



UNIVERSITÀ DEGLI STUDI DI PARMA

Corso di Laurea in  
"Ingegneria dei Sistemi Informativi"

# **Programmazione di Applicazioni Software**

**Strutture dati dinamiche non lineari**

**Andrea Prati**

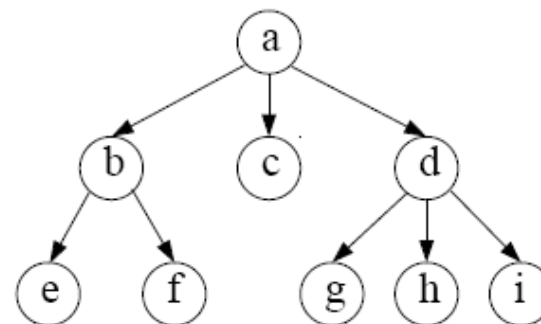
## Strutture dati dinamiche non lineari

- una struttura dati si definisce **dinamica** se permette di rappresentare insiemi dinamici la cui **cardinalità varia** durante l'esecuzione del programma
- strutture dinamiche non lineari
  - **alberi**
    - **alberi binari**
  - **grafi**

# **STRUTTURA DATI DINAMICA NON LINEARE: ALBERO**

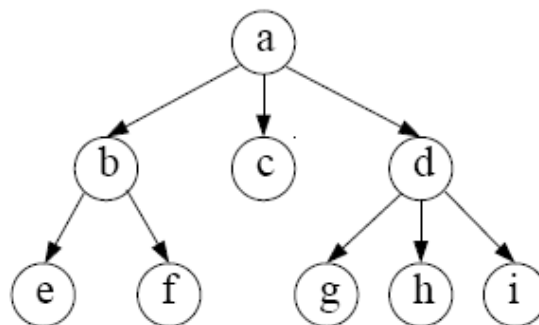
## Albero - definizione

- si dice albero una **tripla**  $T = (N, r, B)$  dove
  - $N$  è un insieme di **nodi**
  - $r \in N$  è detto **radice**
  - $B$  è una **relazione binaria** su  $N$  ( $B \subseteq N \times N$ )
- la relazione  $B$  soddisfa le seguenti proprietà
  - $\forall n \in N, (n, r) \notin B$
  - $\forall n \in N$ , se  $n \neq r$  allora esiste **uno e un solo**  $n' \in N$  tale che  $(n', n) \in B$
  - $\forall n \in N$ , se  $n \neq r$  allora  $n$  è **raggiungibile** da  $r$ , cioè esistono  $n'_1, \dots, n'_k \in N$  con  $k \geq 2$  tali che  $n'_1 = r$ ,  $(n'_i, n'_{i+1}) \in B$  per ogni  $1 \leq i \leq k-1$ , ed  $n'_k = n$



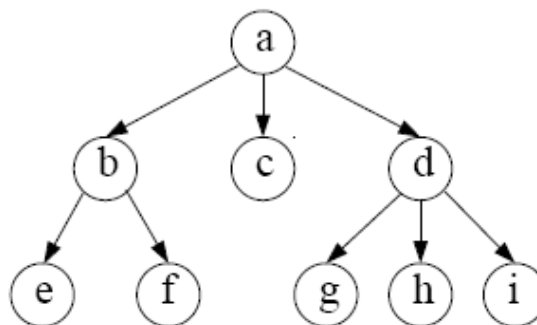
## Sottoalbero - definizione

- sia  $T = (N, r, B)$  un albero ed  $n \in N$ 
  - si dice **sottoalbero** generato da  $n$  l'albero  $T' = (N', n, B')$  dove  $N'$  è il sottoinsieme dei nodi di  $N$  raggiungibili da  $n$  e  $B' = B \cap (N' \times N')$
- sia  $T = (N, r, B)$  un albero e siano  $T_1 = (N_1, n_1, B_1)$  e  $T_2 = (N_2, n_2, B_2)$  i sottoalberi generati da  $n_1, n_2 \in N$ 
  - allora  $N_1 \cap N_2 = \emptyset$  oppure  $N_1 \subseteq N_2$  oppure  $N_2 \subseteq N_1$



