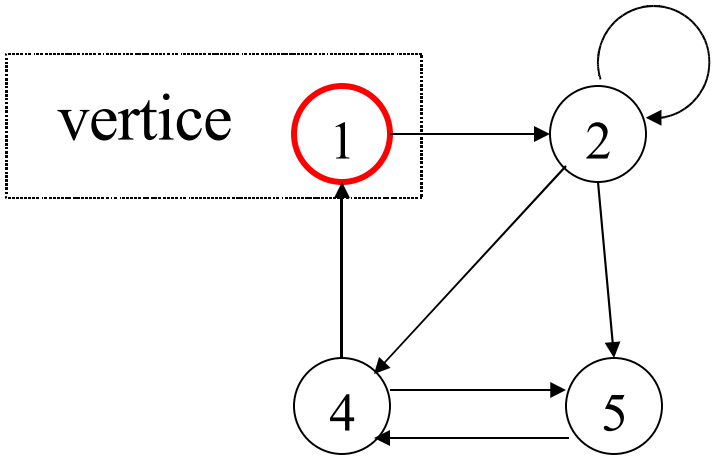
Alberi e Grafi

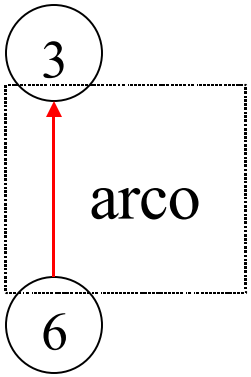
Sommario

- Definizione di Grafi e Alberi
- Implementazione di Grafi e Alberi

Grafi orientati

- Un *grafo orientato* (o diretto) G è una coppia (V,E) dove V è un insieme finito detto dei *vertici* e E è una relazione binaria su V che forma l'insieme degli *archi*.
- Gli archi sono delle coppie ordinate di vertici.



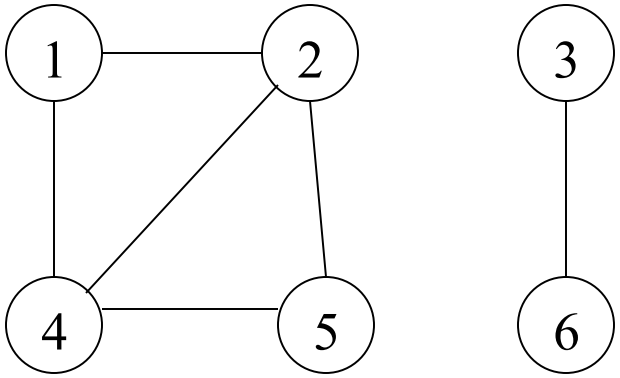


V={1,2,3,4,5,6}

E={(1,2),(2,2),(2,4),
(2,5),(4,1),(4,5),
(5,4),(6,3)}

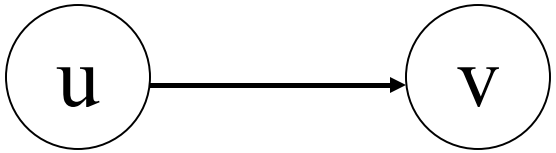
Grafi non orientati

- Un grafo non orientato è un grafo in cui gli archi sono coppie non ordinate di vertici, cioè un arco fra i vertici u,v è un insieme di due elementi {u,v} piuttosto che una coppia (u,v)
- Tuttavia si indica l'arco sempre con notazione (u,v)



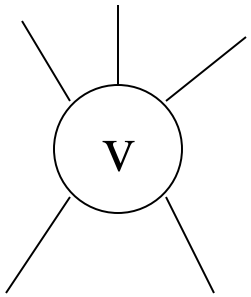
Arco

- Sia (u,v) è un arco di un grafo orientato, si dice che:
  - l'arco esce dal vertice u
  - l'arco entra nel vertice v
- Un arco (u,v) di un un grafo non orientato si dice che è incidente sui vertici v e u
- Si dice che v è adiacente a u
  - in un grafo non orientato la relazione di adiacenza è simmetrica
  - in un grafo orientato v è adiacente a u, ma non è vero il viceversa, e si indica con la notazione u→v

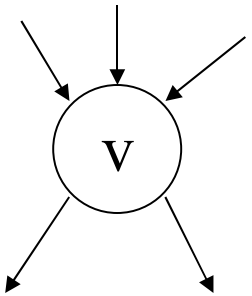


Grado

- Il grado di un vertice in un grafo non orientato è il numero di archi incidenti sul vertice
- In un grafo orientato il grado uscente (entrante) di un vertice è il numero di archi uscenti (entranti) dal vertice



Grado di v=5



Grado entrante di v=3
Grado uscente di v=2

Cammino

- Un cammino da un vertice *a* ad un vertice *b* in un grafo G=(V,E) è una sequenza di vertici

