

GRAFI E PROGRAMMAZIONE DINAMICA

+

o

.

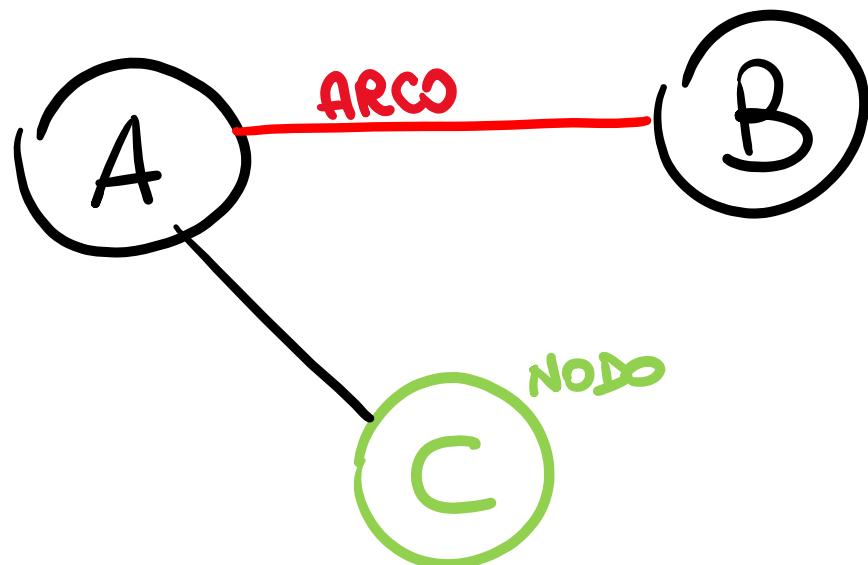


Link utili

- [Esercizi dal sito del Professor Montresor](#)
- [Dispensa del corso di Algoritmi e Strutture Dati](#)
- Se volete farmi domande contattatemi su [telegram](#) oppure via [mail](#)

Grafi: che cosa sono?

- Struttura matematica G rappresentata come una coppia (V, E) di nodi (o vertici) e archi



INSIEME DEI VERTICI (VERTEX)

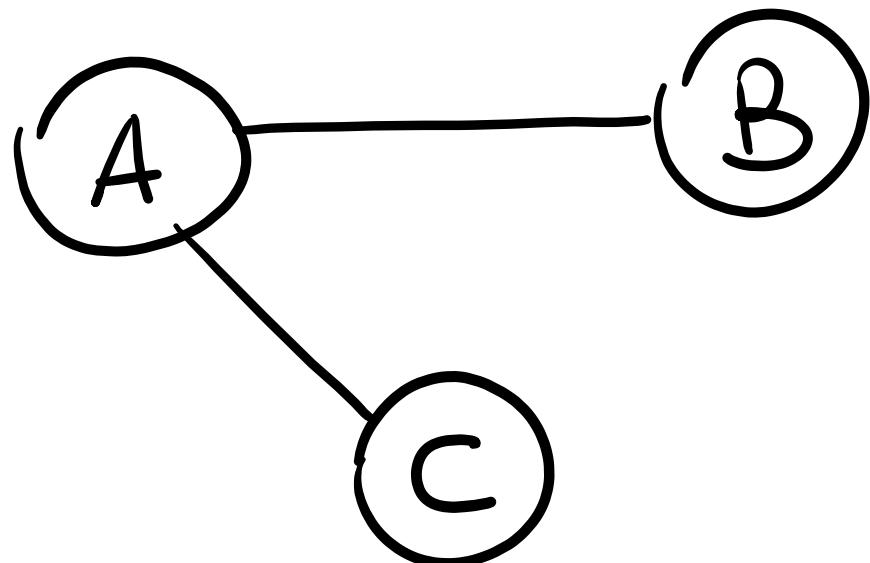
$$V = \{A, B, C\}$$

INSIEME DEGLI ARCHI

$$E = \{(A, B), (B, A), (A, C), (C, A)\}$$

Grafi: a che cosa servono?

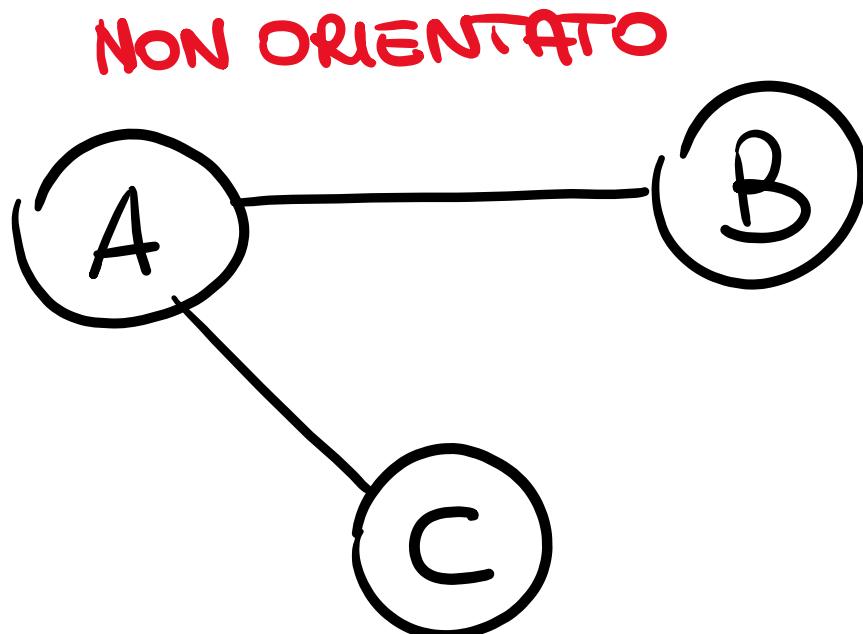
- Notoriamente un grafo G viene utilizzato per rappresentare l'esistenza di un certo tipo di relazione (amicizia, dipendenza, ...) fra due nodi mediante l'utilizzo di un arco.



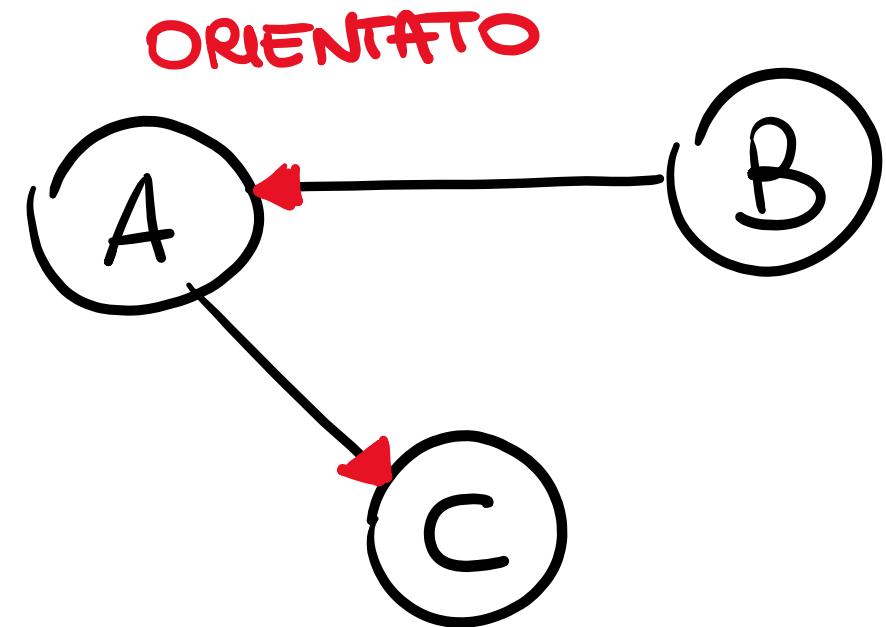
es: A è amico di B
(e viceversa)

Grafi: tipologie

- Esistono due tipologie di grafo e si distinguono in base al tipo di archi che connettono i nodi



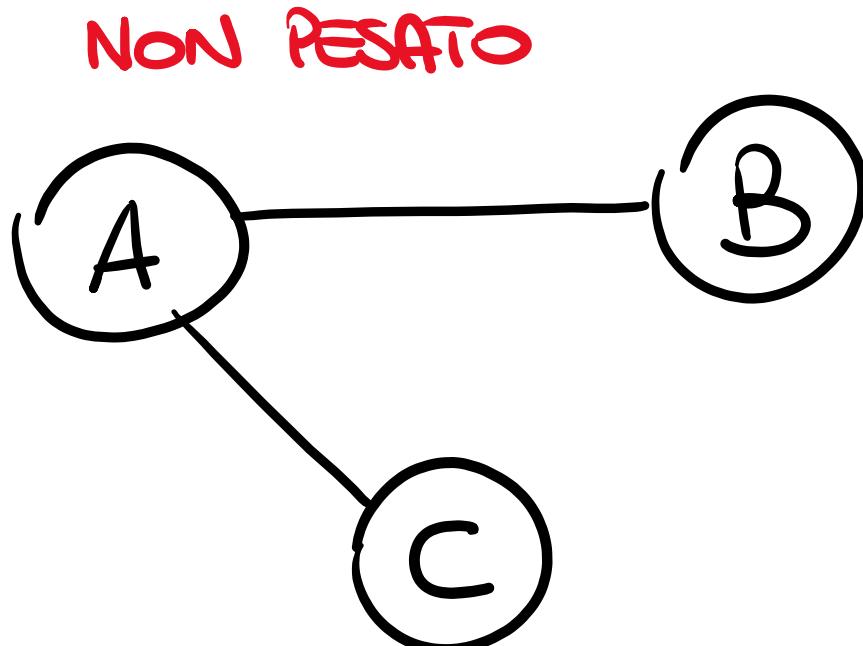
es: A e B sono amici



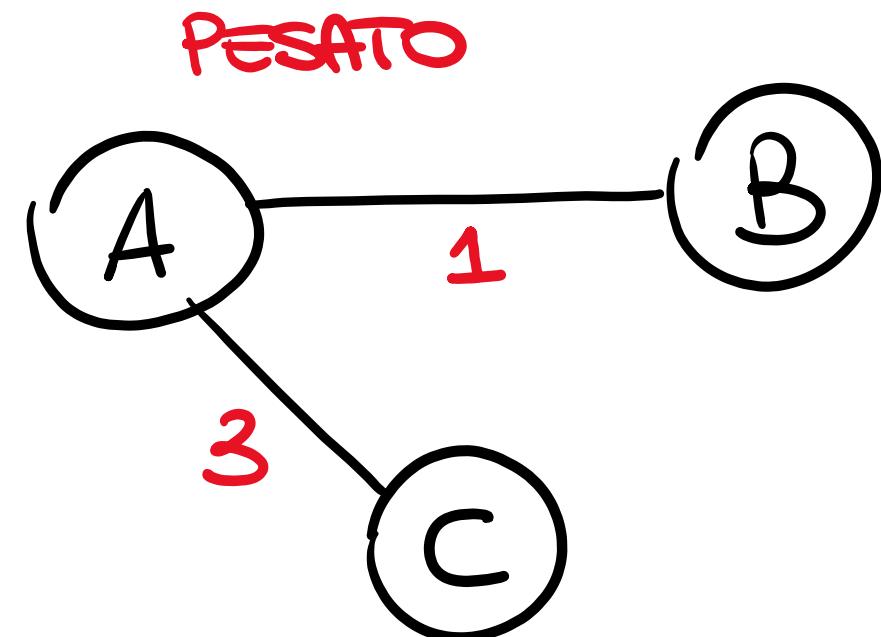
es: B segue A su IG

Grafi: tipologie

- Esistono due tipologie di grafo e si distinguono in base al tipo di archi che connettono i nodi



es: non vi sono costi



es: devo "pagare" 4 per andare da B a C

Grafi: rappresentazione

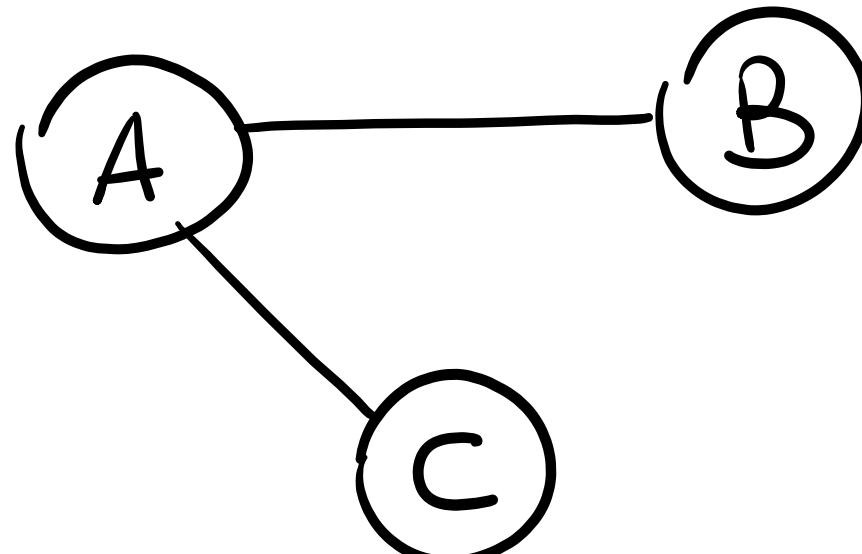
- Per poter rappresentare un grafo a livello informatico si utilizzano due metodi

MATRICE DI ADIACENZA

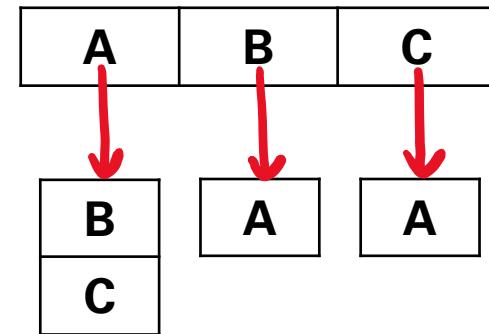
/	A	B	C
A	0	1	1
B	1	0	0
C	1	0	0

$O(n^2)$

$n = \text{n}^{\circ} \text{ nodi}$
 $m = \text{n}^{\circ} \text{ archi}$



LISTE DI ADIACENZA



$O(nm)$

Grafi: rappresentazione

- Per poter rappresentare un grafo a livello informatico si utilizzano due metodi

```
int grafo[N][N];
//c'è un arco da 0 a 0
grafo[0][0] = 1;

//non c'è un arco da 0 a 1
grafo[0][1] = 0;
```

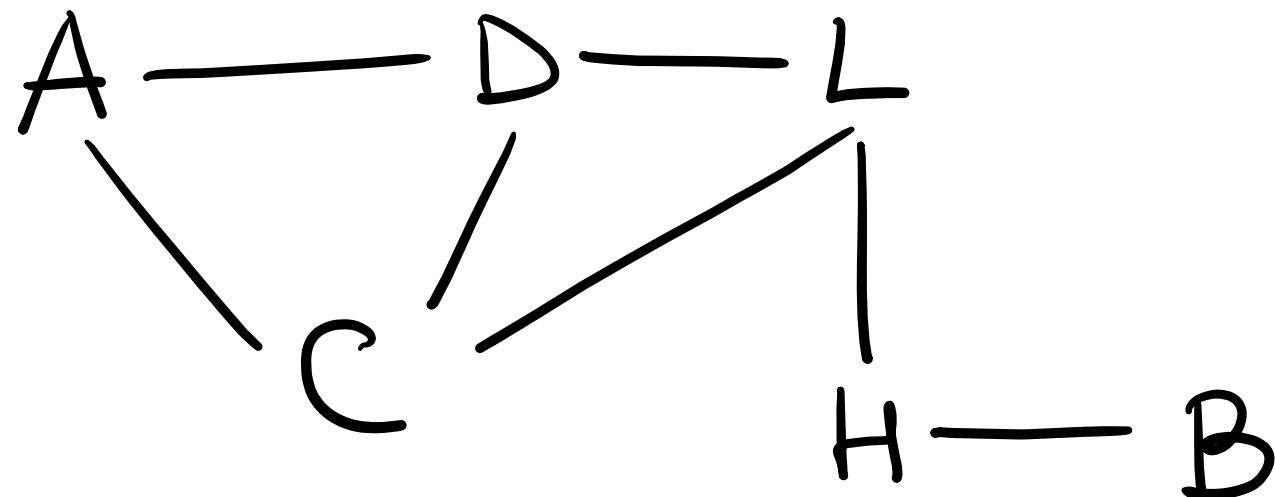
```
struct nodo{
    vector<int> vic;
    int erdos = INT_MAX;
    int weight = 0;
};

vector<nodo> grafo(N);
/* identico a
nodo grafo[N];
*/

// per accedere ai campi su usa il .
grafo[0].erdos = 1;
```

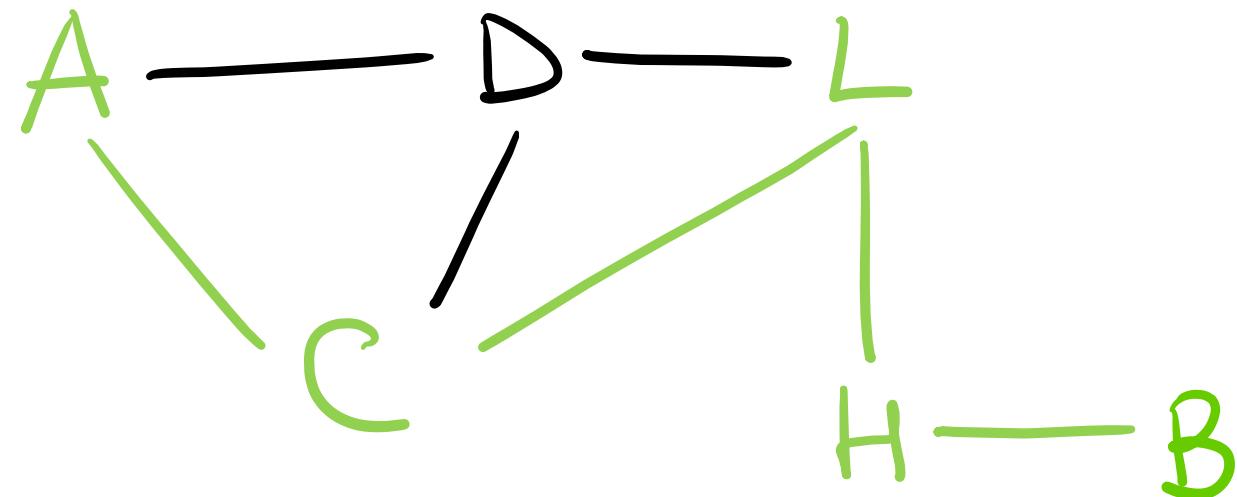
Esercizio 1: Sono connessi?

- Dato in input un grafo $G(V, E)$ non orientato e due nodi A e B, progettare un algoritmo per verificare se A e B sono connessi (cioè se è possibile, partendo da A, arrivare a B seguendo un generico cammino)



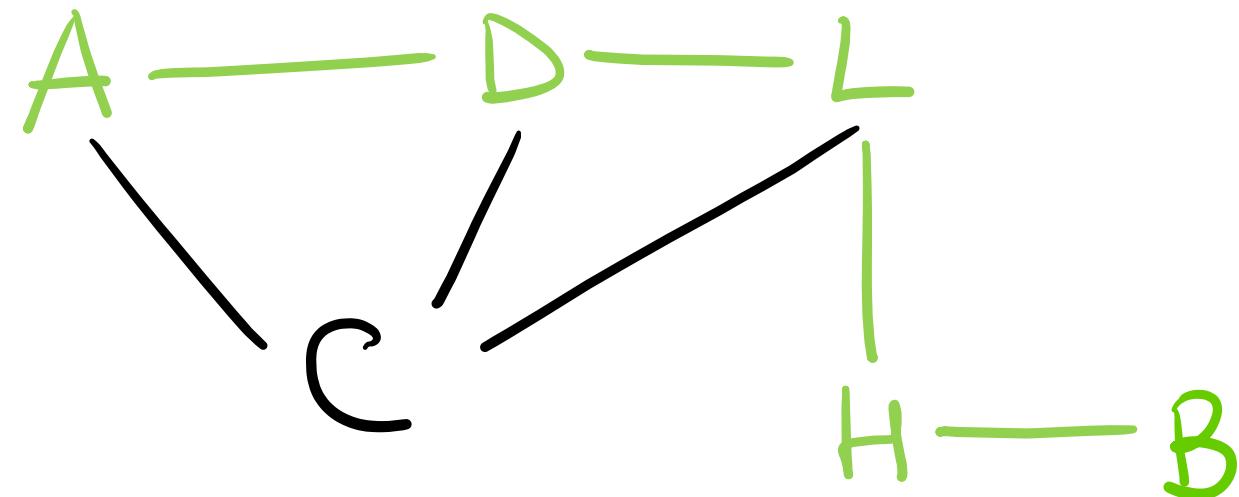
Esercizio 1: Sono connessi?

- Dato in input un grafo $G(V, E)$ non orientato e due nodi A e B , progettare un algoritmo per verificare se A e B sono connessi (cioè se è possibile, partendo da A , arrivare a B seguendo un generico cammino)

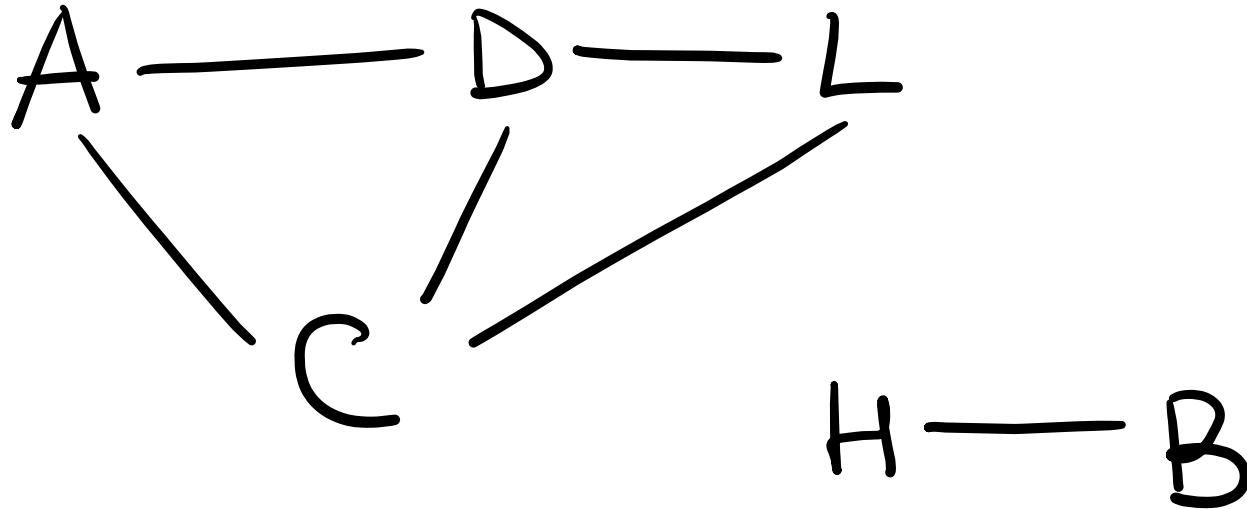


Esercizio 1: Sono connessi?

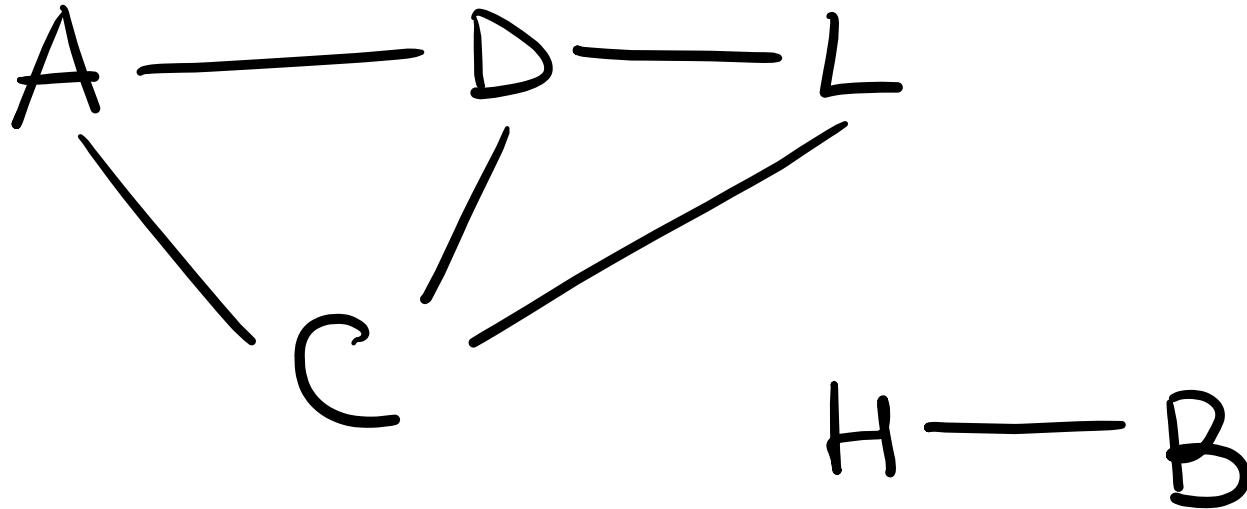
- Dato in input un grafo $G(V, E)$ non orientato e due nodi A e B , progettare un algoritmo per verificare se A e B sono connessi (cioè se è possibile, partendo da A , arrivare a B seguendo un generico cammino)



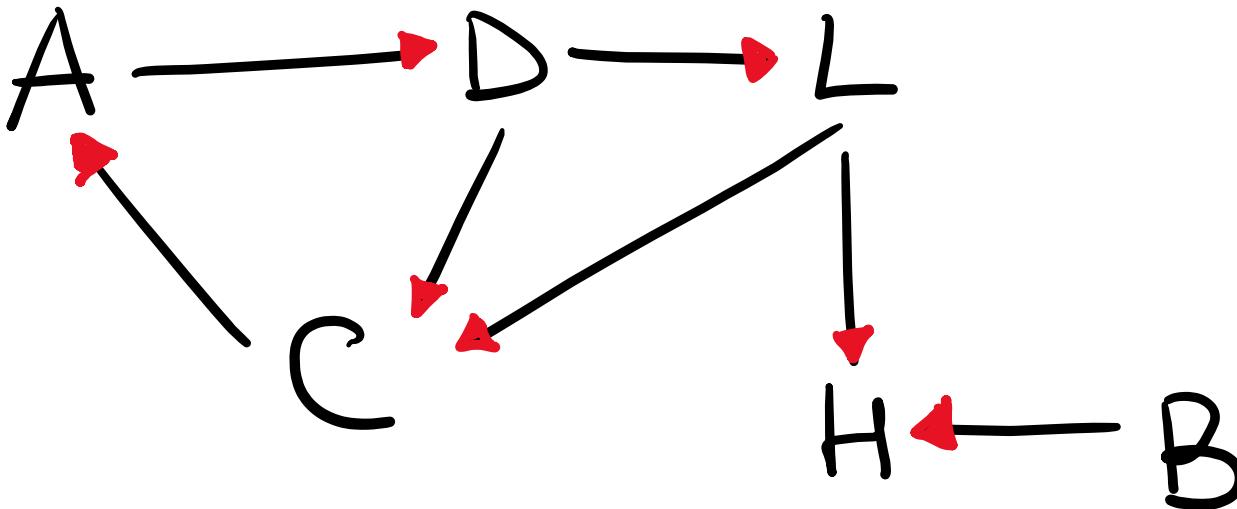
Esercizio 1: Sono connessi?