## Padre - figlio - fratello - foglia

- sia  $T = (N, r, B)$  un albero
- se  $(n, n') \in B$  allora
  - $n$  è detto **padre** di  $n'$
  - $n'$  è detto **figlio** di  $n$
- se  $(n, n_1), (n, n_2) \in B$  allora  $n_1$  ed  $n_2$  sono detti **fratelli**
- i nodi privi di figli sono detti **foglie** (*nodi esterni*)
  - gli altri nodi sono detti *nodi interni*
- gli elementi di  $B$  sono detti **rami**



## Grado - livello - profondità - ampiezza

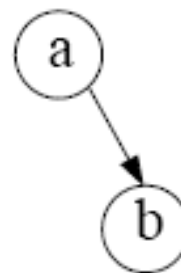
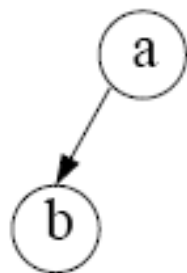
- sia  $T = (N, r, B)$  un albero
- si dice **grado** di  $T$  il massimo numero di figli di un nodo di  $T$
- si dice che  $r$  è a **livello 1**
- se  $n \in N$  è al livello  $i$  e  $(n, n') \in B$  allora  $n'$  è a **livello  $i+1$**
- si dice **profondità** di  $T$  il **massimo numero di nodi** che si **attraversano** per andare **dalla radice alla foglia più distante**
- si dice **ampiezza** di  $T$  il massimo numero di **nodi** di  $T$  che si trovano allo **stesso livello**

# **STRUTTURA DATI DIN. NON LINEARE: ALBERO BINARIO**



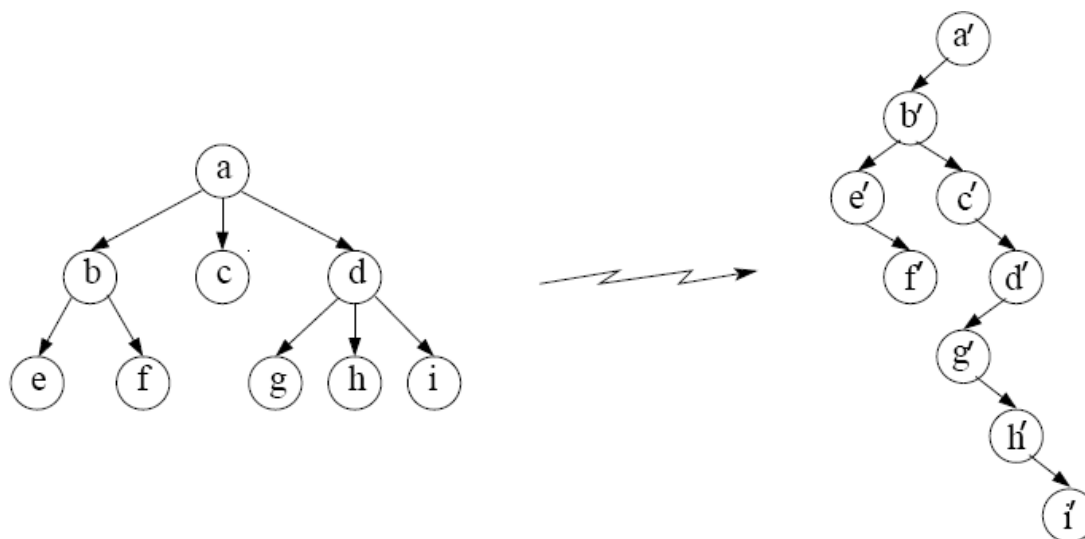
## Albero binario - definizione

- un albero  $T = (N, r, B)$  si dice **binario** se:
  - $B = B_{sx} \cup B_{dx}$
  - $B_{sx} \cap B_{dx} = \emptyset$
  - $\forall n, n_1, n_2 \in N$  se  $(n, n_1) \in B_{sx}$  ed  $(n, n_2) \in B_{sx}$  allora  $n_1 = n_2$
  - idem per  $B_{dx}$
  - se  $(n, n') \in B_{sx}$  allora  $n'$  è detto **figlio sinistro** di  $n$
  - idem per **figlio destro**