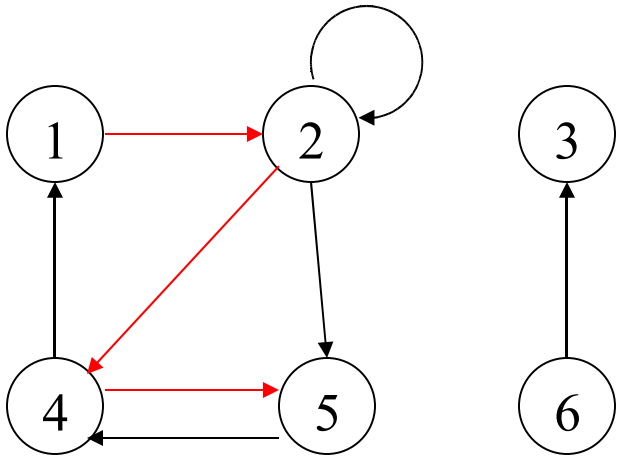
  $< v_0, v_1, \ldots, v_k >$ tali che

  - *a*= v_0
  - *b*= v_k
  - $(v_{i-1}, v_i) \in E$ per i=1,…,k

Cammino valido
<1,2,4,5>

Cammino non valido
<2,5,3>

Cammino non valido
<5,2,1>



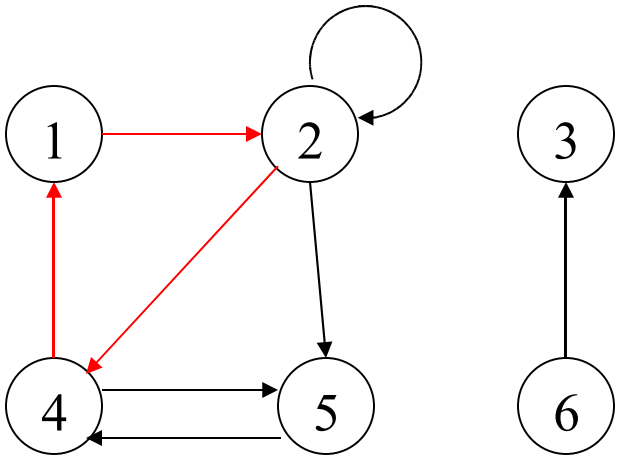
Ciclo

- La lunghezza di un cammino è il suo numero di archi
- Un cammino < v_0, v_1,…,v_k> è un ciclo se v_0= v_k
- Un grafo senza cicli si dice aciclico

Ciclo di lunghezza 3
<1,2,4,1>

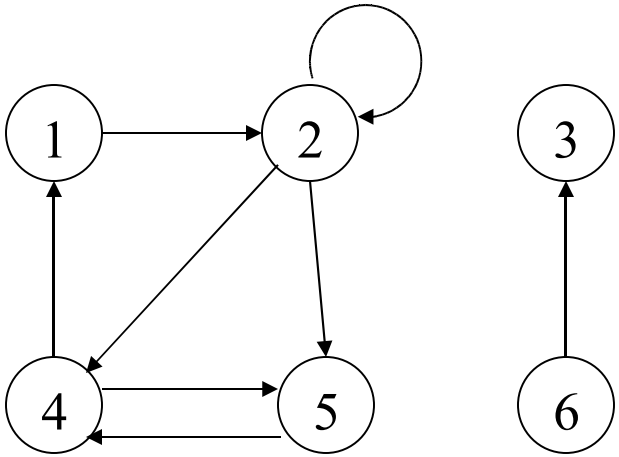
Ciclo di lunghezza 2
<4,5,4>

Ciclo di lunghezza 1
<2,2>



Grafi connessi

- Un grafo non orientato è connesso se ogni coppia di vertici è collegata con un cammino



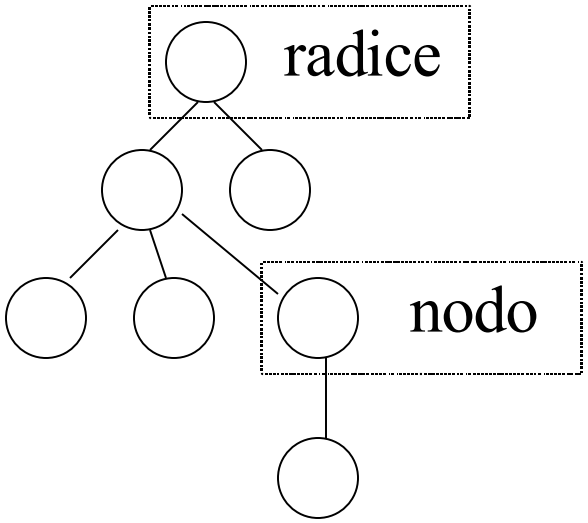
Grafo non connesso: non esiste cammino da 3 a 2

Alberi

- Un albero è:

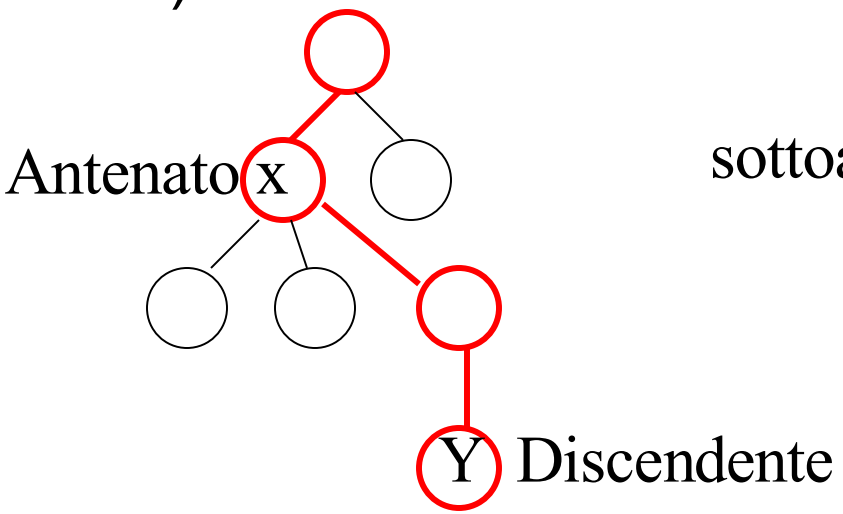
  un grafo non orientato, connesso e aciclico

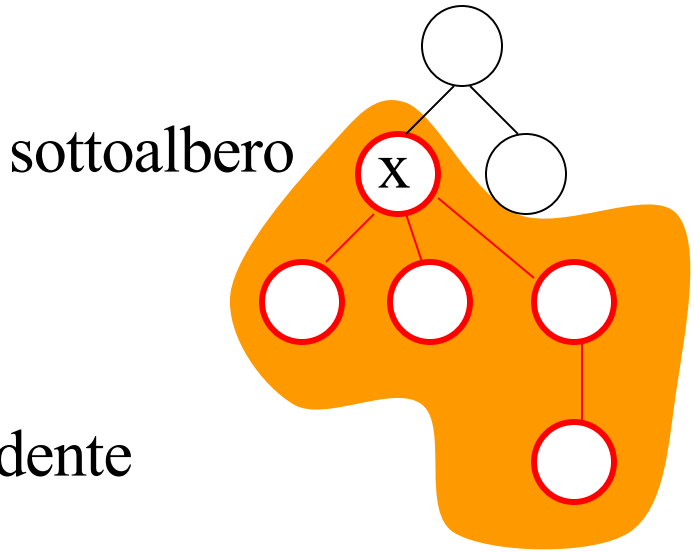
- Un albero radicato è un albero in cui si distingue un vertice (chiamato radice) dagli altri vertici
- I vertici in un albero sono chiamati nodi