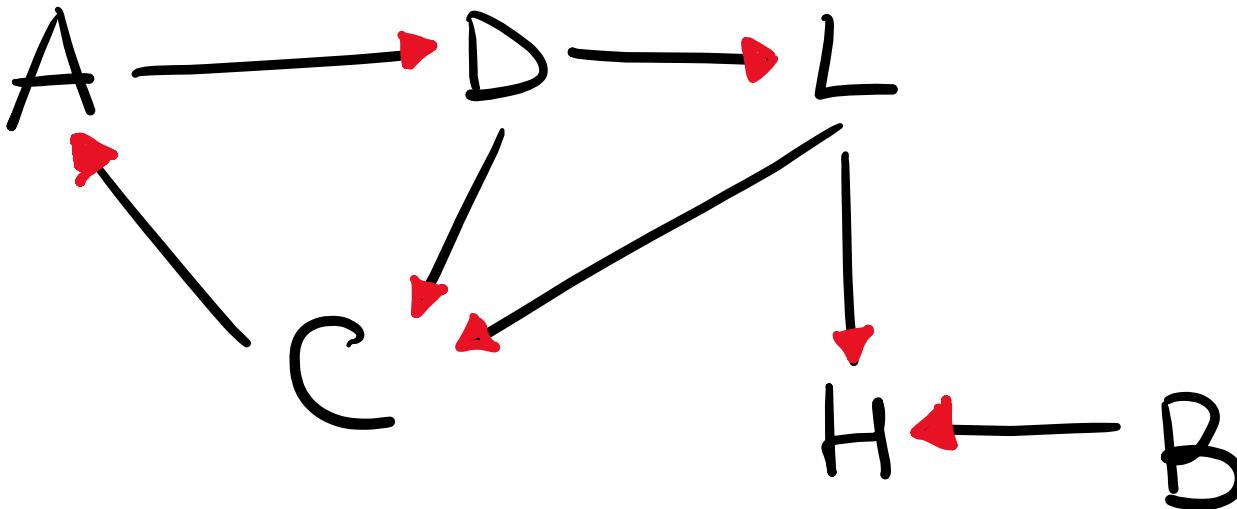
Esercizio 1: Sono connessi? **NO**



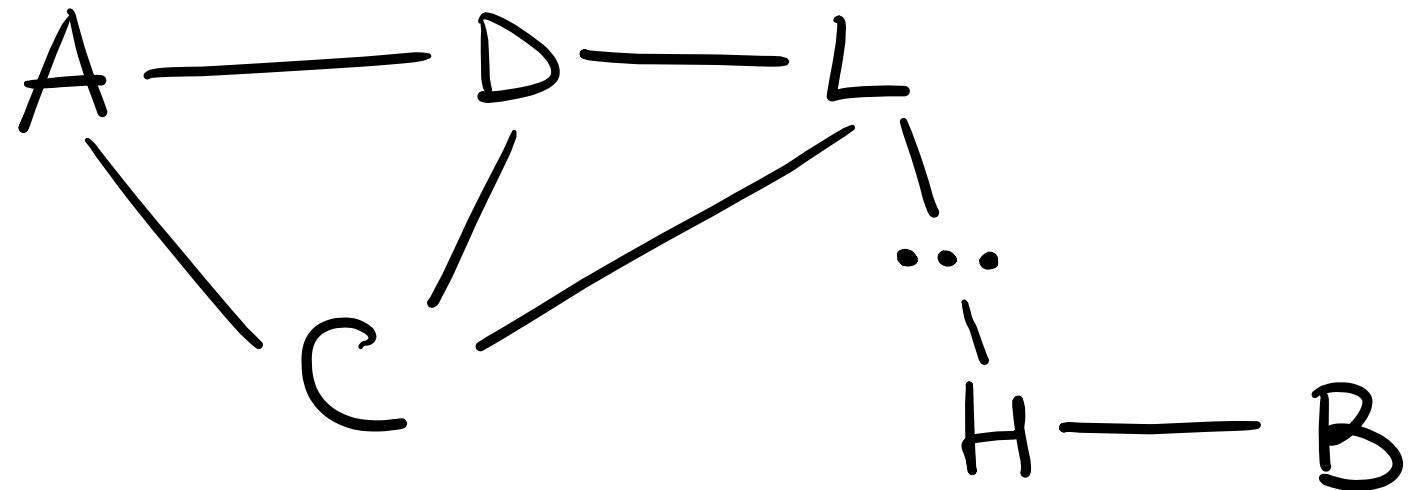
Esercizio 1: Sono connessi?



Esercizio 1: Sono connessi? NO

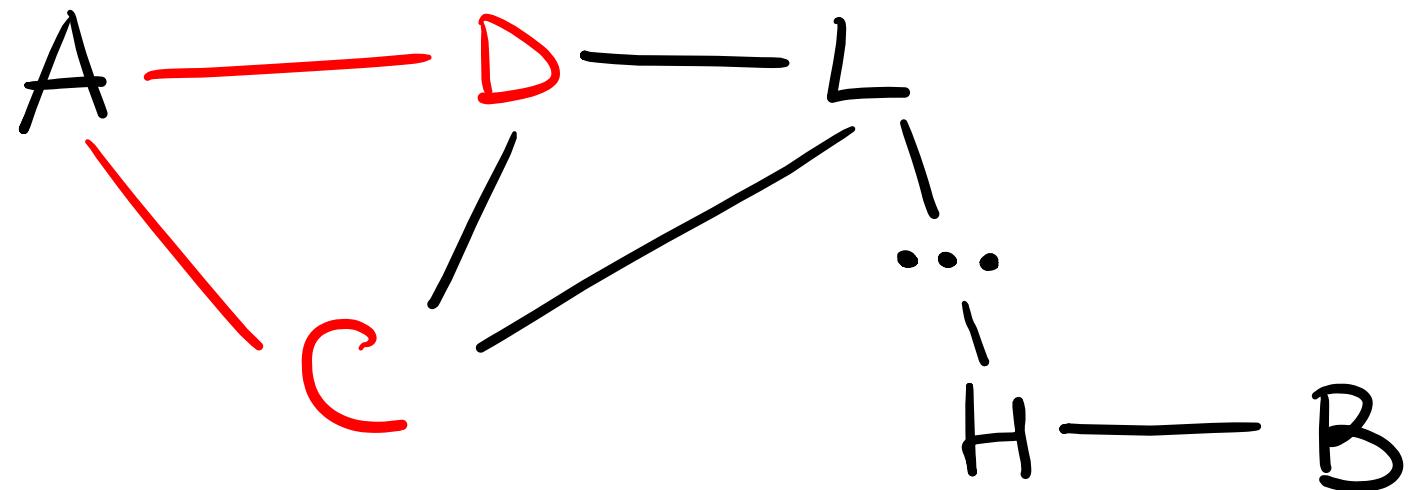


Esercizio 1: Idee?



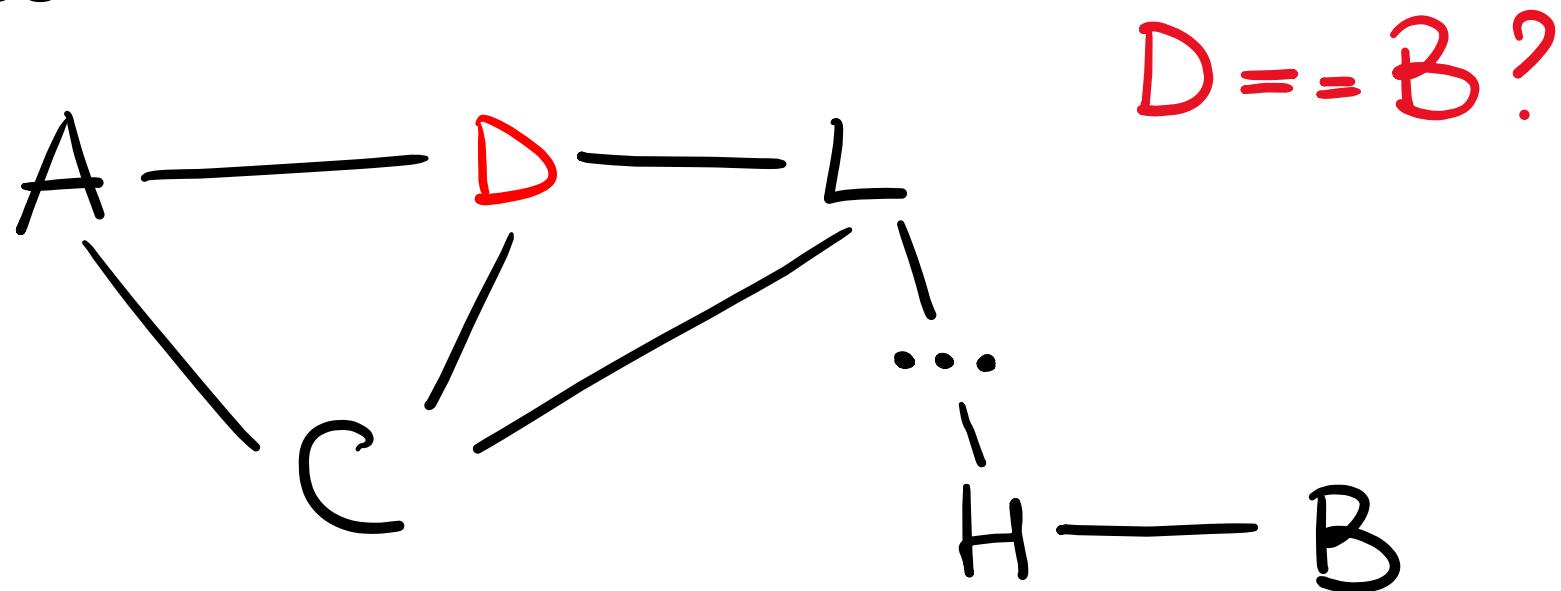
Esercizio 1: Soluzione

1. Partiamo dal nodo A, prendiamo uno dei suoi nodi adiacenti (o vicini), cioè che sono raggiungibili da A attraversando un solo arco



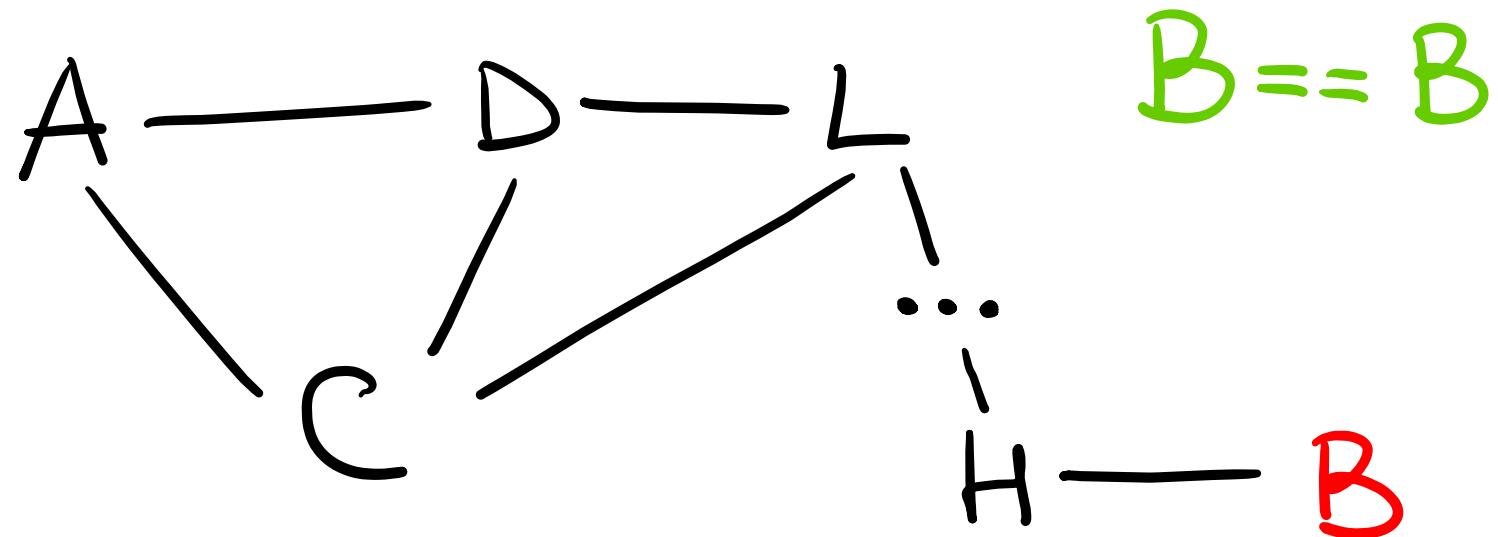
Esercizio 1: Soluzione

2. Verifichiamo se il nodo raggiunto è B, altrimenti ripetiamo l'operazione del punto precedente considerando come A il nodo raggiunto



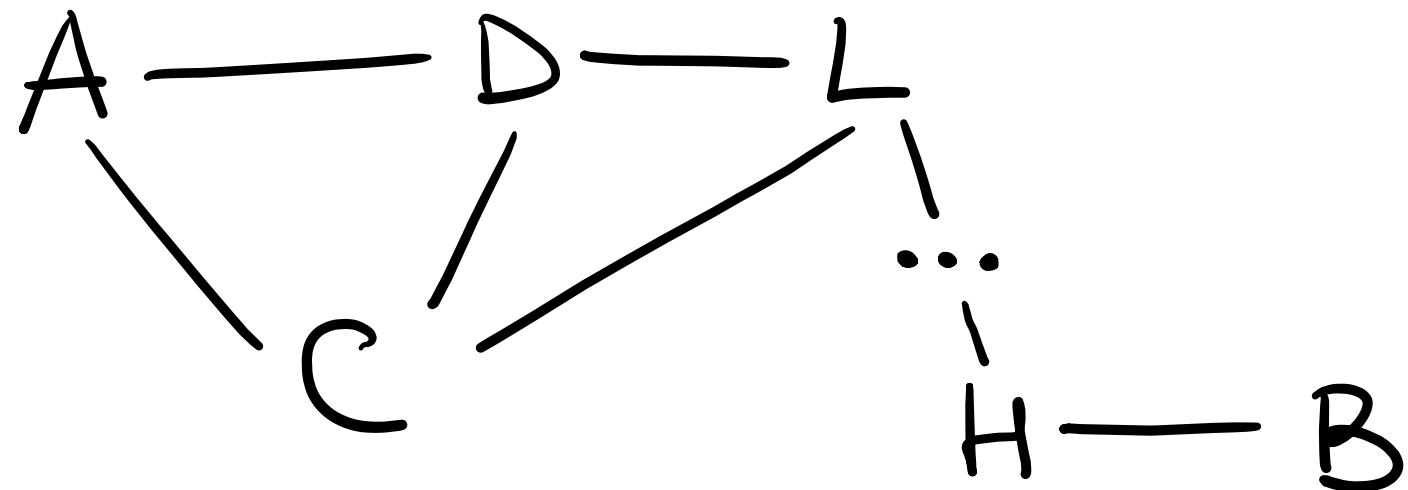
Esercizio 1: Soluzione

3. Se, durante la visita del grafo, riusciamo in qualche modo a raggiungere B, allora vuol dire che esiste un cammino che da A porta a B.



Esercizio 1: Attenzione

1. Come evitiamo di continuare a rimbalzare tra due nodi?
2. Come dichiariamo il risultato finale?



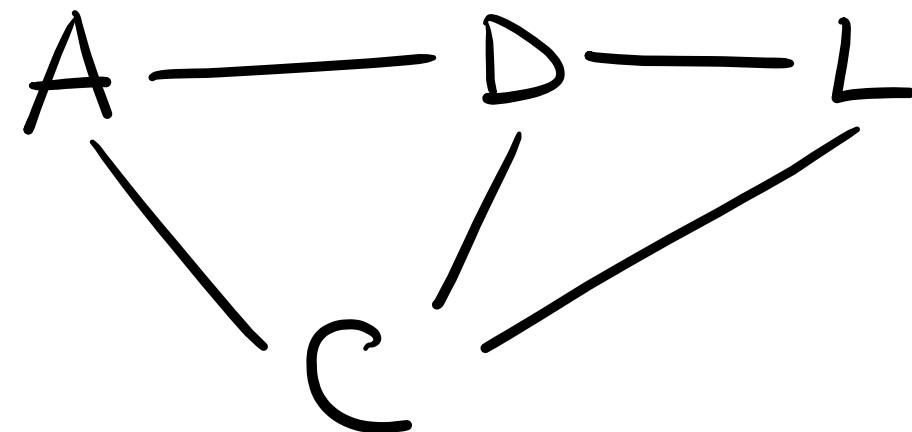
```
bool connected(vector<nodo> graph, int A, int B){  
    return connectedRec(graph, A, B);  
}  
  
bool connectedRec(vector<nodo> graph, int curr, int dest){  
    graph[curr].visited = true;  
  
    if(curr == dest){  
        return true;  
    }  
  
    for(int i = 0; i < graph[curr].vic.size(); i++){  
        int next = graph[curr].vic[i];  
        if(!graph[next].visited){  
            if(connectedRec(graph, next, dest)){  
                return true;  
            }  
        }  
    }  
    return false;  
}
```

```
struct nodo{  
    vector<int> vic;  
    bool visited = false;  
};
```



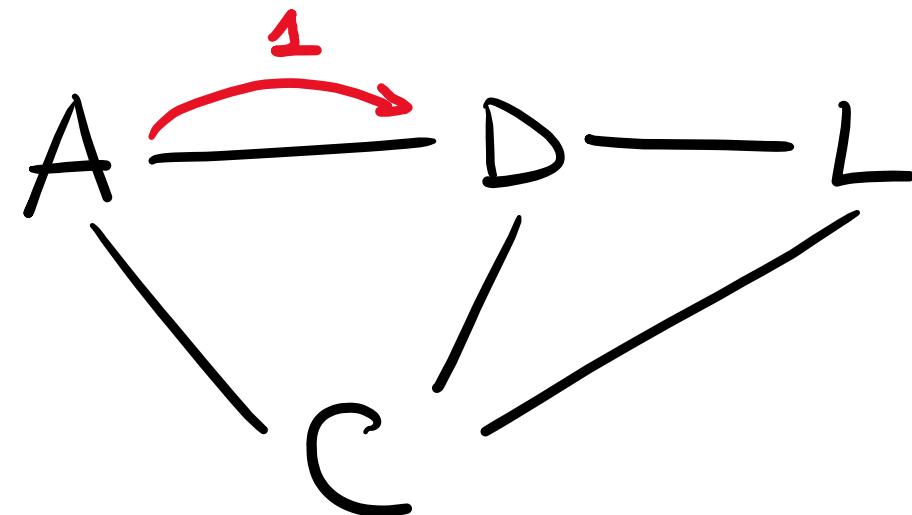
Esercizio 2: distanza massima

- Dato in input un grafo $G(V, E)$ non orientato e un nodo A, progettare un algoritmo che restituisca la **distanza** (espressa come **numero di archi traversati**) del **cammino minimo** tra A e il nodo a maggior distanza da esso.



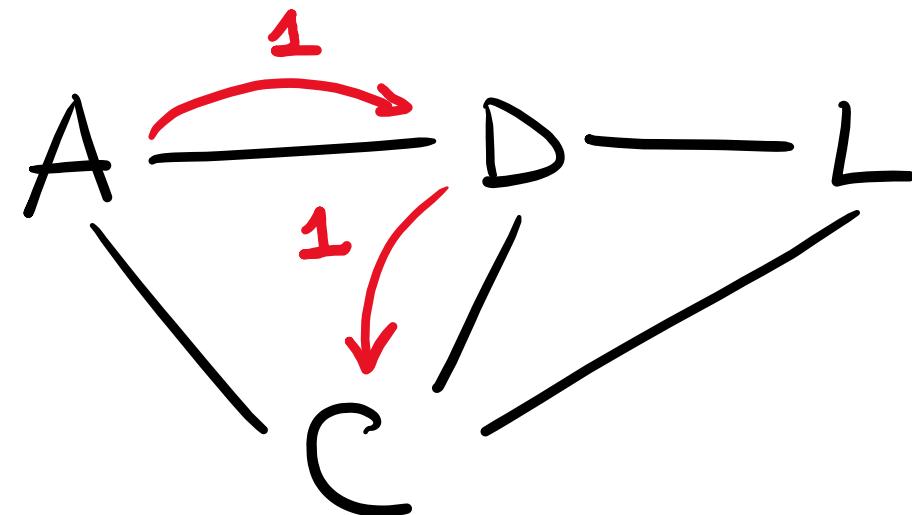
Esercizio 2: distanza massima

- Es: la distanza tra A e D è 1



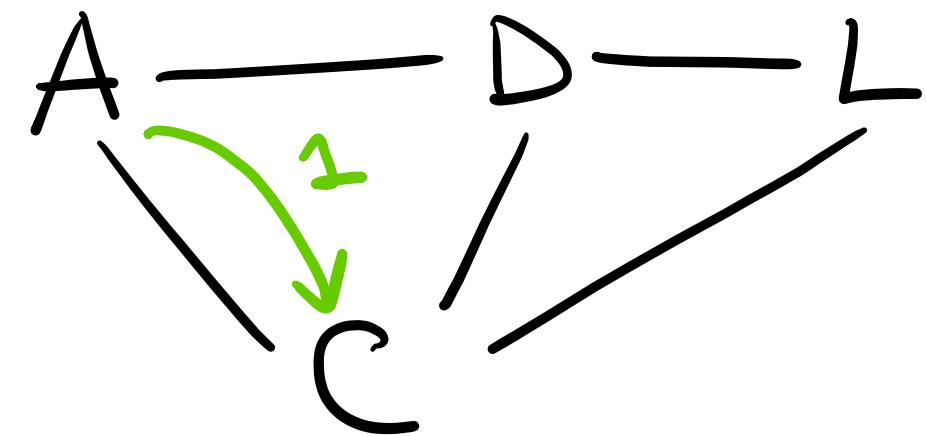
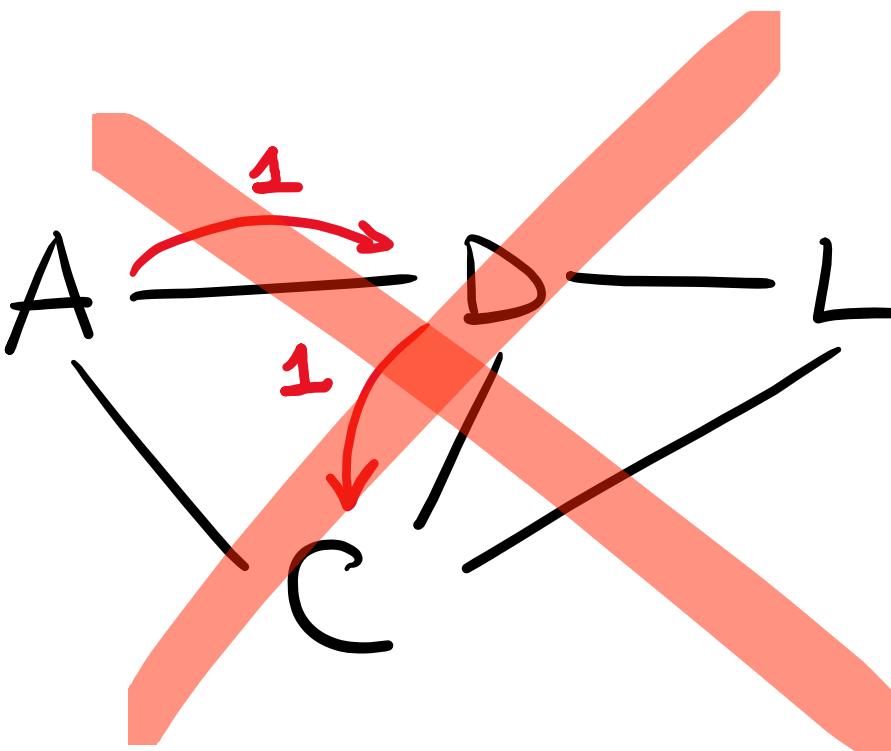
Esercizio 2: distanza massima

- Es: la distanza tra A e C è 2



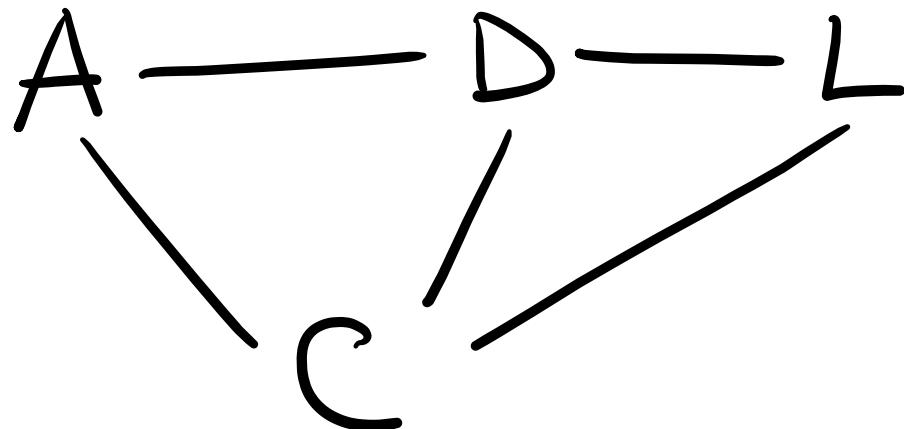
Esercizio 2: distanza massima

- Es: la distanza tra A e C è 1



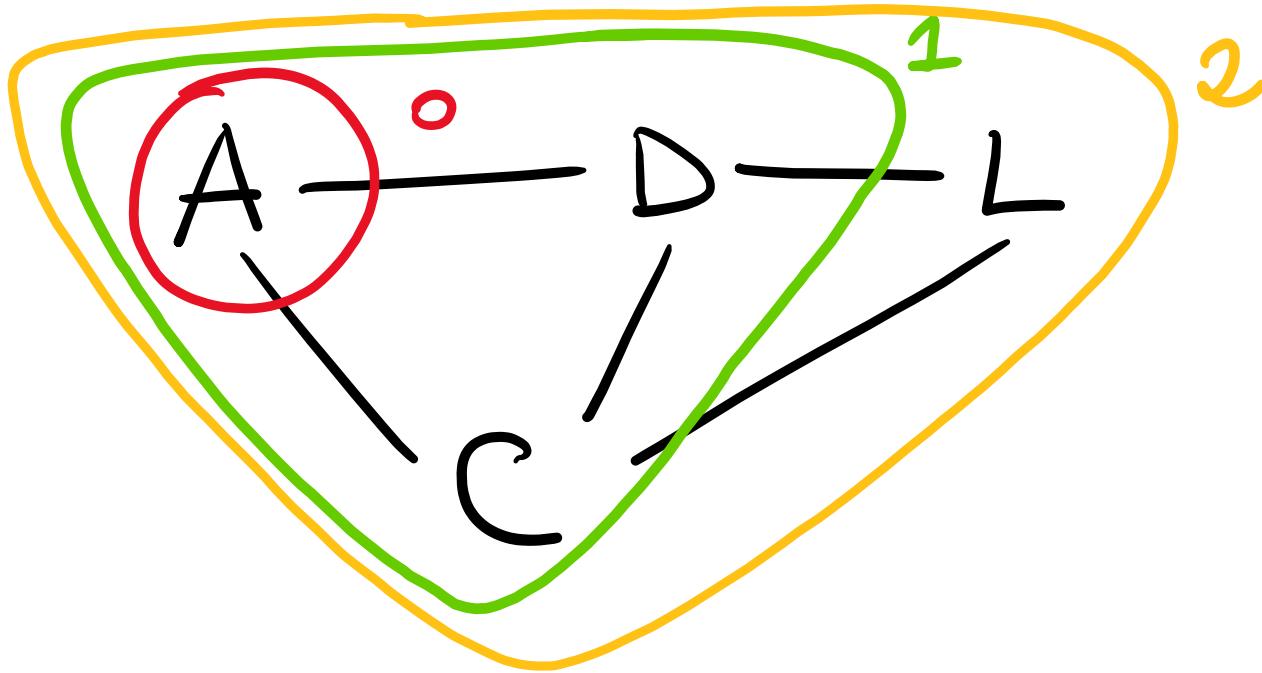
Esercizio 2: Idee?

- Dato in input un grafo $G(V, E)$ non orientato e un nodo A, progettare un algoritmo che restituisca la **distanza** (espressa come **numero di archi traversati**) del **cammino minimo** tra A e il nodo a maggior distanza da esso.



Esercizio 2: Idee?

- Dato in input un grafo $G(V, E)$ non orientato e un nodo A , progettare un algoritmo che restituisca la **distanza** (espressa come **numero di archi attraversati**) del **cammino minimo** tra A e il nodo a maggior distanza da esso.