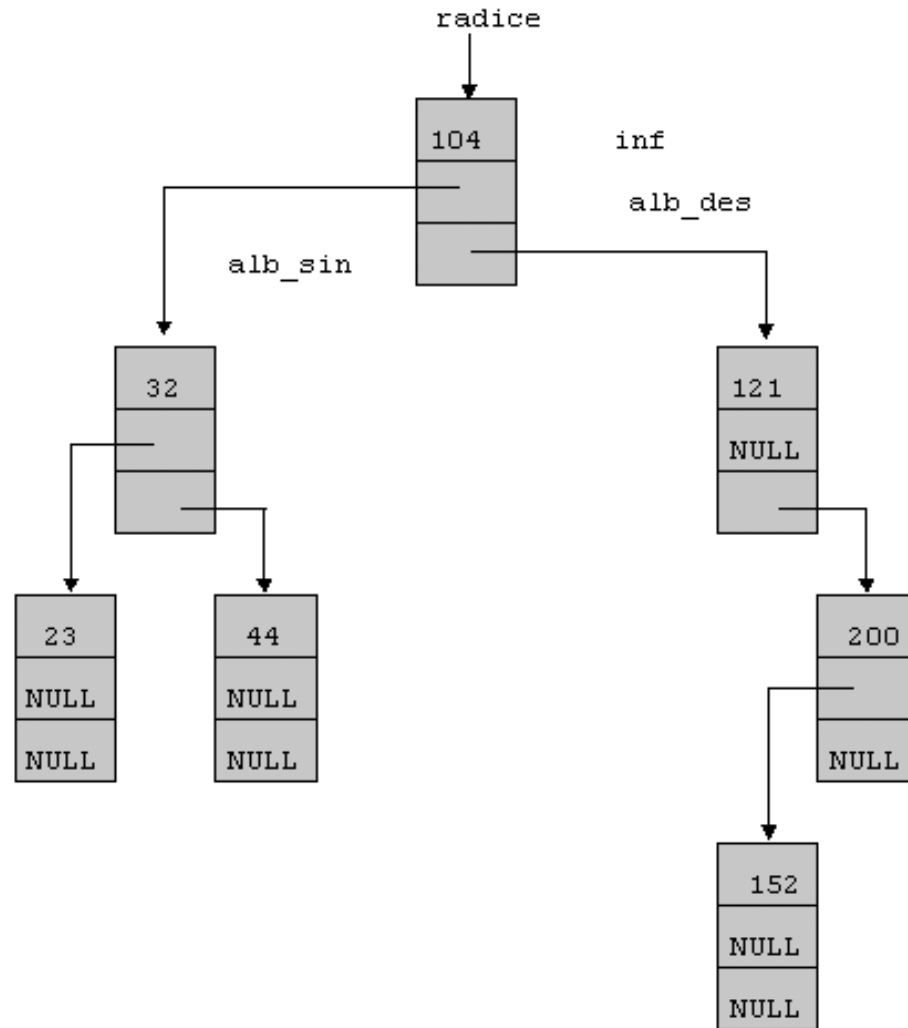


## Trasformazione da albero in albero binario corrispondente

- ogni albero **non binario** è **equivalente** ad un albero **binario** ottenuto applicando la seguente trasformazione **fratello-figlio** ad ogni nodo **n** dell'albero avente come figli i nodi  **$n_1, n_2, \dots, n_k$**
- creare i nuovi nodi  **$n'_1, n'_2, \dots, n'_k$**
- mettere  **$n'_1$**  come **figlio sinistro** di **n**
- per ogni  **$i = 1, \dots, k-1$** , mettere  **$n'_{i+1}$**  come **figlio destro** di  **$n'_i$**



