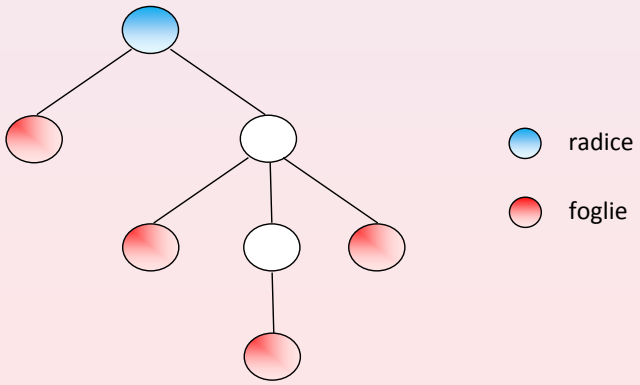
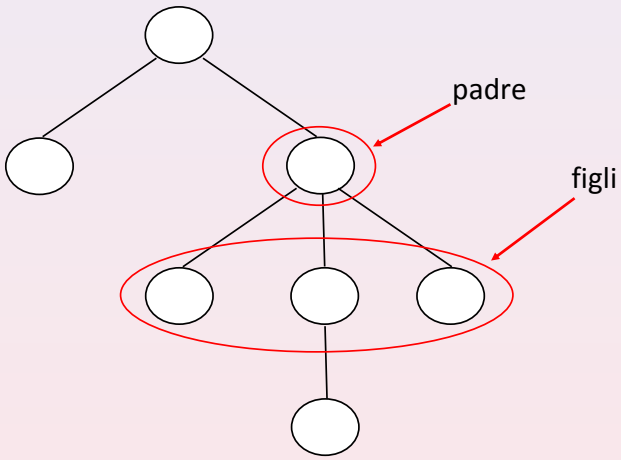
Alberi e alberi binari

- Un albero è un caso particolare di grafo
  - È costituito da un insieme di nodi collegati tra di loro mediante archi
  - Gli archi sono orientati (ogni arco esce da un nodo origine ed entra in un nodo destinazione)
  - Ogni nodo ha al più un arco entrante ed esiste un nodo, la radice dell'albero, che non ha archi entranti
  - I nodi senza archi uscenti sono detti foglie dell'albero
- Questi vincoli ci consentono di rappresentare graficamente un albero come una struttura gerarchica.

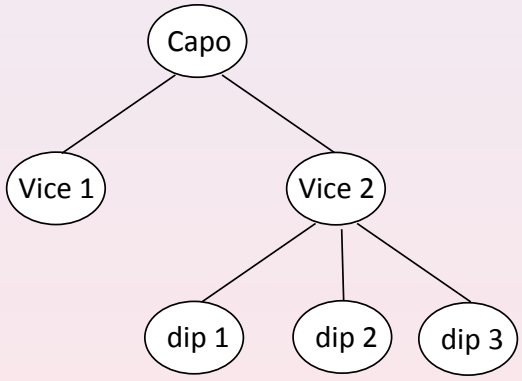


Se un arco esce da un nodo A ed entra in un nodo B si dice che

- A è il padre di B
- B è un figlio di A



- Gli alberi sono utili per rappresentare informazioni che hanno una struttura gerarchica (es. alberi genealogici, organigramma di aziende, ecc.)
- Ai nodi si associano le informazioni di interesse, dette etichette



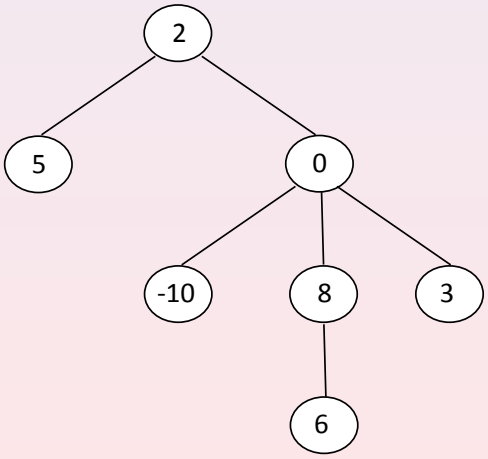
- Nel seguito faremo sempre riferimento ad alberi etichettati e identificheremo un nodo con la sua etichetta, laddove ciò non crei ambiguità!