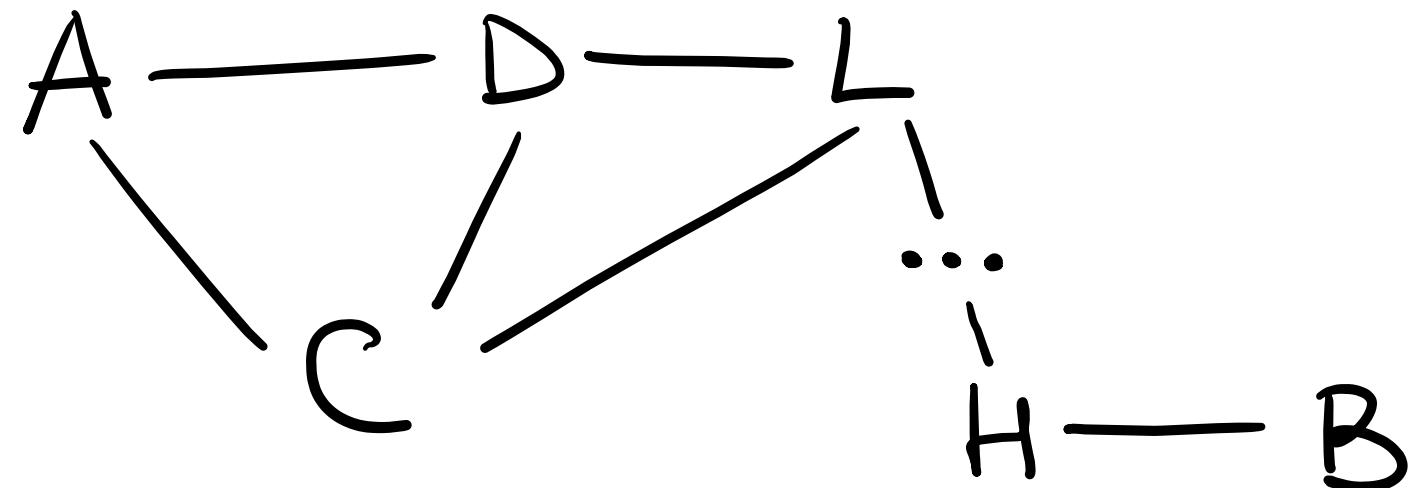
```
int erdos(vector<nodo> graph, int node){  
    queue<int> q;  
    q.push(node);  
    int maxDistance = 0;  
    graph[node].erdos = 0;  
    while(!q.empty()) {  
        int extracted = q.front();  
        q.pop();  
        for(int i=0; i<graph[extracted].vic.size(); i++) {  
            int adj = graph[extracted].vic[i];  
            if(graph[adj].erdos == INT_MAX) {  
                q.push(adj);  
                graph[adj].erdos = graph[extracted].erdos+1;  
                maxDistance = max(maxDistance, graph[adj].erdos);  
            }  
        }  
    }  
    return maxDistance;  
}
```

```
struct nodo{  
    vector<int> vic;  
    int erdos = INT_MAX;  
};
```



Esercizio 2.5: cammino minimo

- Dato in input un grafo $G(V, E)$ non orientato e due nodi A e B, progettare un algoritmo che restituisca un **possibile cammino minimo** tra A e B.



Territoriali: Interruttori

- [...] impianto composto da N lampadine a risparmio energetico di ultima generazione. Queste lampadine sono governate da **interruttori di due tipi**:
- tipo 1: se la lampadina z_i è spenta allora la accende, mentre se è accesa allora la spegne;
- tipo 2: cambia lo stato contemporaneamente a due lampadine x_i e y_i .
- Quindi, per spegnere o accendere una singola lampadina (senza influenzare le altre), potrebbe essere necessario agire su diversi interruttori in sequenza.
- **Problema:** trovare la lampadina che necessita del maggior numero di interruttori per essere spenta.

Territoriali: Interruttori

- Input: La prima riga del file di input contiene un intero T , il numero di casi di test. Seguono T casi di test, numerati da 1 a T . Ogni caso di test è preceduto da una riga vuota.
- Ciascun caso di test è composto da $A+B+1$ righe. La prima riga contiene i tre interi N , A , B separati da uno spazio: rispettivamente
 1. il numero di lampadine (N)
 2. il numero di interruttori di tipo 1 (A)
 3. il numero di interruttori di tipo 2 (B)
- Ciascuna delle seguenti A righe è composta da un unico intero che rappresenta un interruttore di tipo 1. Le successive B righe sono composte da 2 interi ciascuna che rappresentano un interruttore di tipo 2.

Territoriali: Interruttori

1

T

4 1 4

NAB

1

z_i

0 1

$x_i y_i$

0 2

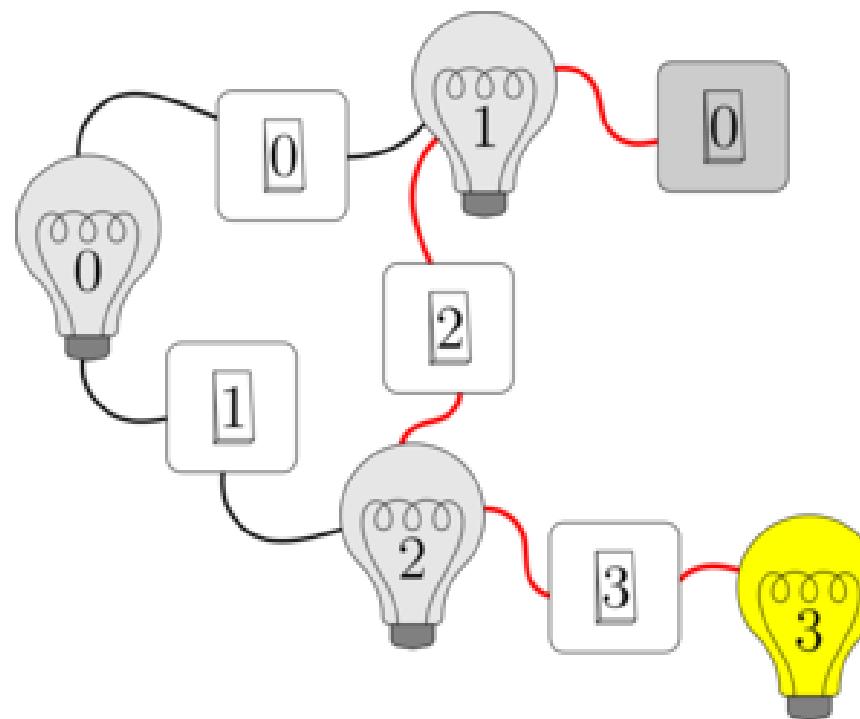
$x_i y_i$

1 2

$x_i y_i$

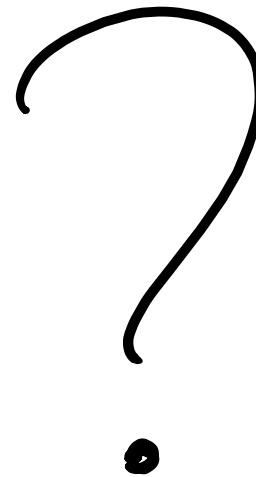
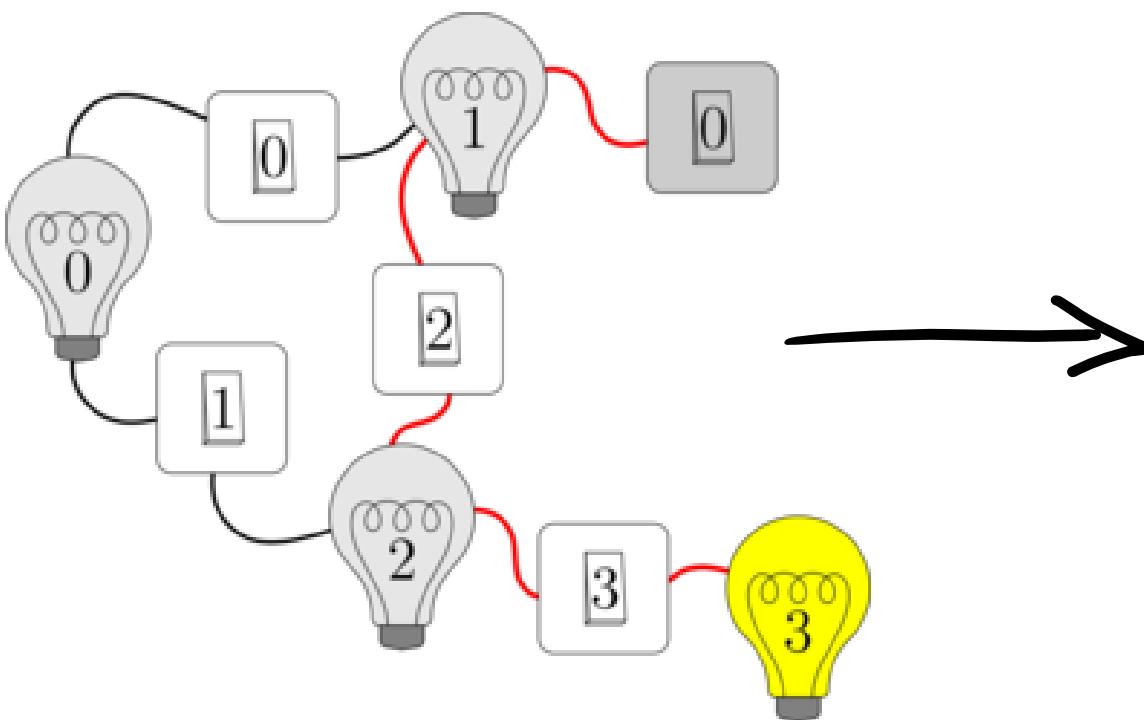
3 2

$x_i y_i$



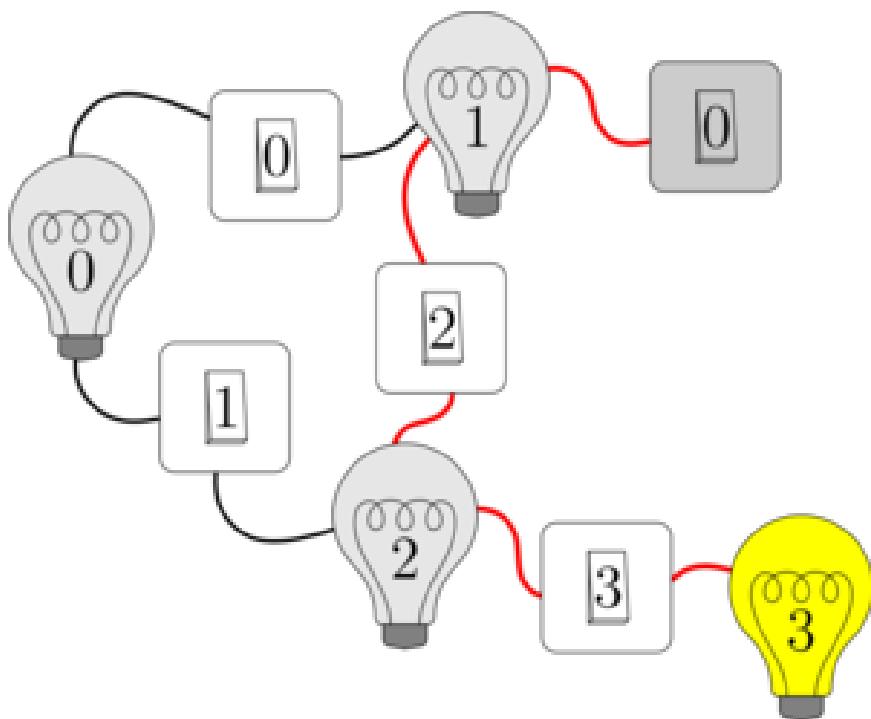
Territoriali: Interruttori

1
4 1 4
1
0 1
0 2
1 2
3 2



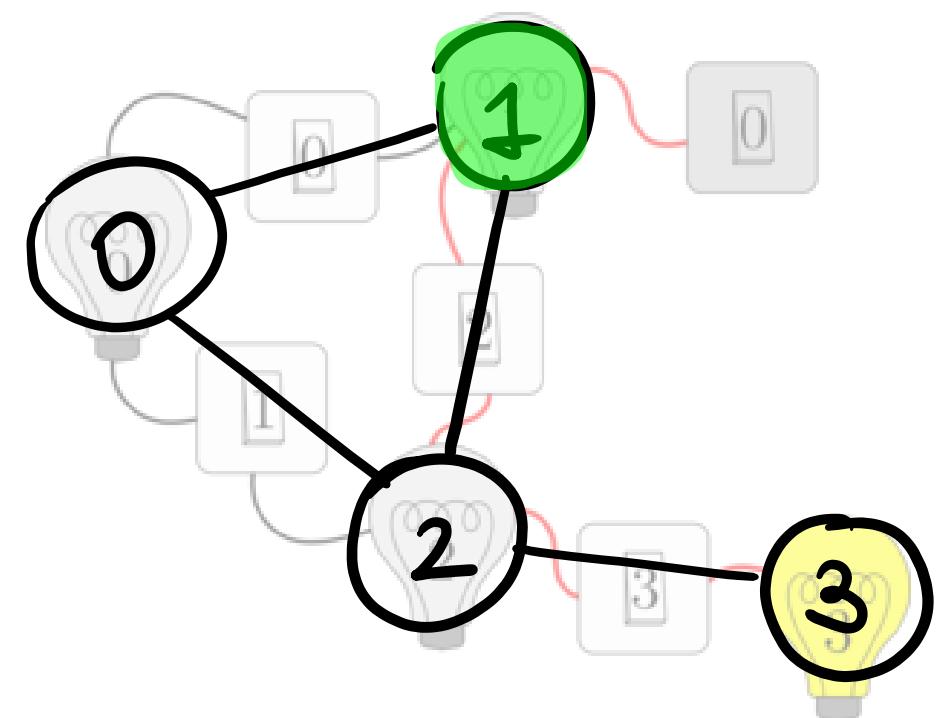
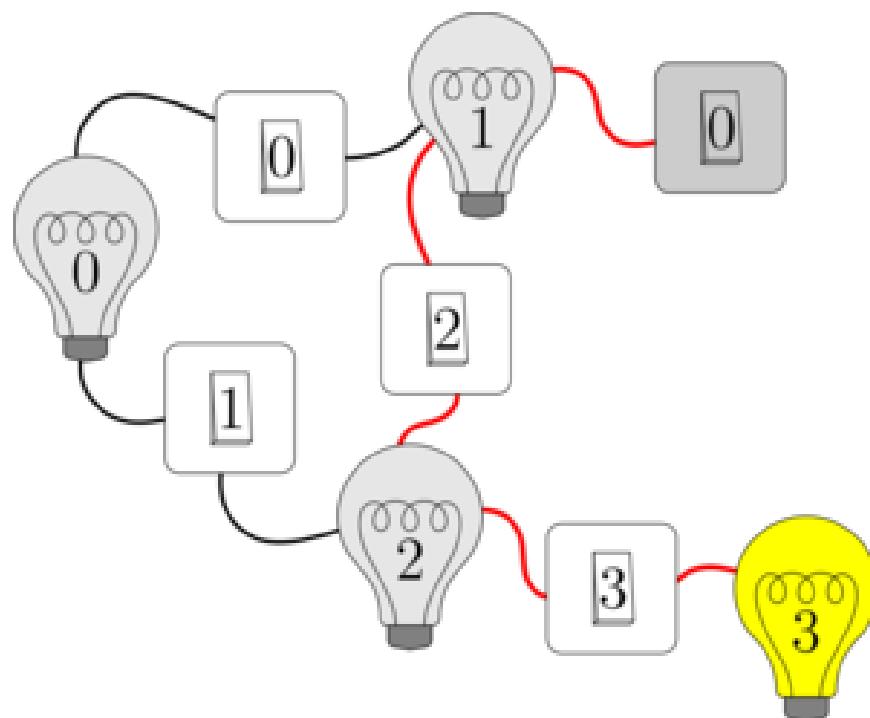
Territoriali: Interruttori

1
4 1 4
1
0 1
0 2
1 2
3 2



Territoriali: Interruttori

1
4 1 4
1
0 1
0 2
1 2
3 2



```
ifstream in("interruttori_input_1.txt");
ofstream out("output.txt");
int T;
in >> T;
for(int i = 1; i <= T; i++){
    int N, A, B;
    in >> N >> A >> B;
    vector<nodo> grafo(N);

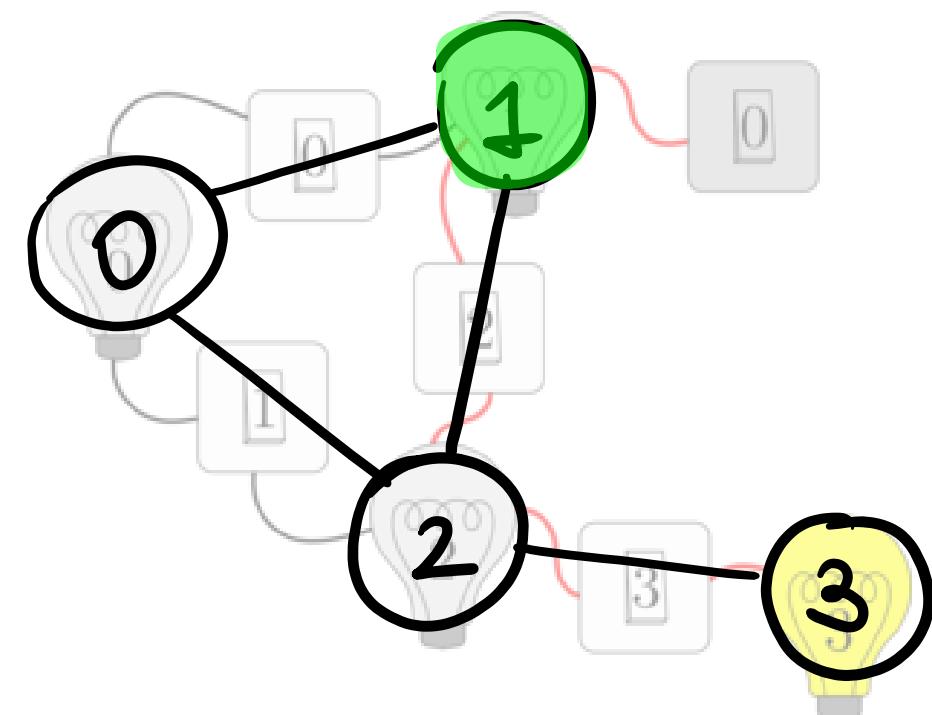
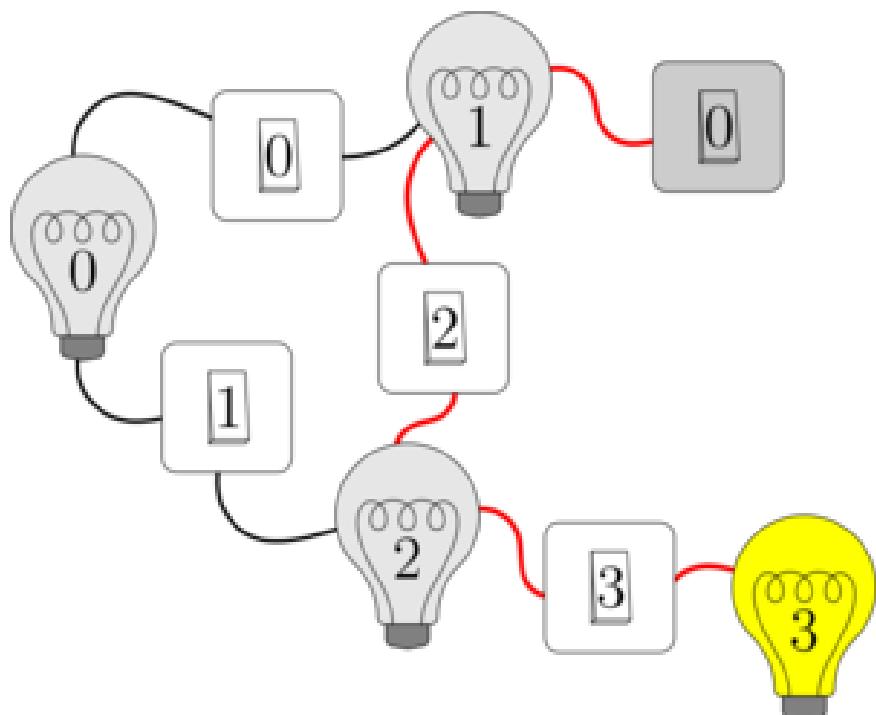
    for(int j = 0; j < A; j++){
        int zero;
        in >> zero;
        grafo[zero].type0 = true;
    }

    for(int j = 0; j < B; j++){
        int from, to;
        in >> from >> to;
        grafo[from].vic.push_back(to);
        grafo[to].vic.push_back(from); //grafo non orientato
    }
}
```

```
struct nodo{
    vector<int> vic;
    bool type0 = false;
};
```

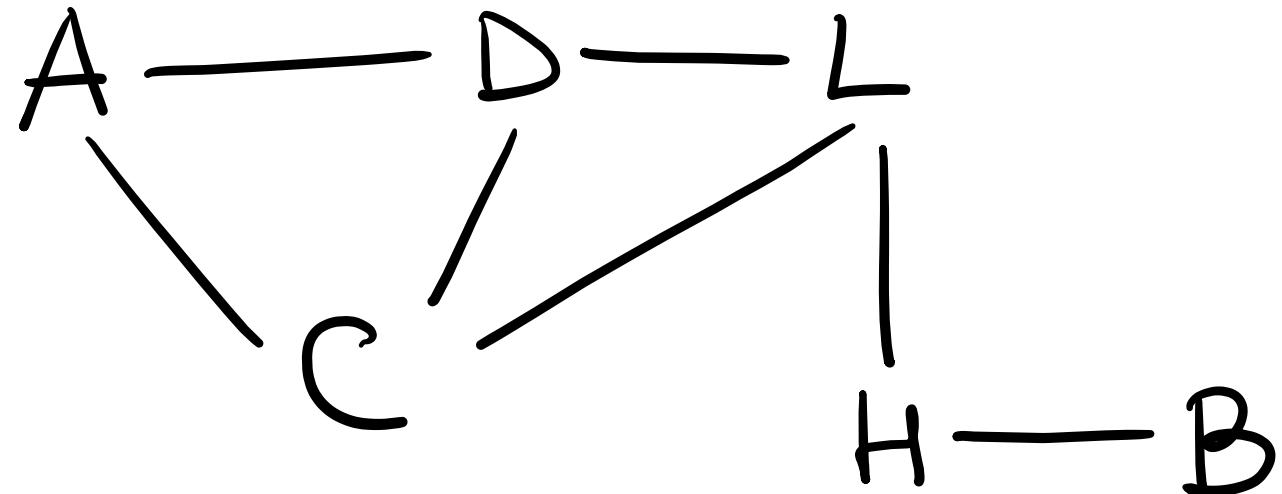
Territoriali: Interruttori

- Risoluzione?



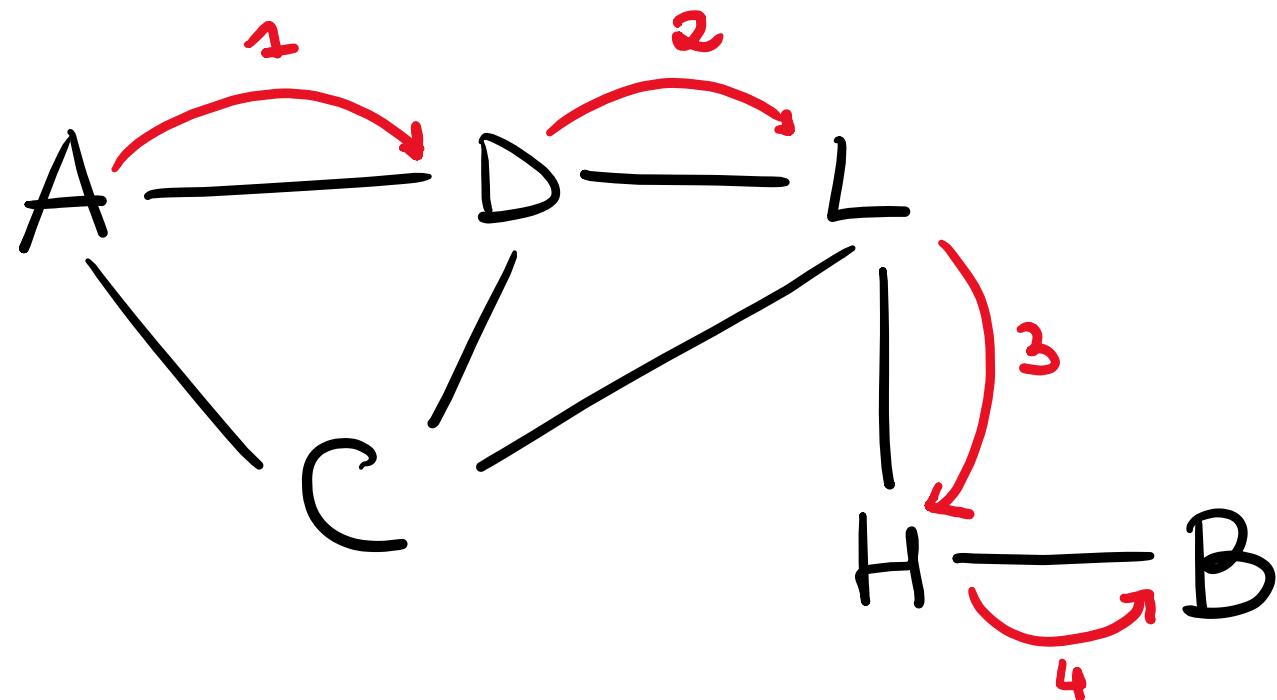
Esercizio 3: diametro del grafo

- Il diametro di un grafo è la lunghezza del cammino minimo (in termini di numero di archi) più lungo fra tutte le coppie di nodi. Progettare un algoritmo che misuri il diametro di un grafo.



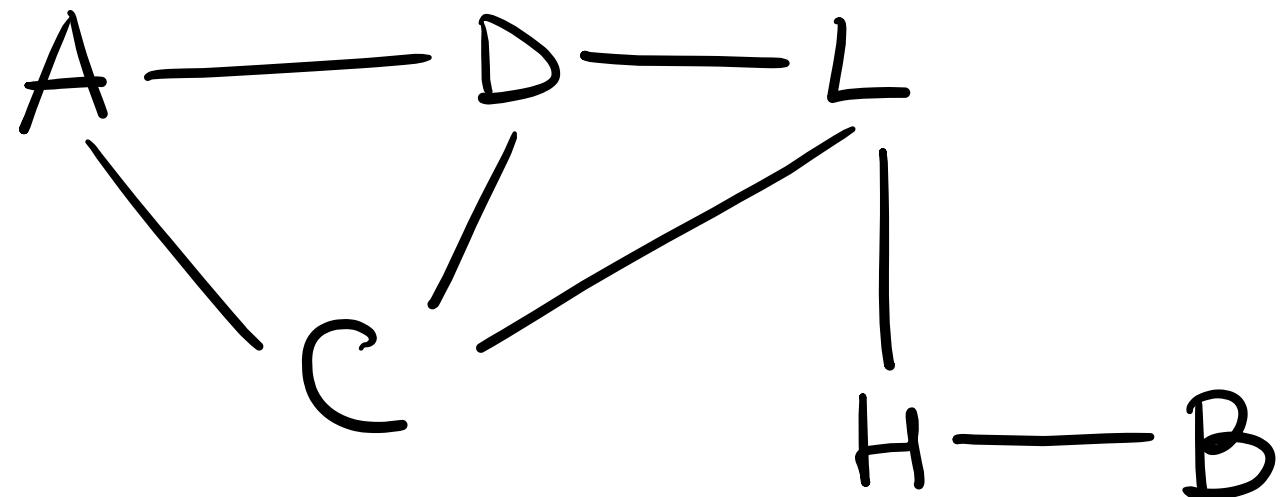
Esercizio 3: diametro del grafo

- Il diametro di un grafo è la lunghezza del cammino minimo (in termini di numero di archi) più lungo fra tutte le coppie di nodi. Progettare un algoritmo che misuri il diametro di un grafo.

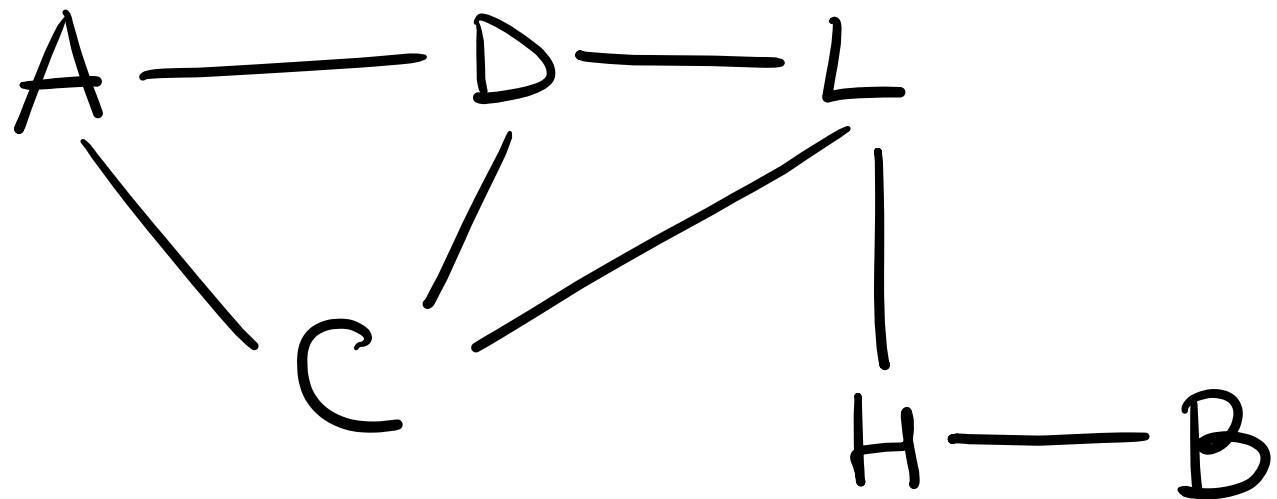


Esercizio 3: Idee?