## Rappresentazione di un albero binario

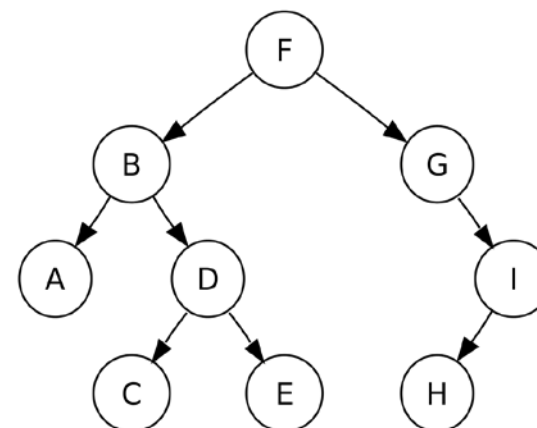


## Algoritmi di visita

- la **visita** consiste nell'accesso **una e una sola volta** a **tutti i nodi** dell'albero
- per gli **alberi binari** sono possibili più algoritmi di visita che generano sequenze diverse (per ordine) di nodi
  - visita in ordine **anticipato**
  - visita in ordine **simmetrico**
  - visita in ordine **posticipato** (*differito*)

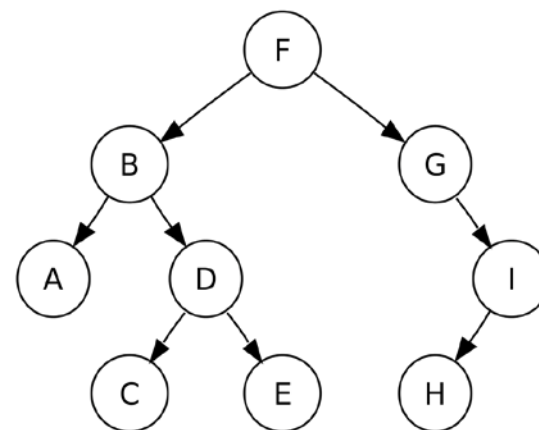
## Visita in ordine anticipato

- visita la **radice**
- visita il sottoalbero **sinistro** in ordine anticipato
- visita il sottoalbero **destro** in ordine anticipato
- lista dei nodi:
  - F, B, A, D, C, E, G, I, H



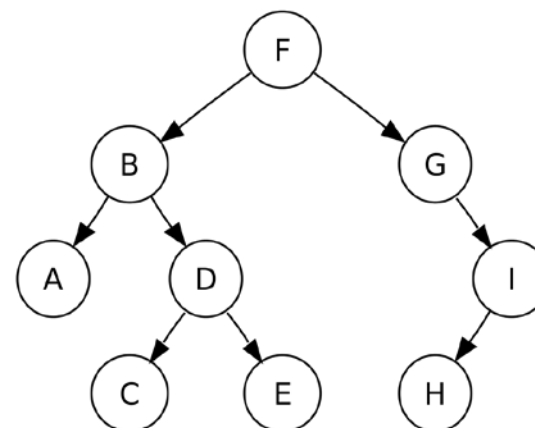
## Visita in ordine simmetrico

- visita il sottoalbero **sinistro** in ordine simmetrico
- visita la **radice**
- visita il sottoalbero **destro** in ordine simmetrico
- lista dei nodi:
  - A, B, C, D, E, F, G, H, I



## Visita in ordine posticipato

- visita il sottoalbero **sinistro** in ordine posticipato
- visita il sottoalbero **destro** in ordine posticipato
- visita la radice
- lista dei nodi:
  - A, C, E, D, B, H, I, G, F

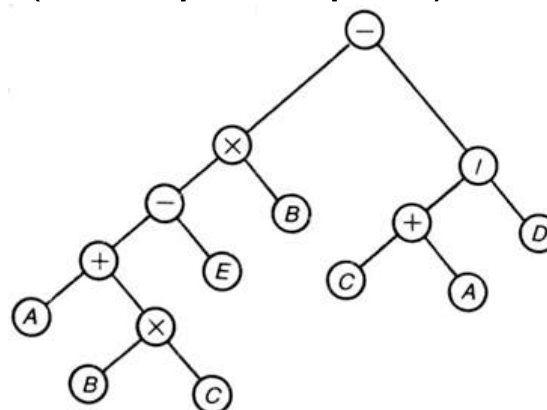


## Alberi ed espressioni

- ogni **nodo** che contiene un **operatore** è **radice** di un sottoalbero
- ogni **foglia** contiene un valore **costante** o una **variabile**
- **notazione polacca** (sintassi) denota formule matematiche
  - gli operatori si trovano tutti a sinistra degli argomenti (**prefissa**)
  - notazione polacca inversa (**postfissa**)