Antenati e discendenti

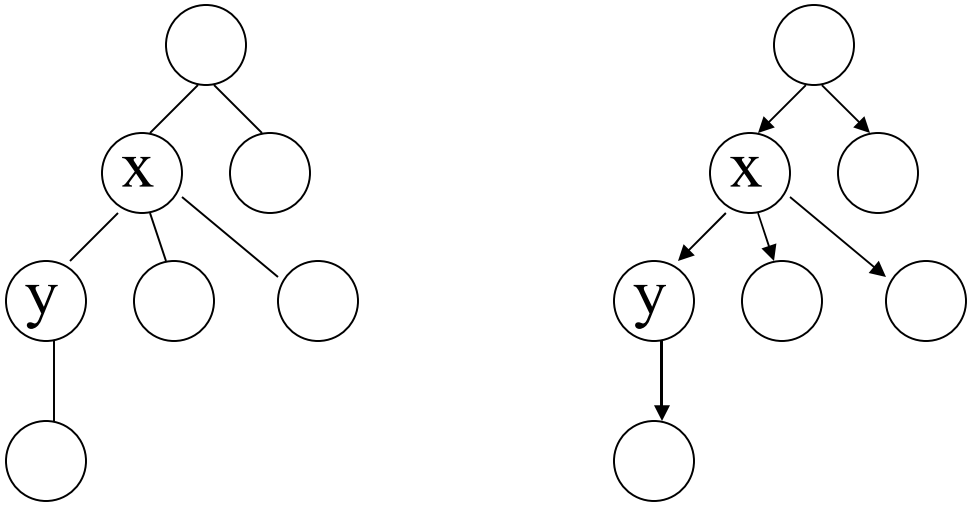
- Sia x un nodo di un albero con radice r: qualunque nodo x sull'unico cammino da r a y è chiamato antenato di y e y si dice discendente di x
- Il sottoalbero radicato in x è l'albero indotto dai discendenti di x, radicato in x (ovvero l'albero formato da tutti i nodi discendenti di x, x stesso e relativi archi)





Radice e orientamento

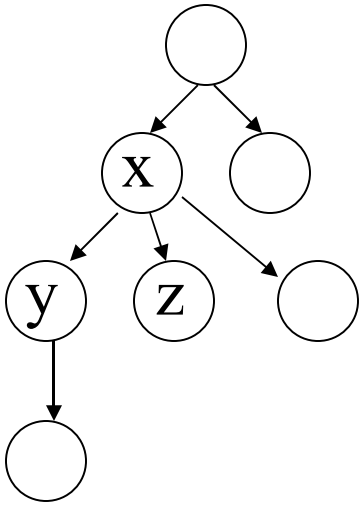
- Una radice induce un orientamento
- Si può cioè predicare l'attributo di predecessore e successore su ogni nodo
- Si rappresenta l'orientamento con frecce



Relazione fra nodi

- Se l'ultimo arco di un cammino dalla radice r ad un nodo x è l'arco (x,y) allora x è il padre di y e y è il figlio di x
- Il numero di figli di un nodo x è il grado di x
- Due nodi con lo stesso padre si dicono fratelli

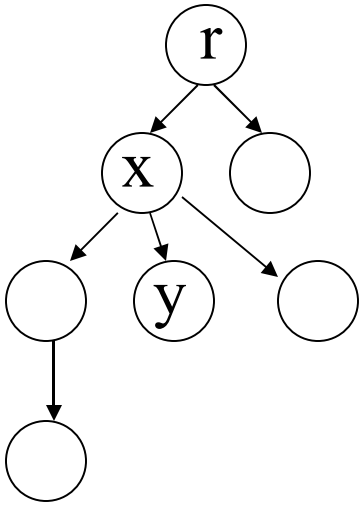
x=padre di y
y=figlio di x
y fratello di z



Nodi particolari

- La radice è l'unico nodo che non ha padre
- Un nodo senza figli si dice nodo esterno o foglia
- Un nodo non foglia è un nodo interno

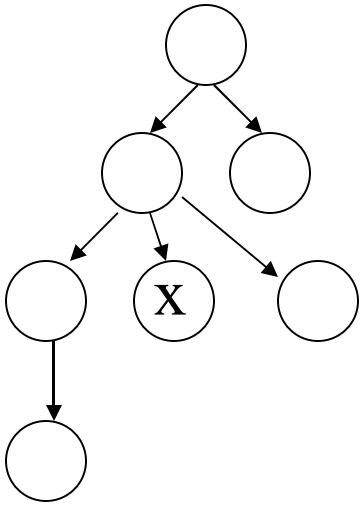
r=radice
x=nodo interno
y=foglia



Altezza

- La lunghezza di un cammino da r a x è la profondità di x
- La profondità massima di un qualunque nodo di un albero è l'altezza dell'albero