- Il diametro di un grafo è la lunghezza del cammino minimo (in termini di numero di archi) più lungo fra tutte le coppie di nodi. Progettare un algoritmo che misuri il diametro di un grafo.

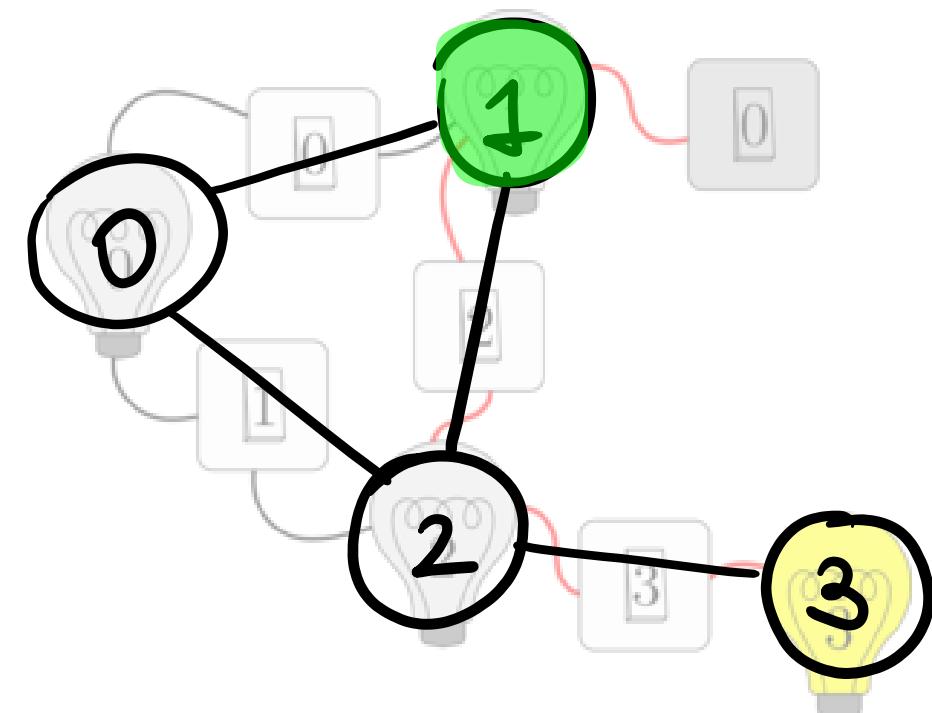
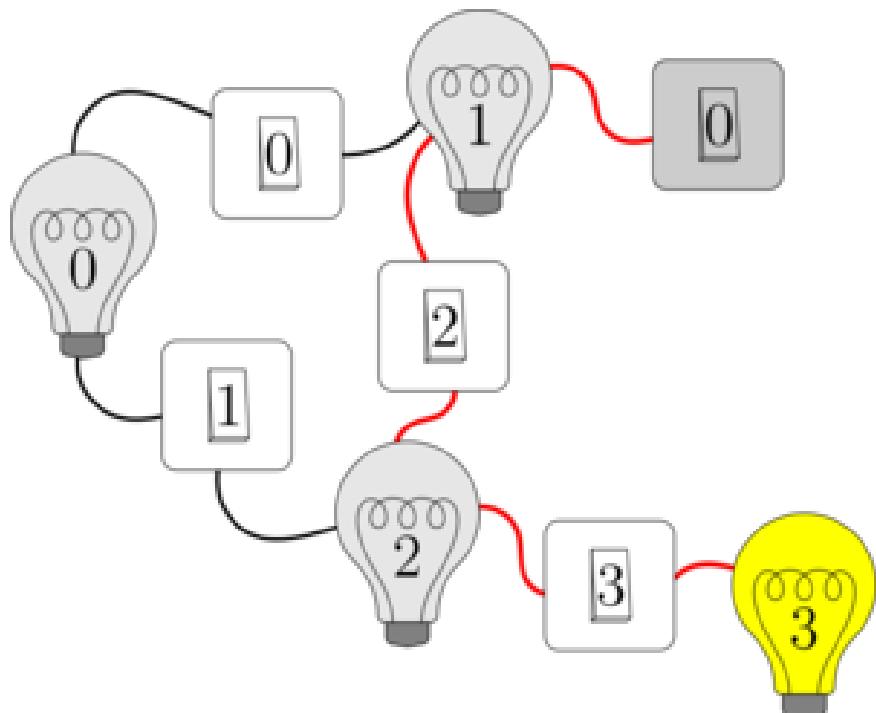


Esercizio 3: Idee? - Whiteboard

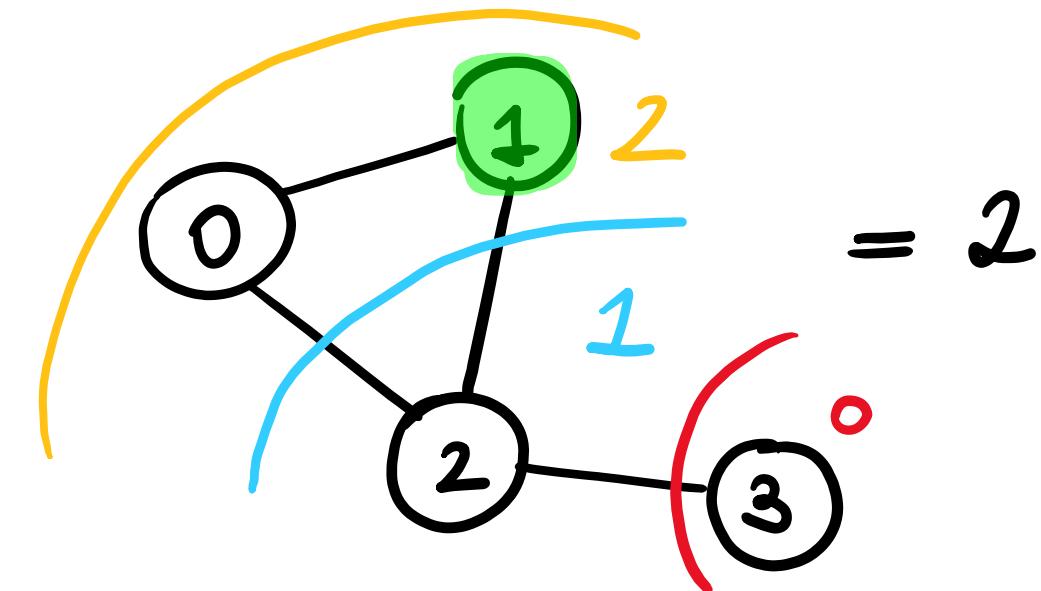
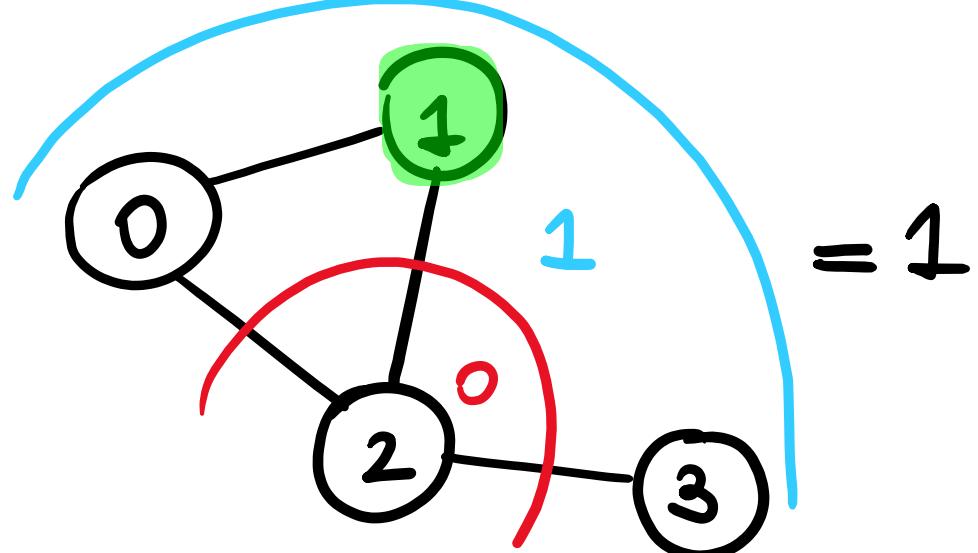
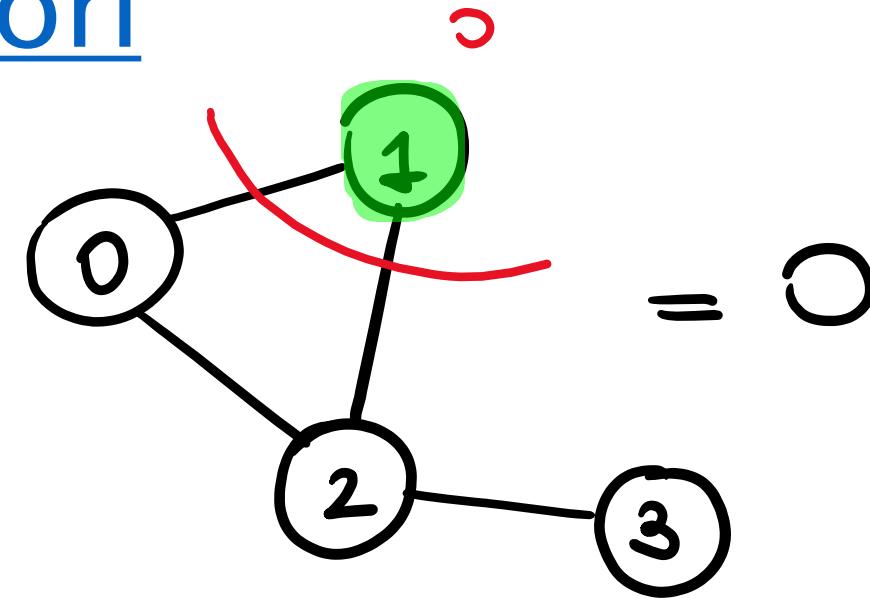
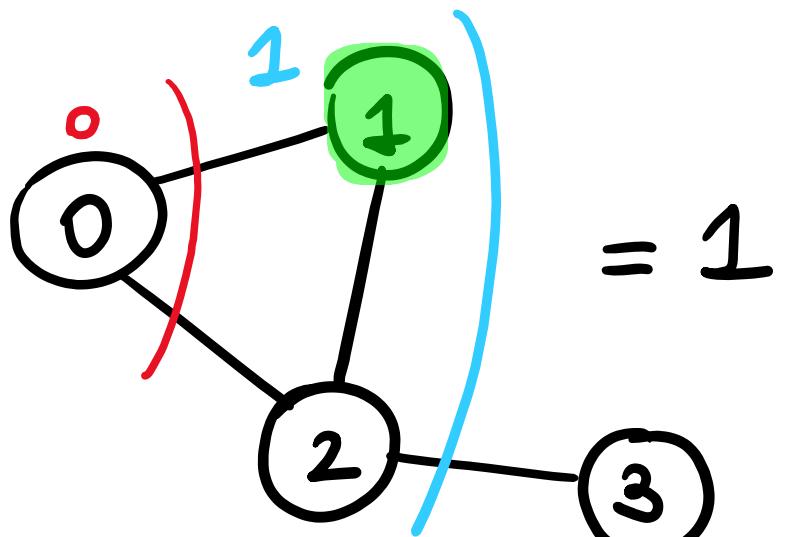


Territoriali: Interruttori

- Risoluzione?

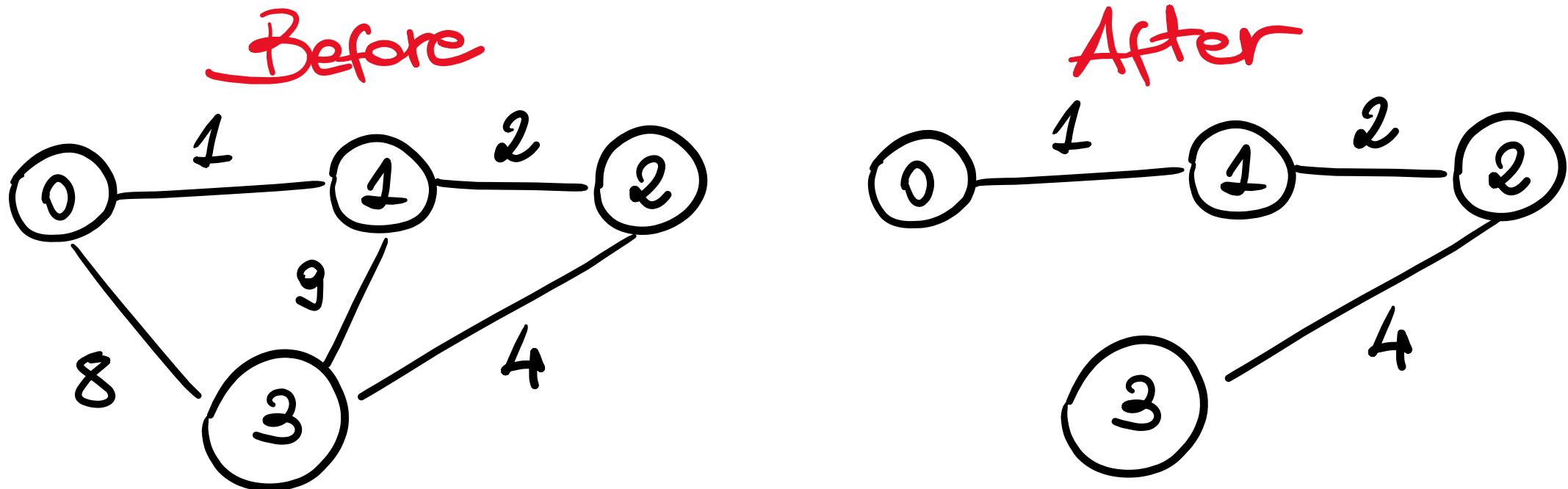


Territoriali: Interruttori



Grafi: Cammini Minimi

- Problema: Dato in input un grafo pesato $G(V, E)$, w funzione di peso tale che $w(E_i) = p \in \mathbb{R}$ e un nodo s , restituisca il cammino di costo minimo dal nodo s a tutti gli altri nodi.



Grafi: Cammini Minimi

- Occorrente per la risoluzione
 1. Grafo $G(V, E)$ con la relativa funzione dei pesi w
 2. Il nodo di partenza s
 3. Un vettore d per memorizzare la distanza di un nodo da s ($d[u]$ conterrà la distanza di u da s)
 4. Un vettore T di padri ($T[u]$ conterrà il padre del nodo u)
 5. Una Struttura dati S in cui memorizzare i nodi raggiunti e tenere conto del costo per arrivarcì
 6. Un vettore per verificare se il nodo u è presente in S (e conoscere la sua posizione all'interno)

```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    vector<int> d(G.size(), INT_MAX);
    vector<int> T(G.size(), -1);
    vector<int> b(G.size(), -1);

    d[s] = 0;
    MinPriorityQueue S(G.size());
    b[s] = insert(S, Element(s, 0));
    while(!isEmpty(S)){
        int u = extract_min(S).label;
        b[u] = -1;
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
    return make_pair(T, d);
}

```

```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

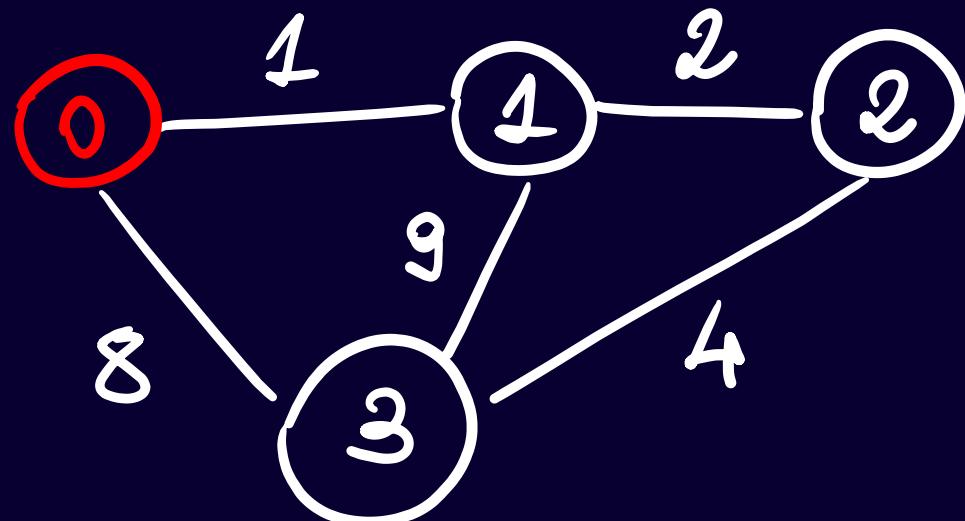
    vector<int> d(G.size(), INT_MAX);
    vector<int> T(G.size(), -1);
    vector<int> b(G.size(), -1);
    d[s] = 0;
    MinPriorityQueue S(G.size());
    b[s] = insert(S, Element(s, 0));
    ...
}

```

\downarrow
Element(label, priority)

	0	1	2	3
d	0	Inf	Inf	Inf
T	-1	-1	-1	-1
b	0	-1	-1	-1
S	(0, 0)	-	-	-

Se $E1.priority < E2.priority$
 $\Rightarrow E1 < E2$ in S



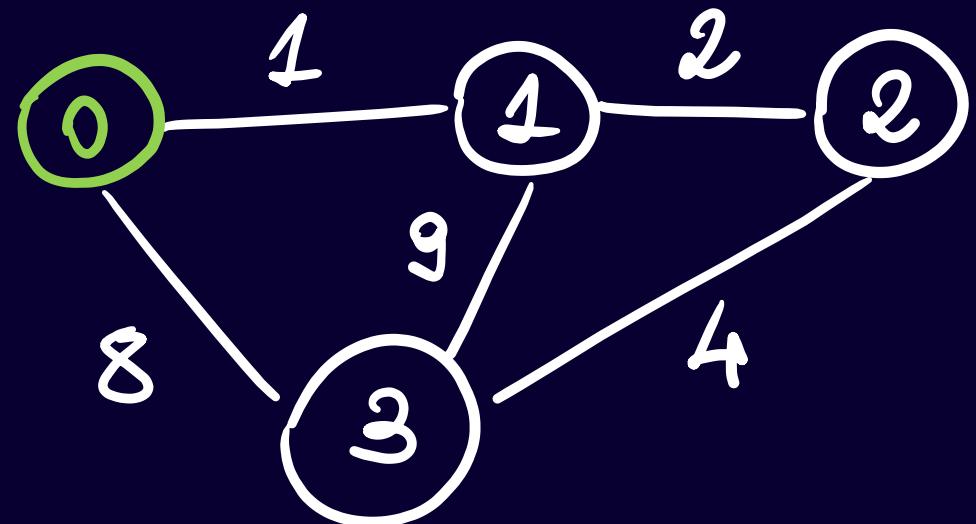
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        int u = extract_min(S).label;
        b[u] = -1;
        ...
    }
    ...
}

```

	0	1	2	3
d	0	Inf	Inf	Inf
T	-1	-1	-1	-1
b	0	-1	-1	-1
S	(0, 0)	-	-	-



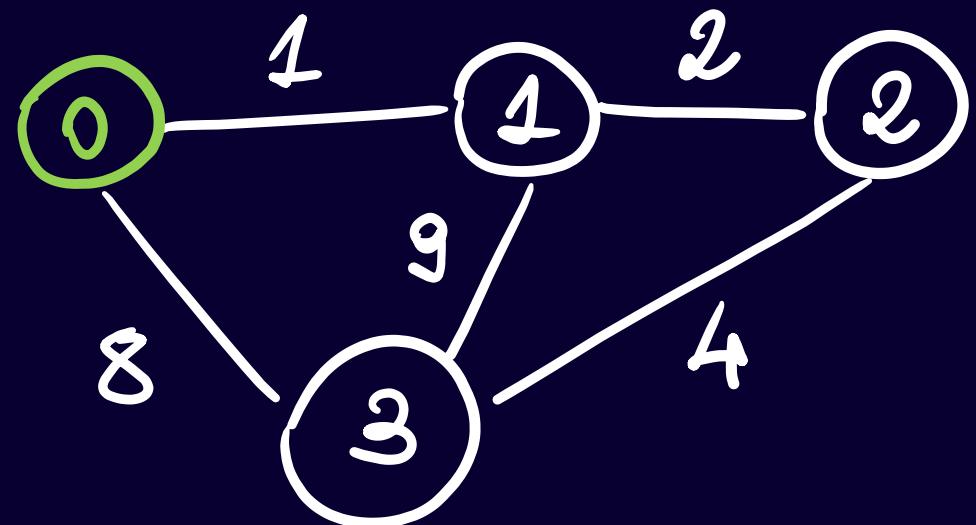
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        int u = extract_min(S).label; // u = 0
        b[u] = -1;
        ...
    }
    ...
}

```

	0	1	2	3
d	0	Inf	Inf	Inf
T	-1	-1	-1	-1
b	-1	-1	-1	-1
S	-	-	-	-



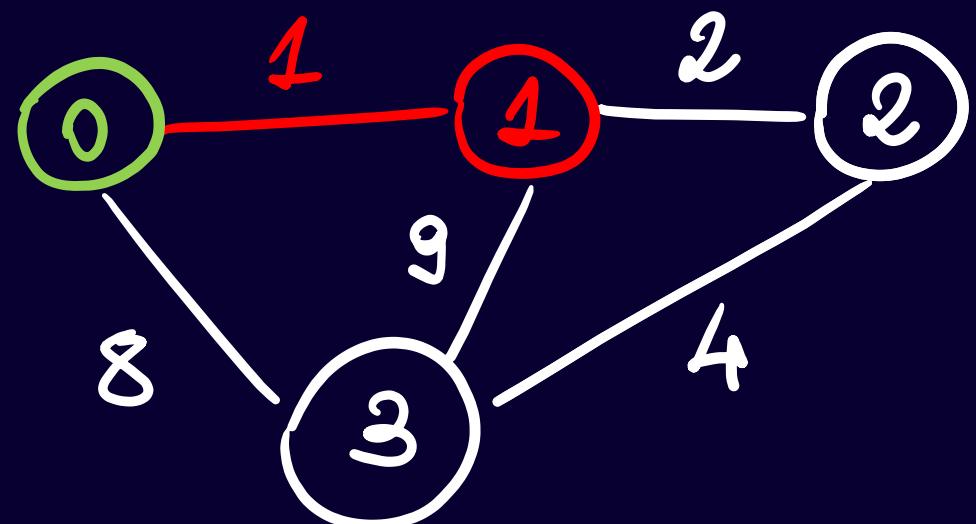
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        → for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

	0	1	2	3
d	0	Inf	Inf	Inf
T	-1	-1	-1	-1
b	-1	-1	-1	-1
S	-	-	-	-



```

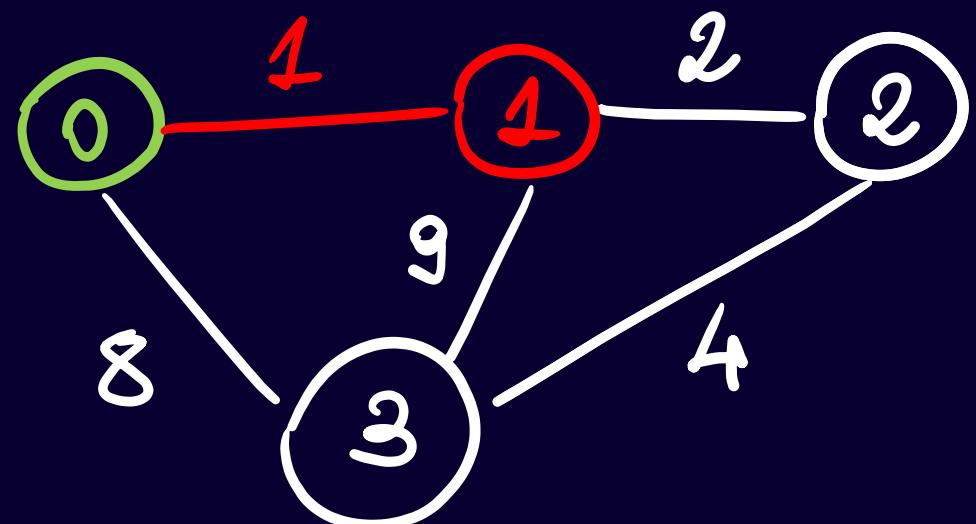
pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 0, v = 1
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
    ...
}

d[0] + w[0][1] < d[1]