$$(((b \times c) + a) - e) \times b - ((c + a) / d)$$

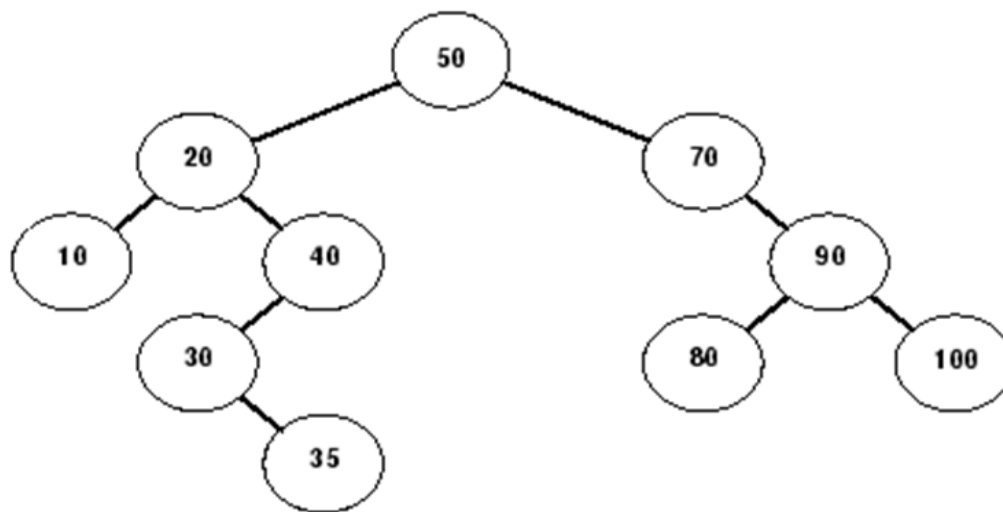
- $- \times - + a \times b \ c \ e \ b \ / + c \ a \ d$  (visita anticipata)
- $a \ b \ c \ \times + e - b \ \times c \ a + d \ / -$  (visita posticipata)





## Alberi binari di ricerca

- un **albero binario di ricerca** è un albero binario tale che:
  - per ogni nodo che contiene una **chiave** di valore **k**
  - ogni nodo del suo sottoalbero **sinistro** contiene una chiave di valore  $\leq k$
  - ogni nodo del suo sottoalbero **destro** contiene una chiave di valore  $\geq k$



## Albero binario: esempio

```
class AlberoBin {
public:
    AlberoBin();
    AlberoBin(string i);
    AlberoBin(string i, AlberoBin* l,
               AlberoBin* r);
    virtual ~AlberoBin();
    ... <setter & getter>
    void preOrder();
    void inOrder();
    void postOrder();
private:
    string info;
    AlberoBin* left;
    AlberoBin* right;
};

void AlberoBin::preOrder() {
    cout << info << " - ";
    if (left!=nullptr)
        left->preOrder();
    if (right!=nullptr)
        right->preOrder();
}

void AlberoBin::inOrder() {
    if (left!=nullptr)
        left->inOrder();
    cout << info << " - ";
    if (right!=nullptr)
        right->inOrder();
}

void AlberoBin::postOrder() {
    if (left!=nullptr)
        left->postOrder();
    if (right!=nullptr)
        right->postOrder();
    cout << info << " - ";
}
```

## Algoritmo di ricerca in alberi binari di ricerca

- **non** è necessario visitare **tutti** i nodi
- basta fare un **unico percorso** tra quelli che partono dalla radice, scendendo ad ogni nodo incontrato che non contiene il valore dato a **sinistra** o a **destra** a seconda che il valore dato sia **minore** o **maggiore**, rispettivamente, della chiave contenuta nel nodo
- la **complessità** della ricerca dipende quindi dalla **profondità** dell'albero

# **STRUTTURA DATI DINAMICA NON LINEARE: GRAFO**

## Grafo

- si dice **grafo diretto** una coppia  $G = (V, \varepsilon)$  dove  $V$  è un insieme di **vertici** ed  $\varepsilon$  è una **relazione binaria** su  $V$
- se  $(v, v') \in \varepsilon$  si dice che  $v'$  è **adiacente** a  $v$  (c'è un **arco da  $v$  a  $v'$** )
- il **grado uscente** di  $v \in V$  è il **numero** di **vertici adiacenti** a  $v$
- il **grado entrante** di  $v \in V$  è il **numero** di vertici a cui  $v$  è **adiacente**
- il **grado** di  $v \in V$  è il **numero** di archi in cui  $v$  è **coinvolto**
- $G$  si dice **completo** se  $\varepsilon = V \times V$

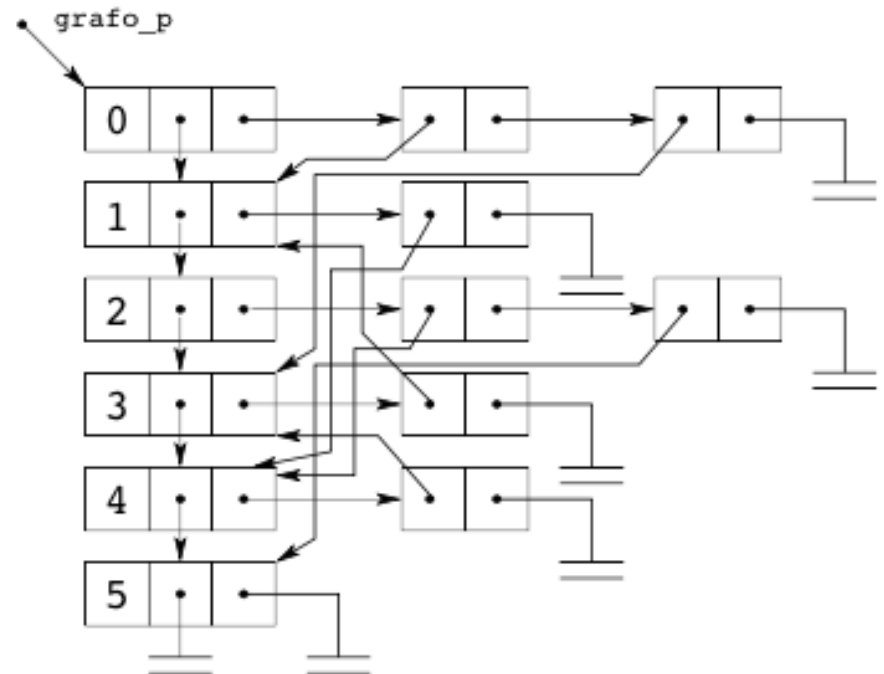
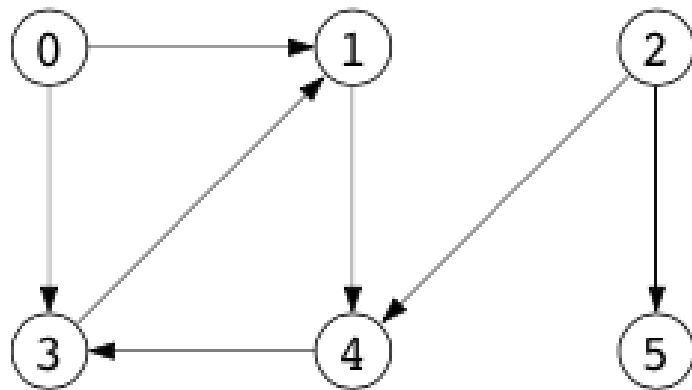
## Grafi connessi

