UNIVERSITÀ
DI PARMA

abstract data type
strutture dati dinamiche non lineari

Alberto Ferrari

strutture dati dinamiche non lineari

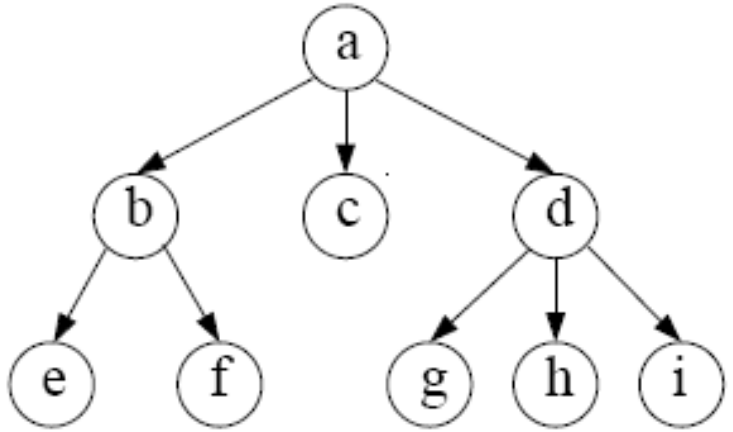
- una struttura dati si definisce ***dinamica*** se permette di rappresentare insiemi dinamici la cui ***cardinalità varia*** durante l'esecuzione del programma
- strutture dinamiche non lineari
  - ***alberi***
    - ***alberi binari***
  - ***grafi***

struttura dati dinamica non lineare

albero

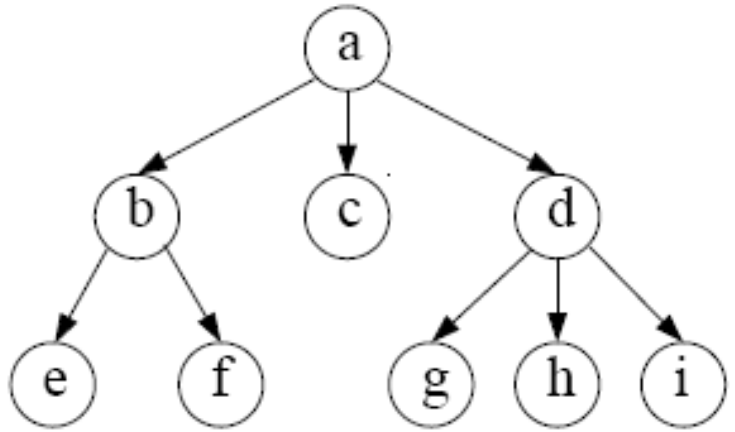
albero - definizione

- si dice albero una ***tripla*** **T = (N, r, *B*)** dove
  - **N** è un insieme di ***nodi***
  - **r ∈ 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 ∈ N, (n,r) ∉ *B***
  - $\forall$ **n ∈ N**, se **n ≠ r** allora esiste ***uno e un solo*** **n' ∈ N** tale che **(n', n) ∈ *B***
  - $\forall$ **n ∈ N**, se **n ≠ r** allora **n** è ***raggiungibile*** da **r**, cioè esistono $\mathbf{n'_1, \ldots, n'_k \in N}$ con $k \geq 2$ tali che $\mathbf{n'_1 = r}$, $\mathbf{(n'_i, n'_{i+1}) \in B}$ per ogni $1 \leq i \leq k-1$, ed $\mathbf{n'_k = n}$



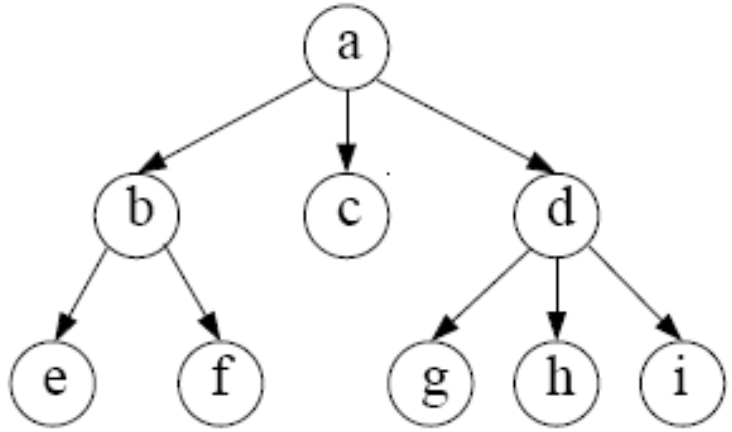
sottoalbero

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



padre - figlio - fratello - foglia

- sia **T = (N, r, *B*)** un albero
- se **(n , n') ∈ *B*** allora
  - n è detto ***padre*** di n'
  - n' è detto ***figlio*** di n
- se **(n , n_1), (n , n_2) ∈ 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 - altezza - larghezza