```

	0	1	2	3
d	0	Inf	Inf	Inf
T	-1	-1	-1	-1
b	-1	-1	-1	-1
S	-	-	-	-



```

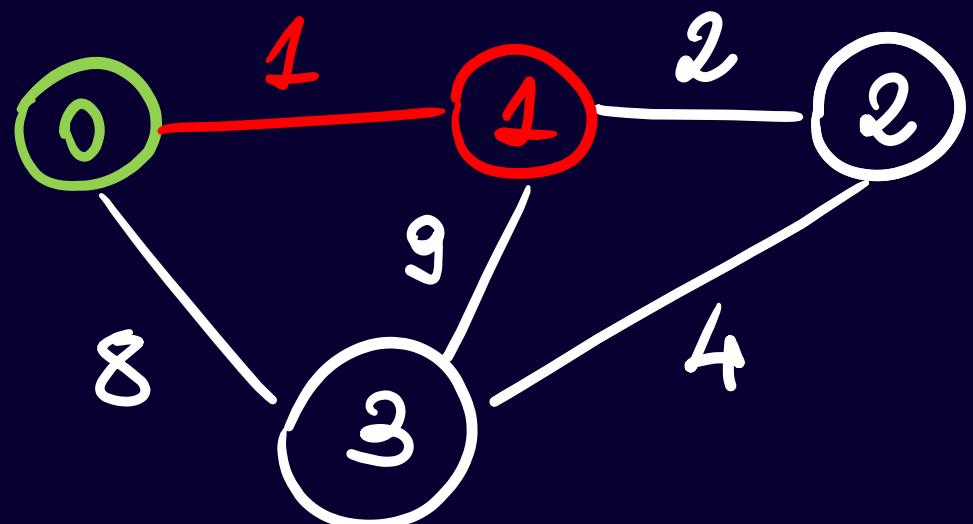
pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 0, v = 1
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

0 + 1 ? + ∞

	0	1	2	3
d	0	Inf	Inf	Inf
T	-1	-1	-1	-1
b	-1	-1	-1	-1
S	-	-	-	-



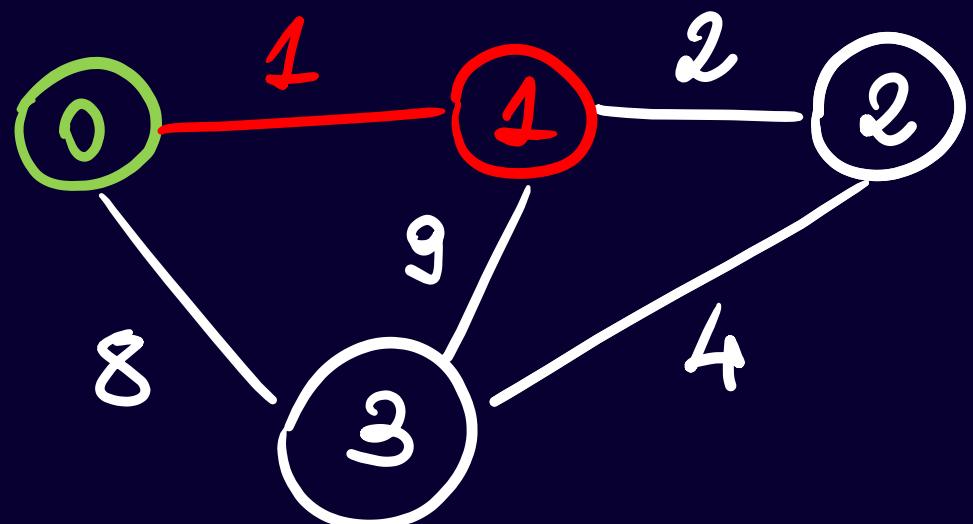
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 0, v = 1
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

	0	1	2	3
d	0	Inf	Inf	Inf
T	-1	-1	-1	-1
b	-1	0	-1	-1
S	(1, 1)	-	-	-



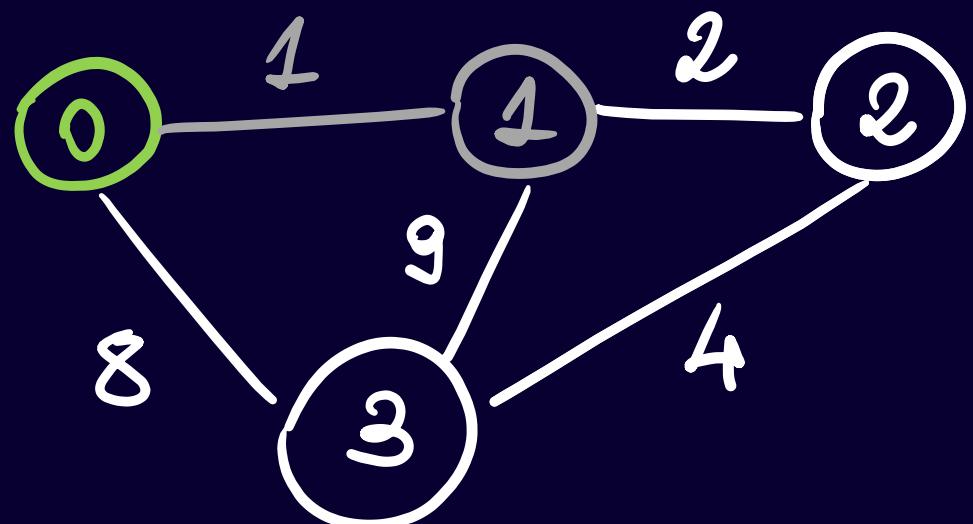
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]) { // u = 0, v = 1
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

	0	1	2	3
d	0	1	Inf	Inf
T	-1	0	-1	-1
b	-1	0	-1	-1
S	(1, 1)	-	-	-



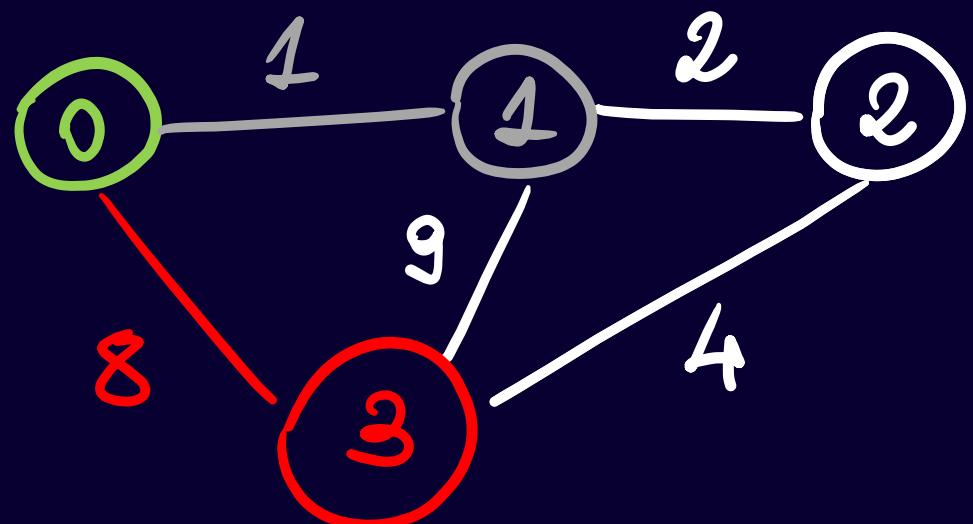
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 0, v = 3
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

	0	1	2	3
d	0	1	Inf	Inf
T	-1	0	-1	-1
b	-1	0	-1	-1
S	(1, 1)	-	-	-



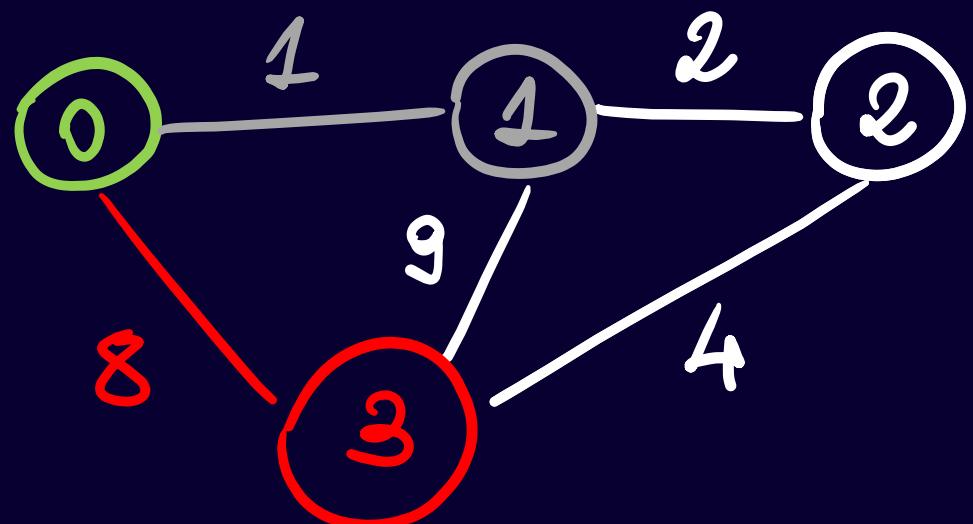
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]) { // u = 0, v = 3
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

	0	1	2	3
d	0	1	Inf	Inf
T	-1	0	-1	-1
b	-1	0	-1	1
S	(1, 1)	(3, 8)	-	-



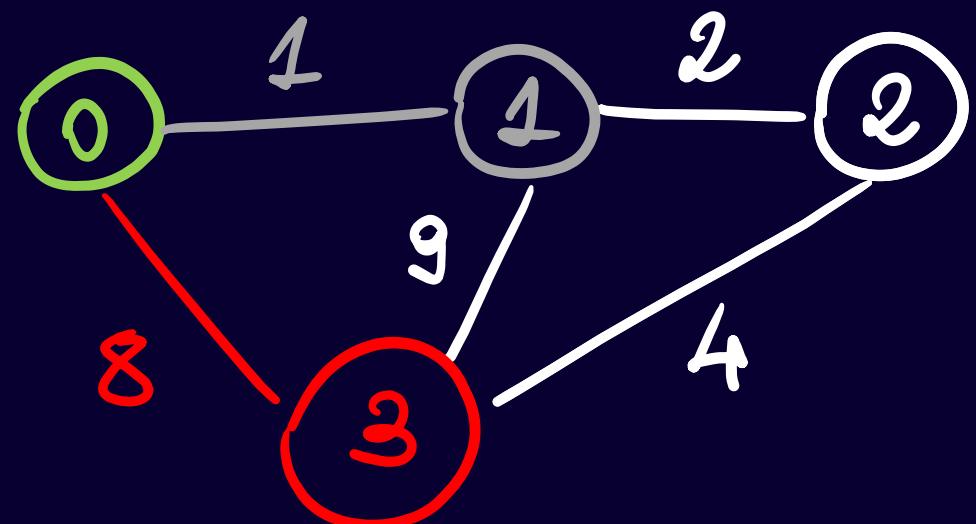
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]) { // u = 0, v = 3
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

d	0	1	2	3
	0	1	Inf	8
T	-1	0	-1	0
b	-1	0	-1	1
S	(1, 1)	(3, 8)	-	-



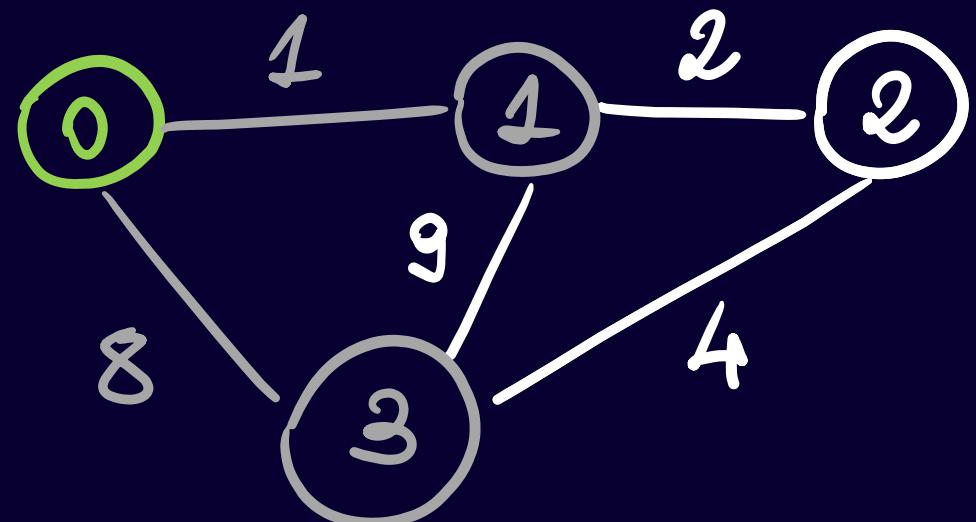
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        int u = extract_min(S).label;
        b[u] = -1;
        ...
    }
    ...
}

```

d	0	1	2	3
	0	1	Inf	8
T	-1	0	-1	0
b	-1	0	-1	1
S	(1, 1)	(3, 8)	-	-



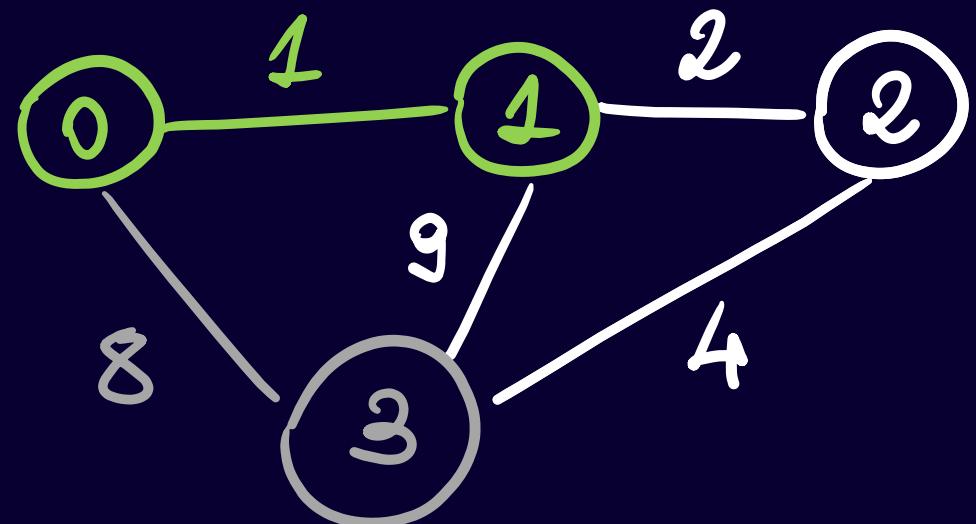
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        int u = extract_min(S).label; // u = 1
        b[u] = -1;
        ...
    }
    ...
}

```

d	0	1	2	3
	0	1	Inf	8
T	-1	0	-1	0
b	-1	-1	-1	0
S	(3, 8)	-	-	-



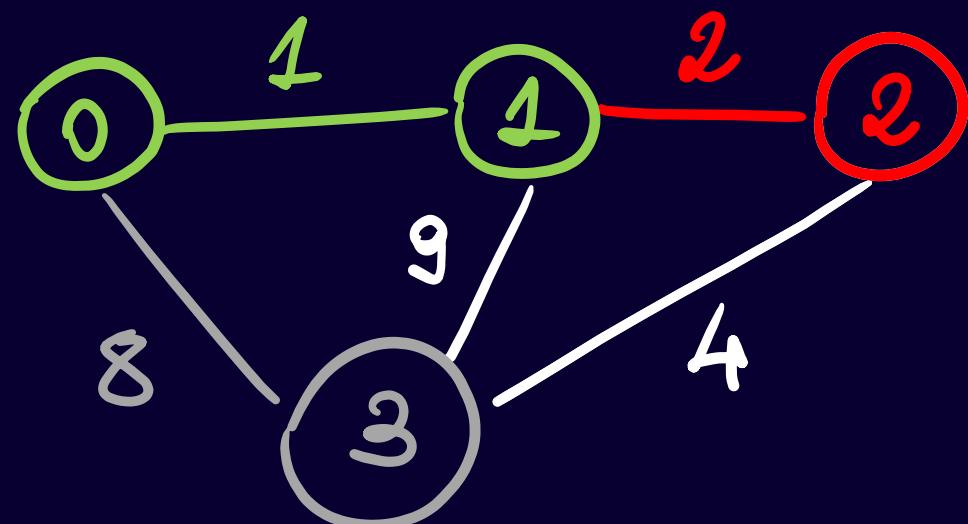
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 1, v = 2
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

	0	1	2	3
d	0	1	Inf	8
T	-1	0	-1	0
b	-1	-1	-1	0
S	(3, 8)	-	-	-



```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 1, v = 2
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

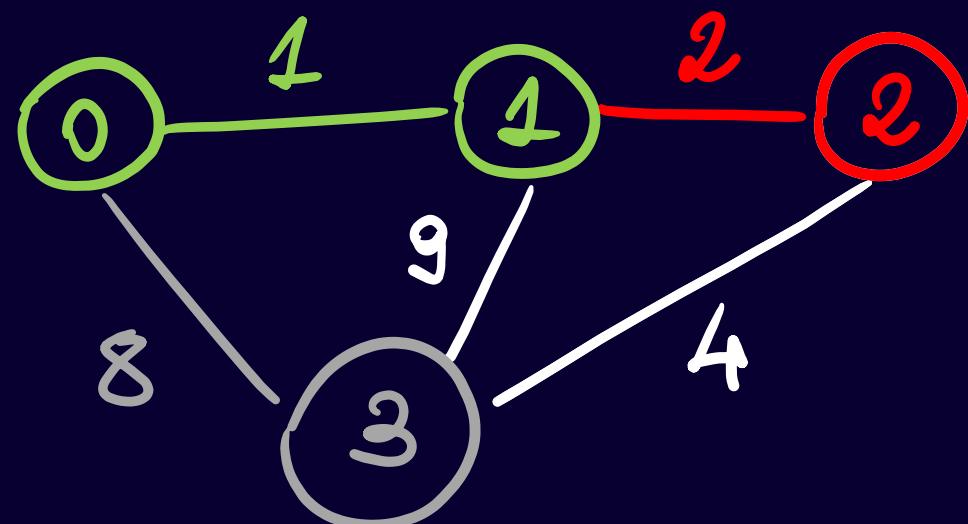
→

<i>d</i>	0	1	2	3
	0	1	Inf	8

<i>T</i>	-1	0	-1	0
	-1	0	-1	0

<i>b</i>	-1	-1	0	1
	-1	-1	0	1

<i>S</i>	(2, 3)	(3, 8)	-	-
	(2, 3)	(3, 8)	-	-



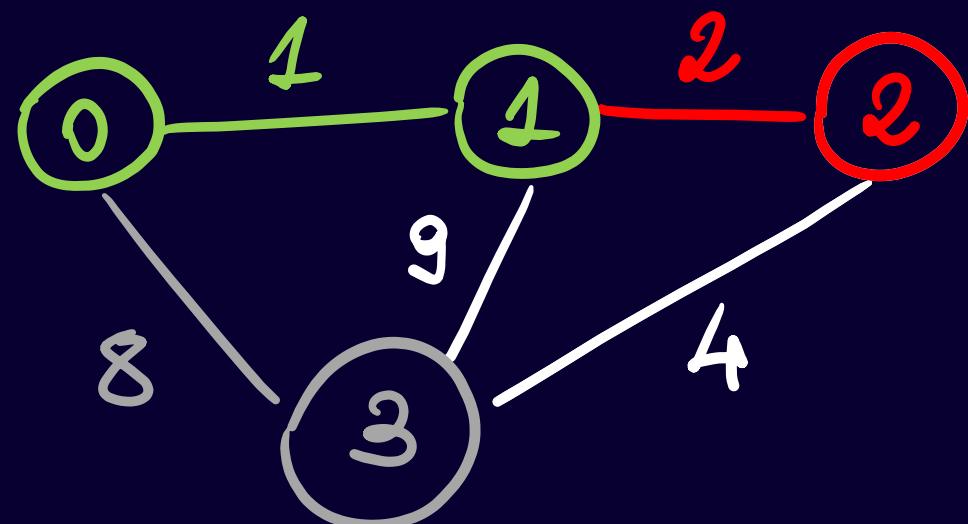
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 1, v = 2
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

d	0	1	2	3
0	1	3	8	
T	-1	0	1	0
b	-1	-1	0	1
S	(2, 3)	(3, 8)	-	-



```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 1, v = 3
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

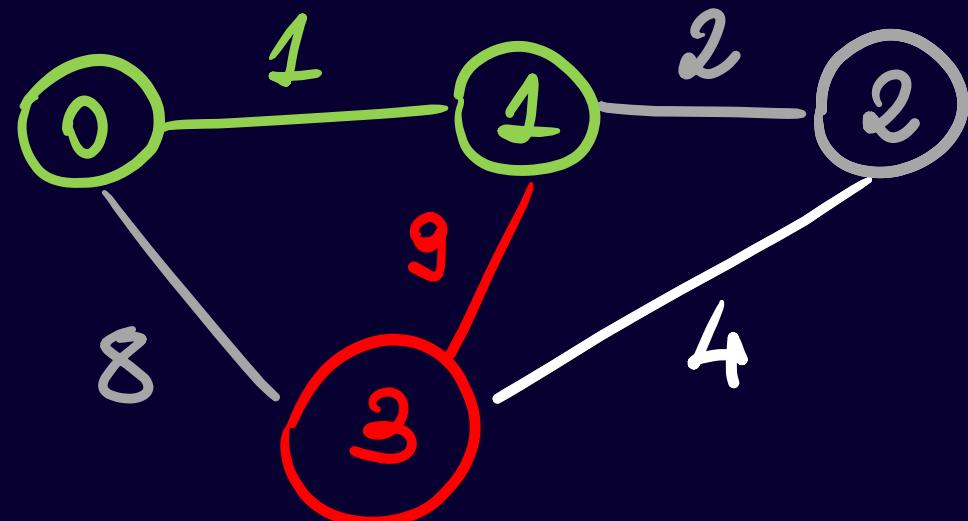
→

d	0	1	2	3
	0	1	3	8

T	-1	0	1	0
	-1	0	1	0

b	-1	-1	0	1
	-1	-1	0	1

S	(2, 3)	(3, 8)	-	-
	(2, 3)	(3, 8)	-	-



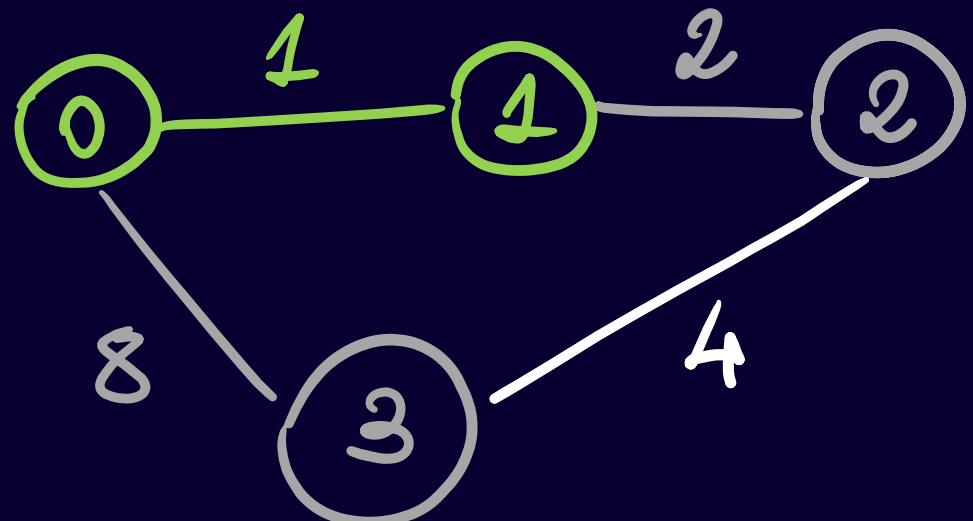
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]) { // u = 1, v = 3
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

d	0	1	2	3
0	1	3	8	
T	-1	0	1	0
b	-1	-1	0	1
S	(2, 3)	(3, 8)	-	-



```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

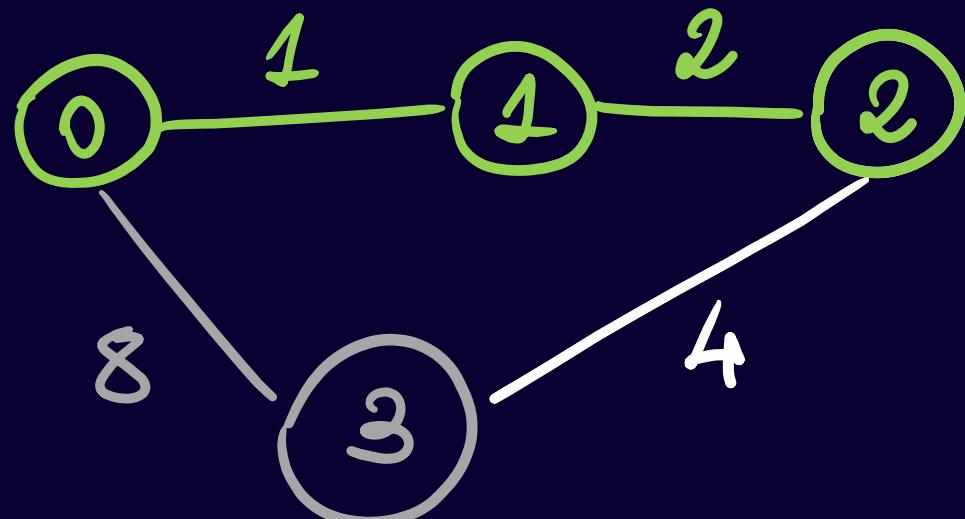
```

<i>d</i>	0	1	2	3
	0	1	3	8

<i>T</i>	-1	0	1	0
----------	----	---	---	---

<i>b</i>	-1	-1	-1	0
----------	----	----	----	---

<i>S</i>	(3, 8)	-	-	-
----------	--------	---	---	---



```

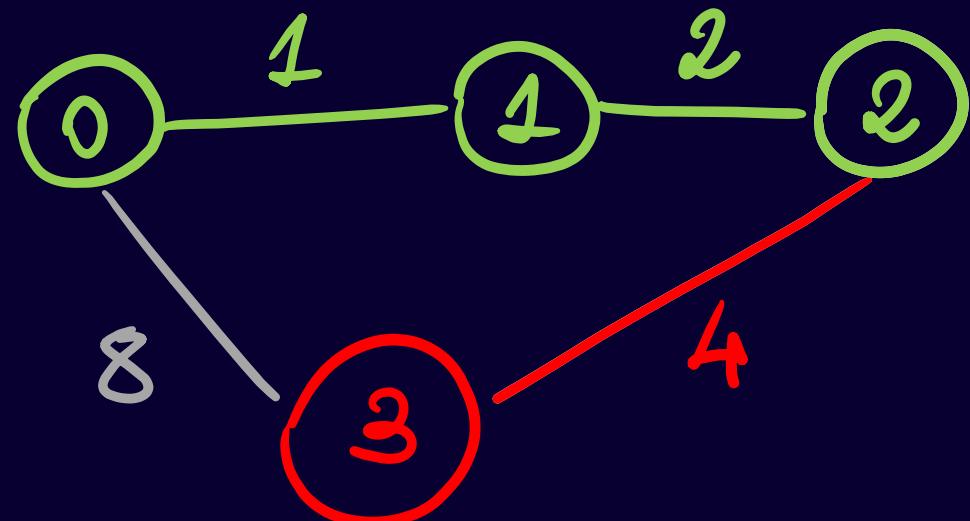
pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 2, v = 3
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

→

d	0	1	2	3
0	1	3	8	
T	-1	0	1	0
b	-1	-1	-1	0
S	(3, 8)	-	-	-



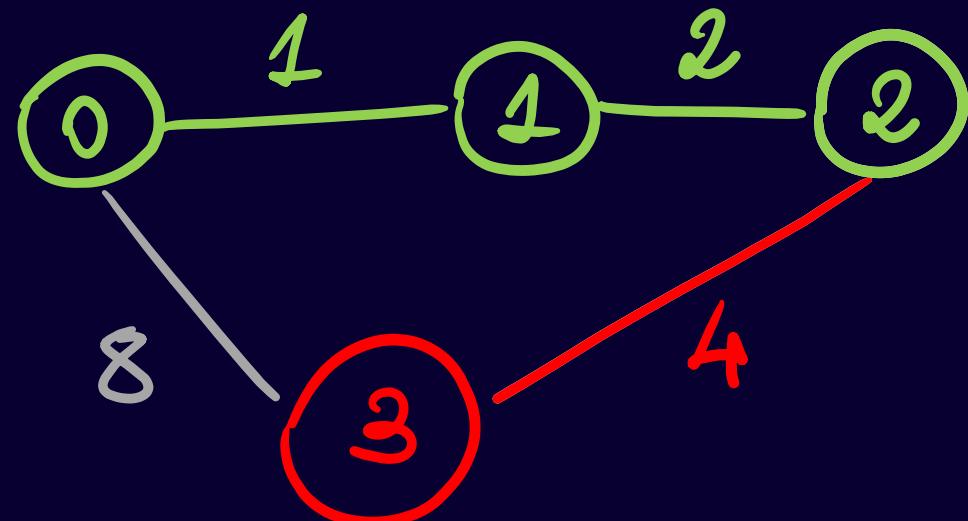
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 2, v = 3
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

d	0	1	2	3
	0	1	3	8
T	-1	0	1	0
b	-1	-1	-1	0
S	(3, 8)	-	-	-



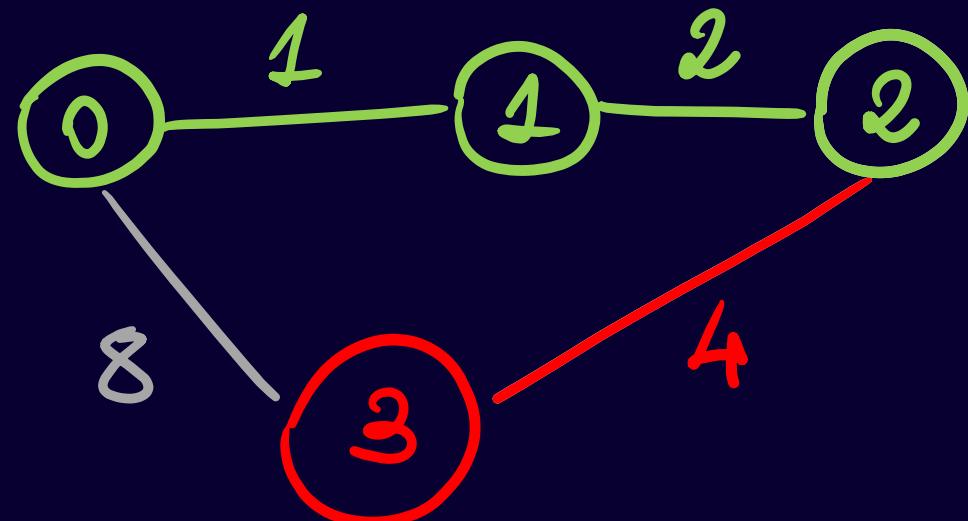
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 2, v = 3
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    → b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

d	0	1	2	3
	0	1	3	8
T	-1	0	1	0
b	-1	-1	-1	0
S	(3, 4)	-	-	-



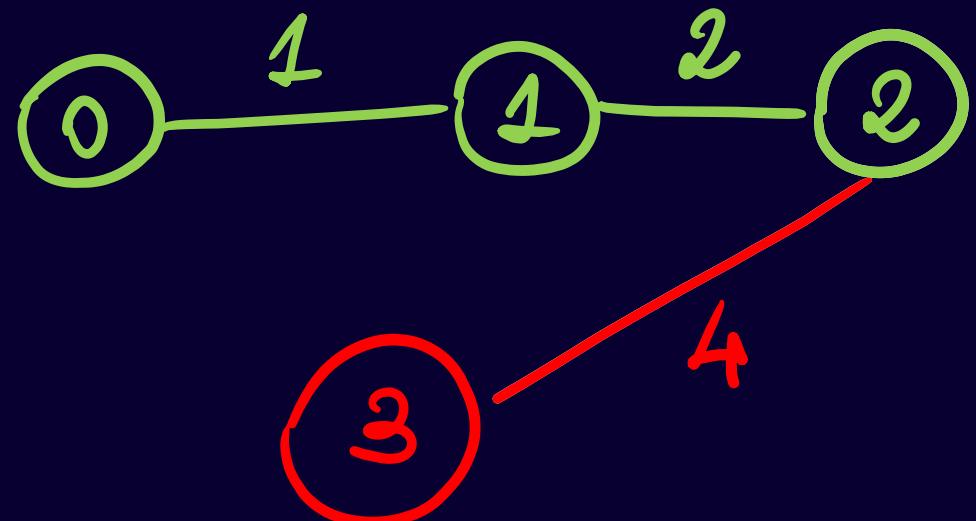
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        ...
        for(int i = 0; i < G[u].vic.size(); i++){
            int v = G[u].vic[i];
            if(d[u] + w[u][v] < d[v]){ // u = 2, v = 3
                if(b[v] == -1){
                    b[v] = insert(S, Element(v, d[u] + w[u][v]));
                } else {
                    b[v] = decrease_priority(S, b[v], d[u] + w[u][v]);
                }
                T[v] = u;
                d[v] = d[u] + w[u][v];
            }
        }
    }
}

```

d	0	1	2	3
0	1	3	7	
T	-1	0	1	2
b	-1	-1	-1	0
S	(3, 4)	-	-	-



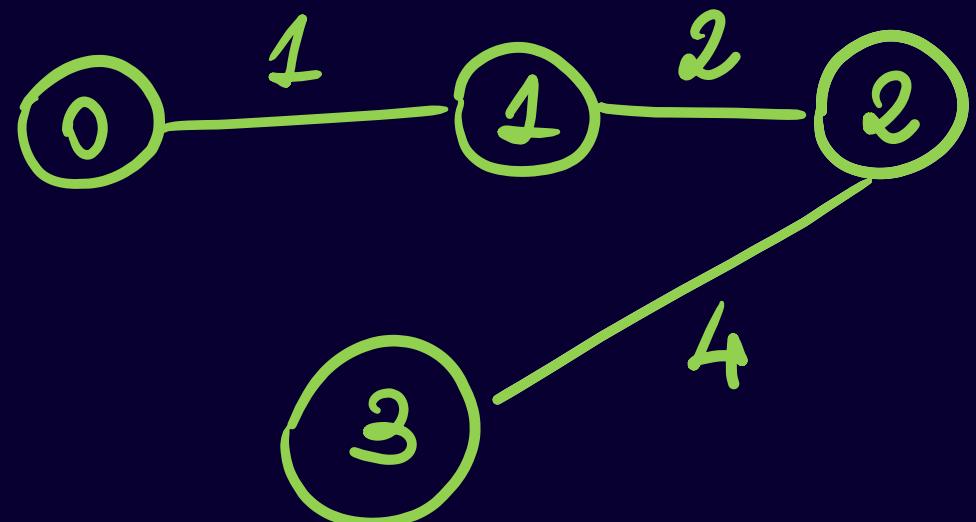
```

pair<vector<int>, vector<int>> shortestPath(vector<node>& G, vector<vector<int>>& w, int s){

