

ASD Laboratorio 03

The A(SD)-Team

UniTN

2026-09-29

CALENDARIO

24/09	Introduzione
22/10	Ad-hoc
29/10	Grafi 1
19/11	Grafi 2
26/11	Grafi 3
03/12	Lab Progetto 1

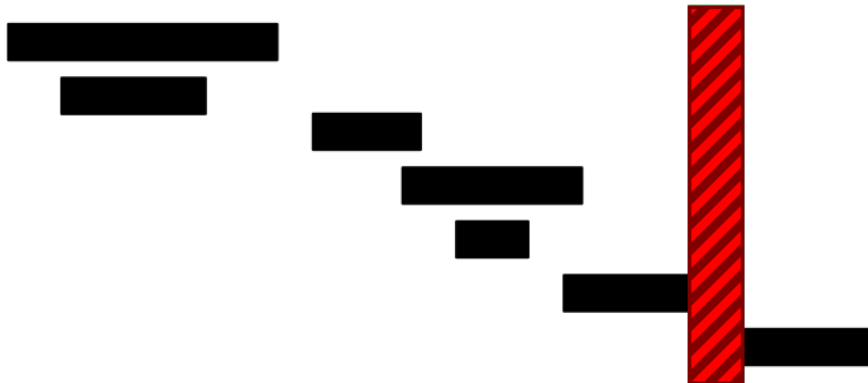
MERGE SORT

```
vector<int> merge_sort(vector<int>& arr) {  
    if(arr.size() <= 1)  
        return arr;  
  
    // Spezzo l'array in due parti  
    int m = arr.size() / 2;  
  
    vector<int> left(&arr[0], &arr[m]);  
    vector<int> right(&arr[m], &arr[arr.size()]);  
  
    // Ordino le due parti  
    left = merge_sort(left);  
    right = merge_sort(right);  
}
```

```
// Inserisco guardie
left.push_back(INT_MAX);
right.push_back(INT_MAX);

vector<int> result;
int l=0, r=0;
for(int i=0; i<arr.size(); i++){
    if(right[r]<left[l]){
        result.push_back(right[r]);
        r++;
    } else {
        result.push_back(left[l]);
        l++;
    }
}
return result;
}
```

Dati N intervalli $\{[Inizio_i, Fine_i)\}_{i=1}^N$, vogliamo sapere quale è il più lungo periodo non coperto da alcun intervallo, considerando solo gli istanti compresi fra il minimo istante di inizio ed il massimo istante di fine degli intervalli.



SOLUZIONI: INTERVALLI (I)

SIMULAZIONE $\mathcal{O}(NM)$

- 1 Se gli intervalli arrivano fino all'istante M , creare un array $A[1..M]$ di booleani.
- 2 Per ogni intervallo, segnare gli istanti coperti dall'intervallo.
- 3 Visitare l'array per trovare i periodi scoperti.

Esempio:

4
2 5
8 12
1 4
7 9

1	2	3	4	5	6	7	8	9	10	11	12

SOLUZIONI: INTERVALLI (I)

SIMULAZIONE $\mathcal{O}(NM)$

- 1 Se gli intervalli arrivano fino all'istante M , creare un array $A[1..M]$ di booleani.
- 2 Per ogni intervallo, segnare gli istanti coperti dall'intervallo.
- 3 Visitare l'array per trovare i periodi scoperti.

Esempio:

4
2 5
8 12
1 4
7 9

1	2	3	4	5	6	7	8	9	10	11	12

SOLUZIONI: INTERVALLI (I)

SIMULAZIONE $\mathcal{O}(NM)$

- 1 Se gli intervalli arrivano fino all'istante M , creare un array $A[1..M]$ di booleani.
- 2 Per ogni intervallo, segnare gli istanti coperti dall'intervallo.
- 3 Visitare l'array per trovare i periodi scoperti.