- 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 dinamica non lineare

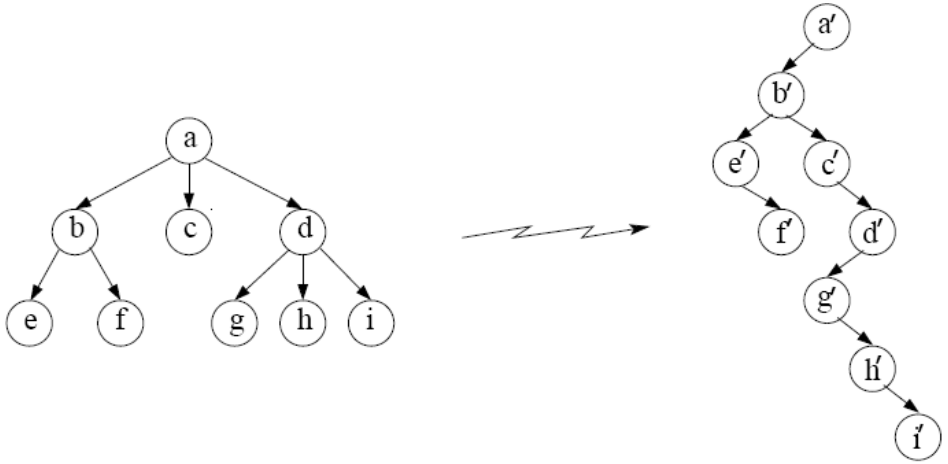
albero binario

albero binario - definizione

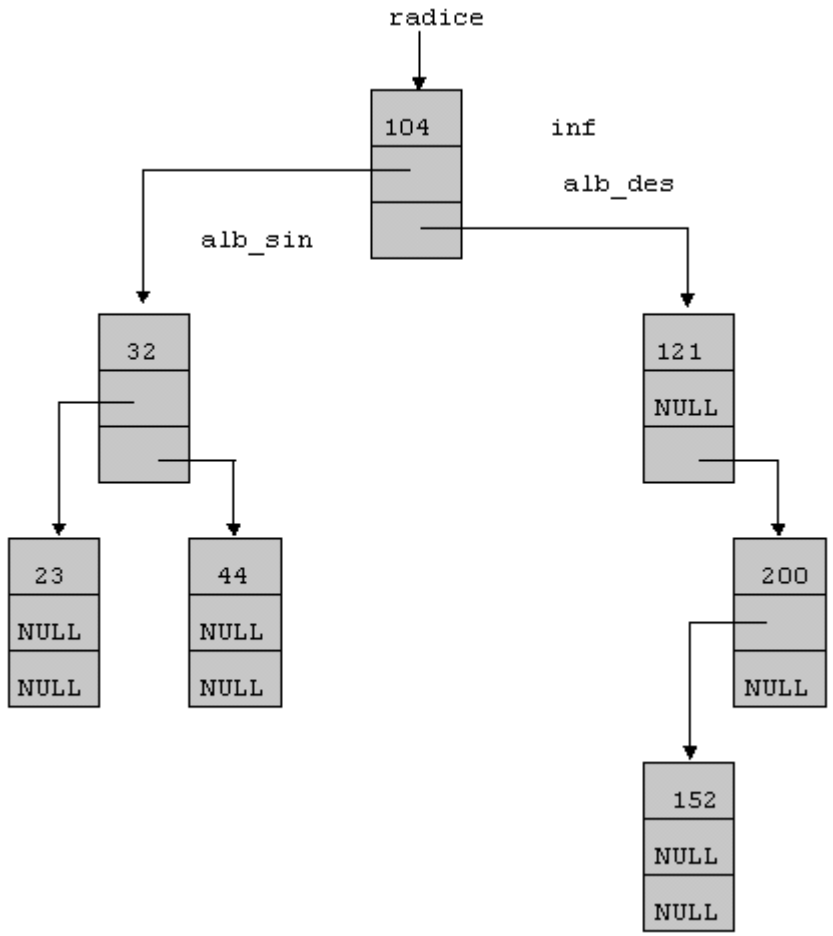
- un albero $\mathbf{T = (N, r, \mathit{B})}$ si dice binario se:
  - $\boldsymbol{B = B_{sx} \cup B_{dx}}$
  - $\mathbf{B_{sx} \cap B_{dx} = \varnothing}$
  - $\forall\ \mathbf{n, n_1, n_2 \in N}$ se $\mathbf{(n, n_1) \in \mathit{B}_{sx}}$ ed $\mathbf{(n, n_2) \in \mathit{B}_{sx}}$ allora $\boldsymbol{n_1 = n_2}$
  - idem per $\mathbf{B_{dx}}$
  - se $\mathbf{(n, n') \in \mathit{B}_{sx}}$ 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 $n_1, n_2, \ldots, n_k$
- creare i nuovi nodi $n'_1, n'_2, \ldots, n'_k$
- mettere n'_1 come ***figlio sinistro*** di ***n'***
- per ogni $i = 1, \ldots, k-1$, mettere n'_{i+1} come figlio destro di n'_k