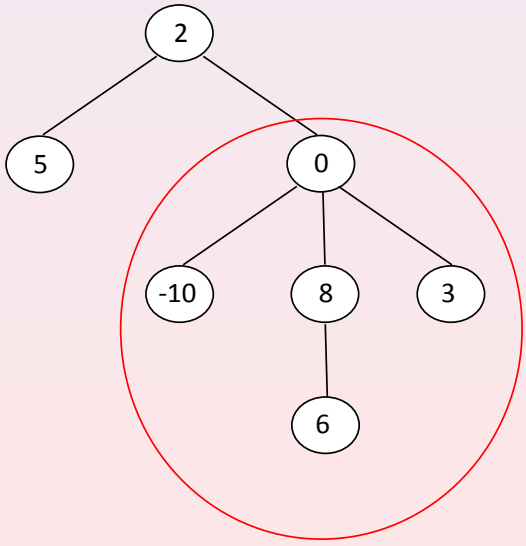
- Dato un albero, definiamo:
  - Cammino nell'albero una sequenza di nodi, in cui ogni nodo è figlio del nodo che lo precede nella sequenza
  - Livello (o profondità) di un nodo, la sua distanza dalla radice (quanto "in basso" si trova nell'albero).

    Il livello di un nodo può essere definito induttivamente come segue:
    - la radice ha livello 0
    - se un nodo ha livello i, allora i suoi figli hanno livello $i+1$
  - Livello k di un albero, come l'insieme di tutti e soli i nodi di livello k.
  - Altezza (o profondità) di un albero come la profondità massima che può avere un nodo dell'albero.
- Osserviamo che ogni nodo di un albero è a sua volta radice di un (sotto) albero

Un albero con etichette intere

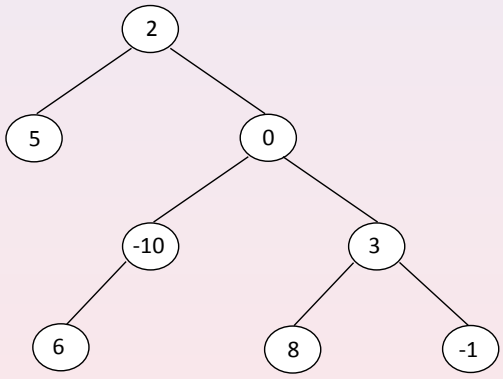
- Alcuni cammini: $\langle 0, 3\rangle$, $\langle 2, 0, 8, 6\rangle$
- Livello: il nodo 0 ha livello 1, il nodo 6 ha livello 3, ...
- Livello 1 dell'albero: {5, 0}
  Livello 2 dell'albero: {-10, 8, 3}
- Un sottoalbero





Alberi binari

- Un albero binario è un albero in cui ogni nodo ha al più 2 figli, detti rispettivamente figlio sinistro e figlio destro



- Per quanto osservato prima il figlio sinistro è a sua volta radice di un sottoalbero binario, detto sottoalbero sinistro. Analogamente per il figlio destro.

Rappresentazione collegata degli alberi binari

- Come possiamo rappresentare in C alberi, e in particolare alberi binari?
- Utilizziamo una rappresentazione collegata simile a quella delle liste
- L'elemento fondamentale è il nodo, che
  - ha un'etichetta
  - è collegato ai sottoalberi sinistro e destro (eventualmente vuoti)
- Possiamo definire una struttura con 3 campi:
  - l'etichetta
  - il puntatore al sottoalbero sinistro
  - il puntatore al sottoalbero destro
- In pratica, rappresentiamo mediante puntatori gli archi che collegano un nodo ai suoi sottoalberi.


```
struct nodoAlberoBinario
{
   TipoInfoAlbero label;
   struct nodoAlberoBinario *left;
   struct nodoAlberoBinario *right;
 }

typedef struct nodoAlberoBinario NodoAlbero;

typedef NodoAlbero *AlberoBinario;
```

- Il tipo `TipoInfoAlbero` definisce il tipo delle etichette dei nodi. Negli esempi
  `typedef int TipoInfoAlbero;`
- Si noti come, analogamente alle liste, un albero binario sia rappresentato dal puntatore al nodo radice

Un esempio

- Vediamo come primo esempio l'implementazione di una procedura che, dato un albero binario di interi, raddoppia l'etichetta di nodi con etichetta pari
- L'implementazione più naturale è di tipo ricorsivo, osservando che un albero binario può essere definito induttivamente come segue:
  - L'albero vuoto è un albero binario
  - Se lt e rt sono alberi binari e n è un intero, allora l'albero con radice un nodo etichettato con n, sottoalbero sinistro lt e sottoalbero destro rt, è un albero binario

```
void raddoppiaPari {AlberoBinario bt)
  if (bt != NULL)
    {
      if even(bt -> label)
         bt -> label = 2 * (bt -> label);
      raddoppiaPari(bt -> left);
      raddoppiaPari(bt -> right);
    }
```