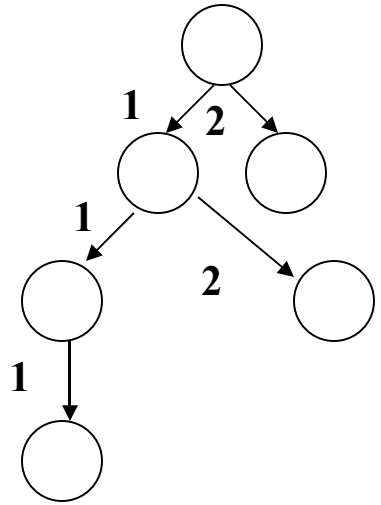
Profondità x=2
Altezza=3

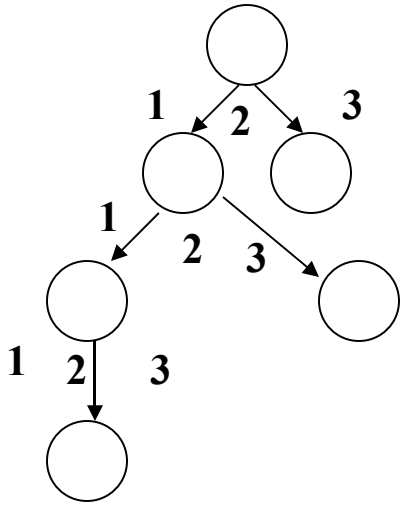


Alberi ordinati e posizionali

- Un albero ordinato è un albero radicato in cui i figli di ciascun nodo sono ordinati (cioè si distingue il primo figlio, il secondo, etc)
- Un alberi si dice posizionale se i figli dei nodi sono etichettati con interi positivi distinti (l'i-esimo figlio di un nodo è assente se nessun figlio è etichettato con l'intero i)

ordinato



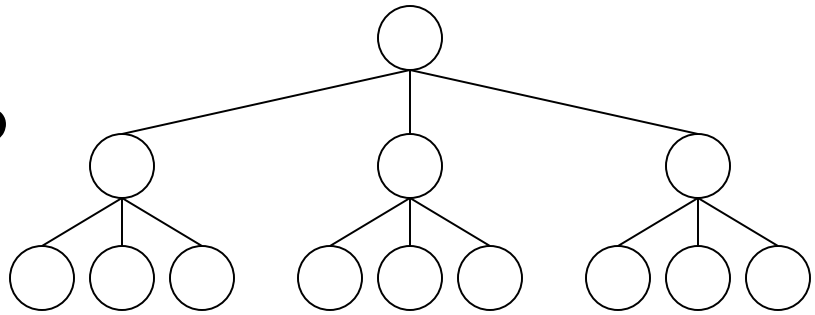


posizionale

Alberi k-ari

- Un albero k-ario è un albero posizionale in cui per ogni nodo tutti i figli con etichetta più grande di k sono assenti
- Un albero k-ario completo è un albero k-ario in cui tutte le foglie hanno la stessa profondità e tutti i nodi interni hanno grado k

Albero 3-ario



Alberi

- Il numero di foglie di un albero k-ario è:
  - la radice ha k figli a profondità 1
  - ognuno dei figli ha k figli a profondità 2 per un totale di k.k foglie
  - a profondità h si hanno k^h foglie
- Il numero di nodi interni di un albero k-ario completo di altezza h è:

$$1+k+ k^2 + \ldots +k^{h-1} = \sum_{i=0..h-1} k^i = (k^h -1)/(k-1)$$

Alberi binari

- Un albero binario è una struttura definita su un insieme finito di nodi che:
  - non contiene nessun nodo, oppure
  - è composto da tre insiemi disgiunti di nodi: un nodo radice, un albero binario chiamato sottoalbero sinistro e un albero binario chiamato sottoalbero destro
- Un albero binario che non contiene nessun nodo è detto albero vuoto o albero nullo (denotato con NIL)
- Se il sottoalbero sinistro (destro) non è vuoto allora la sua radice è detta figlio sinistro (destro)
- Se un sottoalbero è l'albero nullo si dice che il figlio è assente o mancante

Alberi binari

- Un albero binario
  - non è un albero ordinato con nodi con grado al più due
  - ma è un albero posizionale con nodi con grado al più due
- In un albero ordinato non si distingue fra figlio destro o sinistro (ma si considera solo il numero di figli)
- ..mentre in albero posizionale sì
- Un albero binario completo ha 2^h -1 nodi interni

Implementazione alberi binari

- Gli alberi si rappresentano ricorrendo agli stessi metodi usati per rappresentare le liste
- In genere si usano strutture dati con puntatori
- Per gli alberi binari si usano strutture dati per rappresentare i nodi che hanno un campo key e 2 o 3 puntatori ad altri nodi (si può anche non utilizzare il puntatore a padre se non e' necessario)

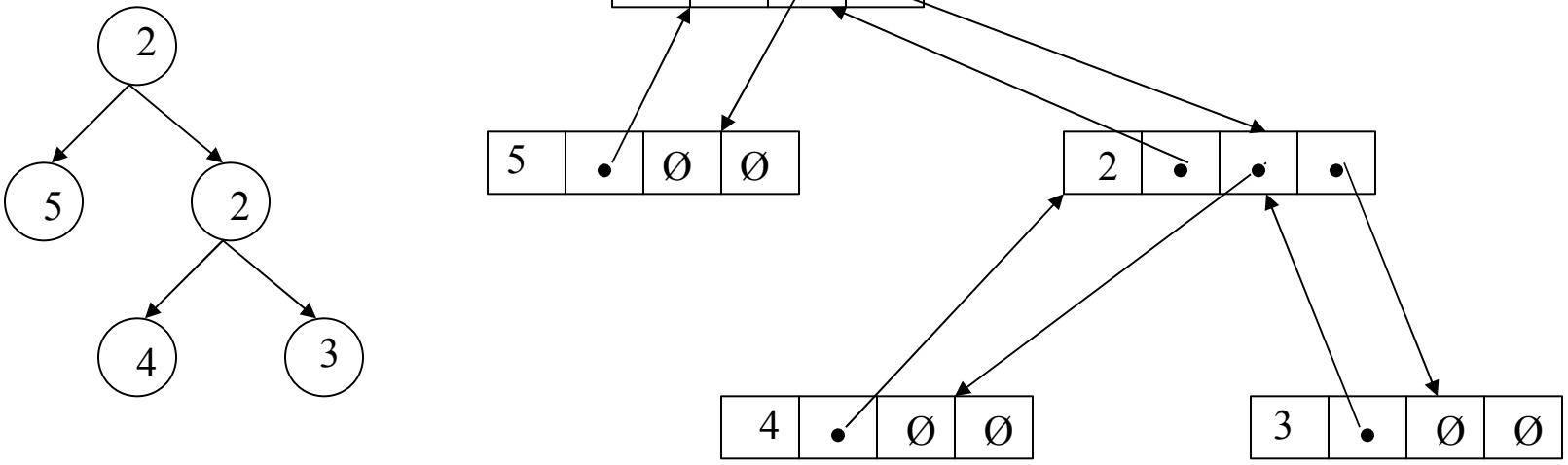
```
struct Node{
    int key;
    Node* p;
    Node * left, * right;
};
```

Implementazione alberi binari

- Se x è un nodo allora
  - se p[x]=NIL il nodo è la radice dell'albero
  - se left[x]=NIL (right[x]=NIL) allora il nodo non ha figlio sinistro (destro)
- Si mantiene il puntatore alla radice dell'albero T memorizzandola nell'attributo root[T]
  - se root[T]=NIL l'albero è vuoto

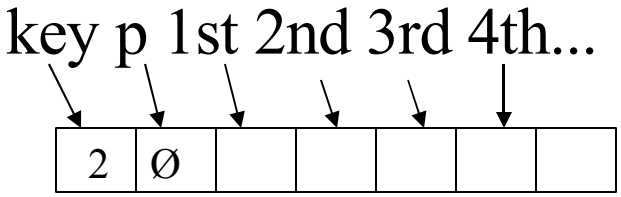
```
class Tree{
    struct Node{
        int key;
        Node* p;
        Node * left, * right;
    };

    Node * root;
};
```

Visualizzazione alberi binari



Implementazione alberi

- Se vogliamo rappresentare alberi con un numero illimitato di figli potremmo pensare di riservare un numero max di link ai figli come:

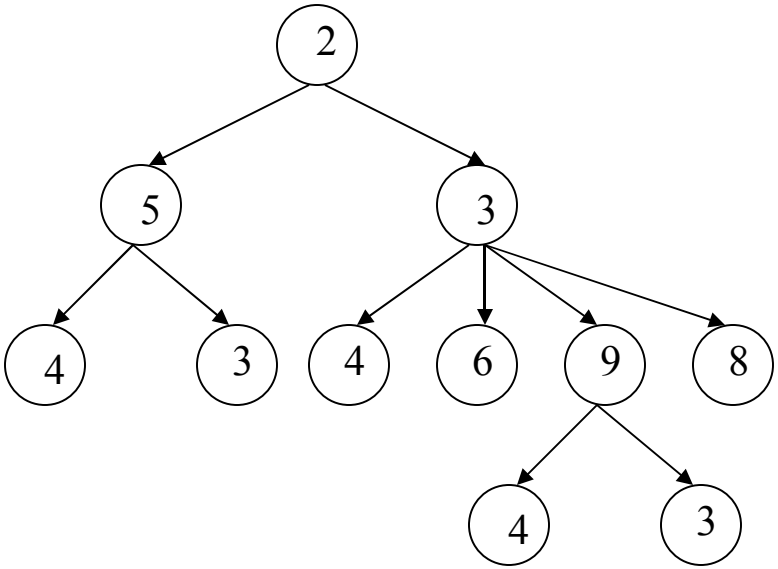


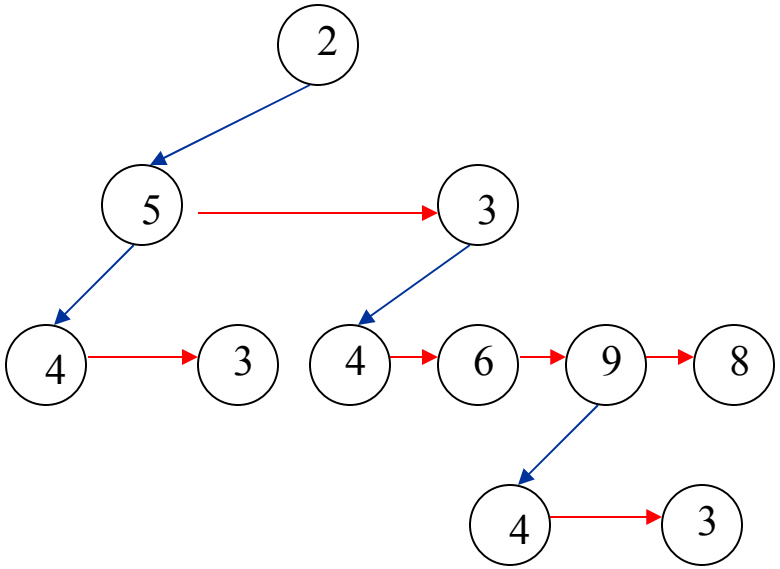
- Ma in questo modo dobbiamo porre un limite sul massimo grado di un nodo
- Inoltre viene sprecata molta memoria per rappresentare i puntatori NIL

Implementazione alberi

- Per rappresentare alberi con un numero illimitato di figli conviene usare la rappresentazione figlio-sinistro fratello-destro
- Ogni nodo conserva il campo key e puntatore a padre
- Invece di avere un puntatore per ogni figlio i nodi hanno solo due puntatori:
  - puntatore al figlio più a sinistra
  - puntatore al fratello immediatamente a destra