rappresentazione di un albero binario

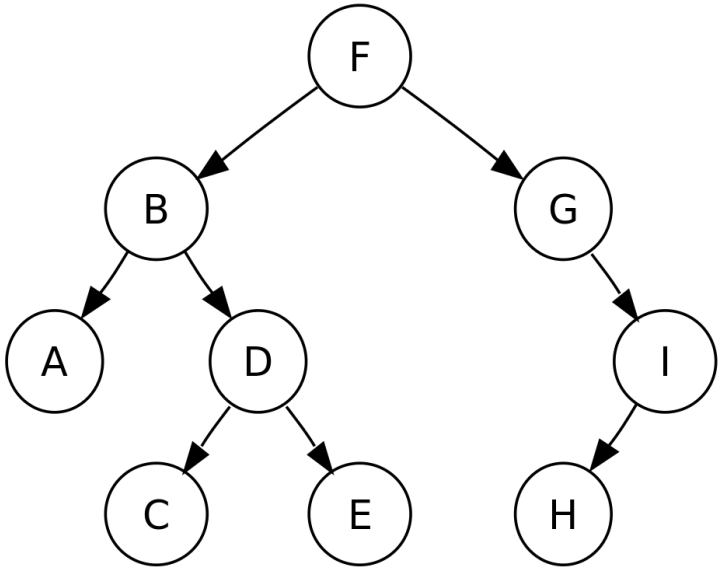


algoritmi di visita

- 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***)