Esempio:

4	
2	5
8	12
1	4
7	9

[illegible]

SOLUZIONI: INTERVALLI (I)

SIMULAZIONE $\mathcal{O}(NM)$

- 1 Se gli intervalli arrivano fino all'istante M , creare un array $A[1..M]$ di booleani.
- 2 Per ogni intervallo, segnare gli istanti coperti dall'intervallo.
- 3 Visitare l'array per trovare i periodi scoperti.

Esempio:

4	
2	5
8	12
1	4
7	9

[illegible]

SOLUZIONI: INTERVALLI (I)

SIMULAZIONE $\mathcal{O}(NM)$

- 1 Se gli intervalli arrivano fino all'istante M , creare un array $A[1..M]$ di booleani.
- 2 Per ogni intervallo, segnare gli istanti coperti dall'intervallo.
- 3 Visitare l'array per trovare i periodi scoperti.

Esempio:

4	
2	5
8	12
1	4
7	9

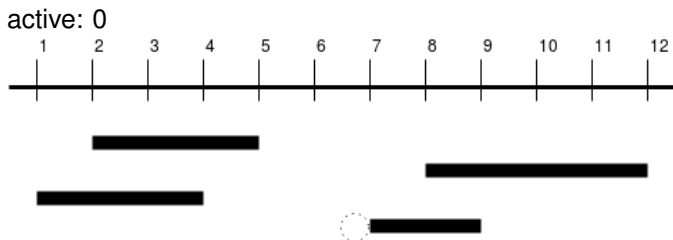
[illegible]

SOLUZIONI: INTERVALLI (II)

SIMULAZIONE EFFICIENTE $\mathcal{O}(N \log(N))$

- 1 Prendere tutti gli istanti di inizio e fine intervallo ed ordinarli tutti insieme, mantenendo in memoria quali si riferiscono ad un inizio intervallo e quali ad una fine intervallo.
- 2 Analizzare questi istanti mantenendo un contatore con il numero di intervalli attivi.
- 3 Se il contatore è uguale a 0, abbiamo trovato un buco.

Esempio:

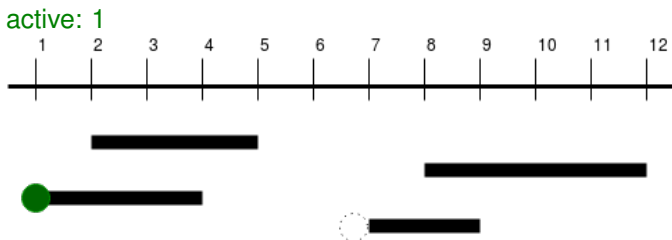


SOLUZIONI: INTERVALLI (II)

SIMULAZIONE EFFICIENTE $\mathcal{O}(N \log(N))$

- 1 Prendere tutti gli istanti di inizio e fine intervallo ed ordinarli tutti insieme, mantenendo in memoria quali si riferiscono ad un inizio intervallo e quali ad una fine intervallo.
- 2 Analizzare questi istanti mantenendo un contatore con il numero di intervalli attivi.
- 3 Se il contatore è uguale a 0, abbiamo trovato un buco.

Esempio:

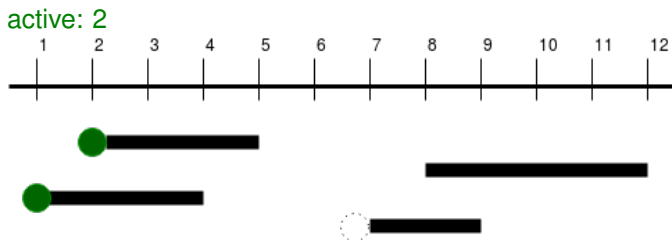


SOLUZIONI: INTERVALLI (II)

SIMULAZIONE EFFICIENTE $\mathcal{O}(N \log(N))$

- 1 Prendere tutti gli istanti di inizio e fine intervallo ed ordinarli tutti insieme, mantenendo in memoria quali si riferiscono ad un inizio intervallo e quali ad una fine intervallo.
- 2 Analizzare questi istanti mantenendo un contatore con il numero di intervalli attivi.
- 3 Se il contatore è uguale a 0, abbiamo trovato un buco.