    ...
    while(!isEmpty(S)){
        int u = extract_min(S).label; // u = 1
        b[u] = -1;
        ...
    }
    ...
}

```

d	0	1	2	3
	0	1	3	7
T	-1	0	1	2
b	-1	-1	-1	0
S	(3, 4)	-	-	-



Grafi: Cammini Minimi

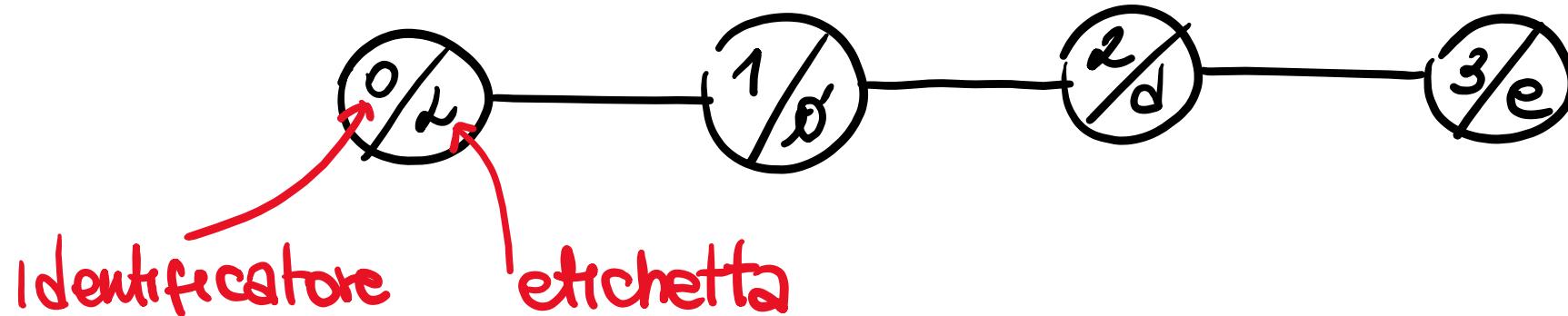
- Inizializzazione:
 - La distanza da s ad ogni nodo è pari ad infinito
 - s è la radice dell'albero (il suo padre è -1, cioè nessuno)
 - Inseriamo s all'interno della struttura S (Coda a Priorità)
- Iterativamente (fino a quando S non è vuota):
 - Estraiamo da S il nodo u con priorità (costo) minimo
 - Controlliamo i nodi v che sono adiacenti ad u
 - Se v non è in S (e non è già stato estratto), lo inseriamo
 - Altrimenti, aggiorniamo la sua priorità se e solo se il costo per andare da u a v è minore di quello attuale ($d[u] + w[u][v] < d[v]$)

Grafi: cosa non abbiamo detto

- Cicli (grafi orientati e non orientati)
- Componenti Connesse
- Componenti Fortemente Connesse
- Grafi Bicolorabili
- Ordinamento Topologico
- Grafi orientati aciclici
- Grafo dei cammini minimi (Dijkstra, Bellman-Ford, ...)
- Grafo di peso minimo (Kruskal, Prim)
- Ricerca Locale e Reti di Flusso
- ...

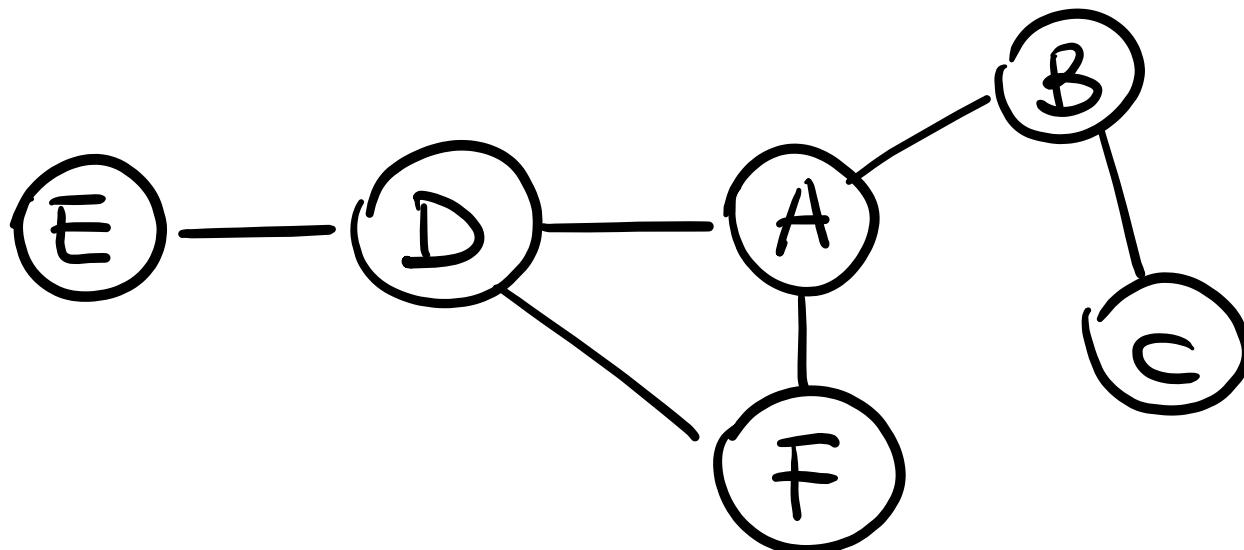
Grafi: Esercizi più ‘seri’

- Il grafo contiene la sotto-sequenza LODE?
 - Considerando un grafo non orientato i cui nodi sono etichettati con delle lettere dalla A alla Z, scrivere un algoritmo che verifichi se è contenuta la sottostringa LODE
 - Una sottostringa è una sequenza contigua di caratteri
 - Potete immaginare di poter accedere ad un campo *u.label* di un nodo per verificare l'etichetta corrispondente.



Grafi: Esercizi più ‘seri’

- Nodi a distanza $\leq d$
 - Dato un grafo non orientato, un nodo A e un intero d, restituire il numero di nodi che si trovano a distanza $\leq d$ partendo da A.



Se $d = 0$

A

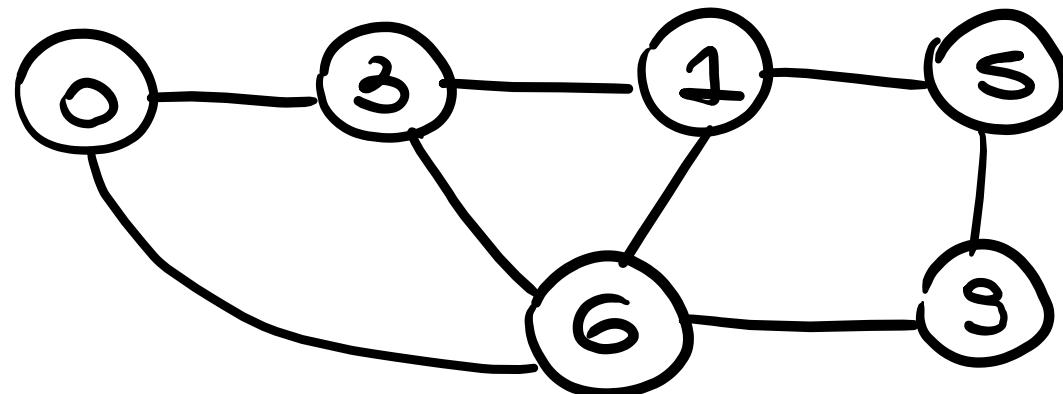
Se $d = 1$

A, B, D, F

Grafi: Esercizi più ‘seri’

- Sequenza incrementale più lunga

- Considerando un grafo non orientato, restituire la lunghezza della più lunga sequenza distinta di nodi V_1, \dots, V_n tale per cui $V_i < V_j$ per ogni $i < j$ (cioè per ogni nodo, il successivo deve avere sempre etichetta strettamente maggiore del precedente)



$$|0, 3, 6, 9| = 4$$

$$|1, 3, 6, 9| = 4$$

Programmazione Dinamica

- Tecnica di programmazione simile a *divide et impera* (si scomponete un problema complesso in sottoproblemi di grandezza unitaria molto semplici), dove però si utilizzano le operazioni eseguite in precedenza (memorizzandole) per poter svolgere calcoli successivi.

Programmazione Dinamica: Fibonacci

- Es: Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

- Problema: qual è l' n -esimo numero della sequenza?

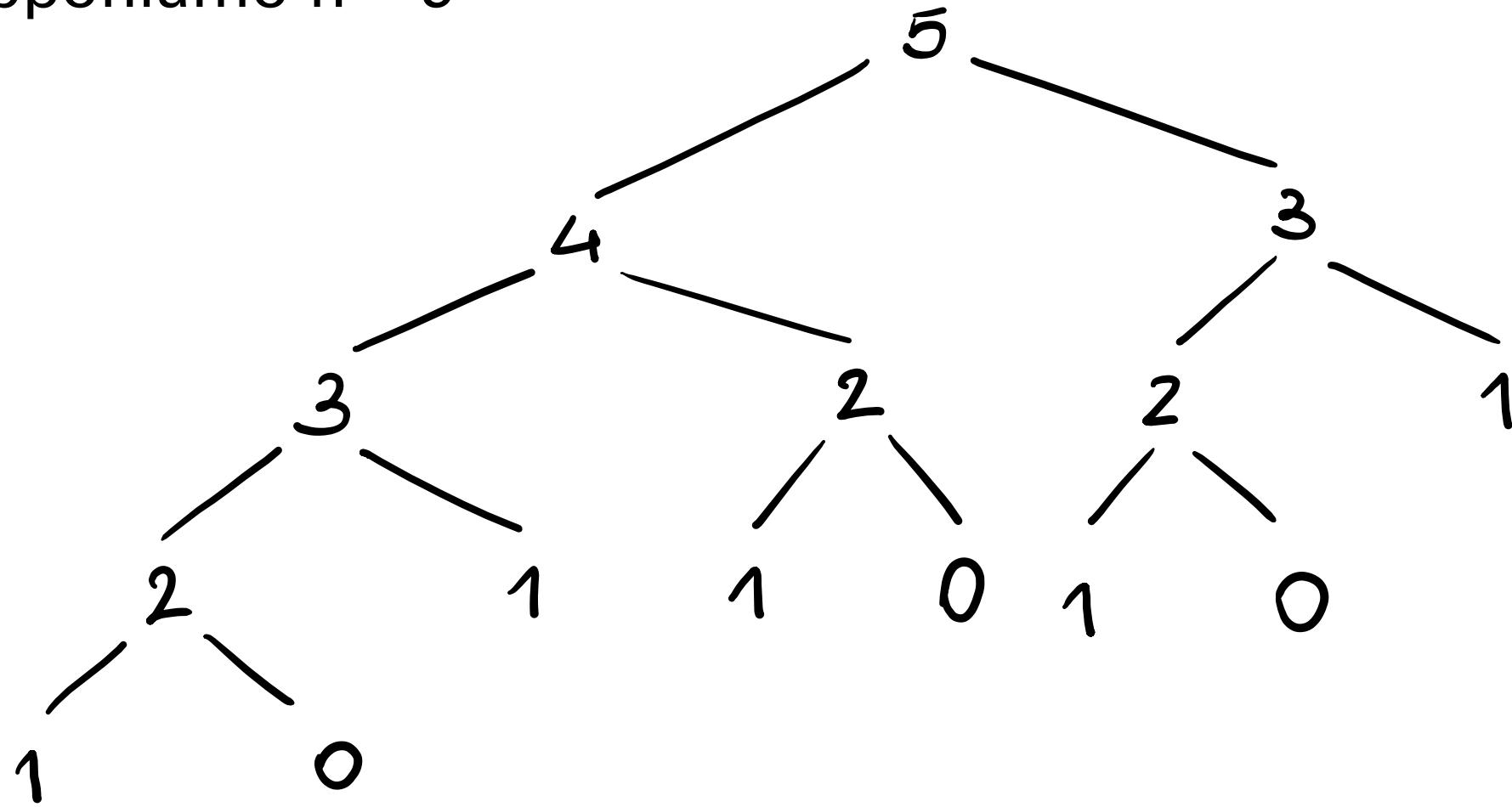
Programmazione Dinamica: Fibonacci

- Per ottenere l'n-esimo numero della sequenza, basta applicare
$$n = n-1 + n-2$$
- Come è possibile osservare, questa formula si presta molto bene alla ricorsione:

```
int fibonacci(int n){  
    if(n <= 1){  
        return n;  
    }  
    return fibonacci(n-1) + fibonacci(n-2);  
}
```

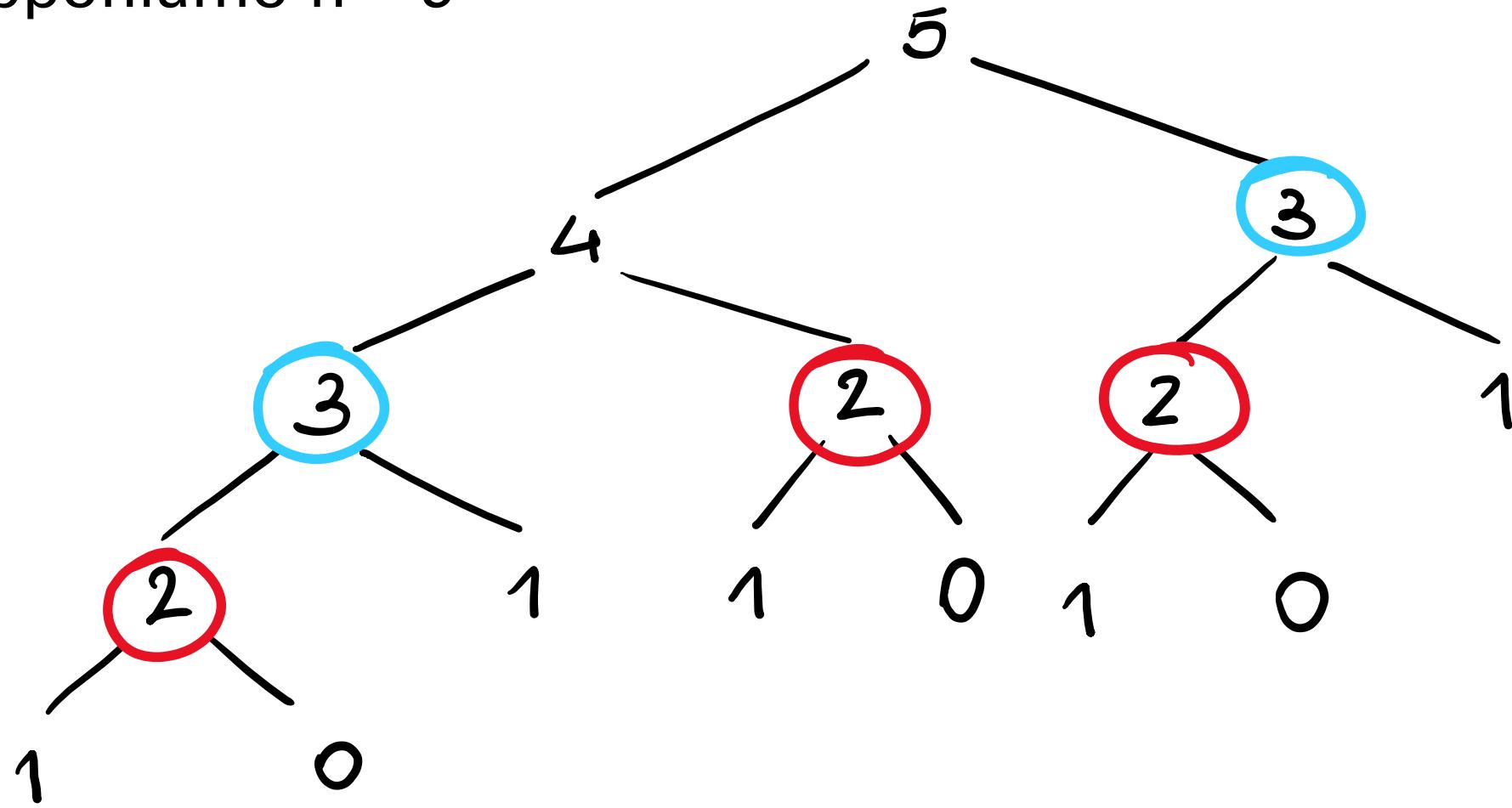
Programmazione Dinamica: Fibonacci

- Supponiamo $n = 5$



Programmazione Dinamica: Fibonacci

- Supponiamo $n = 5$



Programmazione Dinamica: Fibonacci

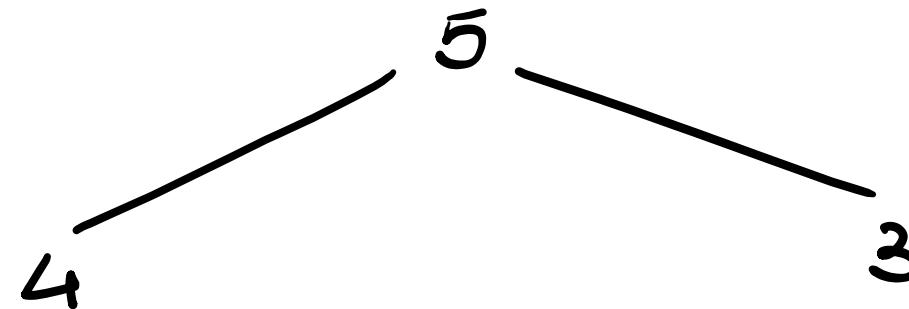
- Abbiamo eseguito 1 volta in più i calcoli per poter ottenere il terzo numero e 2 volte in più quello in seconda posizione. Potevamo fare di meglio?

Programmazione Dinamica: Fibonacci

- Se avessimo memorizzato i risultati precedenti all'interno di un array, avremmo avuto la possibilità, una volta calcolati la prima volta, di potervi accedere immediatamente per risolvere le computazioni seguenti.

Programmazione Dinamica: Fibonacci

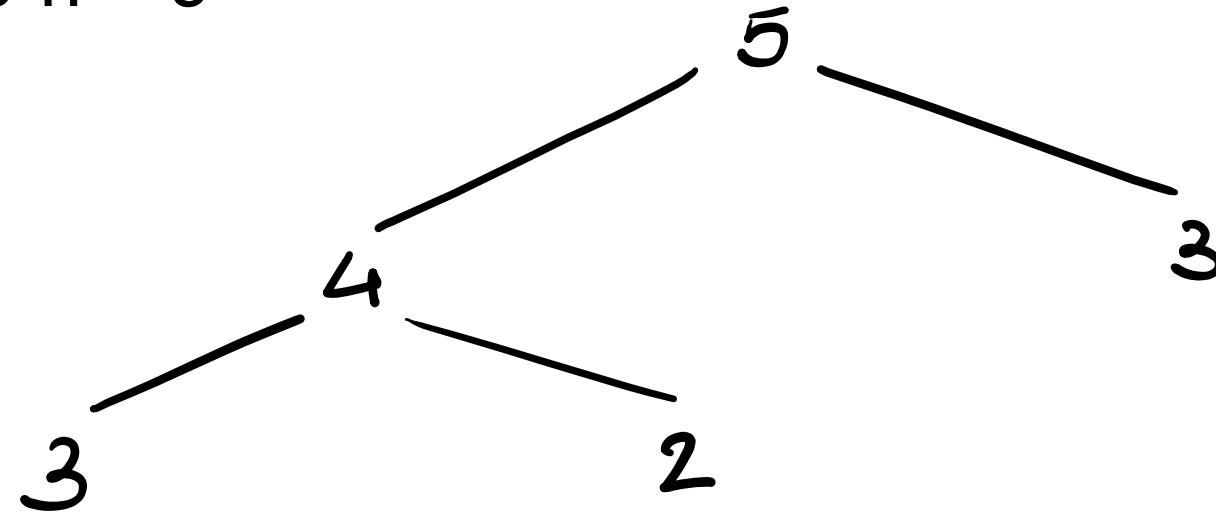
- Supponiamo $n = 5$



0	1				
---	---	--	--	--	--

Programmazione Dinamica: Fibonacci

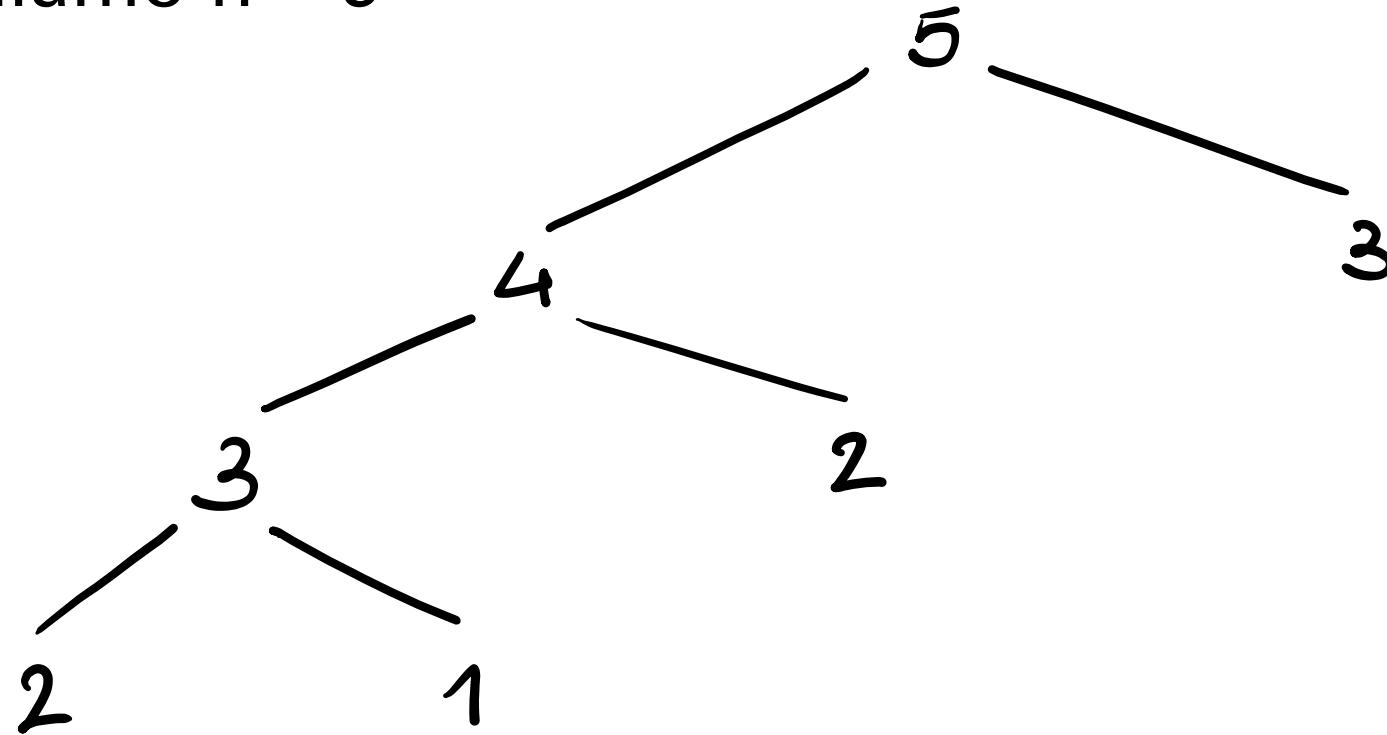
- Supponiamo $n = 5$



0	1				
---	---	--	--	--	--

Programmazione Dinamica: Fibonacci

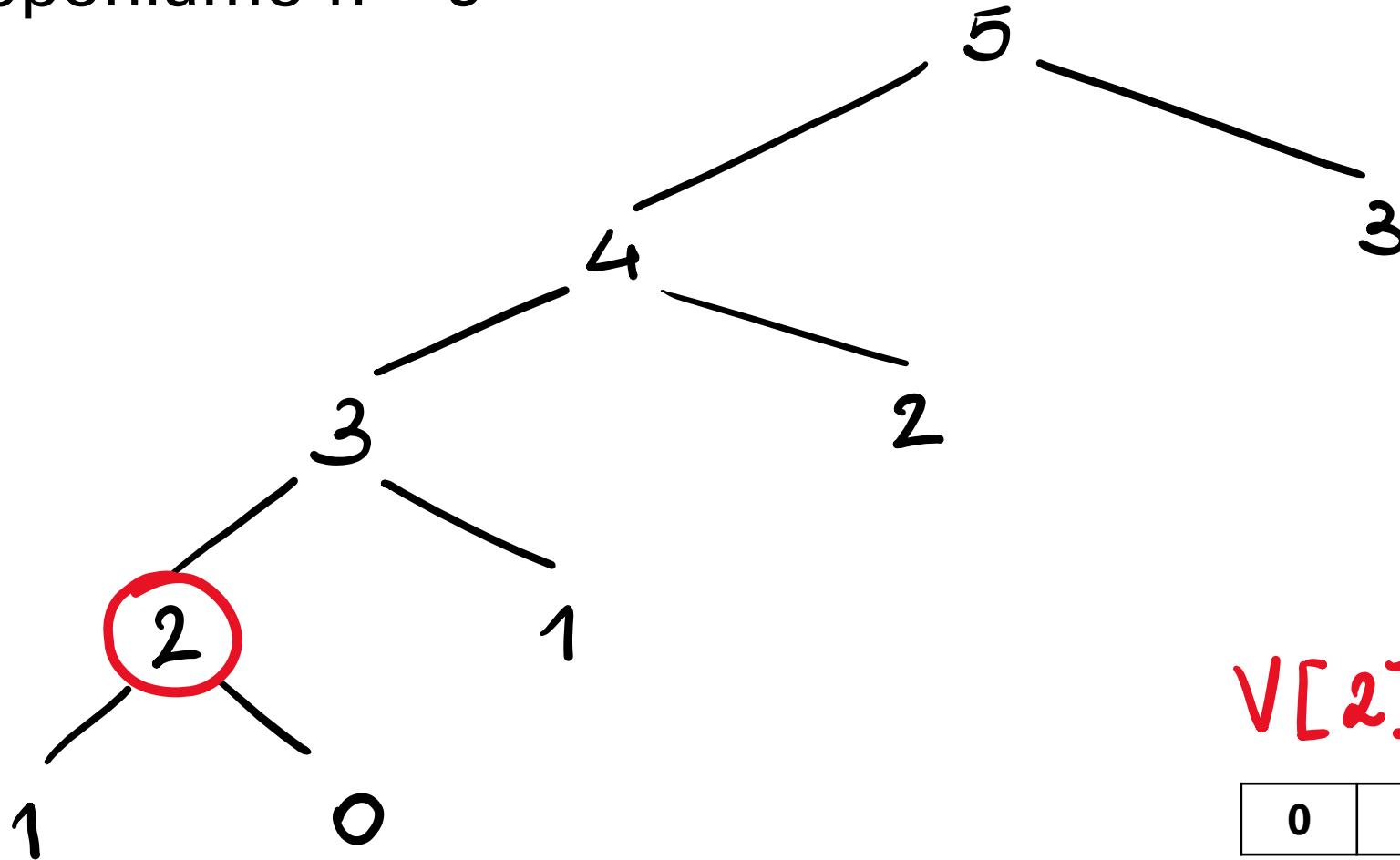
- Supponiamo $n = 5$



0	1				
---	---	--	--	--	--

Programmazione Dinamica: Fibonacci

- Supponiamo $n = 5$

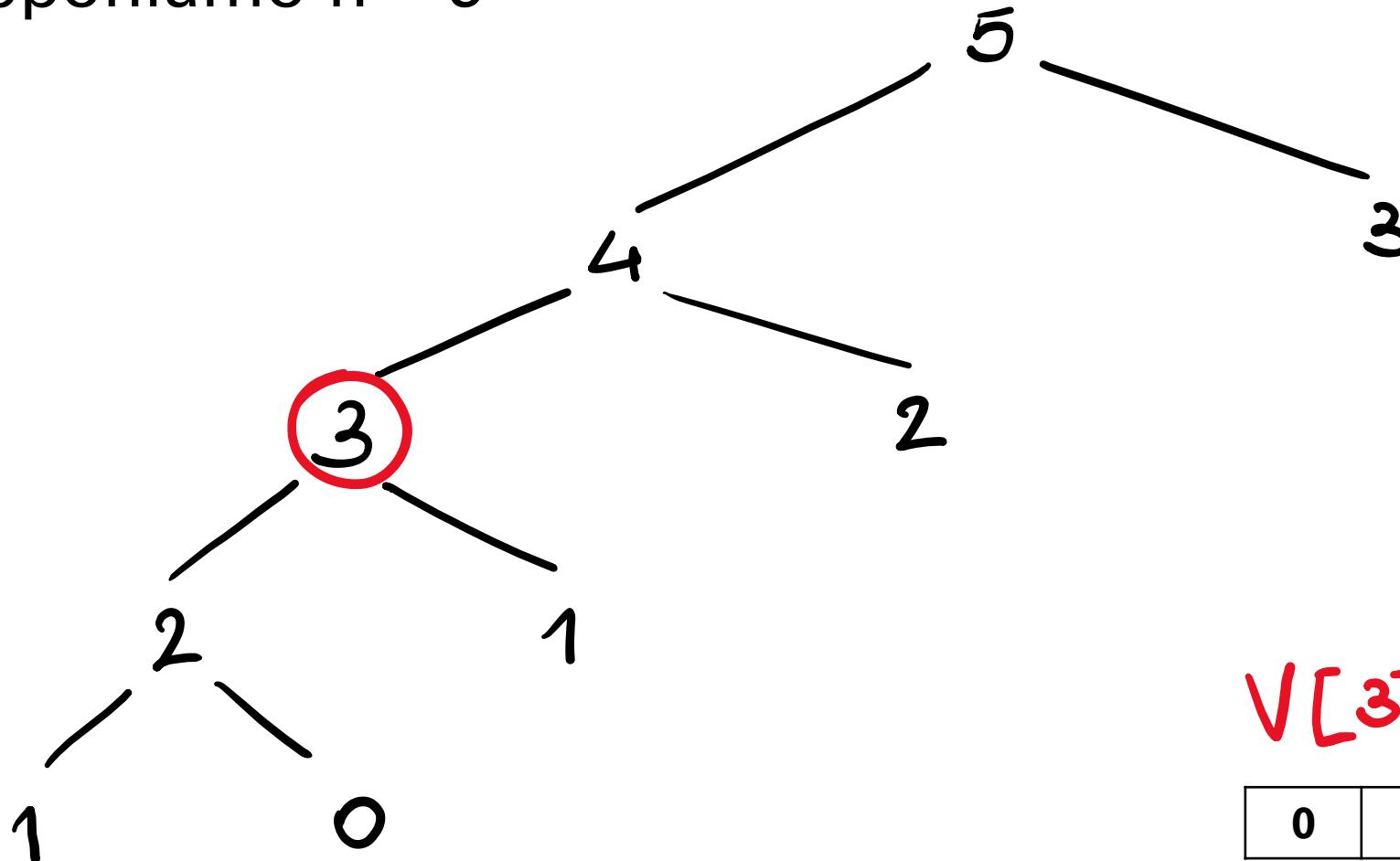


$$V[2] = V[1] + V[0]$$

0	1	1			
---	---	---	--	--	--

Programmazione Dinamica: Fibonacci

- Supponiamo $n = 5$

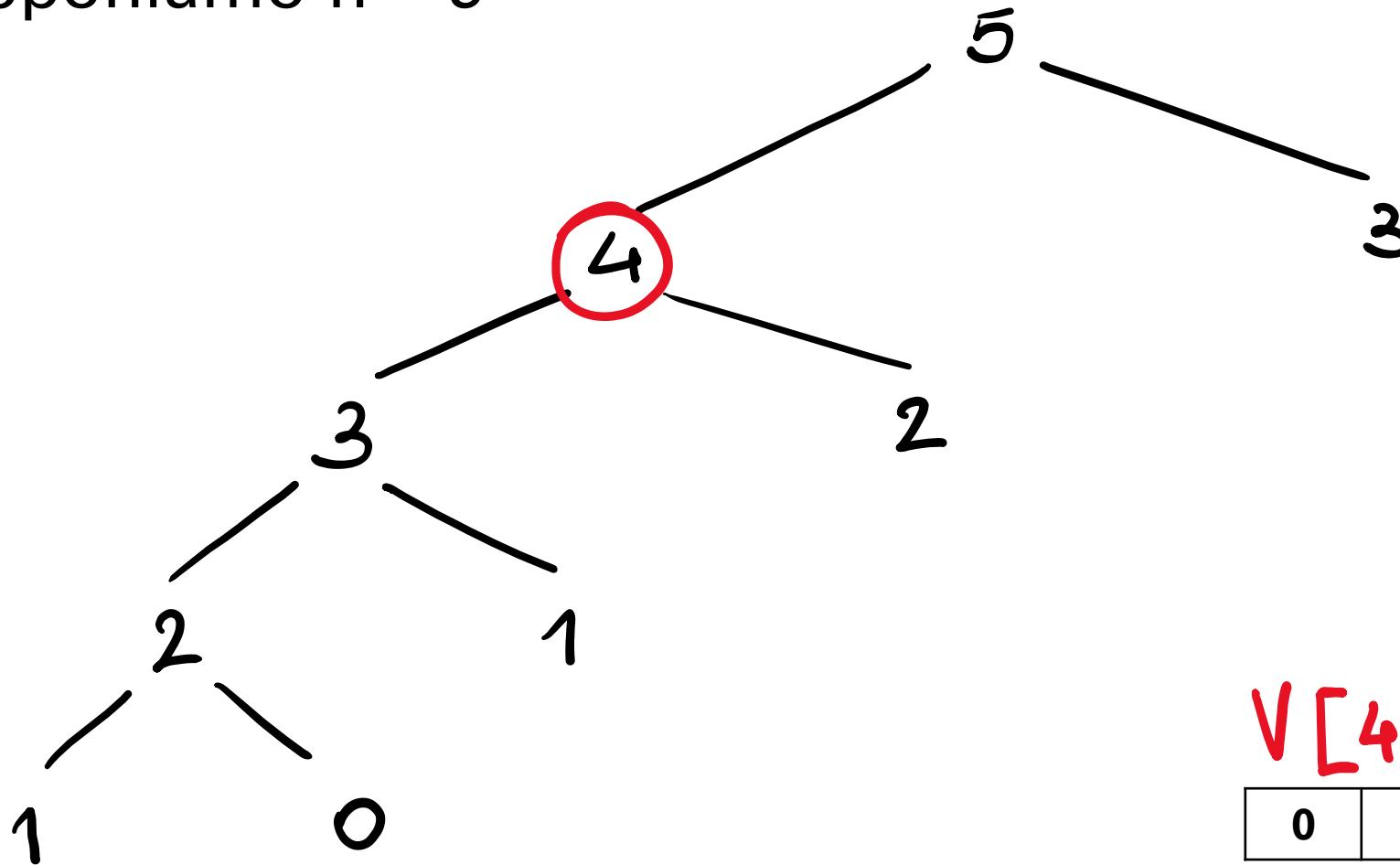


$$V[3] = V[2] + V[1]$$

0	1	1	2		
---	---	---	---	--	--

Programmazione Dinamica: Fibonacci

- Supponiamo $n = 5$

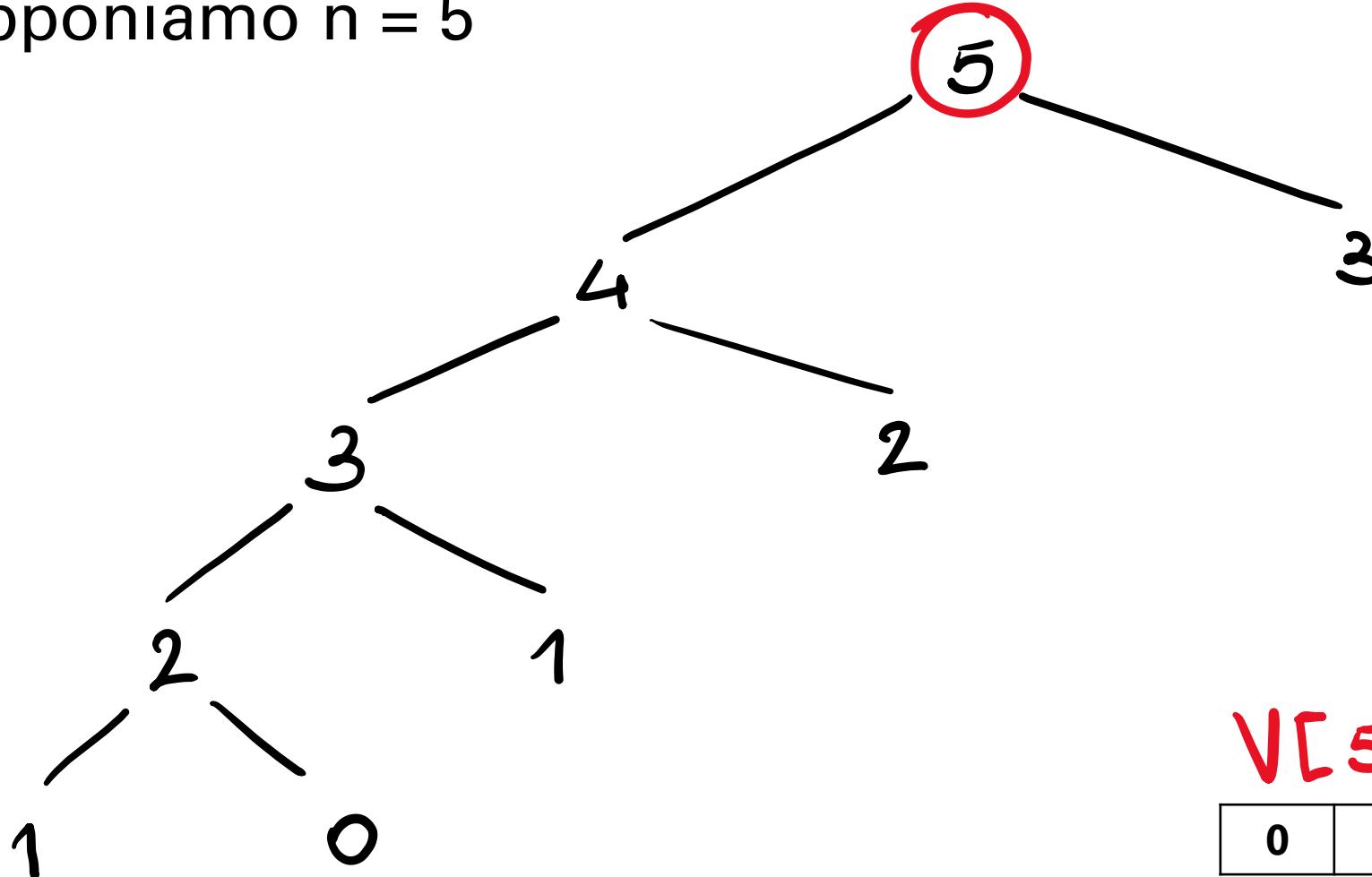


$$V[4] = V[3] + V[2]$$

0	1	1	2	3	
---	---	---	---	---	--

Programmazione Dinamica: Fibonacci

- Supponiamo $n = 5$

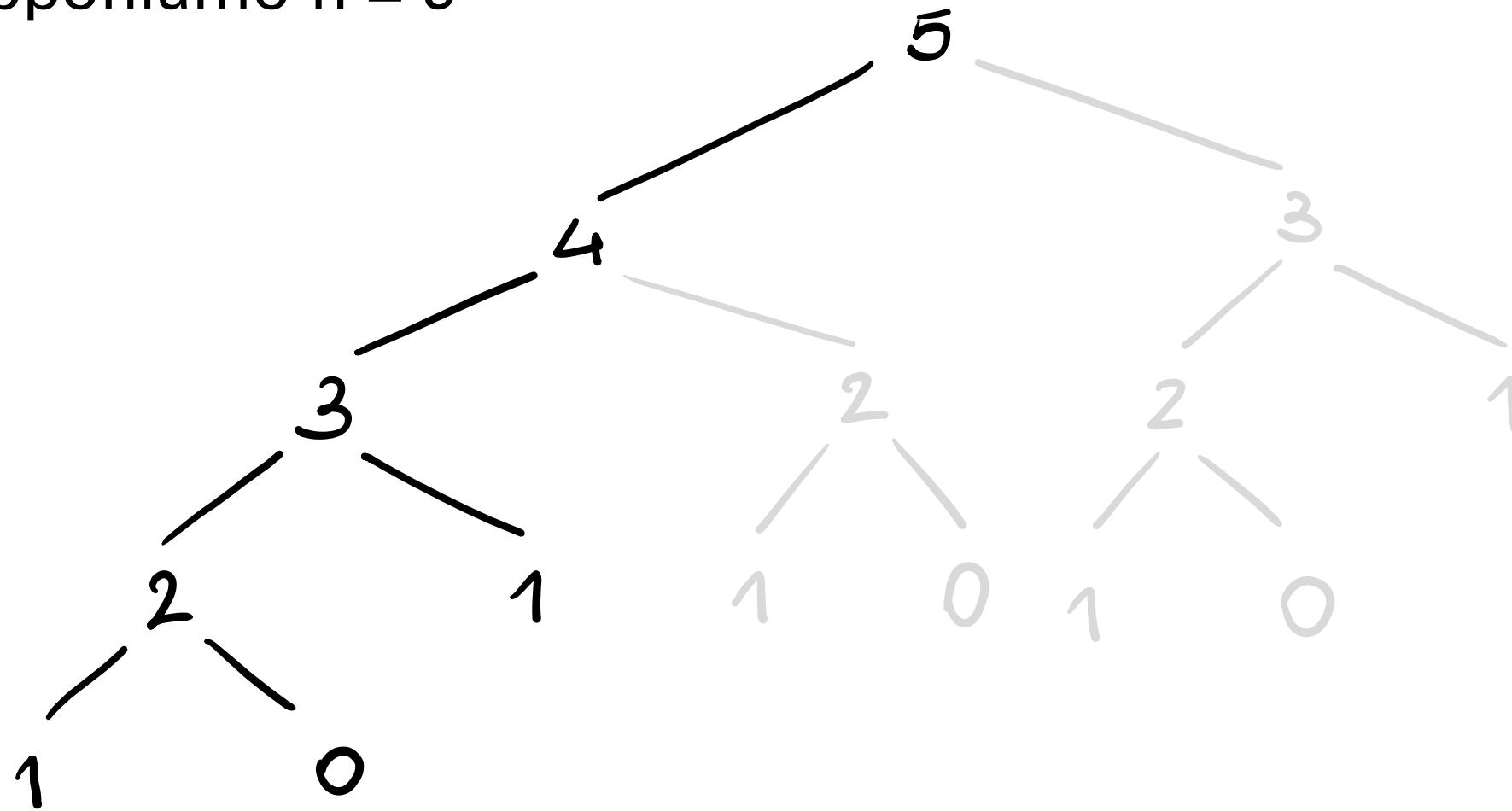


$$V[5] = V[4] + V[3]$$

0	1	1	2	3	5
---	---	---	---	---	---

Programmazione Dinamica: Fibonacci

- Supponiamo $n = 5$



Programmazione Dinamica: Fibonacci

```
int main(){
    vector<int> DP(N, -1);
    DP[0] = 0;
    DP[1] = 1;
    return fibonacci(N, DP);
}

int fibonacci(int n, vector<int>& DP){
    if(DP[n] == -1){
        DP[n] = fibonacci(n-1, DP) + fibonacci(n-2, DP);
    }
    return DP[n];
}
```

Programmazione Dinamica: Fibonacci

- Potevamo fare di meglio?

Programmazione Dinamica: Fibonacci

- Potevamo fare di meglio? Sì, anche se solamente a livello di memoria.

Programmazione Dinamica: Fibonacci

```
vector<int> DP(N, -1);
DP[0] = 0;
DP[1] = 1;
for(int i = 2; i < N; i++){
    DP[i] = DP[i-1] + DP[i-2];
}
return DP[N];
```

0	1				
---	---	--	--	--	--

$i = 2$

0	1	1			
---	---	---	--	--	--

$i = 3$

0	1	1	2		
---	---	---	---	--	--

$i = 4$

0	1	1	2	3	
---	---	---	---	---	--

$i = 5$

0	1	1	2	3	5
---	---	---	---	---	---

Programmazione Dinamica: Fibonacci

```
int a = 0;  
int b = 1;  
int tmp = 0;  
  
for(int i = 2; i < N; i++){  
    tmp = a + b;  
    a = b;  
    b = tmp;  
}  
  
return b;
```



Programmazione Dinamica

- Con il semplice esercizio di Fibonacci abbiamo scoperto esistere due approcci differenti:
 1. Programmazione Dinamica, metodo ITERATIVO (basato sull'uso di cicli) che risolve sistematicamente il problema partendo dai casi noti fino ad arrivare al risultato
 2. Memoization, metodo RICORSIVO che prevede l'inizializzazione di una struttura dati per potersi concentrare soltamente sui casi necessari per risolvere il problema.

Programmazione Dinamica

- Riassumendo
 - Prevede generalmente la creazione di una struttura DP che può avere n dimensioni (ovviamente più dimensioni significa più memoria e più difficoltà nella gestione del problema)
 - Utilizzata principalmente nei problemi di massimizzazione o minimizzazione
 - Ci si può aiutare scrivendo un ‘sistema’ di questo tipo

$$DP[i] = \begin{cases} 1 & n \leq 1 \\ DP[i-2] + DP[i-1] & n > 1 \end{cases}$$

Territoriali: Mostra

- [...] si sono create due lunghe file. Nella prima si sono assiepati **N turisti**, ciascuno con un **grado di preparazione V[i]**. Nella seconda si sono accalcati **M volontari** della scuola d'arte, che si offrono di guidare i turisti attraverso i reperti, ciascuno con un **grado di preparazione G[i]**.
- Per l'accesso alla mostra dei turisti sono disponibili due tipi di biglietti: il biglietto senza guida al costo di **1 euro**, e il biglietto con guida al costo di **2 euro**. I volontari invece non pagano mai. **La grande calca impedisce di riordinare le persone in fila, ma è comunque possibile decidere il modo con cui far entrare le persone:** individualmente da una delle due file, oppure abbinati dalle due file. È possibile abbinare un turista di preparazione V[i] con una guida di preparazione G[j] solo se la guida è più preparata del turista, ovvero se **G[j]>V[i]**.
- Quanto può guadagnare **al massimo** la mostra dai biglietti, decidendo con cura l'assegnamento di guide ai turisti?

Territoriali: Mostra

- Input: La prima riga del file di input contiene un intero T , il numero di casi di test. Seguono T casi di test, numerati da 1 a T . Ogni caso di test è preceduto da una riga vuota.
- Ciascun caso di test è composto da 3 righe:
 - La prima contiene i due numeri interi N ed M , rispettivamente il numero di turisti e il numero di volontari.
 - La riga successiva contiene gli N interi $V[i]$, i gradi di preparazione dei turisti.
 - La terza riga contiene gli M interi $G[i]$, i gradi di preparazione dei volontari.

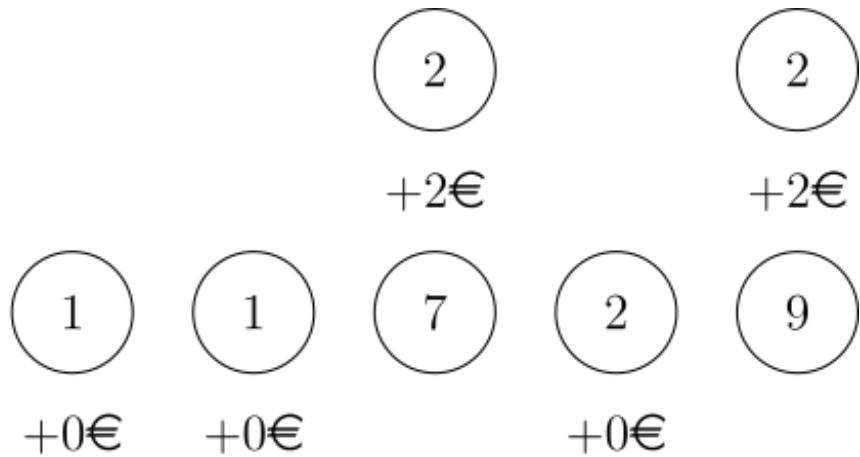
Territoriali: Mostra

1
2 5
2 2
1 1 7 2 9

T
N M
V[0] ... V[N-1]
G[0] ... G[M-1]

Visitatori

Guide



Territoriali: Mostra

- Idee?

Territoriali: Mostra

- Per poter massimizzare i profitti dovremmo ogni volta verificare quale è la scelta più vantaggiosa fra
 1. Far entrare un visitatore (+1)
 2. Far entrare una guida (+0)
 3. Far entrare un visitatore abbinato ad una guida se $G[j] > V[i]$ (+2)

Territoriali: Mostra

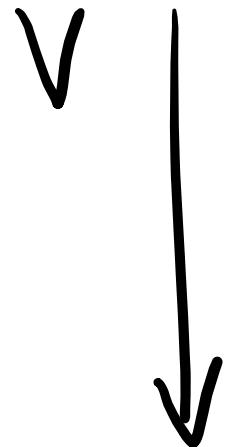
- Struttura

1

2 5

2 2

1 1 7 2 9



Territoriali: Mostra

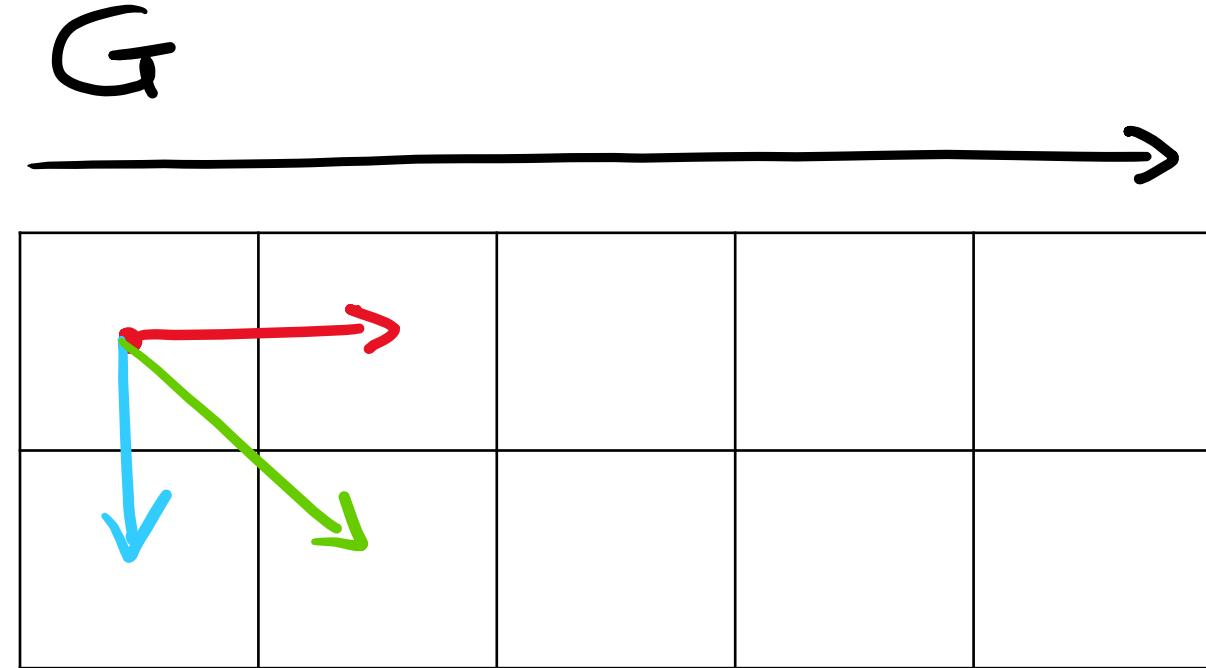
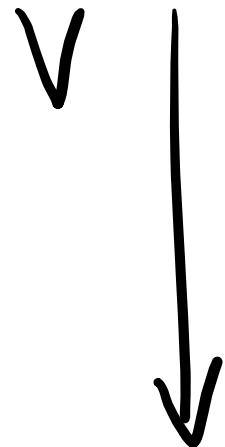
- Struttura

1

2 5

2 2

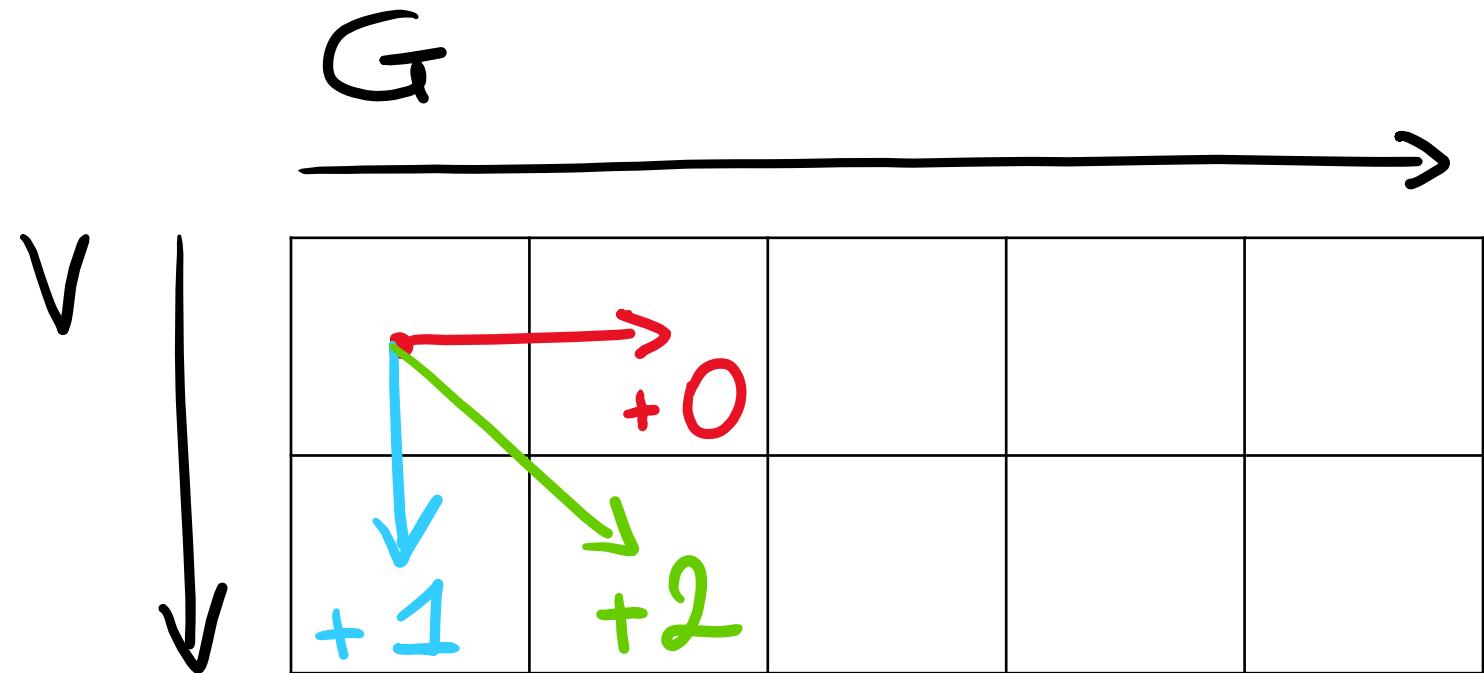
1 1 7 2 9



Territoriali: Mostra

- Struttura

1
2 5
2 2
1 1 7 2 9



Territoriali: Mostra

$$DP[i][j] = \begin{cases} \max(2 + DP[i+1][j+1], 1 + DP[i+1][j], DP[i][j+1]) & \text{Se } i < N \text{ and } j < N \text{ and } G[j] > V[i] \\ \max(1 + DP[i+1][j], DP[i][j+1]) & \text{Se } i < N \text{ and } j < N \\ 0 & \text{Se } i == N \\ N-i & \text{Se } j == M \end{cases}$$

Territoriali: Mostra

1	1	7	2	9	G
---	---	---	---	---	---

2					2
2					1
✓	○	○	○	○	○