Osservazioni

- Nell'esempio precedente abbiamo scelto di operare analizzando, nell'ordine:
  - la radice dell'albero
  - il sottoalbero sinistro
  - il sottoalbero destro
- Poiché l'analisi dei sottoalberi sinistro e destro avviene utilizzando la stessa procedura ricorsiva, anche la loro analisi opera allo stesso modo (prima la radice, poi il sottoalbero sx, quindi il sottoalbero dx . . .)
- Avremmo potuto procedere diversamente, ad esempio:

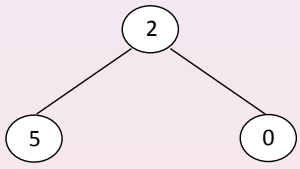
```
void raddoppiaPari {AlberoBinario bt)
  {if (bt != NULL)
    {
      raddoppiaPari(bt -> left);
      raddoppiaPari(bt -> right);
      if even(bt -> label)
         bt -> label = 2 * (bt -> label);
    }}
```

Osservazioni

- Nell'esempio precedente abbiamo scelto di operare analizzando, nell'ordine:
  - la radice dell'albero
  - il sottoalbero sinistro
  - il sottoalbero destro
- Poiché l'analisi dei sottoalberi sinistro e destro avviene utilizzando la stessa procedura ricorsiva, anche la loro analisi opera allo stesso modo (prima la radice, poi il sottoalbero sx, quindi il sottoalbero dx . . .)
- Avremmo potuto procedere diversamente, ad esempio:

```
void raddoppiaPari {AlberoBinario bt)
  {if (bt != NULL)
    {
      raddoppiaPari(bt -> left);
      if even(bt -> label)
         bt -> label = 2 * (bt -> label);
      raddoppiaPari(bt -> right);
    }}
```


- Nel caso della procedura vista, l'ordine è ininfluente ai fini degli effetti finali: tutte le etichette pari vengono comunque raddoppiate
- Ciò non è il caso, però, per altre operazioni
- **Esempio:** : Stampare la sequenza di etichette dell'albero



Possiamo ottenere sequenze diverse:

- 2, 5, 0
- 5, 2, 0
- 5, 0, 2
- . . .

Visita di un albero

- Visitare un albero significa analizzare in sequenza tutti i suoi nodi.
- Molte operazioni sugli alberi possono essere viste come varianti di visite degli stessi.
- Possiamo avere diversi tipi di visita, che differiscono per l'ordine in cui vengono visitati i nodi.
  - Visite depth-first (in profondità)
    - visita anticipata: si analizza la radice, poi si effettua la visita anticipata del sottoalbero sinistro e infine si effettua la visita anticipata del sottoalbero destro
    - visita simmetrica: si effettua la visita simmetrica del sottoalbero sinistro, poi si analizza la radice e infine si effettua la visita simmetrica del sottoalbero destro
    - visita posticipata: si effettua la visita posticipata del sottoalbero sinistro, poi si effettua la visita posticipata del sottoalbero destro, e infine si analizza la radice
  - visita breadth-first (per livelli): si visita prima la radice (livello 0), poi si visitano tutti i nodi di livello 1, poi tutti i nodi di livello 2, ...

Implementazione delle visite

- Vediamo l'implementazione ricorsiva delle visite in profondità (generalizzazione dell'esempio visto), assumendo data una funzione col seguente prototipo
  `void AnalizzaNodo(TipoInfoAlbero)`
- N.B. Se l'analisi del nodo può comportare la modifica dell'etichetta, abbiamo bisogno di una procedura con prototipo
  `void AnalizzaNodo(TipoInfoAlbero *)`.
  Di conseguenza la chiamata nella procedura di visita si modifica in
  `AnalizzaNodo(&(bt -> label))`

Implementazione delle visite (cont.)

Visita simmetrica

```
void visitaSimmetrica {AlberoBinario bt)
 { if (bt != NULL)
    {
      visitaSimmetrica(bt -> left);
      AnalizzaNodo(bt -> label);
      visitaSimmetrica(bt -> right);
    }}
```


Implementazione delle visite (cont.)

Visita anticipata

```
void visitaAnticipata {AlberoBinario bt)
 { if (bt != NULL)
    {
      AnalizzaNodo(bt -> label);
      visitaAnticipata(bt -> left);
      visitaAnticipata(bt -> right);
    }}
```

Implementazione delle visite (cont.)

Visita posticipata

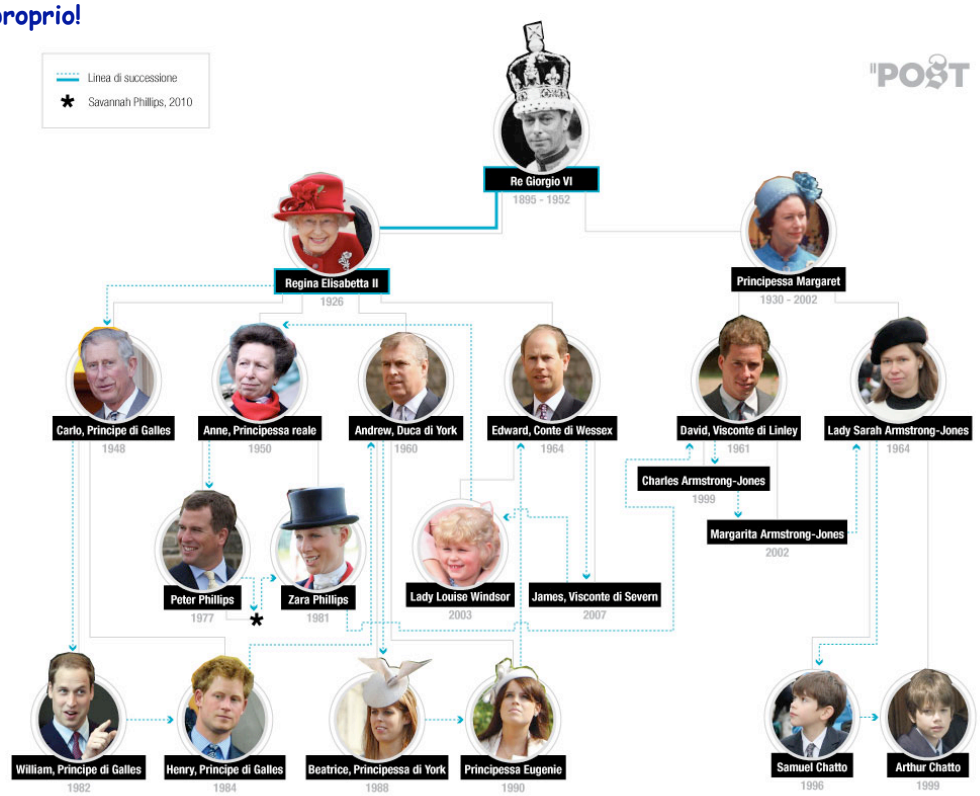
```
void visitaPosticipata {AlberoBinario bt)
  {if (bt != NULL)
    {
      visitaPosticipata(bt -> left);
      visitaPosticipata(bt -> right);
      AnalizzaNodo(bt -> label);
    }}
```


Visita anticipata: una curiosità

La visita in ordine anticipato di un albero genealogico corrisponde all'antichissimo algoritmo usato per determinare l'ordine di successione al titolo in una famiglia nobile o regale.

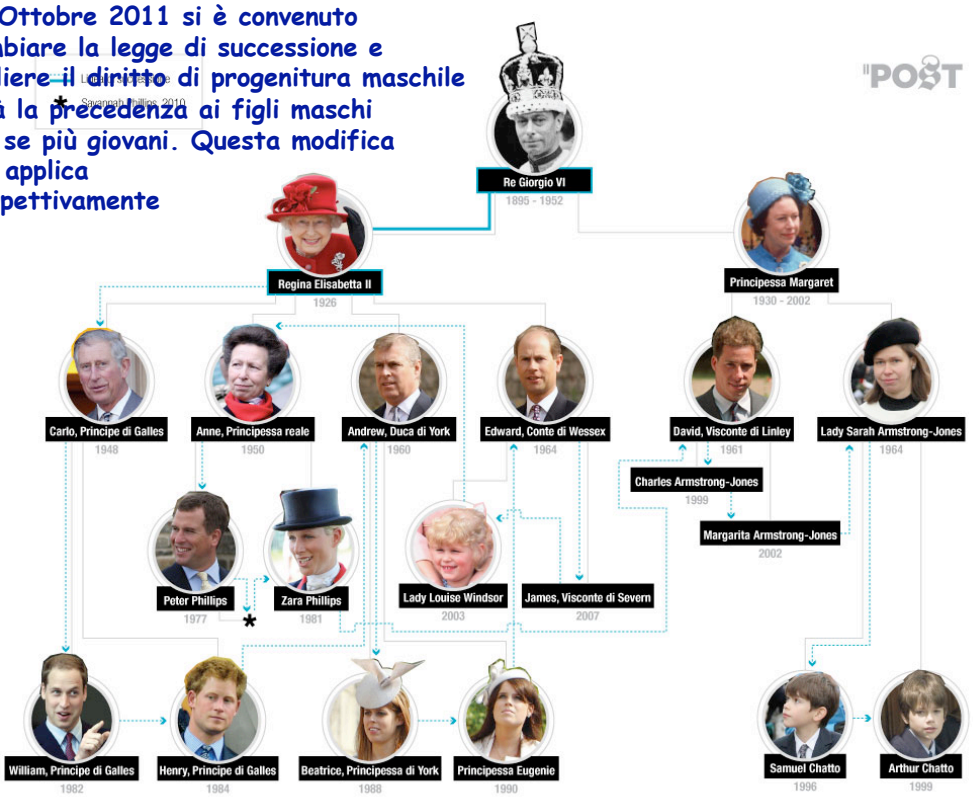
Visita anticipata: una curiosità (2)

