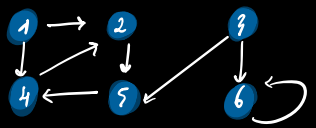
1) SCRIVERE LA MATRICE DI ADIACENZA DEL SEGUENTE GRAFO:



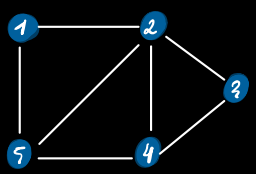
Variabile utilizzata: $\forall (i,j) \in V \times V$

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$

$A =$

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

2) SCRIVERE LA LISTA DI ADIACENZA DEL GRAFO NON ORIENTATO:



1 • → 2 → 5
2 • → 1 → 5 → 4 → 3
3 • → 2 → 4
4 • → 3 → 2 → 5
5 • → 4 → 2 → 1

* SE FACCIAMO LA MATRICE DI ADIACENZA ESCE COME?
SIMMETRICA RISPETTO ALLA DIAGONALE.

ESPLORAZIONE IN AMPIEZZA

BFS (Breadth-first search): esplora un grafo a partire da un v. sorgente. Esplora prima i "vicini" e poi i "vicini dei figli".

* COSA FA L'ALGORITMO:

1) SCOPRE I VERTICI CHE SONO RAGGIUNGIBILI DA s.

RAGGIUNGIBILE = v è raggiungibile da u sse ∃ cammino da u a v.

CAMMINO = è una sequenza finita di vertici $u_0, u_1 \dots u_k$ dove $u_0 = u$, $u_k = v$ e $\forall i \in \{0 \dots k-1\}$ $(u_i, u_{i+1}) \in E$.
↓ insieme archi

2) Calcola $\forall v \in V$, la distanza di quel vertice DALLA SORGENTE. (numero minimo di archi).

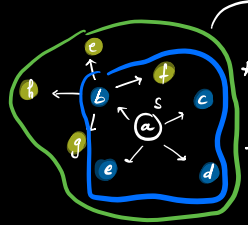
DISTANZA: # di archi su un cammino minimo da u a v.

- PRINCIPIO DI FUNZIONAMENTO:

3) Amplia la frontiera tra vertici scoperti e non scoperti in modo tale che prima scopre i vertici a distanza 0 da s, poi:

scopre i vertici a distanza 1 da s.
" " " 2 da s.
" " " 3 da s.
. . . . . .

→ FRONTIERA CHE SI ALLARGA AD OGNI NUOVA SCOPERTA.



* ESPLORO PRIMA I VICINI, POI UNO ALLA VOLTA
I "VICINI DEI VICINI". → uno alla volta.

4) Genera un albero con una radice s contenente tutti i vertici raggiungibili da s, tramite i valori $*.\pi$ al termine dell'esecuzione. ↓

ISTANZA: $G(V,E)$, $s \in V$

* grafo orientato o non
* un suo vertice

→ **LOGICA DI ESPANSIONE DEI GRAFI IN AMPIEZZA:**

* nodi scoperti direttamente da s → $d=1$, $\pi = s$
* nodi figli dei figli diretti di s → $d=2$, $\pi = (s).next()$
* figli dei figli di s → $d = n$, $(\pi) = (s).next().next()$ .... (u);

↳ con liste / matrici di adiacenza diventa MOLTO SEMPLICE.

*ALGORITMO:

BFS (G,s): $G(V,E)$ $s \in V$

for $u \in V \setminus \{s\}$

```
u.col = "WHITE"
u.d   = ∞
u.π   = NIL
```

} inizializzazione!

Enqueue (Q, s);

```
s.col = "gray"
s.d   = 0 ;
s.π   = ∅ ;
```

→ inizializzazione del nodo sorgente.

→ **INFORMAZIONI AGGIUNTIVE:**

- **c** → colore, dello "stato di esplorazione" del nodo.
- **d** → distanza dalla sorgente (come definita prima)
- **π** → "predecessore", da "chi era stato scoperto."

* è il "parent" del vertice corrente.

Occhio! I non collegati al nodo s (componenti indipendenti) **NON VENGONO ESPLORATI.**

while $Q \neq \emptyset$ ↳ coda dei vertici del grafo a distanza d

```
u = Dequeue(Q)   // Estraggo 1 vertice dalla frontiera corrente.
for v ∈ Adj[u] {   // Espando, per quel vertice, di +1 la distanza di frontiera.
    IF ( v.color == "white" ):
        v.color = "gray"
        v.π     = u ;
        v.d     = u.d + 1 ;
    Enqueue(Q,v) → Lo metto in coda! Cioè, "è grigio" e ne devo ancora esplorare gli adiacenti.
u.color = "Black" ;
    ↳ Ho esplorato TOTALMENTE l'ampiezza di un nodo.
}
```

Aggiorno il vertice appena scoperta.

ALBERO GENERATO DA s:

$G(V,E)$ se V ALBERO → grafo non orientato connesso e aciclico

ALBERO BFS:

$G_\pi = (V_\pi, E_\pi)$ $V_\pi: \{ u \in V \mid u.d \neq \infty \} \cup \{s\}$ → POSSO SCRIVERLA IN ALTRI 2 MODI → * con colore. $\{ n \in V \mid n.col = \text{"BLACK"} \}$ * con parent. $\{...\}$

$E_\pi: \{ (u.\pi, u) \mid u \in V_\pi \setminus \{s\} \}$

ALGORITMO FUNZIONA BENE ANCHE SENZA I COLORI! (uso $0 \dots \infty$, per le distanze).

For $u \in V \setminus \{s\}$

$u.d = \infty$

$s.d = \phi$

$Q := \emptyset$

ENQUEUE (Q, s)

While $Q \neq \emptyset$

$u =$ DEQUEUE (Q)

For $u \in Adj[u]$

IF $(u.d == \infty)$

ENQUEUE (Q, u)

$u.d = u.d + 1$

COSÌ CONOSCO I VERTICI RAGGIUNGIBILI CON CAMMINO MINIMO.

↳ Conosco la distanza.

Riassumendo:

* Inizializzazione ⟶ Ogni nodo in più, aggiunge → colore, → parent, → distanza dalla sorgente

→ Viene utilizzata una queue come struttura dati di supporto.

Contiene → "vertici da visitare" (in ampiezza)