Esempio:

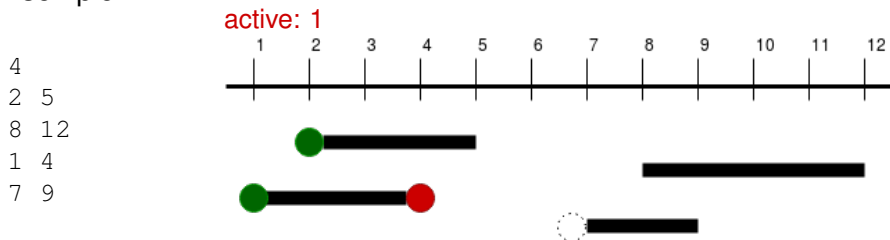


SOLUZIONI: INTERVALLI (II)

SIMULAZIONE EFFICIENTE $\mathcal{O}(N \log(N))$

- 1 Prendere tutti gli istanti di inizio e fine intervallo ed ordinarli tutti insieme, mantenendo in memoria quali si riferiscono ad un inizio intervallo e quali ad una fine intervallo.
- 2 Analizzare questi istanti mantenendo un contatore con il numero di intervalli attivi.
- 3 Se il contatore è uguale a 0, abbiamo trovato un buco.

Esempio:

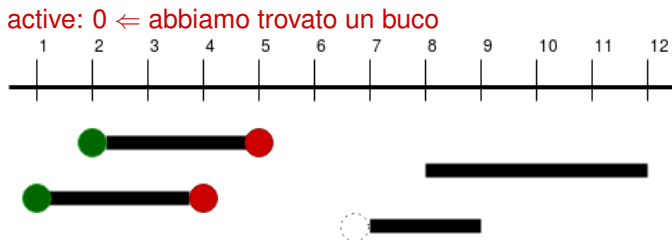


SOLUZIONI: INTERVALLI (II)

SIMULAZIONE EFFICIENTE $\mathcal{O}(N \log(N))$

- 1 Prendere tutti gli istanti di inizio e fine intervallo ed ordinarli tutti insieme, mantenendo in memoria quali si riferiscono ad un inizio intervallo e quali ad una fine intervallo.
- 2 Analizzare questi istanti mantenendo un contatore con il numero di intervalli attivi.
- 3 Se il contatore è uguale a 0, abbiamo trovato un buco.

Esempio:

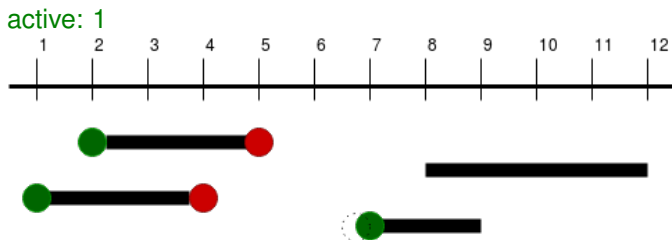


SOLUZIONI: INTERVALLI (II)

SIMULAZIONE EFFICIENTE $\mathcal{O}(N \log(N))$

- 1 Prendere tutti gli istanti di inizio e fine intervallo ed ordinarli tutti insieme, mantenendo in memoria quali si riferiscono ad un inizio intervallo e quali ad una fine intervallo.
- 2 Analizzare questi istanti mantenendo un contatore con il numero di intervalli attivi.
- 3 Se il contatore è uguale a 0, abbiamo trovato un buco.

Esempio:

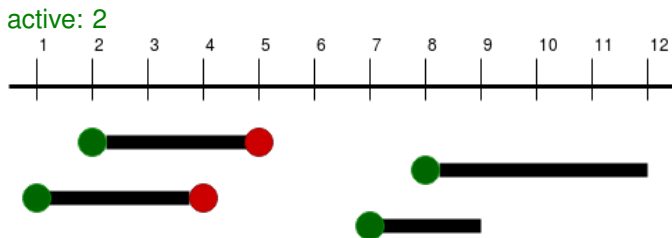


SOLUZIONI: INTERVALLI (II)

SIMULAZIONE EFFICIENTE $\mathcal{O}(N \log(N))$

- 1 Prendere tutti gli istanti di inizio e fine intervallo ed ordinarli tutti insieme, mantenendo in memoria quali si riferiscono ad un inizio intervallo e quali ad una fine intervallo.
- 2 Analizzare questi istanti mantenendo un contatore con il numero di intervalli attivi.
- 3 Se il contatore è uguale a 0, abbiamo trovato un buco.

Esempio:

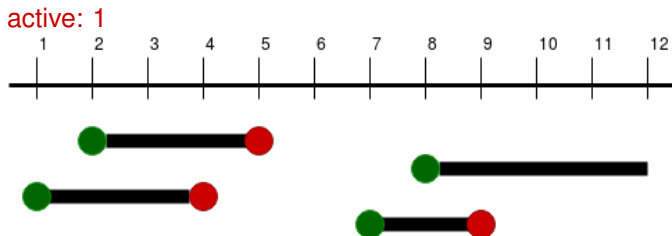


SOLUZIONI: INTERVALLI (II)

SIMULAZIONE EFFICIENTE $\mathcal{O}(N \log(N))$

- 1 Prendere tutti gli istanti di inizio e fine intervallo ed ordinarli tutti insieme, mantenendo in memoria quali si riferiscono ad un inizio intervallo e quali ad una fine intervallo.
- 2 Analizzare questi istanti mantenendo un contatore con il numero di intervalli attivi.
- 3 Se il contatore è uguale a 0, abbiamo trovato un buco.

Esempio:

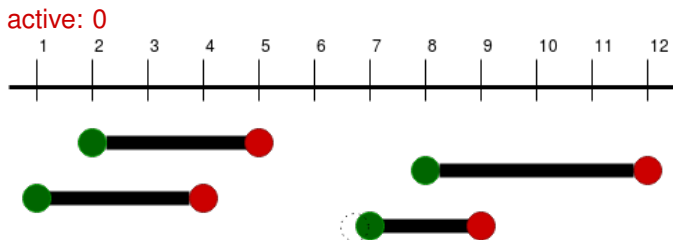


SOLUZIONI: INTERVALLI (II)

SIMULAZIONE EFFICIENTE $\mathcal{O}(N \log(N))$

- 1 Prendere tutti gli istanti di inizio e fine intervallo ed ordinarli tutti insieme, mantenendo in memoria quali si riferiscono ad un inizio intervallo e quali ad una fine intervallo.
- 2 Analizzare questi istanti mantenendo un contatore con il numero di intervalli attivi.
- 3 Se il contatore è uguale a 0, abbiamo trovato un buco.

Esempio:

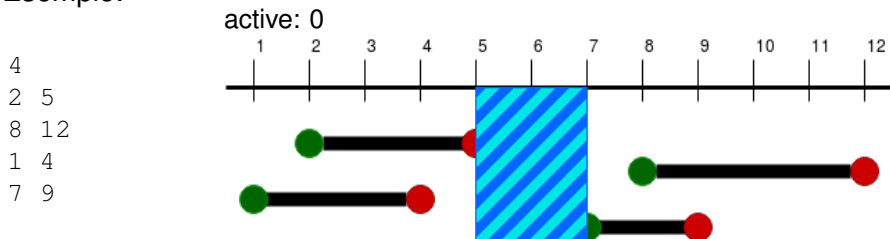


SOLUZIONI: INTERVALLI (II)

SIMULAZIONE EFFICIENTE $\mathcal{O}(N \log(N))$

- 1 Prendere tutti gli istanti di inizio e fine intervallo ed ordinarli tutti insieme, mantenendo in memoria quali si riferiscono ad un inizio intervallo e quali ad una fine intervallo.
- 2 Analizzare questi istanti mantenendo un contatore con il numero di intervalli attivi.
- 3 Se il contatore è uguale a 0, abbiamo trovato un buco.

Esempio:



SOLUZIONI: INTERVALLI (III)

SOLUZIONE EFFICIENTE $\mathcal{O}(N \log(N))$

- 1 Ordiniamo gli intervalli per istante di inizio.
- 2 L'intervallo i termina un periodo scoperto se $start[i] > end[j]$ per ogni $j < i$.
- 3 Visitiamo gli intervalli in ordine mantenendo in memoria il massimo degli istanti di fine.

Algorithm 1 LongestHole

```
// intervals può essere un const vector<pair<int,int>>&
1: function LONGESTHOLE(intervals[], N)
2:   sort<start(intervals)           // intervals è ordinato per istanti di inizio
3:    $R \leftarrow \text{intervals}[1].\text{end}$ 
4:    $s_h \leftarrow -1$                                // Inizio del buco
5:    $e_h \leftarrow -1$                                // Fine del buco
6:   for  $i = 2 \rightarrow N$  do                       //  $N = \text{intervals.size}()$ 
7:     if  $(\text{intervals}[i].\text{start} > R) \wedge (\text{intervals}[i].\text{start} - R > e_h - s_h)$  then
      // Periodo scoperto da  $R$  a  $\text{intervals}[i].\text{start}$ ,
      // più lungo del migliore
8:        $s_h \leftarrow R$ 
9:        $e_h \leftarrow \text{intervals}[i].\text{start}$ 
10:    end if
11:     $R \leftarrow \max(R, \text{intervals}[i].\text{end})$ 
12:  end for
13:  return  $(s_h, e_h)$ 
14: end function
```

FUNZIONE CEIL: $\text{CEIL}(I)$

Sia V un vettore contenente N_v interi **distinti** $[v_j]_{j=1}^{N_v}$. Sia A un vettore **ordinato in modo crescente**, contenente N interi positivi $[a_i]_{i=1}^N$, nei quali possono comparire valori ripetuti. Scrivere un algoritmo che, per ogni $v_j \in V$, restituisca il corrispondente c_j , definito come il più piccolo elemento di A strettamente maggiore di v_j , oppure -1 se tale elemento non esiste.

OSSERVAZIONE

Nel caso in cui di un solo valore v ($N_v = 1$), la complessità dell'algoritmo di ricerca binaria $\mathcal{O}(\log N)$ è dominata dalla lettura dell'input $\mathcal{O}(N)$.

FUNZIONE CEIL: `CEIL` (II)

RICERCA BINARIA

Per ogni $v \in V$, utilizziamo un algoritmo di **ricerca binaria**.

Funzione **`upper_bound`** (<algorithm>) nelle STL:

```
int ceil(vector<int> const& vec, int value) {  
    auto const it = upper_bound(vec.begin(), vec.end(),  
                                value);  
    if (it == vec.end()) { return -1; }  
    return *it;  
}
```

COMPLESSITÀ

N (lettura A) + N_v (lettura V) + $N_v \log(N)$ (ricerca binaria su A ($\log N$) per N_v volte)

FUNZIONE CEIL: `CEIL` (III)

ORDINAMENTO

Ordiniamo i valori di V e li confrontiamo progressivamente con A .

Esempio:

4 10
11 19 2 6
1 1 2 2 5 5 7 7 9 9

- 1 Ordiniamo i valori di V , tenendo gli indici delle posizioni originali.

11	19	2	6
1	2	3	4

sort →

2	6	11	19
3	4	1	2

FUNZIONE CEIL: `CEIL` (IV)

- 2 Inizializziamo un vettore dove salveremo i risultati con il valore -1 .

-1	-1	-1	-1
1	2	3	4

- 3 Utilizzando due indici confrontiamo i valori di V e A , avanziamo lungo A finche troviamo un valore $a_j > v_i$.



2	6	11	19
3	4	1	2



2	6	11	19
3	4	1	2



1	1	2	2	5	5	7	7	9	9
---	---	---	---	---	---	---	---	---	---



1	1	2	2	5	5	7	7	9	9
---	---	---	---	---	---	---	---	---	---

FUNZIONE CEIL: `CEIL` (IV)

- ④ Quando troviamo un valore $a_j > v_i$, salviamo il valore a_j nella posizione i nell'array dei risultati. Avanziamo con l'indice sul vettore V .

-1	-1	5	7
1	2	3	4



2	6	11	19
3	4	1	2



1	1	2	2	5	5	7	7	9	9
---	---	---	---	---	---	---	---	---	---

FUNZIONE CEIL: $\text{CEIL}(V)$

5 Ripetiamo il procedimento per i successivi valori di V_s .



2	6	11	19
3	4	1	2



1	1	2	2	5	5	7	7	9	9
---	---	---	---	---	---	---	---	---	---

6 Quando arriviamo alla fine del vettore A il vettore dei risultati è completo.



2	6	11	19
3	4	1	2



1	1	2	2	5	5	7	7	9	9	
---	---	---	---	---	---	---	---	---	---	--

-1	-1	5	7
1	2	3	4

FUNZIONE CEIL: `CEIL` (VI)

COMPLESSITÀ

N (lettura A) + N_v (lettura V) + $N_v \log(N_v)$ (ordinamento di V) + $N + N_v$ (scorrimento di A e V)

FUNZIONE CEIL: CEIL (VI)

COMPLESSITÀ

La complessità dell'algoritmo dipende sia dalla lunghezza di A (N) che di V (N_v).

Algoritmo:	Ricerca binaria	Ordinamento
Complessità:	$N_v \log(N) + N + N_v$	$N_v \log(N_v) + 2N + 2N_v$
$N \ll N_v = M$	M	$M \log(M)$
$N \sim N_v = M$	$M \log(M)$	$M \log(M)$
$M = N \gg N_v$	M	M

ORDINAMENTO PESATO: SORT PESATO (I)

Dato un array di N interi da ordinare. Gli elementi sono precisamente tutti gli interi fra 1 e N . Ad ogni turno potete scambiare due elementi a scelta dell'array. Per fare ciò, pagate un prezzo pari alla somma dei due elementi.

ESEMPIO

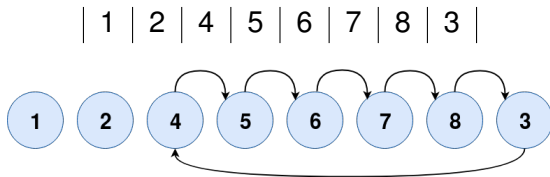
Per scambiare di posto l'elemento 3 e l'elemento 4 impiegate un turno e pagate 7.

Due problemi:

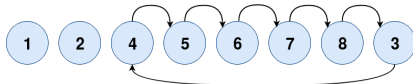
- individuare il metodo più veloce, che ottimizza il numero di turni;
- individuare il metodo più economico, che ottimizza il prezzo;

SOLUZIONE SORT PESATO: SORTPESATO (I)

- Ogni elemento deve finire in una certa posizione.
- Possiamo vederli come archi in un grafo da n nodi e n archi.
- Individuiamo i cicli e valutiamo separatamente.
 - ▶ Il numero minimo di mosse per un ciclo di n elementi è $n - 1$.
 - ▶ La soluzione ovvia sposta sempre l'elemento più piccolo con gli altri elementi.
 - ▶ Può convenire prendere in prestito l'elemento 1, scambiandolo con l'elemento più piccolo del ciclo, ordinare il resto del ciclo e rimettere a posto l'elemento 1.



SOLUZIONE SORT PESATO: SORTPESATO (II)



Possibili sequenze di scambi:

- scambiando l'elemento più piccolo, m :

$$\underbrace{(3 + 8) + (3 + 7) + (3 + 6) + (3 + 5) + (3 + 4)}_{n-1 \text{ scambi}} = 45$$

$$\rightarrow C_m = (n - 1) \cdot m + S - m$$

- scambiando con 1:

$$\begin{aligned} & (1 + 3) \\ & + \underbrace{(1 + 8) + (1 + 7) + (1 + 6) + (1 + 5) + (1 + 4)}_{n-1 \text{ scambi}} \\ & + (3 + 1) \\ & = 43 \end{aligned}$$

$$\rightarrow C_1 = 2 \cdot (1 + m) + (n - 1) + S - m$$

nota: $S = \sum_i n_i$, in questo esempio $[m = 3, n = 6, S = 33]$

Dato un grafo **DIRETTO** $G(V, E)$, esso può essere rappresentato in vari modi¹:

- 1 Liste di adiacenza: per ogni nodo $v \in V$, lista dei vicini;
- 2 Matrice di adiacenza: $M[v][w] = 1$ se esiste l'arco $(v, w) \in E$, 0 altrimenti;
- 3 Lista di archi: lista degli archi $(v, w) \in E$

⇒ se il grafo è **INDIRETTO** per ogni (v, w) aggiungiamo anche (w, v) .

Esempi scaricabili dal sito: `https:`

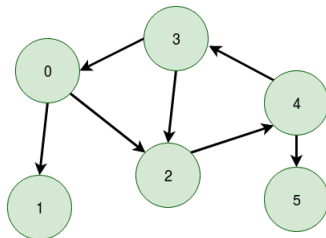
`//judge.science.unitn.it/slides/asd20/graphs.zip.`

¹([link di approfondimento](#))

FORMATO DI INPUT: ESEMPIO (I)

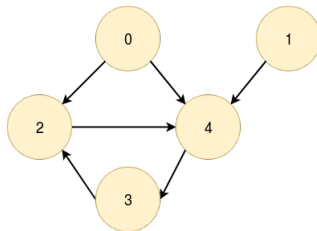
- riga 1: due interi N e M , rispettivamente il numero di nodi e di archi;
- righe 2 ... $M+1$: due interi s_i, t_i che denotano l'arco (s_i, t_i) . Eventualmente possono esserci altri attributi, per esempio un peso w_i .

```
6 7
0 1
0 2
4 3
4 5
3 2
2 4
3 0
```

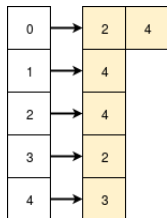


FORMATO DI INPUT: ESEMPIO (II)

5 6
0 2
0 4
1 4
3 2
2 4
4 3



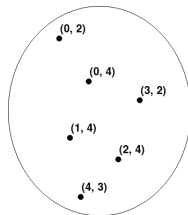
Liste di adiacenza:



Matrice di adiacenza:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Lista di archi:



LISTE DI ADIACENZA

ARRAY DI VECTOR

Lista di adiacenza implementata come array di vector o vector di vector. Se servono altre variabili (open, close) si creano altri array.

```
vector< vector <int> > adj;  
vector<bool> visitato;
```

ARRAY DI STRUCT

Struct nodo contenente lista di adiacenza e altre informazioni.

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

LISTE DI ADIACENZA

```
struct nodo{  
    vector<int> vic;  
    bool visitato= false;  
};  
vector<nodo> grafo;  
...  
in >> N;  
grafo.resize(N);  
for(int i=0;i<M;i++){  
    int from, to;  
    in >> from >> to;  
    grafo[from].vic.push_back(to);  
}
```

PROBLEMI

VISITA DI UN ALBERO

Dato un albero T radicato in R , visitare l'albero in pre-order, in-order e post-order. Questo task ha 20 casi di test, di cui 6 sono alberi binari.

VISITA DI GRAFO ORIENTATO

Dato un grafo ed un nodo di partenza, trovare il numero di nodi raggiungibili a partire da quel nodo.

DIAMETRO SU GRAFO NON ORIENTATO

Dato un grafo non orientato trovare il cammino minimo di lunghezza massima tra ogni coppia di nodi del grafo, ovvero il più lungo percorso minimo fra due nodi.

PRIMO PROGETTO A. A. 2016/2017

Pokémon Park