Non proprio!



Visita anticipata: una curiosità (3)

Il 28 Ottobre 2011 si è convenuto di cambiare la legge di successione e di togliere il diritto di progenitura maschile che dà la precedenza ai figli maschi anche se più giovani. Questa modifica non si applica retrospettivamente



Esercizi

- Scrivere una funzione che determina se un albero contiene un nodo con una certa etichetta. Il prototipo è

  `boolean member (AlberoBinario, TipoInfoAlbero)`

- Scrivere una funzione che conta il numero di occorrenze di una certa etichetta in un albero binario.

  `int contaOccorrenze (AlberoBinario, TipoInfoAlbero)`

- Per alberi binari con etichette di tipo `int`, scrivere una funzione che calcoli la somma delle etichette
  - di tutti i nodi dell'albero
  - di tutte le foglie dell'albero

Ricerca di un'etichetta

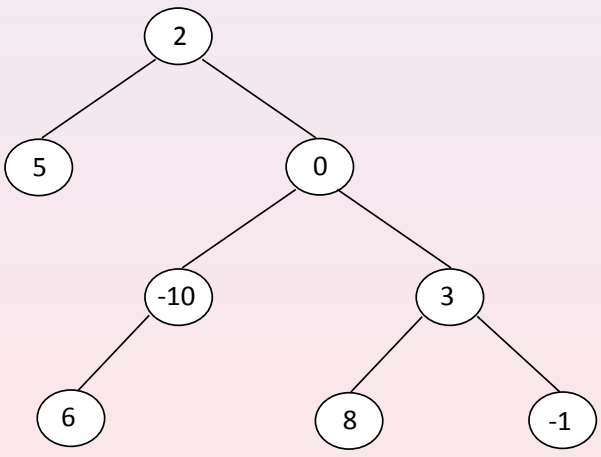
Diamo due tra le tante possibili soluzioni:

```
boolean member (AlberoBinario bt, TipoInfoAlbero etichetta)
{
   boolean risultato = false;
   if (bt != NULL)
      risultato = ((bt -> label) == etichetta) || member(bt -> left, etichetta)
      |I              member(bt -> right, etichetta);
   return risultato;
 }

boolean member (AlberoBinario bt, TipoInfoAlbero etichetta)
{
   boolean risultato = false;
   if (bt != NULL)
      if ((bt -> label)==etichetta)
         risultato = true;
      else  {
              risultato = member(bt->left, etichetta);
              if (!risultato)
                 risultato = member(bt -> right, etichetta);
            }
   return risultato;
 }
```

Anche in questo caso due tra le varie soluzioni possibili (la prima visita in ordine simmetrico, la seconda in ordine posticipato)

```
int contaOccorrenze (AlberoBinario bt, TipoInfoAlbero etichetta)
{
   int risultato = 0;
   if (bt != NULL)
      {
        if ((bt -> label) == etichetta) risultato = risultato + 1;
        risultato = risultato + contaOccorrenze(bt -> left, etichetta) +
                                contaOccorrenze(bt -> right, etichetta);
      }
    return risultato;
 }

int contaOccorrenze (AlberoBinario bt, TipoInfoAlbero etichetta)
{
   int risultato = 0;
   if (bt != NULL)
      {
        risultato = contaOccorrenze(bt -> right, etichetta);
        risultato  = risultato + contaOccorrenze(bt -> left, etichetta);
        if ((bt -> label) == etichetta) risultato = risultato + 1;
      }
    return risultato;
 }
```


- **Esempio:** Dato un albero binario di interi, costruire la lista delle etichette dei suoi nodi, ottenuta visitando l'albero in ordine simmetrico
- In corrispondenza dell'albero



vogliamo dunque ottenere la lista

```
5 --> 2 --> 6 --> -10 --> 0 --> 8 --> 3 --> -1 --> //
```