Visualizzazione

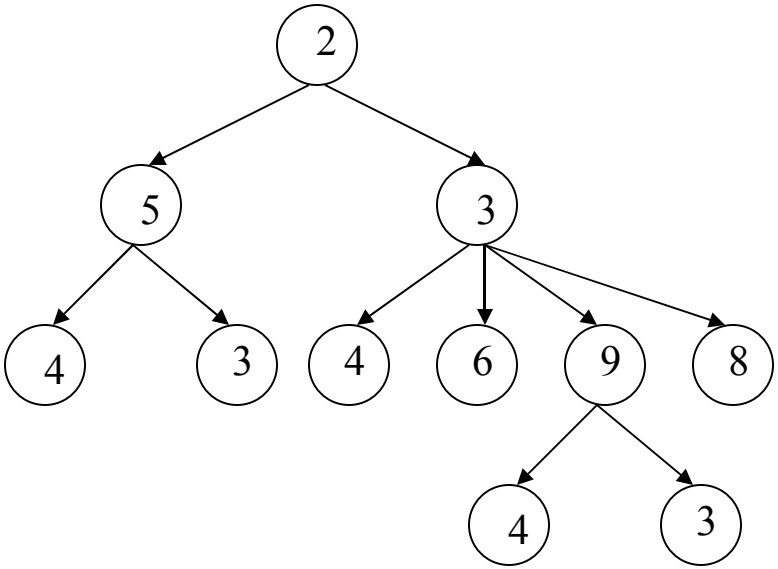


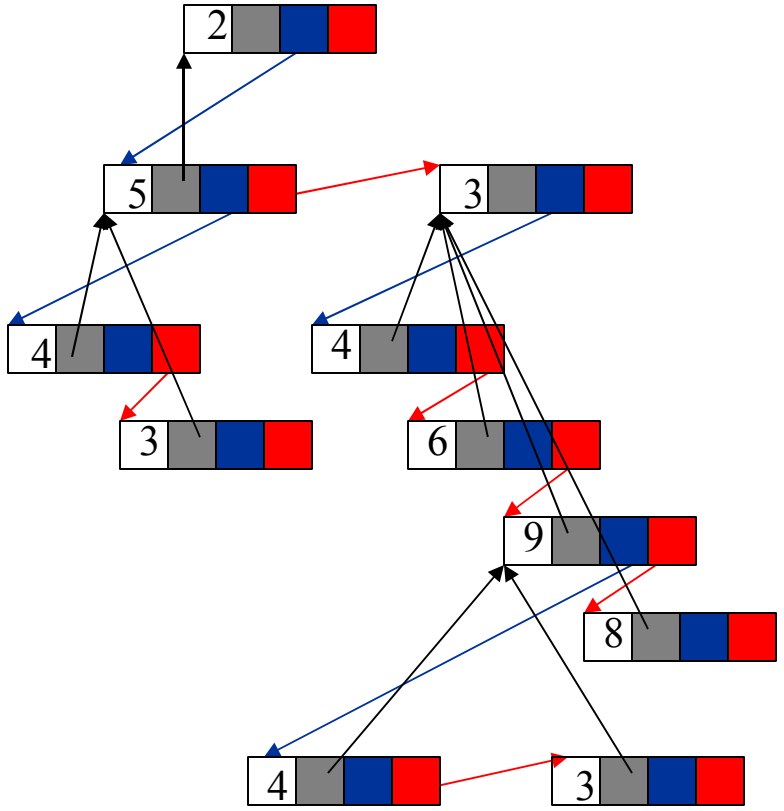


fratello destro

figlio sinistro

Visualizzazione





Grafi

- I grafi sono strutture dati molto diffuse in informatica
- Vengono utilizzati per rappresentare reti e organizzazioni dati complesse e articolate
- Per elaborare i grafi in genere è necessario visitarne in modo ordinato i vertici
- Vedremo a questo proposito due modi fondamentali di visita: per ampiezza e per profondità

Nota sulla notazione asintotica

- Il tempo di esecuzione di un algoritmo su un grafo G=(V,E) viene dato in funzione del numero di vertici |V| e del numero di archi |E|
- Utilizzando la notazione asintotica adotteremo la convenzione di rappresentare |V| con il simbolo V e |E| con E: quando diremo che il tempo di calcolo è O(E+V) vorremo significare O(|E|+|V|)

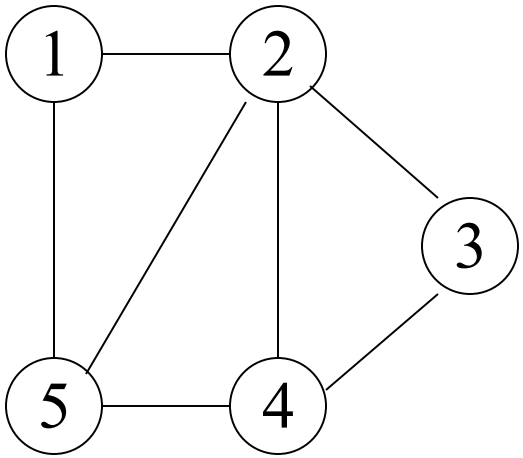
Rappresentazione di un grafo

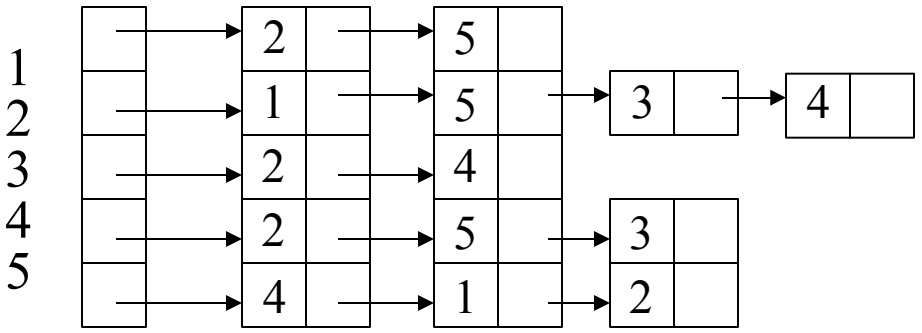
- Vi sono due modi per rappresentare un grafo:
  - collezione (array) di *liste di adiacenza*
  - *matrice di adiacenza*
- Si preferisce la rappresentazione tramite liste di adiacenza quando il grafo è *sparso*, cioè con |E| molto minore di $|V|^2$
- Si preferisce la rappresentazione tramite matrice di adiacenza quando, al contrario, il grafo è *denso* o quando occorre alta efficienza nel rilevare se vi è un arco fra due vertici dati

Liste di adiacenza

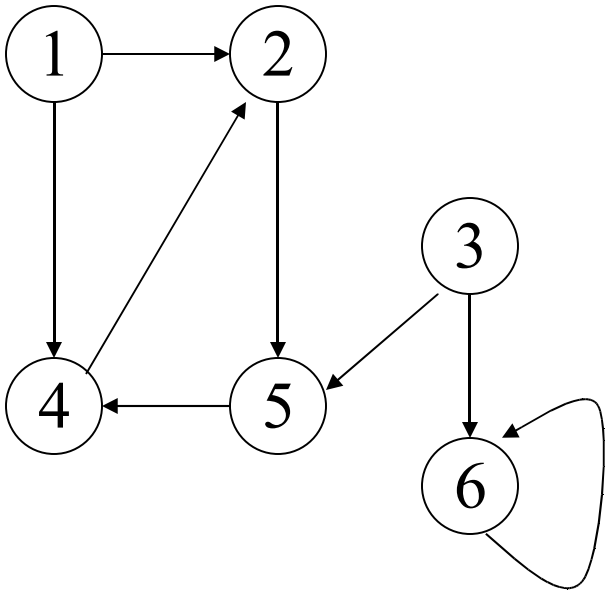
- Si rappresenta un grafo G=(V,E) con un vettore Adj di liste, una lista per ogni vertice del grafo
- Per ogni vertice u, Adj[u] contiene tutti i vertici v adiacenti a u, ovvero quei vertici v tali per cui esiste un arco $(u,v) \in E$
- In particolare questo insieme di vertici è memorizzato come una lista
- L'ordine dei vertici nella lista è arbitrario

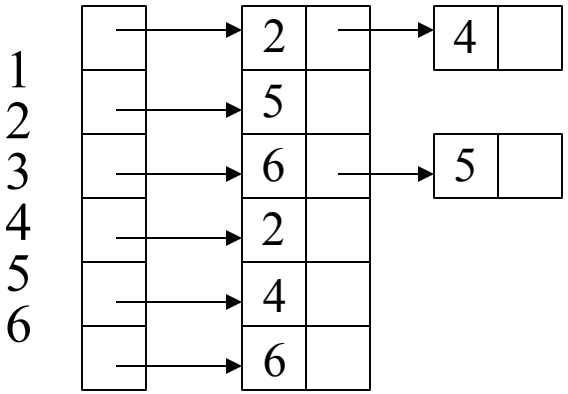
Grafo non orientato con liste di adiacenza





Grafo orientato con liste di adiacenza





Proprietà liste di adiacenza

- Se un grafo è orientato allora la somma delle lunghezze di tutte le liste di adiacenza è |E|
  - infatti per ogni arco (u,v) c'è un vertice v nella lista di posizione u
- Se un grafo non è orientato allora la somma delle lunghezze di tutte le liste di adiacenza è 2|E|
  - infatti per ogni arco (u,v) c'è un vertice v nella lista di posizione u e un vertice u nella lista di posizione v
- La quantità di memoria necessaria per memorizzare un grafo (orientato o non) è O(max(V,E)) = O(V+E)

Grafi pesati

- In alcuni problemi si vuole poter associare una informazione (chiamata *peso*) ad ogni arco
- Un grafo con archi con peso si dice *grafo pesato*
- Si dice che esiste una funzione peso che associa ad un arco un valore

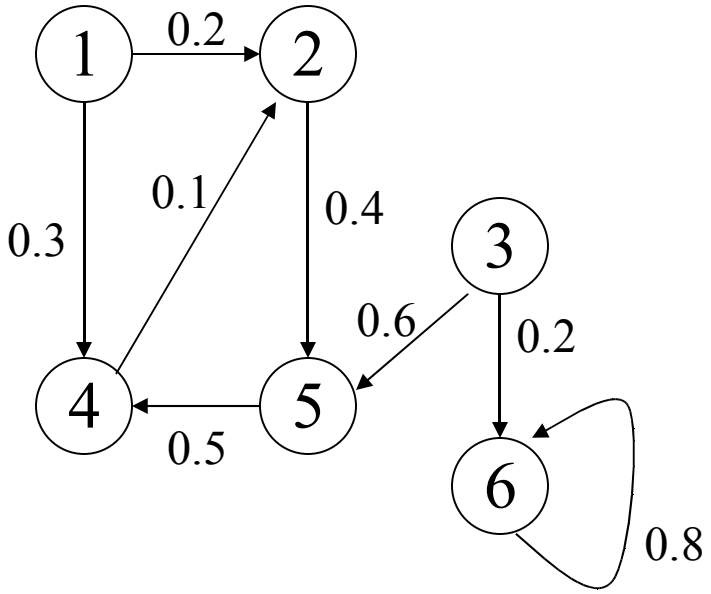
$$w : E \rightarrow \mathbf{R}$$

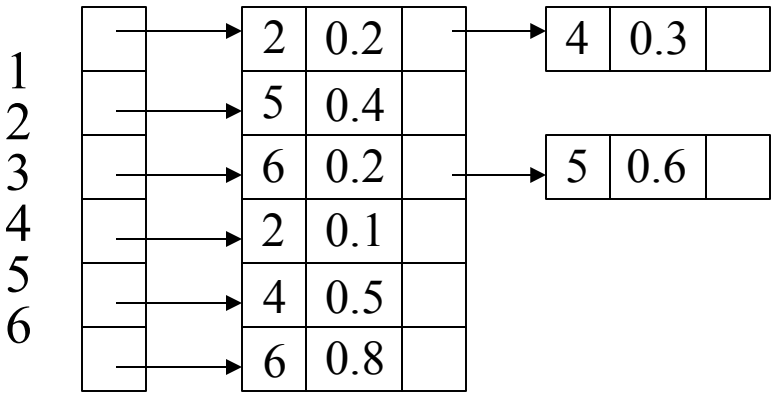
- Ovvero un arco (u,v) ha peso w(u,v)

Grafi pesati con liste di adiacenza

- Si memorizza il peso w(u,v) insieme al vertice v nella lista per il vertice u

Grafo orientato pesato con liste di adiacenza





Svantaggi liste di adiacenza

- Per sapere se un arco (u,v) è presente nel grafo si deve scandire la lista degli archi di u
- Se si deve accedere spesso a tutti i vertici adiacenti di un dato vertice e' il metodo piu' conveniente