- sia  $G = (V, \varepsilon)$  un grafo diretto
  - siano  $v_1, v_2 \in V$  – si dice che  $v_2$  è **raggiungibile** da  $v_1$  se esiste un percorso da  $v_1$  a  $v_2$
  - si dice che  $G$  è **connesso** se per ogni  $v_1, v_2 \in V$  esiste un **percorso** da  $v_1$  a  $v_2$  o da  $v_2$  a  $v_1$
  - si dice che  $G$  è **fortemente connesso** se per ogni  $v_1, v_2 \in V$  esistono un percorso da  $v_1$  a  $v_2$  E un percorso da  $v_2$  a  $v_1$

## Implementazione con liste di adiacenza

- il grafo viene rappresentato come una struttura dati dinamica reticolare detta **lista di adiacenza**, formata da una **lista primaria** dei **vertici** e **più liste secondarie** degli **archi**
- la lista **primaria** contiene un **elemento** per **ciascun vertice** del grafo, il quale contiene a sua volta la **testa** della relativa **lista secondaria**
- la lista **secondaria** associata ad un vertice descrive tutti gli **archi uscenti** da quel vertice

## Liste di adiacenza

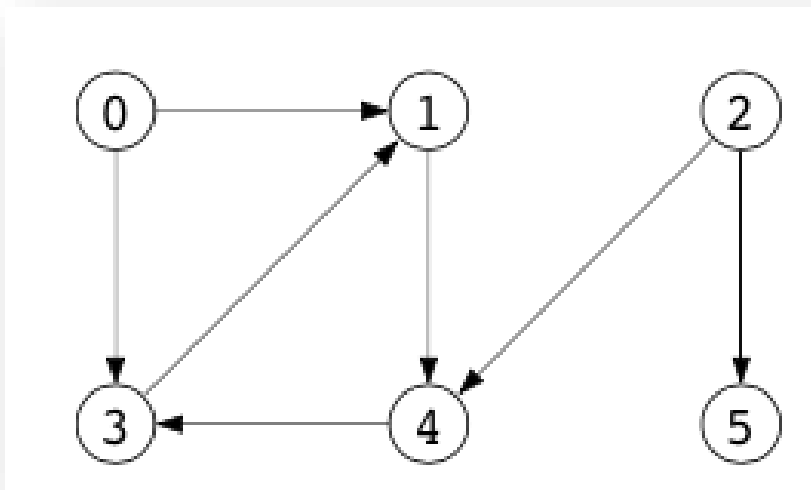




## Implementazione con matrice di adiacenza

- se la struttura di un grafo non cambia oppure è importante fare accesso rapidamente alle informazioni contenute nel grafo, allora conviene ricorrere ad una rappresentazione a **matrice di adiacenza**
- la **matrice** ha tante **righe** e tante **colonne** quanti sono i **vertici**
- l'elemento di indici **i** e **j** vale **1** se **esiste un arco** dal **vertice i** al **vertice j**, **0** altrimenti
- per i grafi **pesati** si può sostituire il valore 1 con il **peso** del grafo

## matrice di adiacenza



0	1	0	1	0	0
0	0	0	0	1	0
0	0	0	0	1	1
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	0

## Applicazione dei grafi: esempio

- il problema dei **sette ponti di Königsberg** è un problema ispirato da una città reale e da una situazione concreta
- *Königsberg è percorsa dal fiume Pregel e da suoi affluenti e presenta due estese isole che sono connesse tra di loro e con le due aree principali della città da **sette ponti***
- nel corso dei secoli è stata più volte proposta la questione **se sia possibile con una passeggiata seguire un percorso che attraversi ogni ponte una e una volta soltanto e tornare al punto di partenza**
- nel 1736 Leonhard Euler affrontò tale problema, dimostrando che la passeggiata ipotizzata **non era possibile**

## Astrazione

- Eulero ha formulato il problema in termini di **teoria dei grafi**, **astruendo** dalla situazione specifica di Königsberg
  - **eliminazione** di tutti gli **aspetti contingenti** ad esclusione delle aree urbane delimitate dai bracci fluviali e dai ponti che le collegano
  - **rappresentazione** di ogni **area urbana** con un **vertice**
  - **rappresentazione** di ogni **ponte** con un **arco**