```
int max_profit(vector<vector<int>>& DP, vector<int>& G, vector<int>& V, int i, int j, int N, int M){  
    if(j == M || i == N){  
        return DP[i][j];  
    } else {  
        if(DP[i][j] == -1){  
            if(G[j] > V[i]){  
                DP[i][j] = max(2 + max_profit(DP, G, V, i+1, j+1, N, M),  
                               max(max_profit(DP, G, V, i, j+1, N, M), 1 + max_profit(DP, G, V, i+1, j, N, M)));  
            } else {  
                DP[i][j] = max(1 + max_profit(DP, G, V, i+1, j, N, M), max_profit(DP, G, V, i, j+1, N, M));  
            }  
        }  
        return DP[i][j];  
    }  
}
```

```
//nel main  
...  
  
vector<vector<int>> DP(N+1, vector<int>(M+1, -1));  
for(int i = 0; i < N; i++){  
    DP[i][M] = N-i;  
}  
for(int j = 0; j <= M; j++){  
    DP[N][j] = 0;  
}  
  
out << "Case #" << i << ":" << max_profit(DP, G, V, 0, 0, N, M) << endl;
```

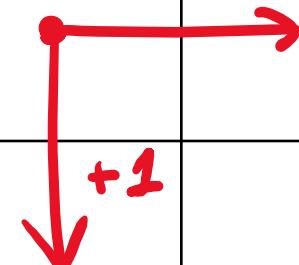
Territoriali: Mostra

1	1	7	2	9	G
---	---	---	---	---	---

2
2

✓

					2
+1					1
0	0	0	0	0	0



Territoriali: Mostra

1	1	7	2	9	G
---	---	---	---	---	---

2
2

✓

					2
					1
	0	0	0	0	0

+1

+1

Territoriali: Mostra

1	1	+	2	9	G
---	---	---	---	---	---

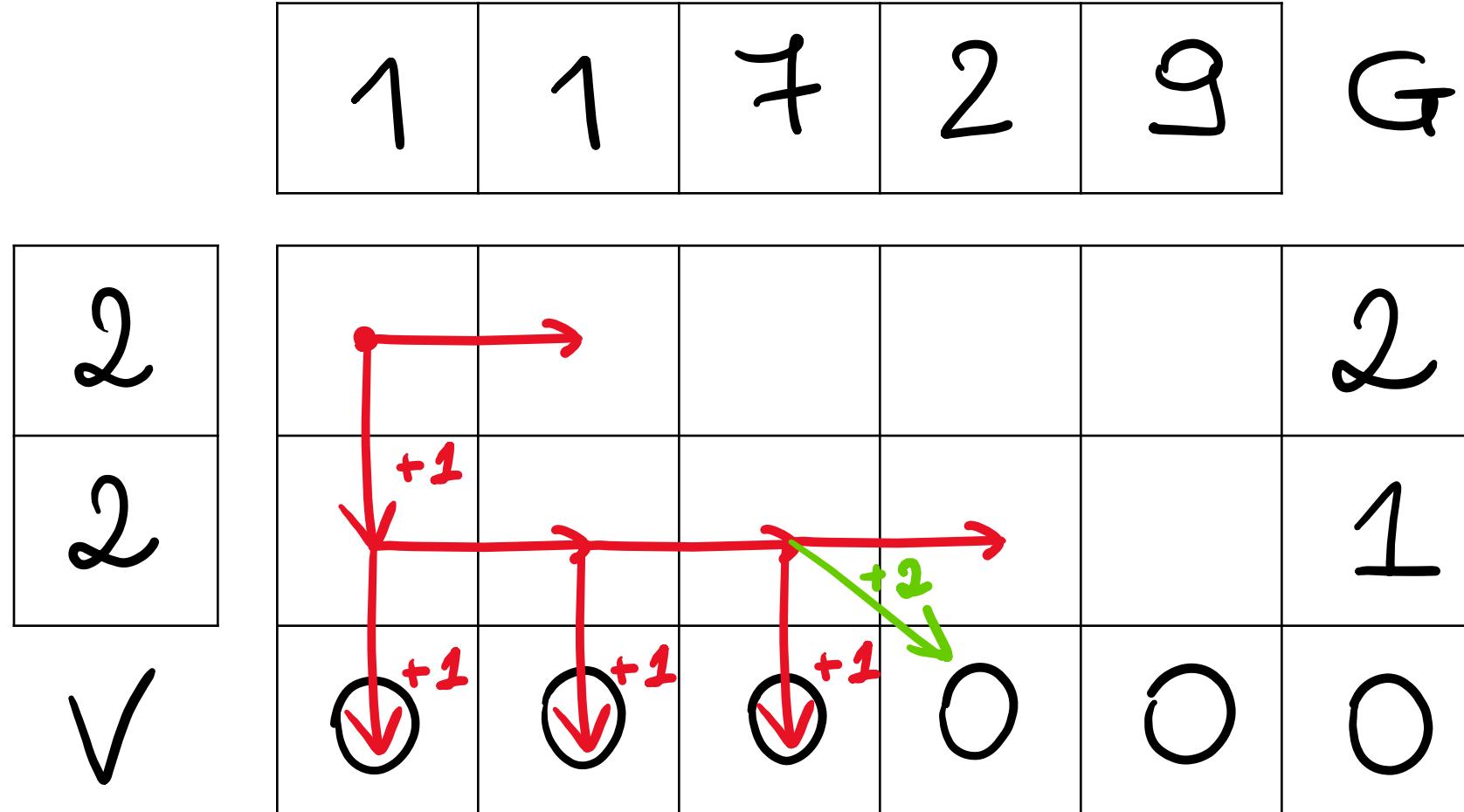
2
2

✓

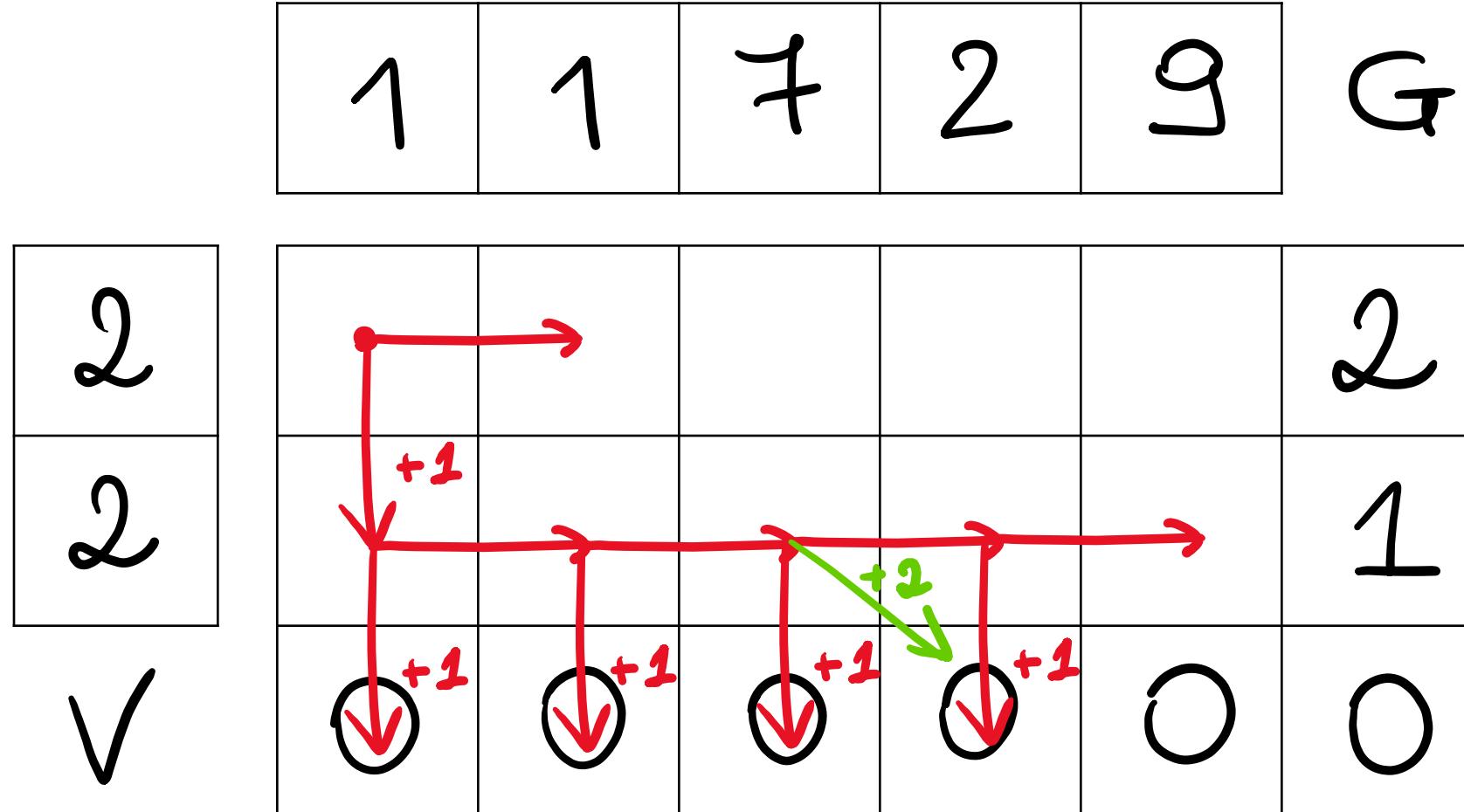
					2
					1
			0	0	0

The diagram illustrates a sequence of moves on a board. It starts with a red arrow pointing right from the first cell (2, 2). From the second cell, a vertical red arrow points down to a circled '0'. From this circled '0', a horizontal red arrow points right, passing over another circled '0'.

Territoriali: Mostra



Territoriali: Mostra

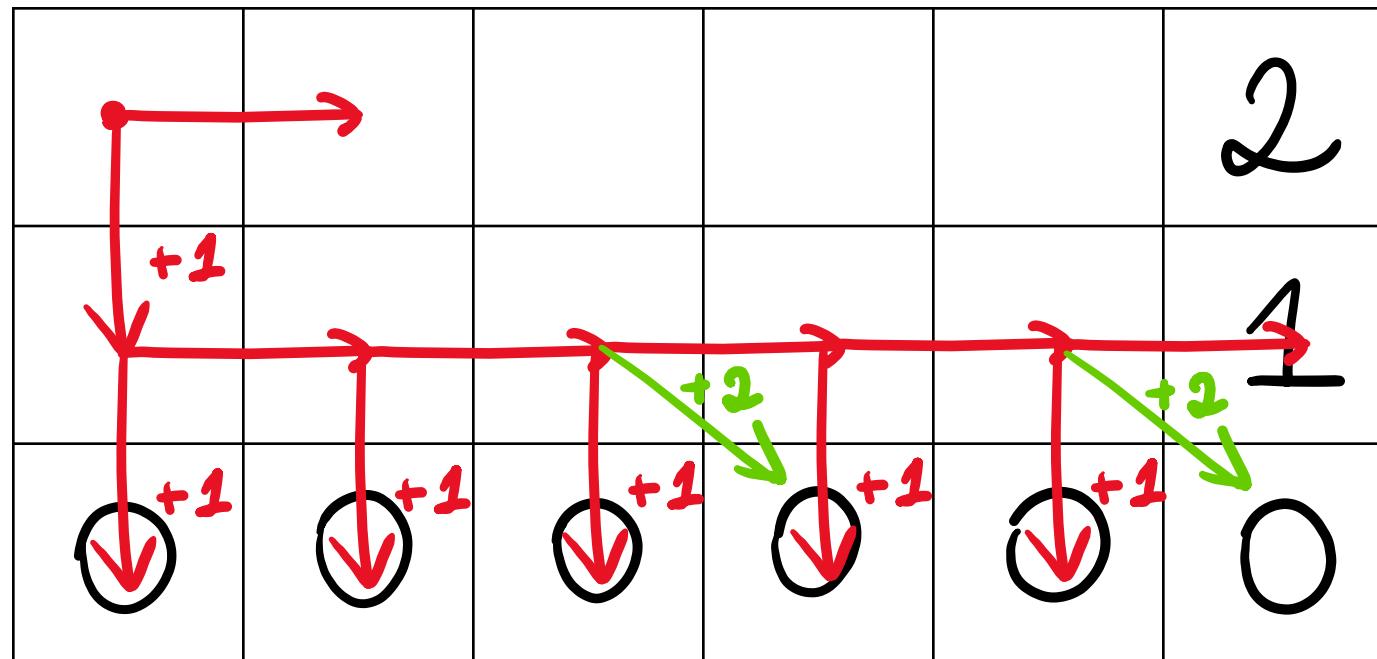


Territoriali: Mostra

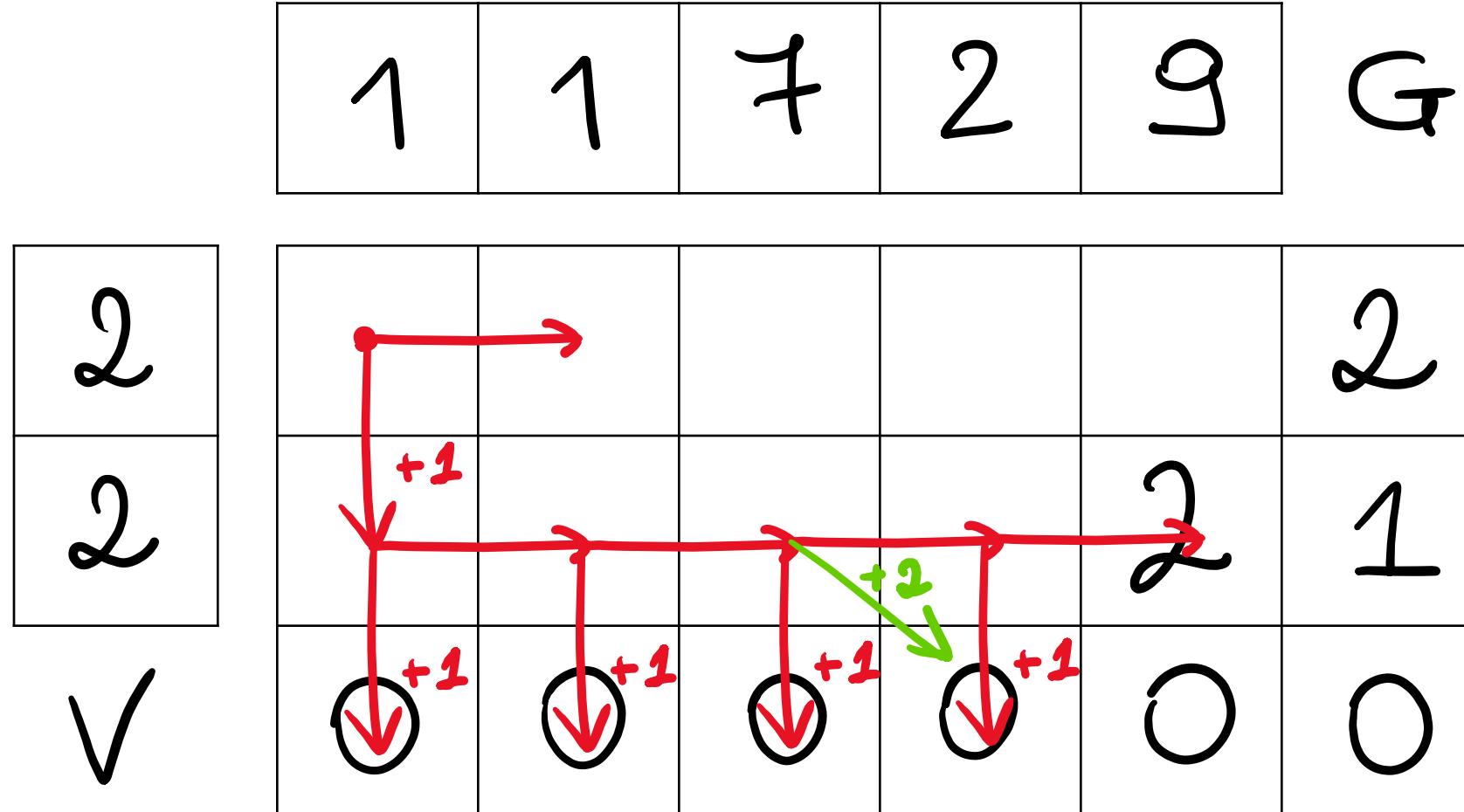
1	1	7	2	9	G
---	---	---	---	---	---

2
2

✓



Territoriali: Mostra



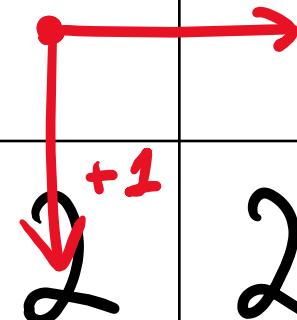
Territoriali: Mostra

1	1	7	2	9	G
---	---	---	---	---	---

2
2

✓

					2
2					1
2	2	2	2	2	0



Territoriali: Mostra

1	1	7	2	9	G
---	---	---	---	---	---

2
2

✓

0	0	0	0	0	0

The diagram illustrates a sequence of moves in a territorial game. A horizontal red line connects the top row's '1' and '7' to the second row's '2'. From the first '2', a red arrow labeled '+1' points down to the bottom row's '0'. From the '7', a red arrow labeled '+1' points down to the bottom row's '0'. From the second '2', a red arrow labeled '+1' points down to the bottom row's '0'. From the '9', a green arrow labeled '+2' points down to the bottom row's '0'. A black '2' is written next to the final '0'.

Territoriali: Mostra

1	1	7	2	9	G
---	---	---	---	---	---

2	4	4	4	3	3	2
2	2	2	2	2	2	1
✓	0	0	0	0	0	0

Territoriali: Antivirus

- Il nuovo sistema di gara delle Selezioni Territoriali funziona alla grande, ma Mojito non è così convinto... sembra infatti che la nota mascotte delle Olimpiadi abbia fiutato un **virus** nascosto fra i file inviati da un partecipante!
- **Conosciamo la lunghezza del virus e sappiamo che si ripete uguale nei quattro file** che abbiamo ricevuto, ma non sappiamo dove. Aiutaci ad individuare il virus!

Territoriali: Antivirus

- I quattro file F_1, F_2, F_3, F_4 sono dati in input, rappresentati come quattro stringhe di caratteri di lunghezza rispettivamente N_1, N_2, N_3, N_4 .
- Il virus è una stringa di caratteri V di lunghezza M . La lunghezza M è data in input, ma non si conosce il contenuto della stringa V del virus.
- Sappiamo con certezza che il virus V appare all'interno di tutti e quattro i file, come sottostringa di caratteri *consecutivi*. Sappiamo inoltre che *NON* ci sono altre sottostringhe consecutive di lunghezza M che si ripetono uguali in tutti e quattro i file.
- Le posizioni dei caratteri nelle stringhe sono numerati a partire da 0. Per ciascuno dei quattro file F_i , trova la posizione in cui è inserito il virus, ovvero la posizione dove appare il primo carattere del virus V all'interno della stringa F_i .

Territoriali: Antivirus

- Input: La prima riga del file di input contiene un intero T , il numero di casi di test. Seguono T casi di test, numerati da 1 a T . Ogni caso di test è preceduto da una riga vuota.
- In ciascun caso di test:
 - La prima riga contiene quattro interi, N_1 , N_2 , N_3 e N_4 , separati da uno spazio, che corrispondono alla lunghezza di ciascuno dei quattro file.
 - La seconda riga contiene un solo intero M , che corrisponde alla lunghezza del virus.
 - Le successive 4 righe contengono rispettivamente le quattro stringhe F_1 , F_2 , F_3 e F_4 .

Territoriali: Antivirus

1	T
8 12 10 7	N1 N2 N3 N4
4	M
ananasso	F1
associazione	F2
tassonomia	F3
massone	F4

**Nel primo caso d'esempio il virus è
asso: ananasso, associazione,
tassonomia, massone**

Territoriali: Antivirus

- Idee?

Territoriali: Antivirus

- Per poter comprendere meglio il problema osserviamo un esempio in ‘piccolo’.

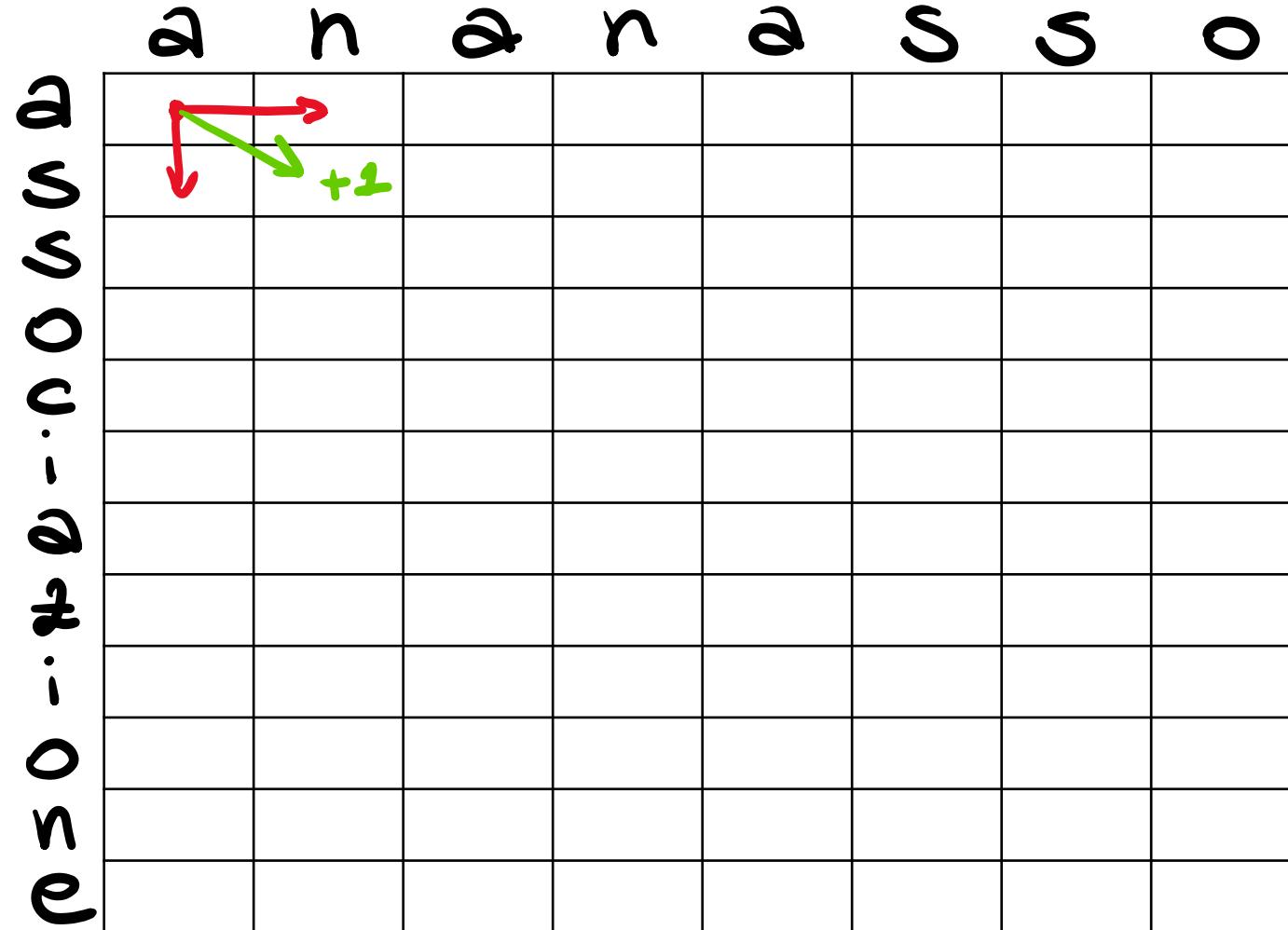
Territoriali: Antivirus

- Esempio:

a	n	q	r	a	s	s	o
s	v	o	u	-	d	-	e
s	o	c	o	-	t	-	e
o	u	n	o	-	e	-	n
u	o	n	u	-	n	-	e
n	o	u	o	-	e	-	n
o	u	o	u	-	n	-	e
u	o	u	o	-	e	-	n
o	u	o	u	-	n	-	e
e	o	u	o	-	n	-	e

Territoriali: Antivirus

- Esempio:



Territoriali: Antivirus

- Esempio:

assou - at - once

Territoriali: Antivirus

- Esempio:

as you do. - at. - one

Territoriali: Antivirus

- Esempio:

as you do. — That is one

Territoriali: Antivirus

- Esempio:

as seen on - at - 05e

Territoriali: Antivirus

- Esempio:

as you do. — That is one

Territoriali: Antivirus

- Esempio:

	a	n	a	n	a	s	s	o
a	0	0	0	0	0	0	0	0
s	0	1	0	1	0	1	0	0
s	0							
s	0							
s	0							
s	0							
s	0							
s	0							
s	0							
e	0							

Territoriali: Antivirus

- Esempio:

assou - a - n - e

Territoriali: Antivirus

- Esempio:

as you do. — at. — one

```
for (int i = 0; i <= N1; i++){
    for (int j = 0; j <= N2; j++){
        for (int k = 0; k <= N3; k++){
            for (int l = 0; l <= N4; l++){
                if (i == 0 || j == 0 || k == 0 || l == 0){
                    DP[i][j][k][l] = 0;
                }else if (F1[i-1] == F2[j-1] && F2[j-1] == F3[k-1] && F3[k-1] == F4[l-1]) {
                    DP[i][j][k][l] = DP[i-1][j-1][k-1][l-1] + 1;
                    if(DP[i][j][k][l] == M){
                        sol[0] = i - M;
                        sol[1] = j - M;
                        sol[2] = k - M;
                        sol[3] = l - M;
                        return;
                    }
                }
                else{
                    DP[i][j][k][l] = 0;
                }
            }
        }
    }
}
```