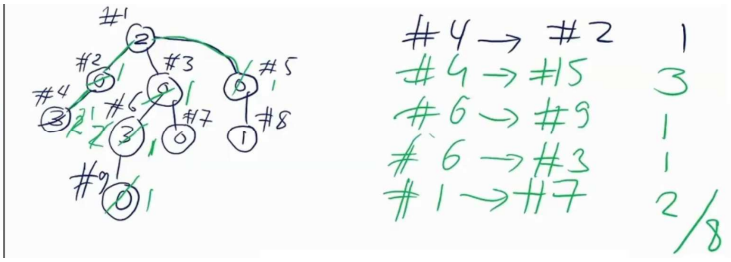
a_1=2 il contenitore **#1** contiene 2 monete

a_2=0 il contenitore **#2** contiene 0 monete

8 operazioni da fare per riordinare le monete:

- William sposta 1 moneta dal contenitore #4 al contenitore #2 = 1 operazione
- William sposta 1 moneta dal contenitore #4 al contenitore #5 = 3 operazioni, etc etc





Suggerimento: partendo dalle foglie, lasciare o portare ad 1 ogni nodo aggiornando il nodo al quale è appeso sottraendogli una moneta nel caso il nodo foglia sia vuoto (G diventerebbe 1 e C **-1**).

ATTENZIONE: possono determinarsi valori intermedi negativi: