Alberi

Strutturare i dati….

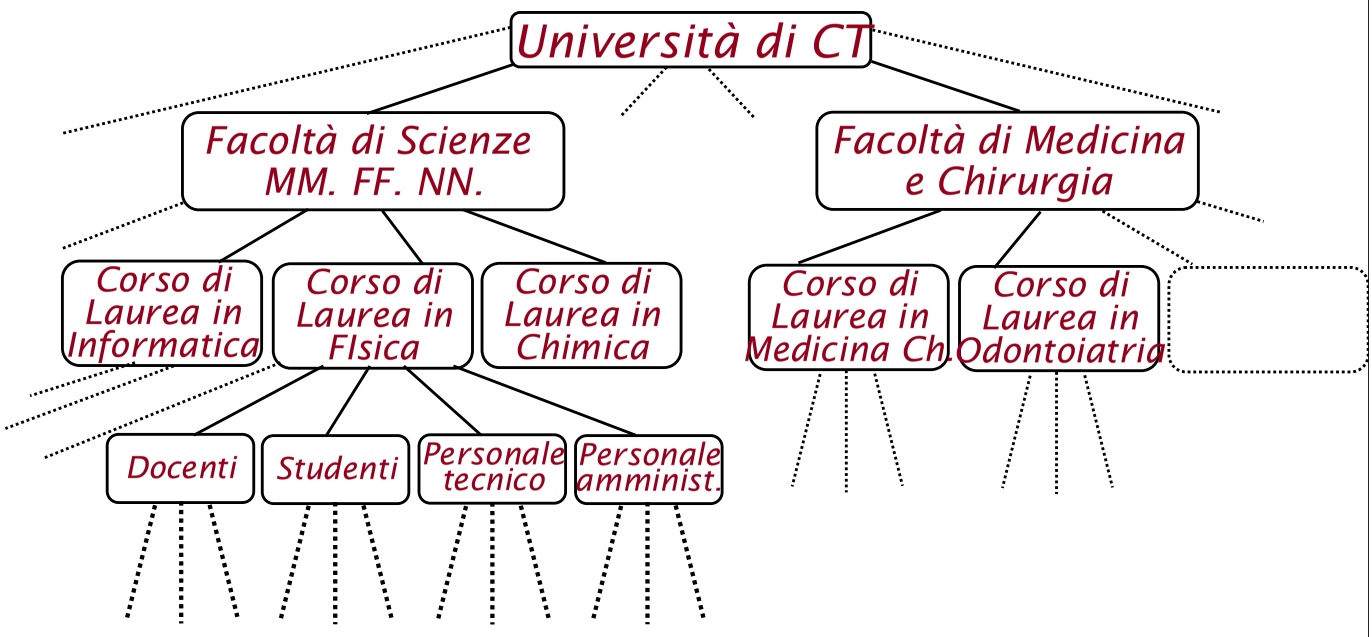
- alberi genealogici
- gerarchie di ereditarietà
  - ad es., in Java
- classificazioni di specie animali
- organizzazione del territorio
  - continente, nazione, regione, provincia ecc
- (alcuni) organigrammi
- file system
- domini Internet

Preliminari

- Array, liste concatenate, pile, code sono strutture dati **lineari**
- Difficile (ma possibile) utilizzarle per una **rappresentazione gerarchica** dei dati
- Una rappresentazione gerarchica dei dati è naturalmente possibile con una struttura dati chiamata ***albero***
- Struttura ispirata all'omonimo concetto esistente in natura
- Gli alberi in informatica sono tipicamente disegnati a testa in giù…

Esempio

Possiamo usare un albero per una rappresentazione gerarchica della struttura nostra Università.



Motivazioni

Qual è lo scopo di una rappresentazione gerarchica dei dati? Ovvero: perchè usare gli alberi?

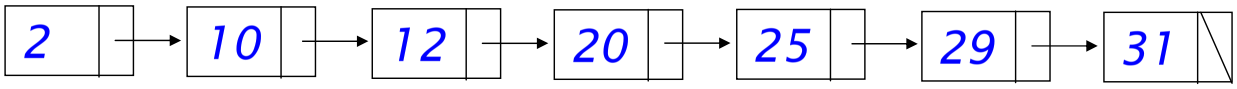
- Le operazioni di ricerca, inserimento e cancellazione sulle strutture dati lineari prendono tempo ***lineare***.
- Possono essere più efficienti su una struttura gerarchica come un albero

Motivazioni — esempio

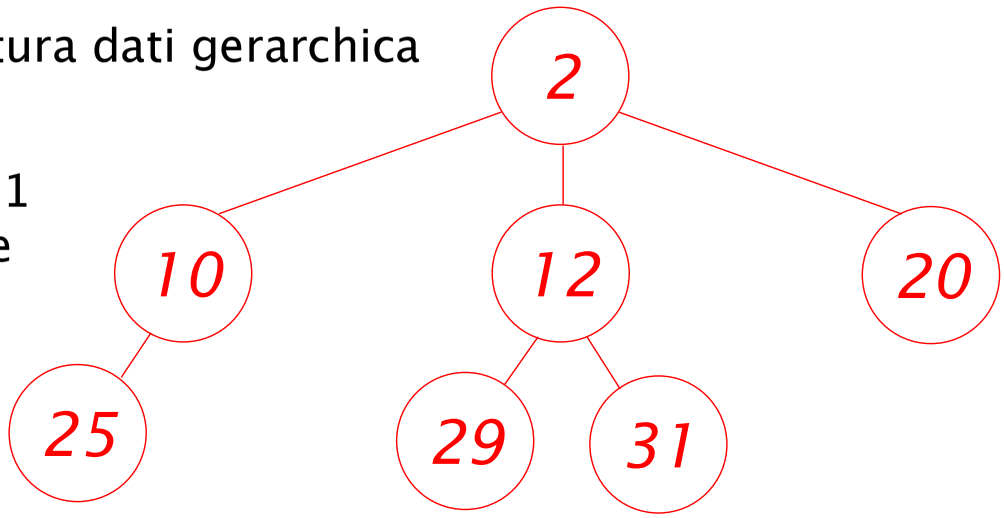
Consideriamo l'insieme di numeri naturali

{ 2, 10, 12, 20, 25, 29, 31 }

Lo rappresentiamo con la struttura dati lineare lista concatenata:



e con la struttura dati gerarchica albero:



Ricerchiamo 31 su ambedue le strutture.

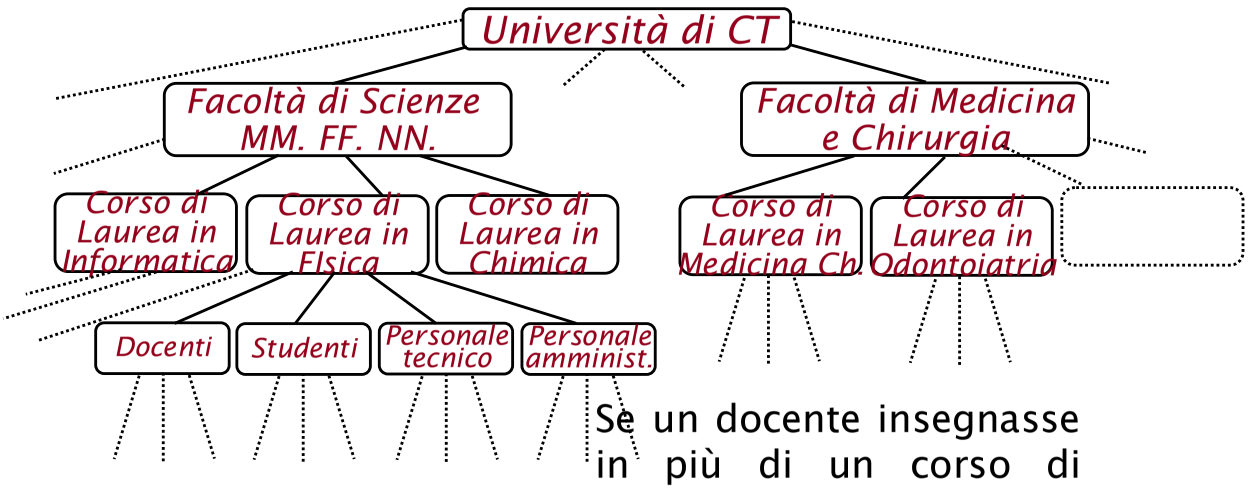
Complessità della ricerca?

Definizioni

- Un *albero* è un insieme (eventualmente vuoto) di nodi connessi mediante *rami:*
  - vertice è sinonimo di nodo
  - arco è sinonimo di ramo
- Ogni nodo (**eccetto il nodo radice**) è connesso tramine un ramo a un altro nodo che ne è il padre, e di cui rappresenta un figlio
- Quindi la radice è l'unico nodo privo di padre

Esercizio

Individuare nodi, archi, radice, nodi padre, nodi figli nel seguente albero esempio.



Se un docente insegnasse in più di un corso di laurea??

Definizione ricorsiva di albero

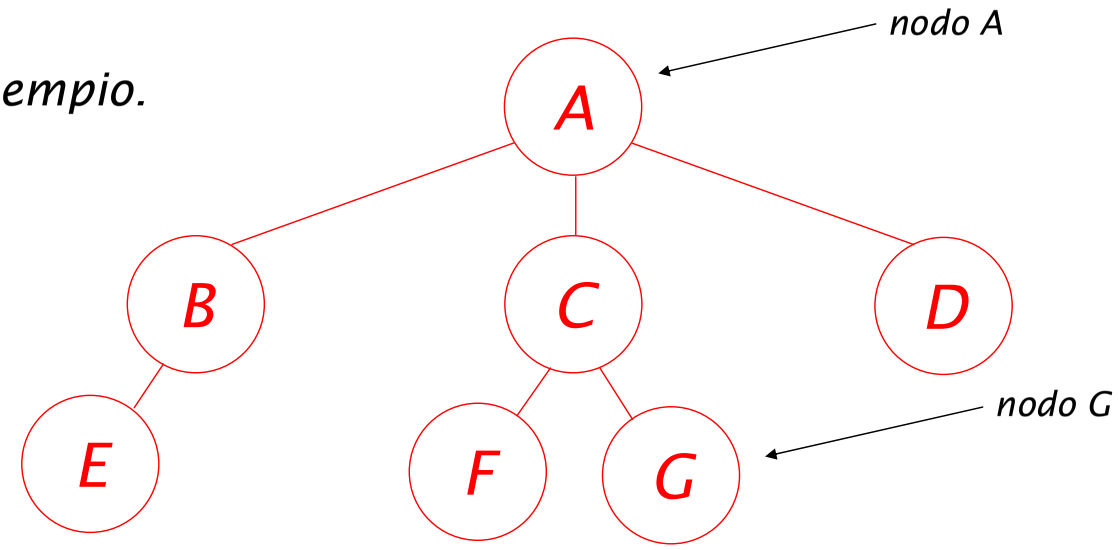
1. Una struttura vuota è un albero (vuoto)
2. Se $\mathcal{T}_1$, $\mathcal{T}_2$, ... , $\mathcal{T}_n$ sono alberi privi di nodi comuni, e X e' un nodo, allora inserendo $\mathcal{T}_1$, $\mathcal{T}_2$, ... , $\mathcal{T}_n$ come sotto-alberi di X, si ottiene un albero

Chiave o Etichetta

Ogni nodo può contenere informazioni di vario tipo, rappresentate dalla sua chiave, o etichetta.

Spesso si indica un nodo mediante la sua etichetta.

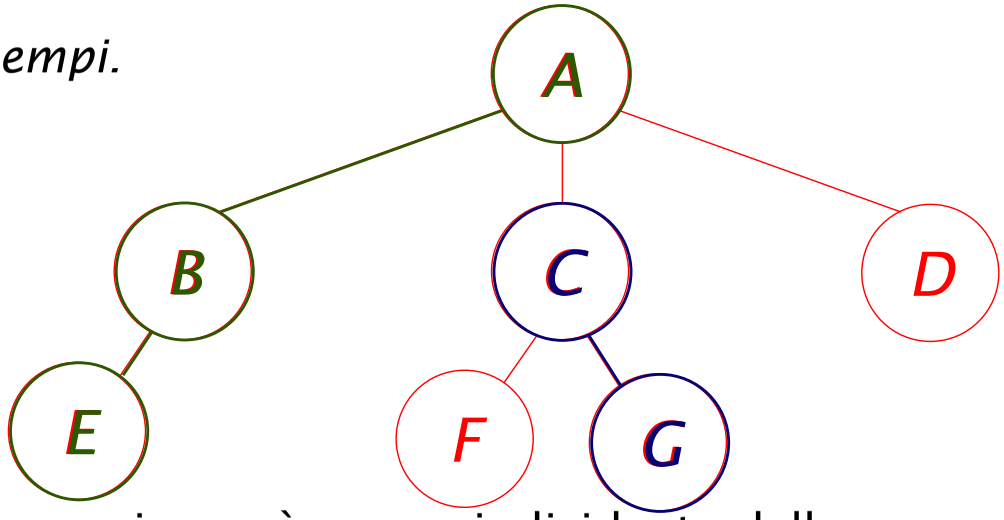
Esempio.



Cammino

Una sequenza di nodi e rami tali che ciascun nodo (tranne l'ultimo) sia padre del successivo si chiama cammino.

Esempi.

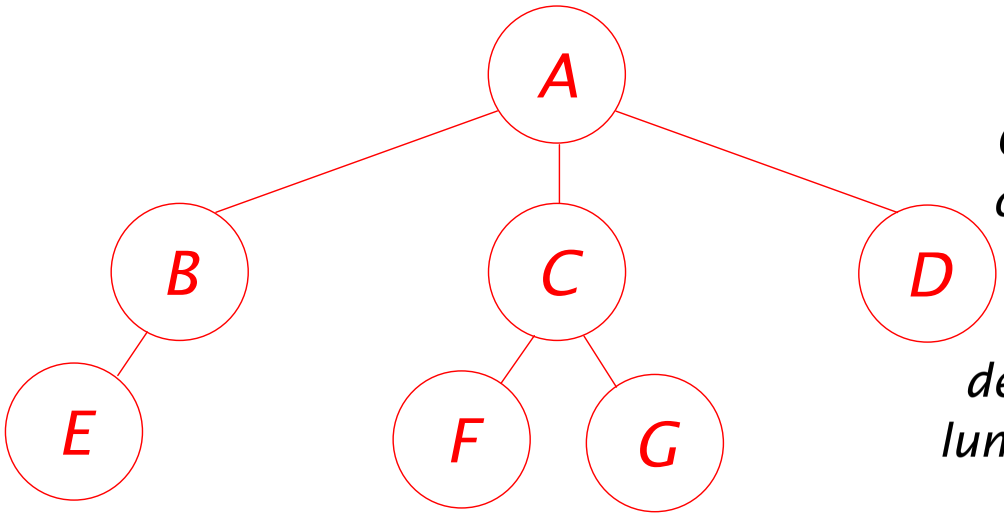


Un cammino può essere individuato dalla sequenza delle etichette dei suoi nodi.

Gli esempi riguardano i cammini (A,C,G) ed (A,B,E) .

Lunghezza di un cammino

Il numero di archi coinvolti in un cammino si chiama lunghezza del cammino.



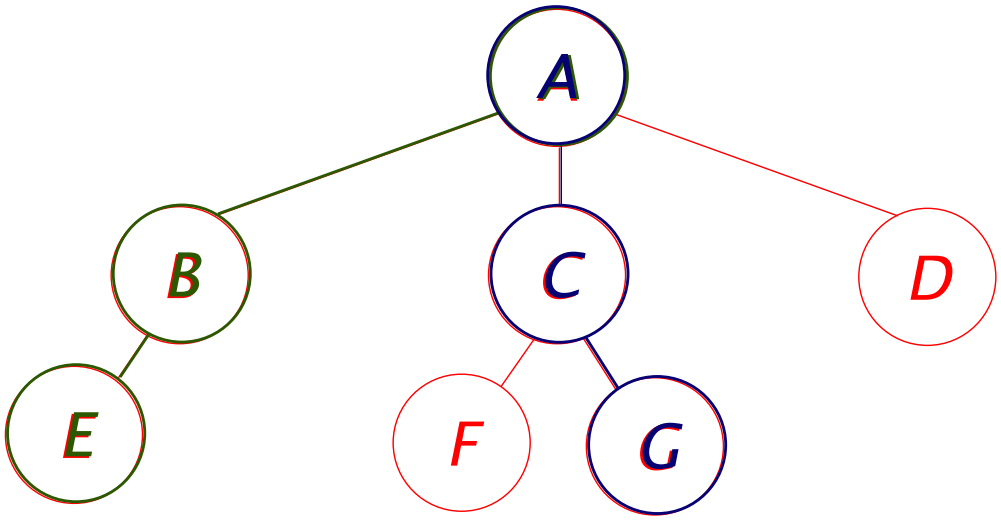
Ogni nodo costituisce un cammino degenere di lunghezza 0.

Il cammino C,F ha lunghezza 1; A,B,E ha lunghezza 2.

In generale, un cammino con K nodi ha K–1 archi, ossia ha lunghezza K–1.

Cammino caratteristico di un nodo

Per ciascun nodo esiste uno ed un solo cammino che lo collega alla radice; tale cammino si chiama cammino caratteristico del nodo.

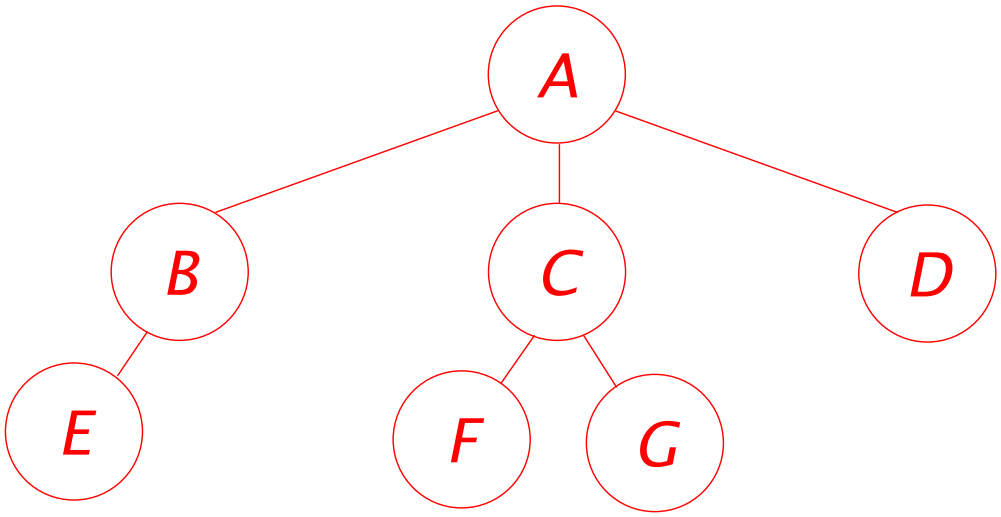


A,B,E è il cammino caratteristico del nodo E.

A,C,G è il cammino caratteristico del nodo G.

Livello di un nodo

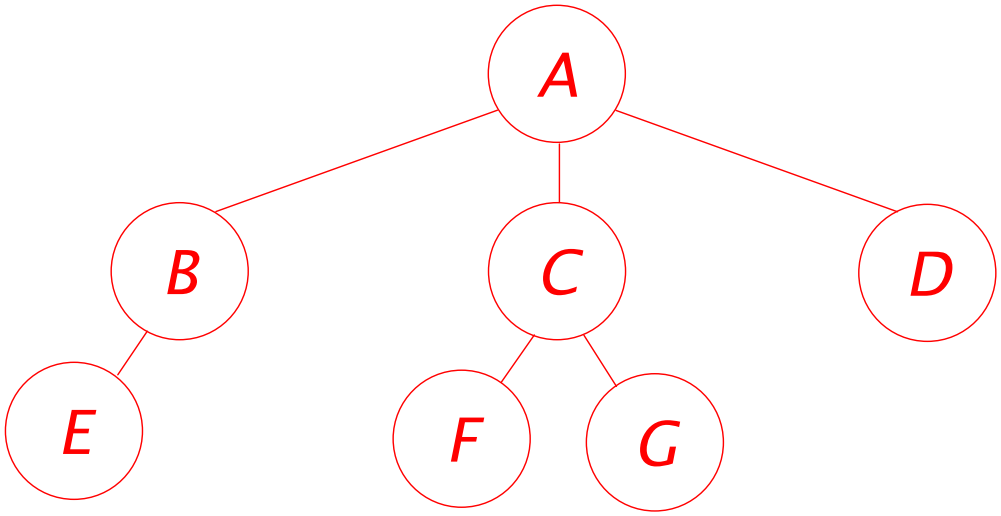
La lunghezza del cammino caratteristico di un nodo si chiama livello del nodo.



Il nodo A ha livello 0; il nodo C ha livello 1; il nodo E ha livello 2.

Antenati e Discendenti

Se il cammino caratteristico di un nodo Y contiene un nodo X, diciamo che X è antenato di Y e Y è discendente di X.

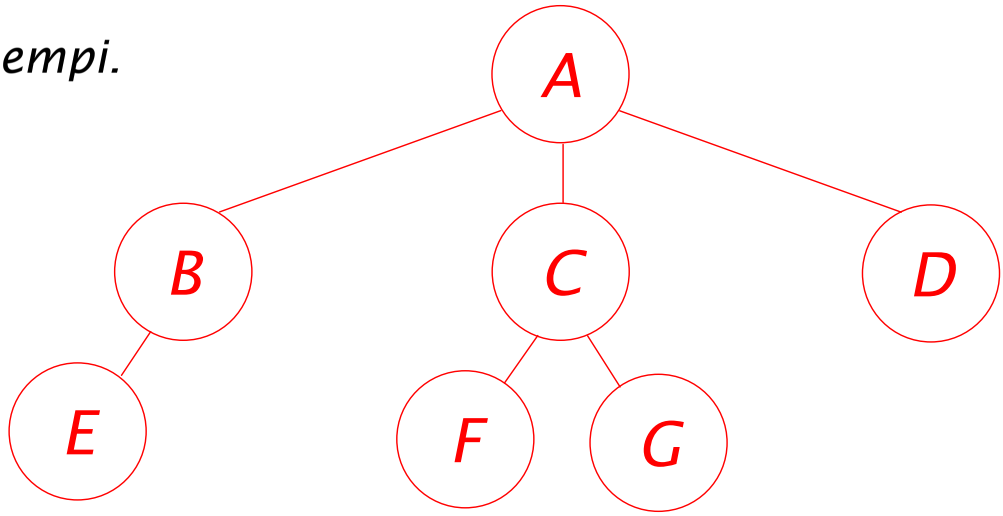


Trovare gli antenati di F e i discendenti di A.
A è un antenato di E ? D è un antenato di E ?

Tipi di nodo

- Un nodo senza figli si chiama foglia
- Un nodo con almeno un figlio si chiama nodo interno
- Il nodo interno senza padre si chiama radice
- Due o più nodi con lo stesso padre si chiamano fratelli

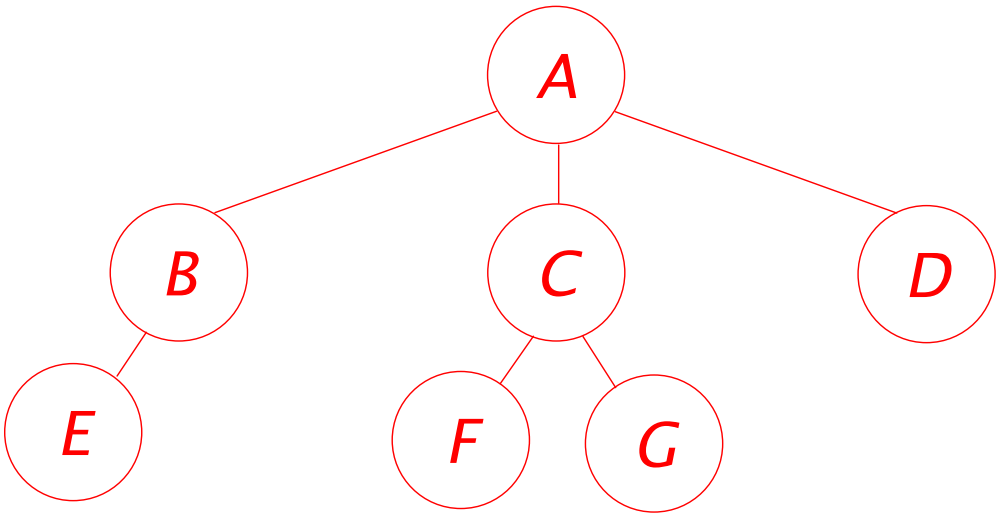
Esempi.



Altezza di un nodo

La lunghezza del più lungo cammino da un nodo ad una foglia si chiama altezza del nodo.

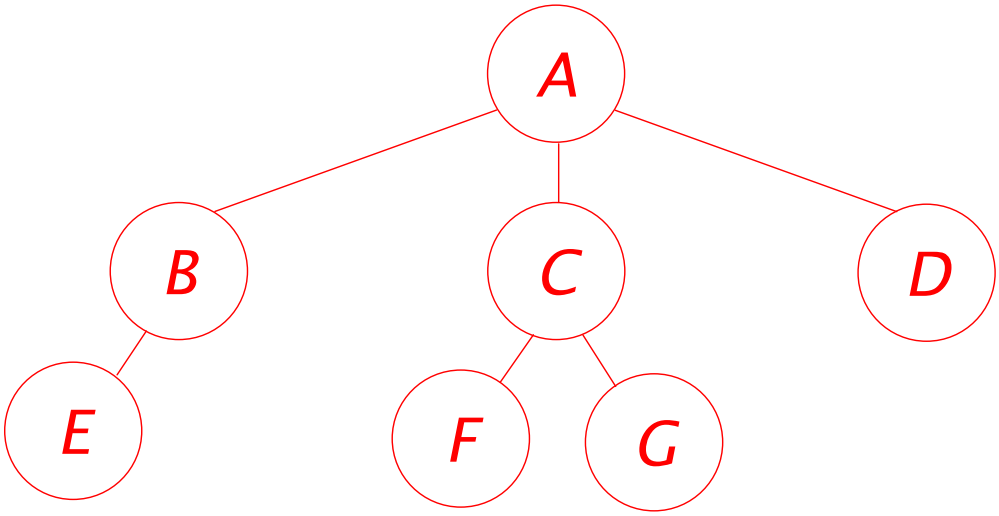
Tutte le foglie hanno altezza 0.



Altezza di un albero

L'altezza di un albero si definisce equivalentemente come:

- l'altezza della sua radice, o
- il massimo livello delle sue foglie

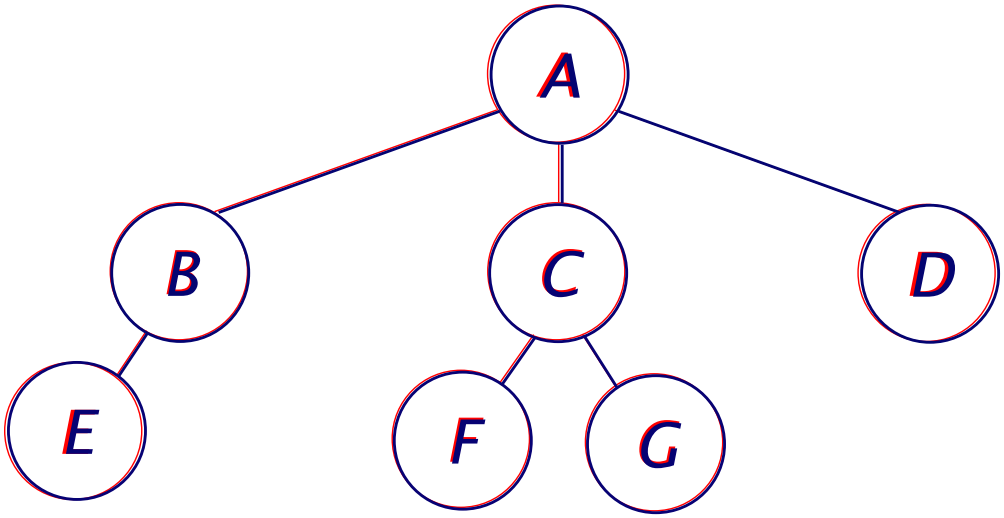


Quest'albero ha altezza … 2.

Sotto-albero

Se T è un albero e X è un suo nodo, l'insieme dei nodi di T contenente X e tutti i suoi discendenti si chiama sotto-albero di T.

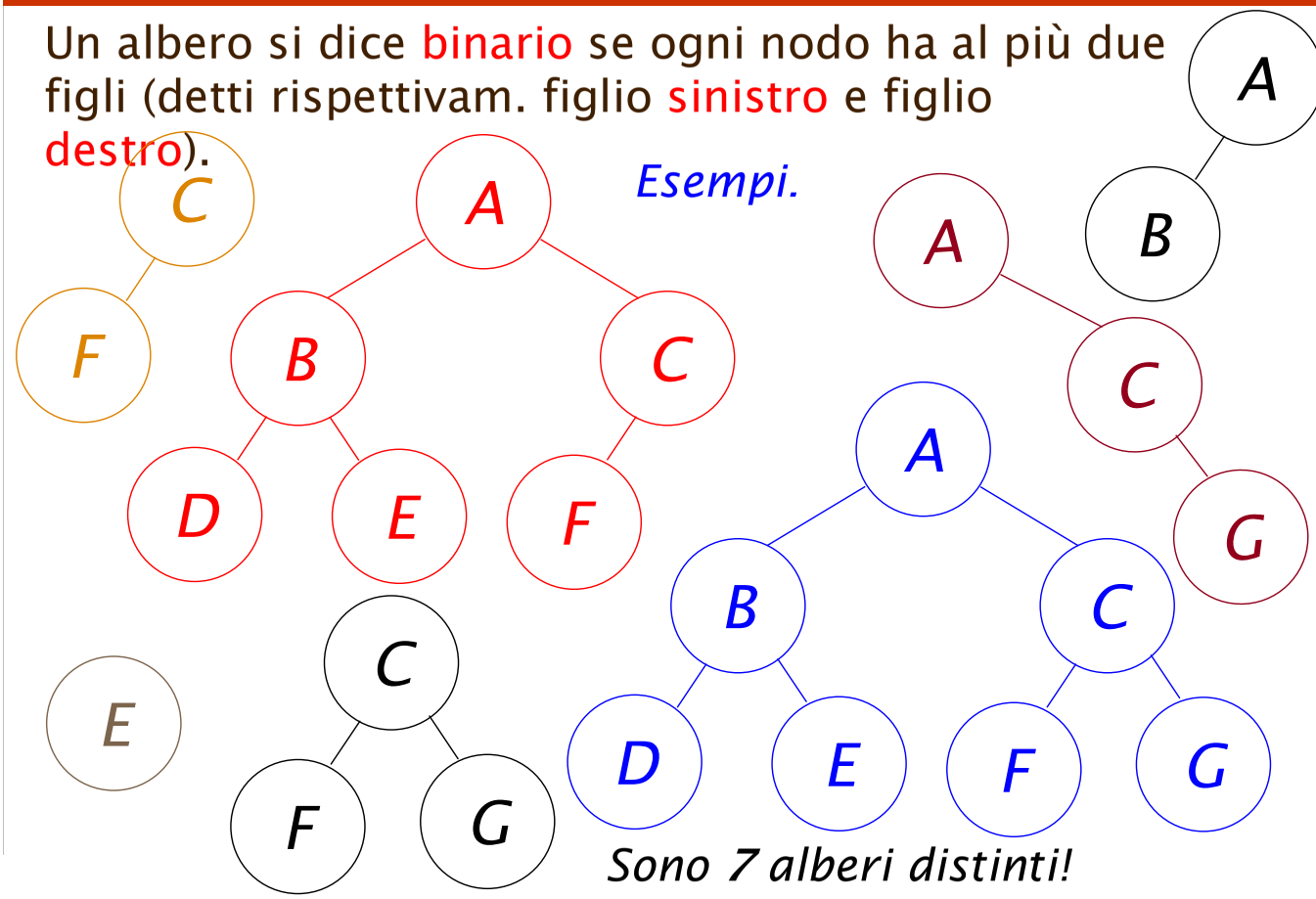
X si chiama radice del sotto-albero.



Albero binario

Un albero si dice binario se ogni nodo ha al più due figli (detti rispettivam. figlio sinistro e figlio destro).

Esempi.



Sono 7 alberi distinti!

Albero binario completo

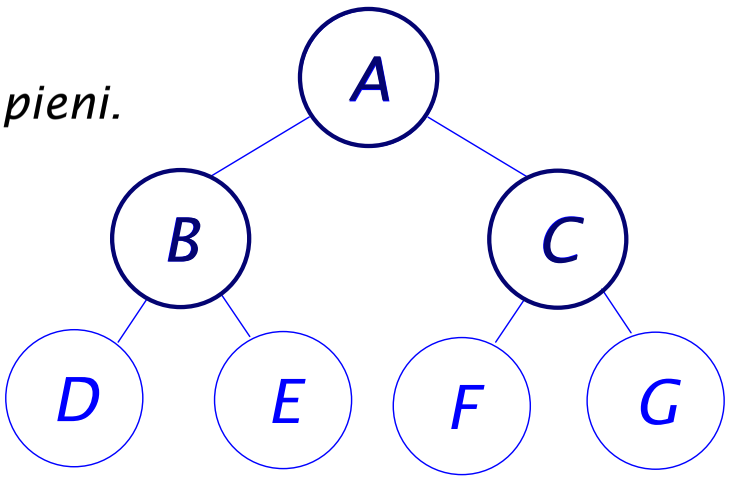
In un albero binario, un nodo avente due figli si dice pieno.

Un albero binario di altezza **h** si dice completo se tutti i nodi di livello minore di **h** sono pieni.

Esempio.

Trovare i nodi pieni.

È completo?



Livello 0

Livello 1

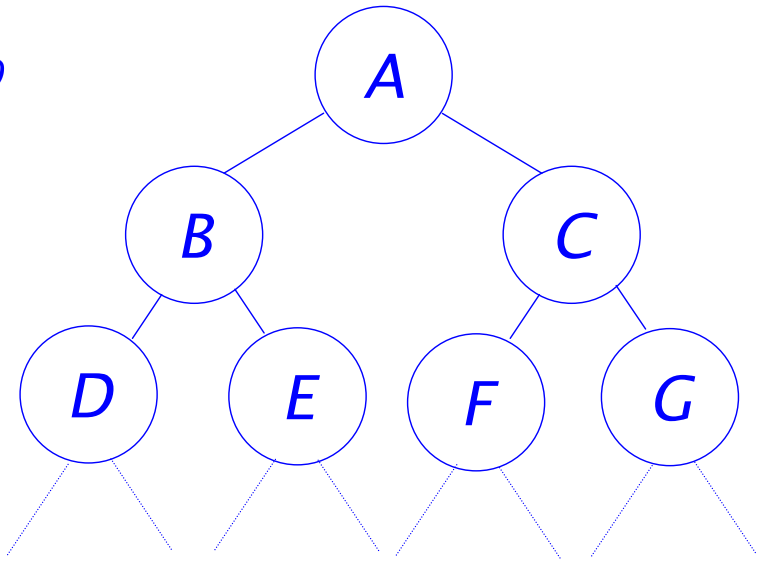
Livello 2

nodi in un albero binario completo
(livello per livello)

$2^0 = 1$ *nodo al livello* 0

$2^1 = 2$ *nodi al livello* 1

$2^2 = 4$ *nodi al livello* 2



Livello 0

Livello 1

Livello 2

2^i *nodi al livello* i

Livello i

Relazioni fra altezza e numero di nodi
in un albero binario completo

Consideriamo un albero binario completo avente altezza h ed n nodi.

Possiamo esprimere *n* in funzione di *h*.

Teorema 1. $n = 2^{h+1} - 1$

Dim. Per induzione su h. Caso base: h=0.
Caso ind.: tesi valga per h–1, si dimostri per h.

(*il lucido precedente può aiutare l'intuizione*)

Possiamo esprimere *h* in funzione di *n*.

Teorema 2. $h = \lg(n+1) - 1$

Dim. Dal teorema 1 mediante passaggio ai logaritmi.

Attraversamento di albero

Per attraversamento o visita di un albero si intende l'ispezione dei nodi dell'albero in modo che tutti i nodi vengano ispezionati una ed una sola volta.

Quindi, un attraversamento di un albero definisce un ordinamento totale tra i nodi dell'albero

- in base alla loro posizione nell'albero,
- NON in base alle loro etichette (che possono essere non ordinabili)

Trattandosi di ordinamento totale, ogni nodo ha un precedente e un successivo all'interno di un attraversamento.

Esempi di attraversamento di albero

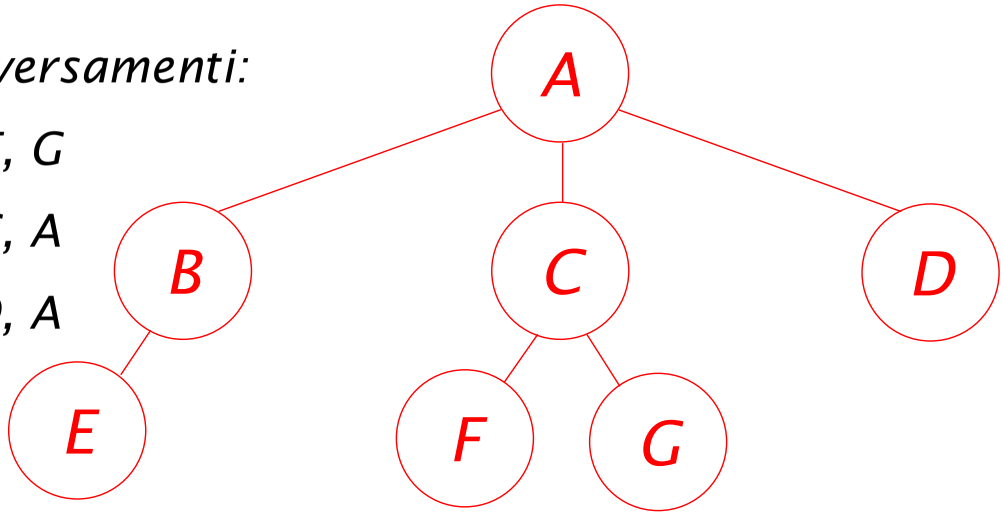
Possibili attraversamenti:

A, B, C, D, E, F, G

B, C, D, E, F, G, A

E, F, G, B, C, D, A

. . .



- Ogni permutazione dei nodi definisce un diverso attraversamento
- Certi attraversamenti procedono a salti e in pratica sono poco utili

Attraversamento di albero binario: Preorder

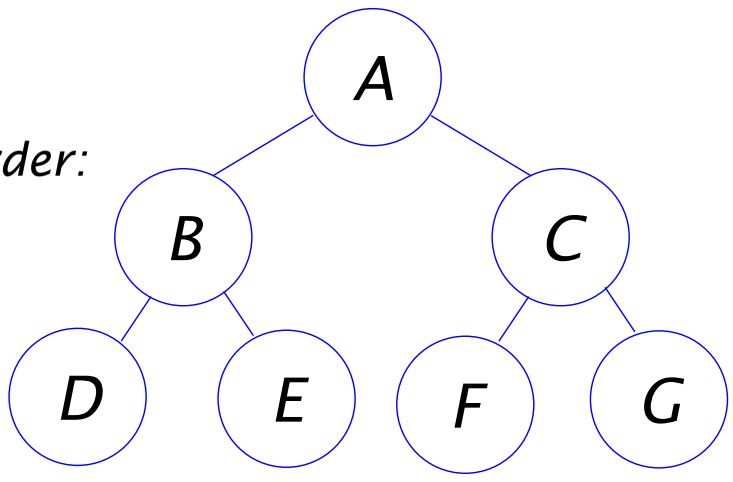
Gli attraversamenti fondamentali di un albero binario sono i seguenti tre:

- ***Preorder***: visitare la radice, attraversare ricorsivamente il sotto-albero sinistro della radice e infine il sotto-albero destro della radice

Esempio.

Attraversamento preorder:

A, B, D, E, C, F, G



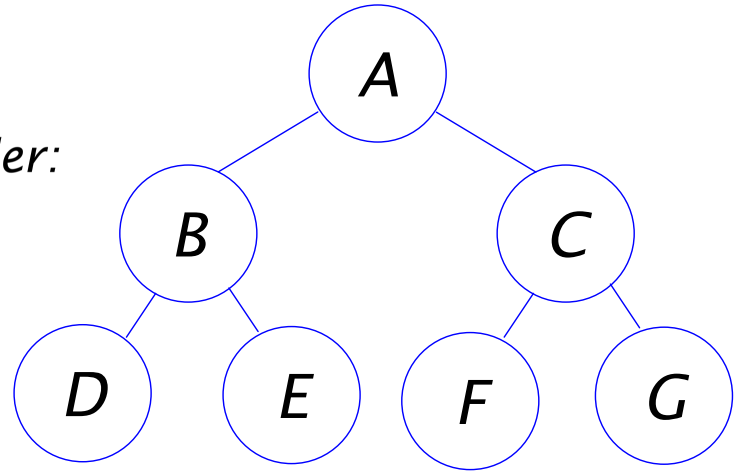
Attraversamento di albero binario: Inorder

- ***Inorder***: attraversare ricorsivamente il sotto-albero sinistro della radice, visitare la radice, e infine attraversare ricorsivamente il sotto-albero destro della radice

Esempio.

Attraversamento inorder:

D, B, E, A, F, C, G



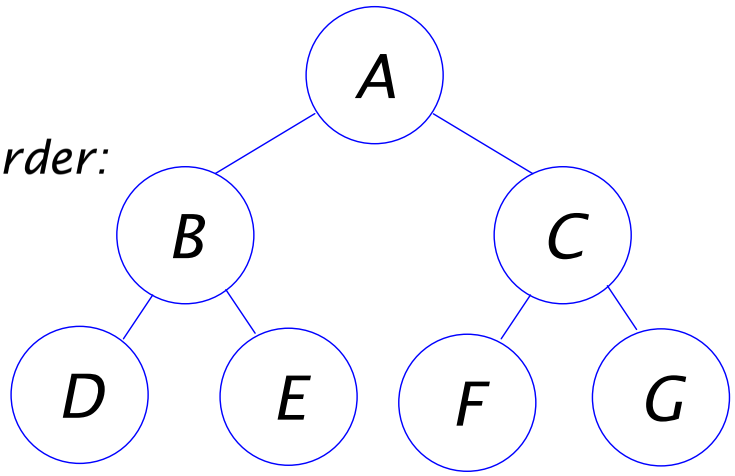
Attraversamento di albero binario: Postorder

- ***Postorder***: attraversare ricorsivamente il sotto-albero sinistro della radice, il sotto-albero destro della radice, e infine visitare la radice.

Esempio.

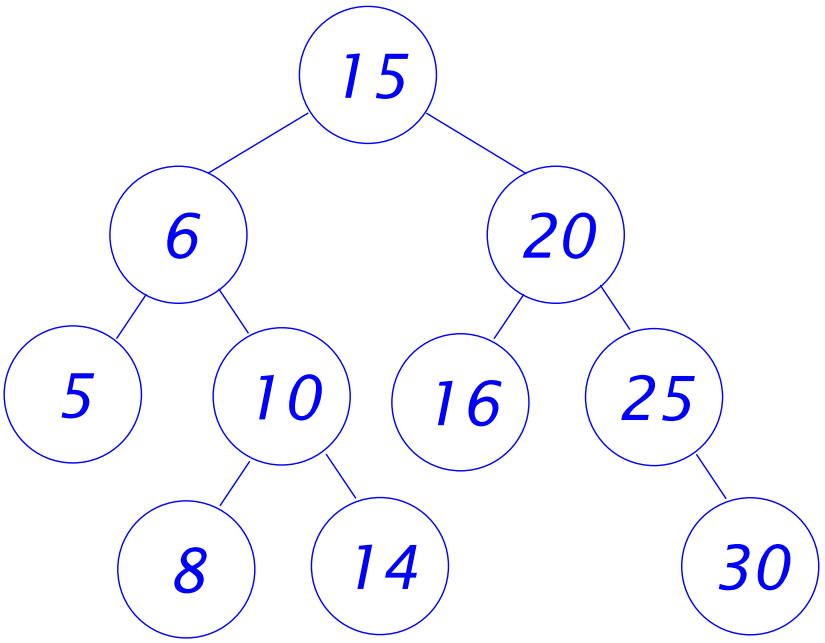
Attraversamento postorder:

D, E, B, F, G, C, A



Esercizio

Simulare sul seguente albero, che vedremo nelle prossime lezioni essere un BST (Binary Search Tree) i metodi di attraversamento visti.

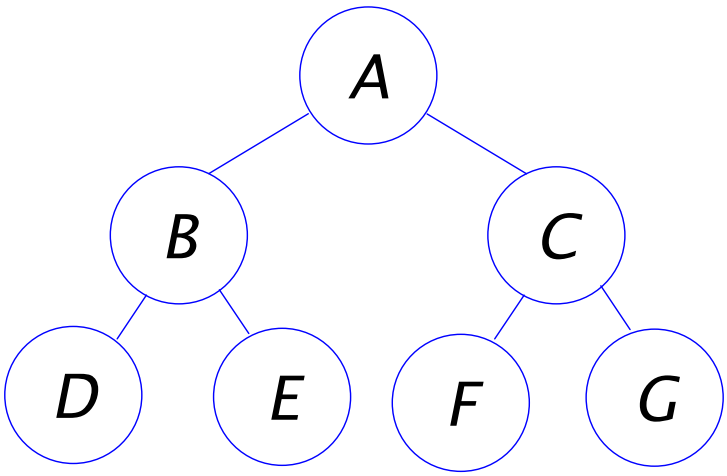


Albero binario bilanciato

Un albero binario si dice bilanciato (in altezza) se, per ogni nodo, la differenza di altezza dei due sottoalberi è zero o uno.

Esempio.

È bilanciato?



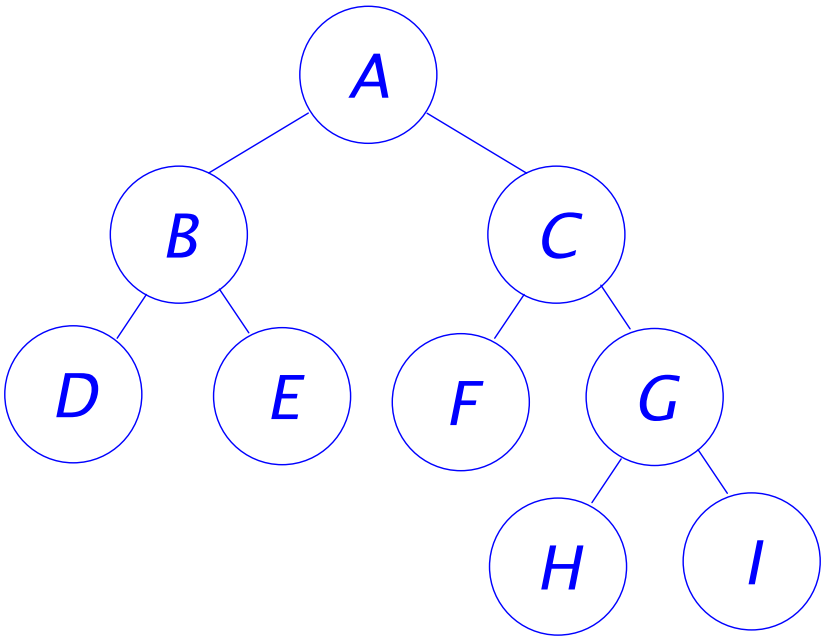
Un albero binario completo è bilanciato.

Albero binario bilanciato

Un albero binario si dice bilanciato (in altezza) se, per ogni nodo, la differenza di altezza dei due sottoalberi è zero o uno.

Esempio.

È bilanciato?



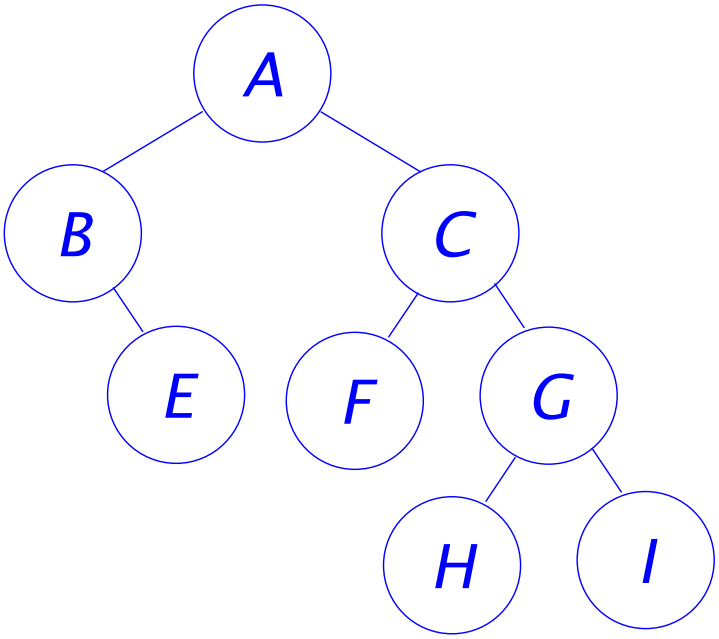
SI'!

Albero binario bilanciato

Un albero binario si dice bilanciato (in altezza) se, per ogni nodo, la differenza di altezza dei due sottoalberi è zero o uno.

Esempio.

È bilanciato?



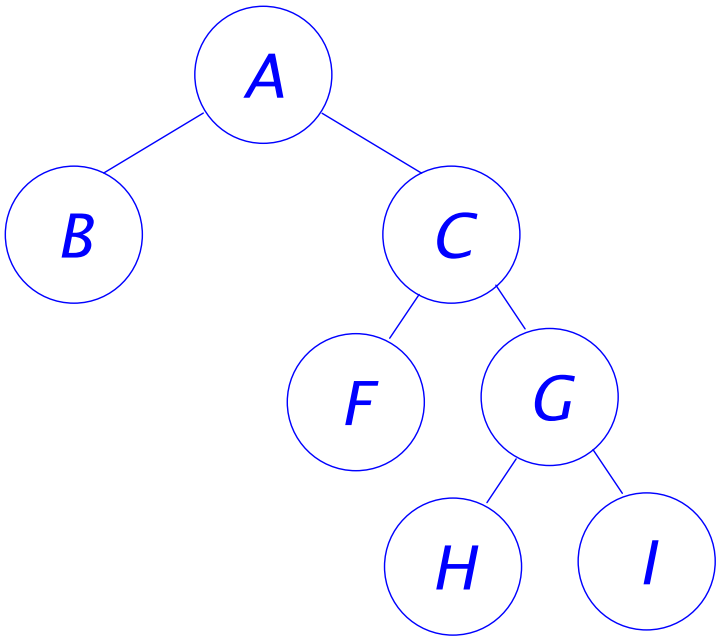
SI'!

Albero binario bilanciato

Un albero binario si dice bilanciato (in altezza) se, per ogni nodo, la differenza di altezza dei due sottoalberi è zero o uno.

Esempio.

È bilanciato?



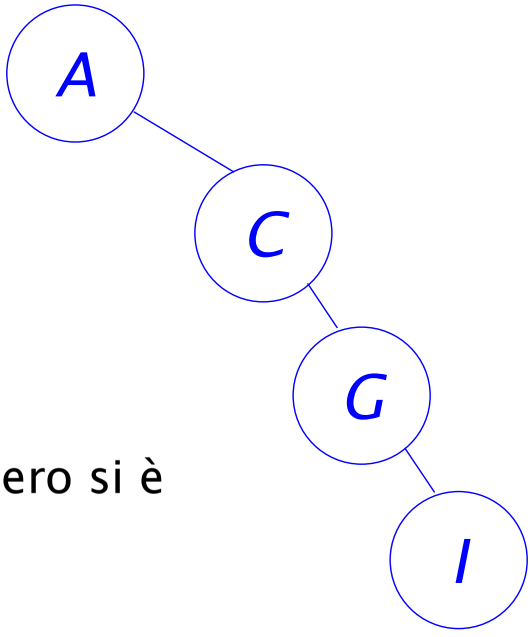
NO!

Albero binario bilanciato

Un albero binario si dice bilanciato (in altezza) se, per ogni nodo, la differenza di altezza dei due sottoalberi è zero o uno.

Esempio.

È bilanciato?



Massimo sbilanciamento: l'albero si è ridotto a una struttura lineare.

operazioni sugli alberi

operazioni più importanti

- `element(v)`: restituisce l'elemento memorizzato nel nodo `v`
- `root()`: restituisce la radice dell'albero
- `children(v)`: restituisce i figli di `v`
- `isLeaf(v)`: restituisce `true` se `v` è una foglia
- `isRoot(v)`: restituisce `true` se `v` è la radice

- `parent(v)`: restituisce il genitore del nodo `v`

operazioni sugli alberi

Altre operazioni possibili:

- `livello di un nodo`
- `altezza dell'albero`
- `# nodi`
- `# foglie`
- `arità`
  - max # di figli di un nodo dell'albero
- `isEmpty`
  - `true` se l'albero ha zero nodi

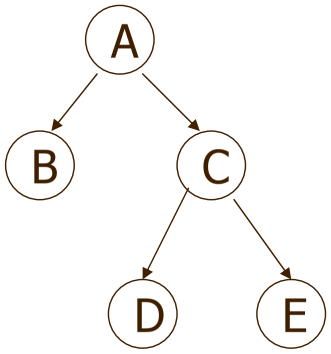
Alberi binari: rappr. sequenziale

Gli alberi binari possono anche essere implementati utilizzando degli array di lunghezza predefinita, nella seguente maniera:

ogni nodo v è memorizzato in posizione p(v)

- se v è la radice allora p(v)=1 (indice 0 in Java, C, C++)
- se v è il figlio sinistro di u allora p(v)=2p(u)
- se v è il figlio destro di u allora p(v)=2p(u)+1

Per gli alberi non completi serve un *tag* booleano che indica se il nodo esiste



1	A	true
2	B	true
3	C	true
4	0	false
5	0	false
6	D	true
7	E	true

```
Left(i)=2

Right(i)=2i+1

Parent(i)=(int)(i/2)
```

Alberi binari: rappr. sequenziale

Si tratta di un'implementazione statica:

- è necessaria una stima del numero massimo di nodi dell'albero
- può portare a spreco di risorse
- nel caso peggiore, un albero con n nodi richiede un vettore con 2^n-1 elementi (se l'albero degenera in una catena "destra")

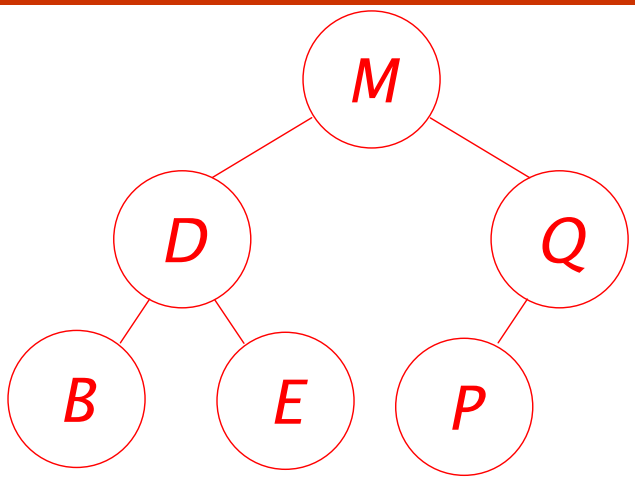
Alberi Binari: rappresentazione collegata mediante array

Ciascuna cella dell'array rappresenta un nodo.

Ciascuna cella è una struttura con 3 campi:

1. L'etichetta del nodo che la cella rappresenta
2. L'indice della cella dell'array che rappresenta il nodo figlio sinistro
3. L'indice della cella dell'array che rappresenta il nodo figlio destro

Esempio

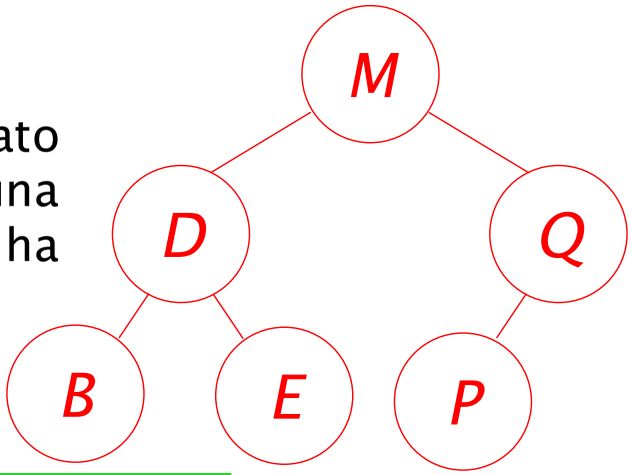


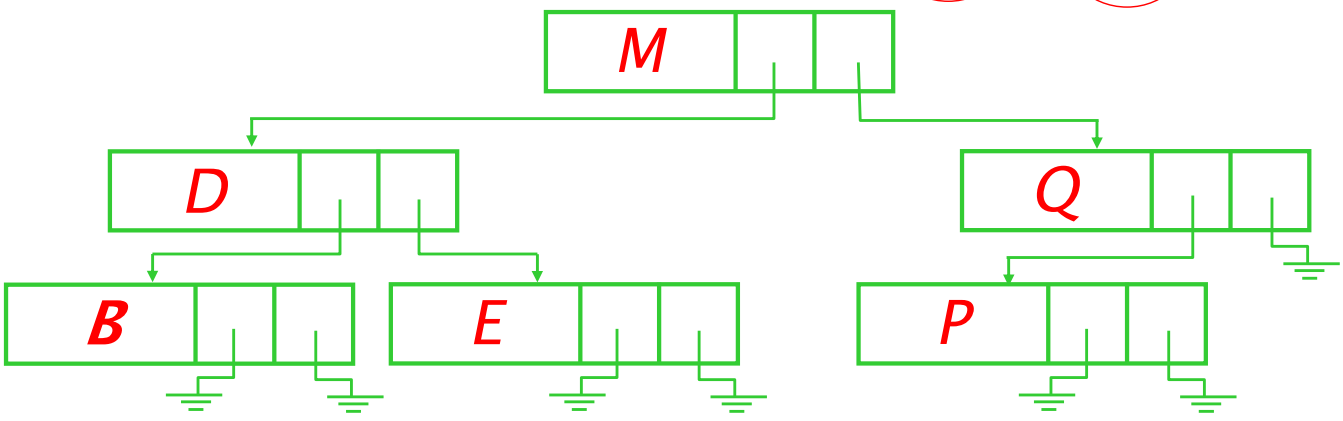
5	2	-1	-1	4	-1	-1	-1	-1	-1	3	1
M		E		Q		B		P		D	
0		1		2		3		4		5	

Tipicamente, la radice sta nella prima cella e –1 indica il figlio nullo.

Alberi binari implementazione dinamica

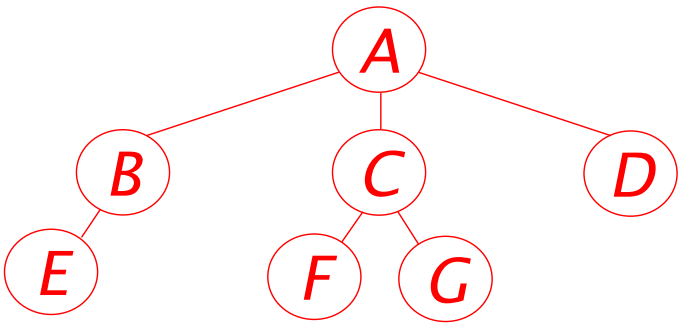
Può essere implementato dinamicamente mediante una struttura in cui ogni nodo ha due riferimenti ai nodi figli.

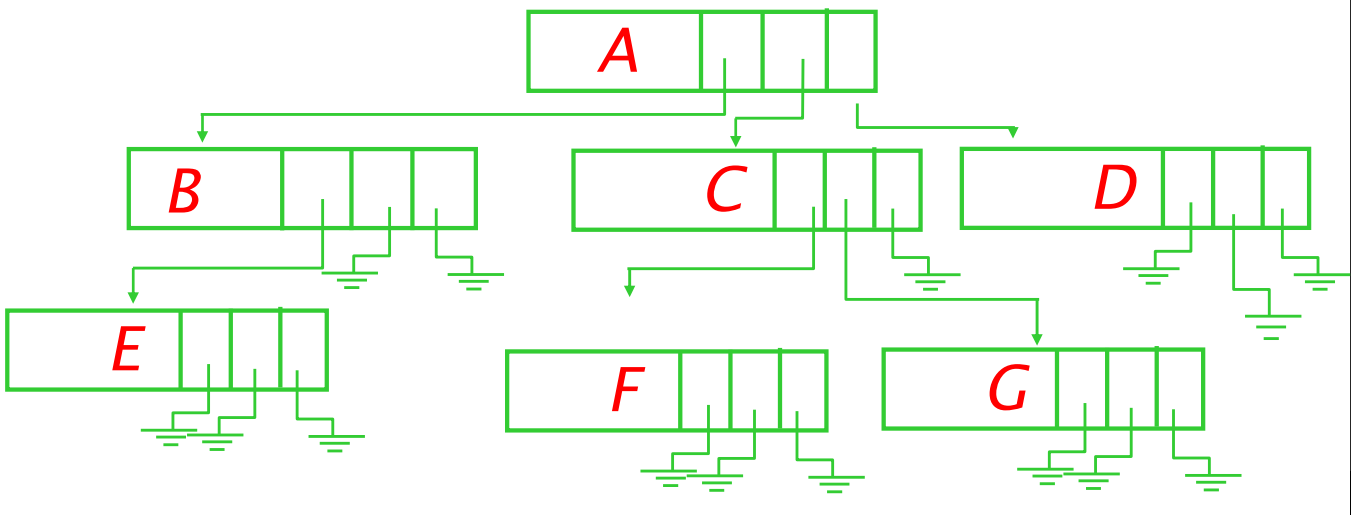




Implementazione ADT albero

Per alberi *n-ari*, si potrebbe quindi considerare una struttura in cui ogni nodo oltre al campo info ha un numero di riferimenti pari all'arietà dell'albero (cioè *n*).

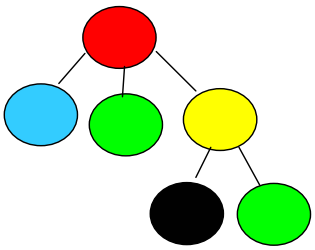


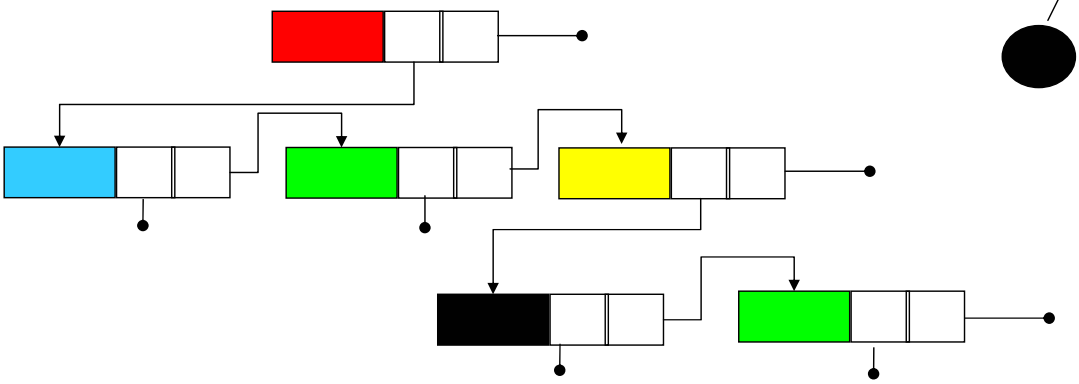


Alberi n-ari: impl. dinamica

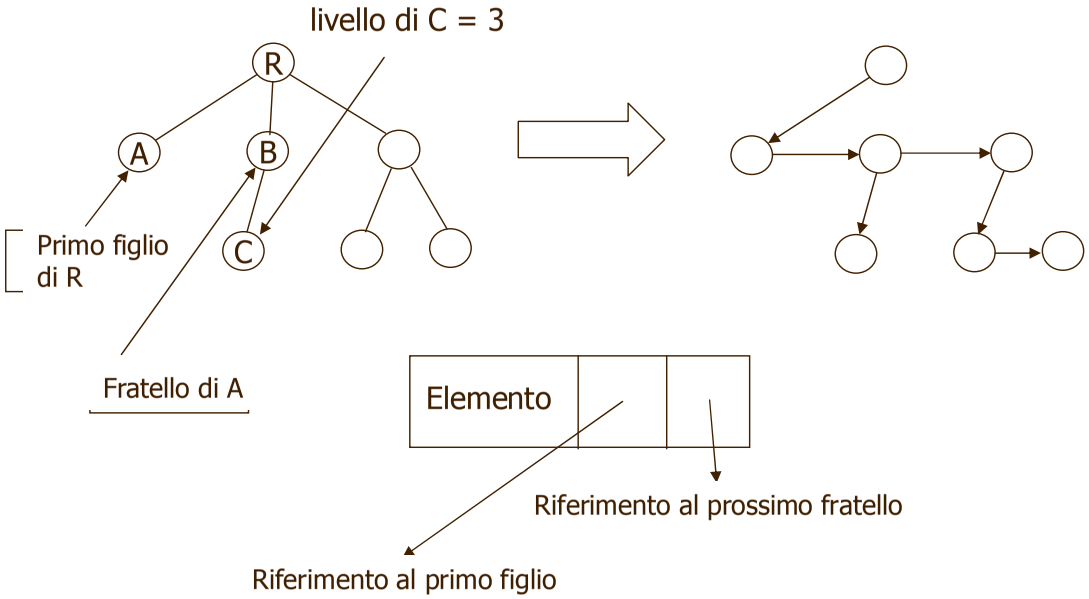
Un'implementazione più conveniente che prescinde dal numero massimo di figli soprattutto quando il numero di possibili "figli" di ciascun nodo è superiore a 2, è una struttura in cui ogni nodo ha sempre solo due riferimenti:

- il primo figlio (a sinistra);
- il primo fratello (a destra);





Alberi n-ari: impl. dinamica



Codice per implementazione dinamica

(nodo di un albero binario di interi)

```
public class IntBTNodo
{ protected int key;       // è un BST di numeri interi
  protected IntBSTNodo left, right;

  public IntBTNodo()
  {    left = right = null;
  }

  public IntBTNodo(int val)
  {    this(val, null, null);
  }
  ...
}
```

Codice per implementazione dinamica
(nodo di un albero binario di interi)

```
public class IntBTNodo
{
  ...

  public IntBTNodo(int val, IntBSTNodo lt, IntBSTNodo rt)
  {    key = val; left = lt; right = rt;
  }

  public int visit()
  {    return key;
  }

}
```

Codice per implementazione dinamica
(BST di interi)

```
public class IntBT
{ protected IntBTNodo root;

  public IntBST()
  {    root = null;
  }

  // metodi per
  // attraversamento (preorder, inorder, postorder),
  // ricerca, inserimento, cancellazione

}
```

non modificano l'albero

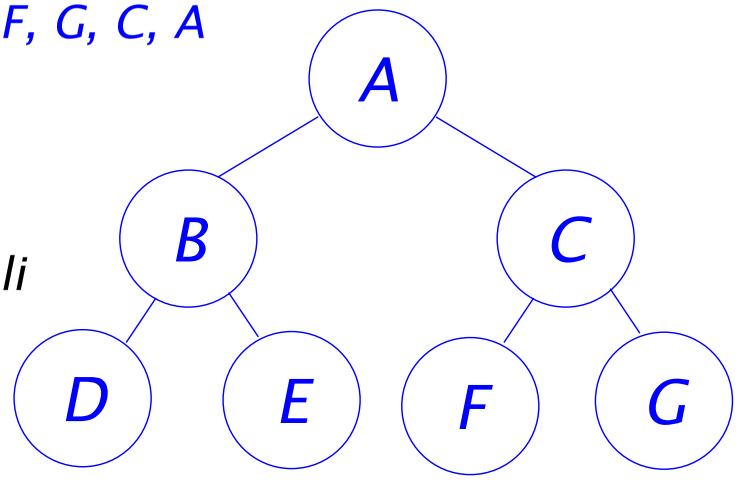
modificano l'albero

Attraversamento di un albero binario
(non necessariamente BST)

Abbiamo visto 3 attraversamenti per un albero binario.

- Preorder *A, B, D, E, C, F, G*
- Inorder *D, B, E, A, F, C, G*
- Postorder *D, E, B, F, G, C, A*

Come implementare tali attraversamenti?



Implementazione ricorsiva per Preorder

```
protected void preorder(IntBSTNodo p)
{       if (p != null)
        {       p.visit();          // da gestire !!
                preorder(p.left);
                preorder(p.right);
        }
}
```

La vera potenza di questo e dei due metodi che seguono sta nell'uso della ricorsione.

Implementazione ricorsiva per Inorder

```
protected void inorder(IntBSTNodo p)
{       if (p != null)
        {       inorder(p.left);
                p.visit();              // da gestire !!
                inorder(p.right);
        }
}
```

Si tratta di ricorsione doppia.

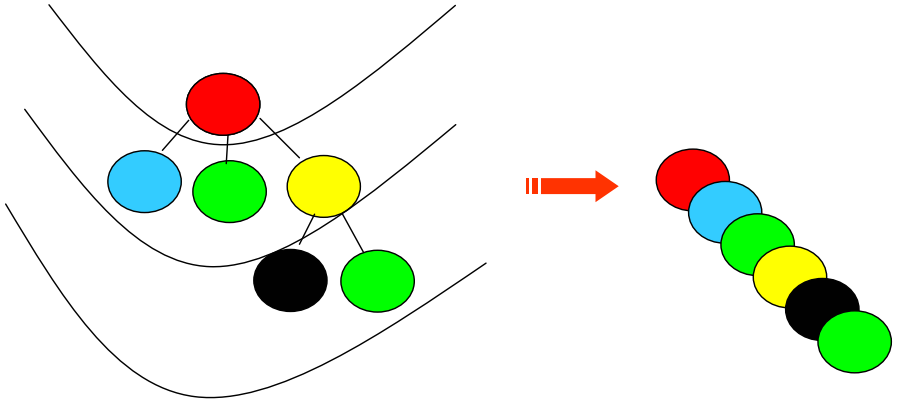
Implementazione ricorsiva per Postorder

```
protected void postorder(IntBSTNodo p)
{       if (p != null)
        {       postorder(p.left);
                postorder(p.right);
                p.visit();              // da gestire !!
        }
}
```

Il codice per le tre procedure differisce solo per l'ordine delle chiamate ricorsive.

Visita x livelli

Scrivere un metodo che crei una lista contenente tutti in nodi di un albero ordinati per livello.



IDEA: Bisogna implementare una visita per livelli dell'albero. E' necessario utilizzare una coda come struttura d'appoggio per tenere traccia dei nodi già visitati livello per livello.

Visita per livelli

```
public static LinkedList getInfoList(BTNode p) throws
  Underflow {

QueueListByComposition queue=new
                              QueueListByComposition();
LinkedList l = new LinkedList();

if (p != null) {
    queue.enqueue(p);
    while (!queue.isEmpty()) {
           p = (BTNode) queue.dequeue();
           l.insertTail(p.getInfo());
           if (p.getLeft() != null)
              queue.enqueue(p.getLeft());
           if (p.getRight() != null)
              queue.enqueue(p.getRight());
        }
    }
return l;
  }
```