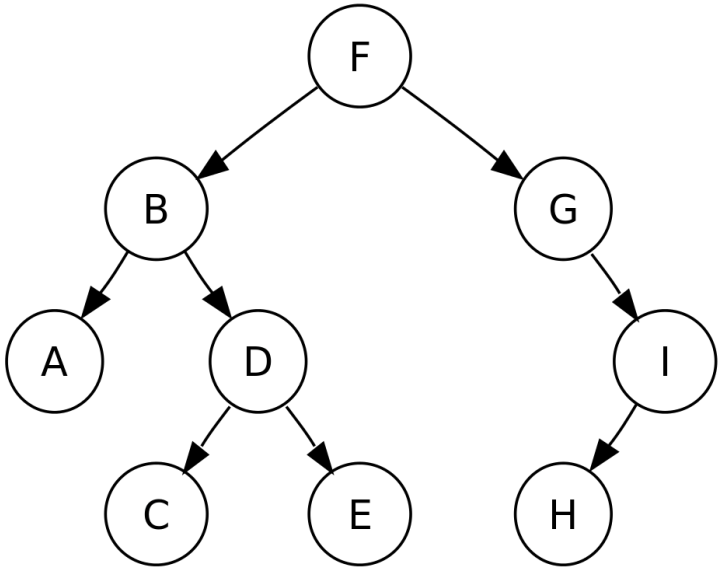
visita in ordine anticipato

- 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



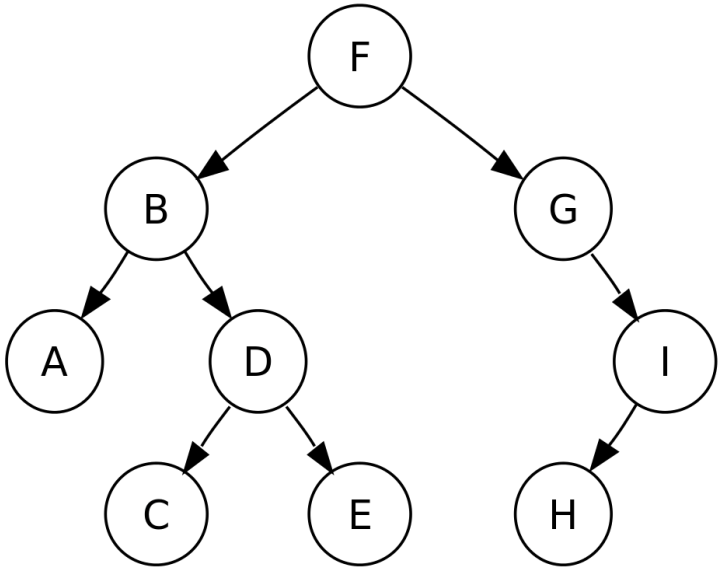
visita in ordine simmetrico

- 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



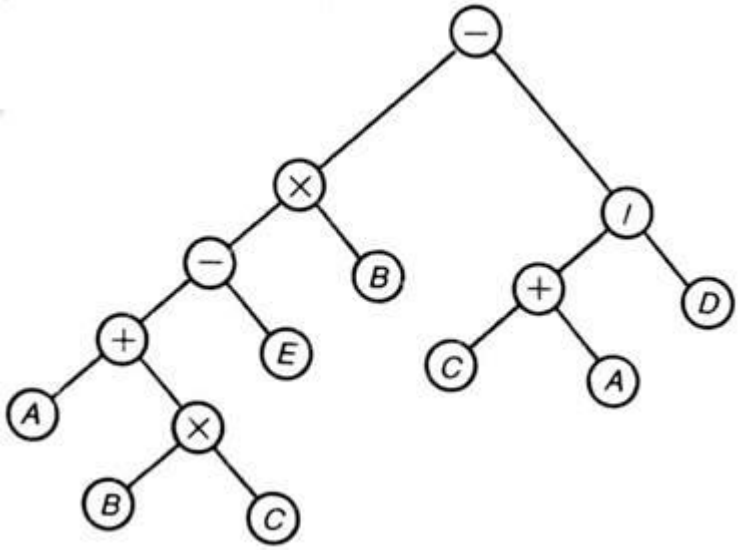
visita in ordine differito

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



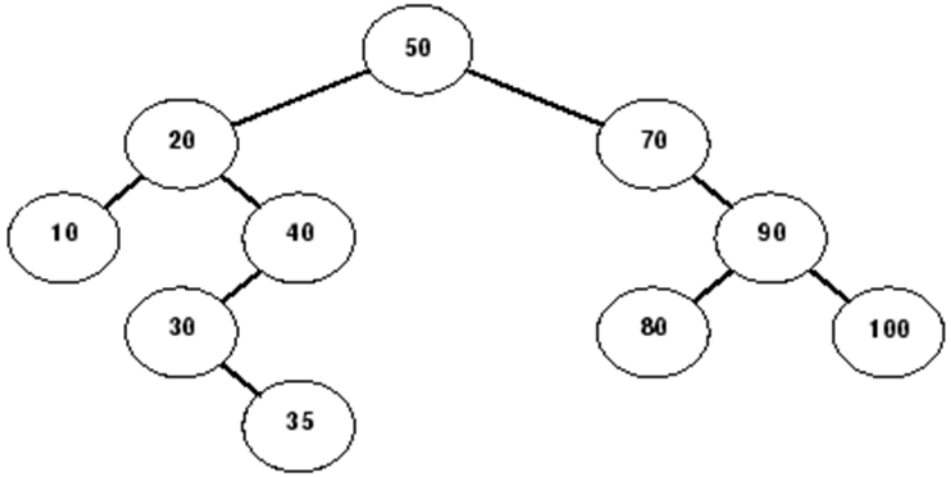
alberi ed espressioni

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



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

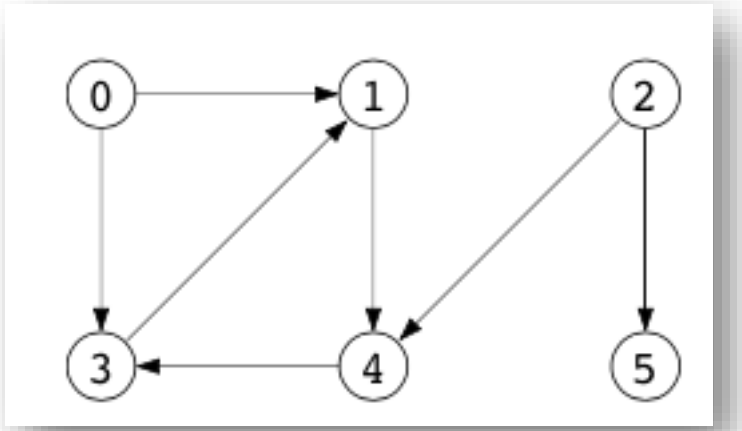
grafi connessi

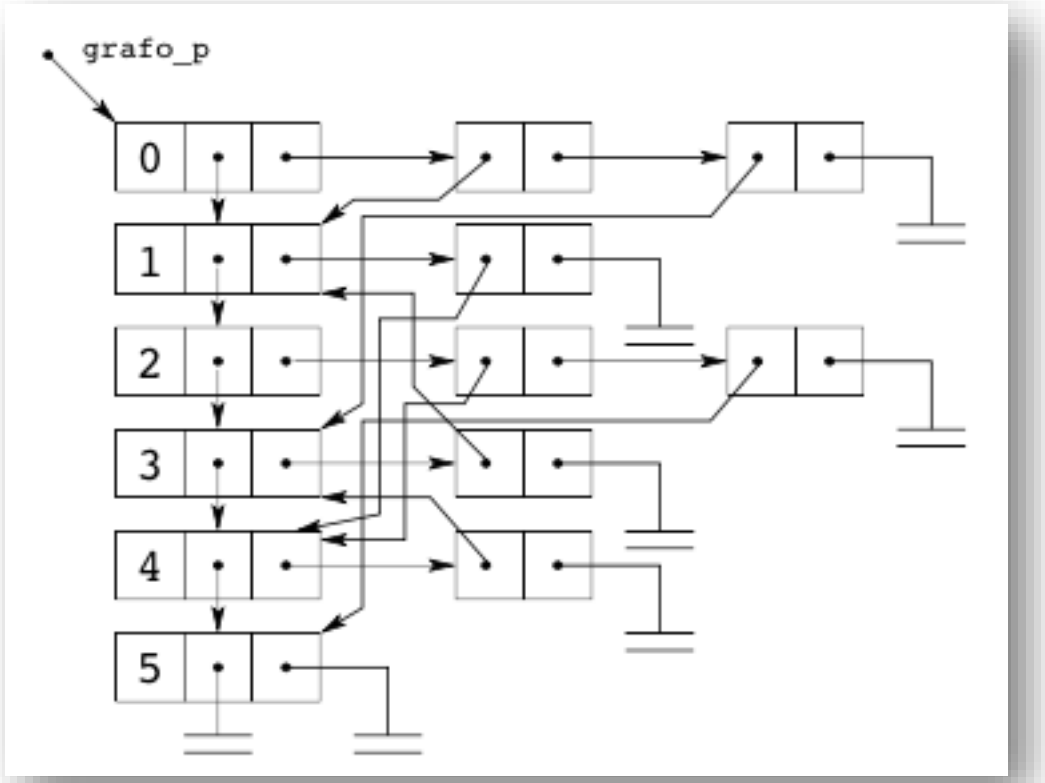