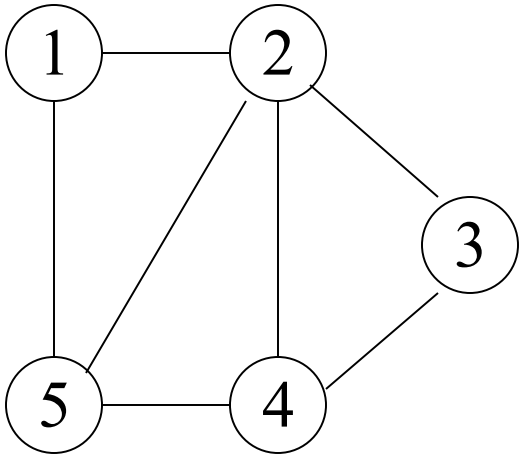
Matrici di adiacenza

- Per la rappresentazione con matrici di adiacenza si assume che i vertici siano numerati in sequenza da 1 a |V|
- Si rappresenta un grafo G=(V,E) con una matrice A=(a_{ij}) di dimensione |V|x|V| tale che:

  a_{ij}=1 se (i,j) $\in$ E

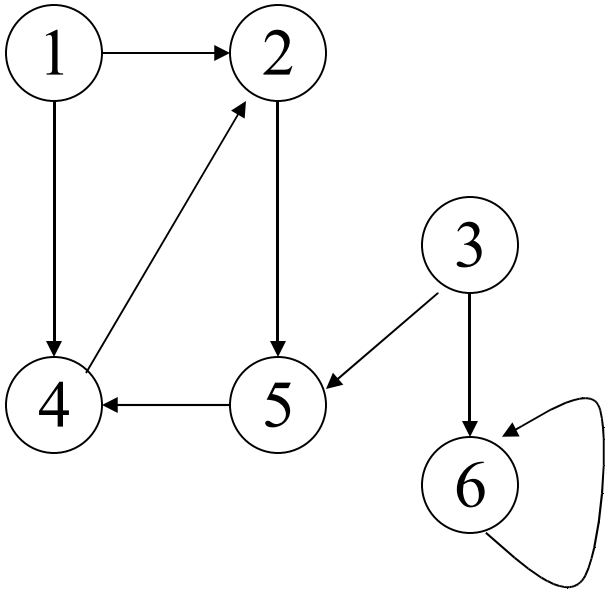
  a_{ij}=0 altrimenti

Grafo non orientato con matrice di adiacenza



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

Grafo orientato

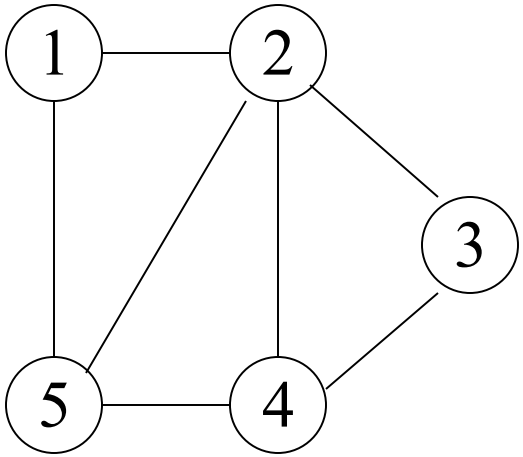


	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Proprietà matrice di adiacenza

- La rappresentazione di un grafo G=(V,E) con matrice di adiacenza richiede memoria $\Theta(V^2)$ indipendentemente dal numero di archi
- La matrice di adiacenza di un grafo non orientato è simmetrica ovvero $a_{ij} = a_{ji}$
- Per un grafo non orientato si può allora memorizzare solo i dati sopra la diagonale (diagonale inclusa), riducendo della metà lo spazio per memorizzare la matrice

Grafo non orientato con matrice di adiacenza

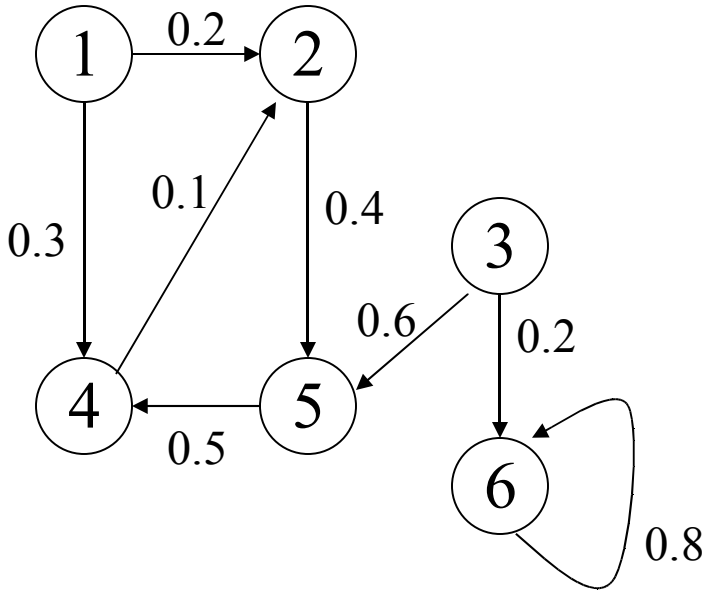


	1	2	3	4	5
1	0	1	0	0	1
2		0	1	1	1
3			0	1	0
4				0	1
5					0

Grafi pesati con matrici di adiacenza

- Si memorizza il peso nell'elemento a_{ij} invece di 1
- Se l'arco non esiste si indica con 0 o ∞ o NIL a secondo del problema

Grafo orientato pesato con matrice di adiacenza



	1	2	3	4	5	6
1	0	.2	0	.3	0	0
2	0	0	0	0	.4	0
3	0	0	0	0	.6	.2
4	0	.1	0	0	0	0
5	0	0	0	.5	0	0
6	0	0	0	0	0	.8

Vantaggi della matrice di adiacenza

- La rappresentazione con matrice di adiacenza è semplice
- Se il grafo è piccolo non vi è sostanziale differenza di efficienza con la rappresentazione con liste di adiacenza
- Per grafi non pesati si può rappresentare ogni singolo elemento della matrice non con una parola ma con un singolo bit