- sia **G = (*V*,*ε*)** un grafo diretto
  - siano $v_1, v_2 \in V$ – si dice che v_2 è ***raggiungibile*** da v_1 se esiste un percorso da v_1 a v_2
  - si dice che G è ***connesso*** se per ogni $v_1, v_2 \in V$ esiste un ***percorso*** da v_1 a v_2 o da v_2 a v_1
  - si dice che G è ***fortemente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

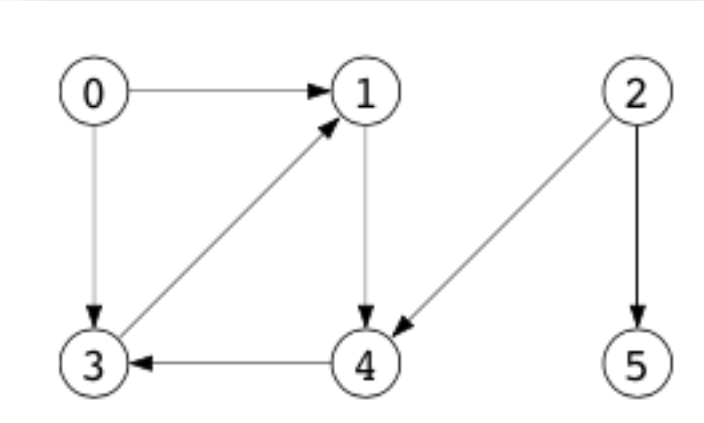




implementazione con matrice di adiacenza

- 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***

- Eulero ha formulato il problema in termini di ***teoria dei grafi***, ***astraendo*** 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***