POLITECNICO DI MILANO

Algoritmi e Strutture Dati

Laboratorio 20/10/2008

Prof. Pier Luca Lanzi

Primo Esercizio

- Scrivere un programma per misurare il tempo necessario per ordinare un vettore di interi contenente 10-10^7 elementi utilizzando l'insertion sort e il merge sort

- Il programma deve generare un file con tre colonne: la prima contiene il numero di dati, la seconda il tempo necessario per l'insertion sort e la terza contiene il tempo necessario per il merge sort

Secondo Esercizio

- Scrivere il template di una classe che implementi una coda derivata dalla seguente classe virtuale astratta

```
template <class Elem> class CodaBase {
public:
   virtual bool enque(Elem) = 0;
   virtual bool deque() = 0;
   virtual Elem head() = 0;
   virtual bool isempty() = 0;
};
```

Secondo Esercizio

- Usare la classe coda per eseguire il seguente codice

```
Coda<int> c;

for(i=0;i<10;i++) c.enqueue(rand());

for(i=0;i<10;i++) {
  cout << c.head();
  c.dequeue();
}
```

Terzo Esercizio

- Creare un template di una classe CodaConPriorita derivata dal template della classe Coda in cui l'inserimento di un elemento in coda dipende anche da un valore di priorita'.

Terzo Problema

```
class point {
private:
   float x, y;
public:
   // costruttore
   point (float a, float b)
     {x = a; y = b;};
   // distruttore
   ~point() {};
   void x_move (float a)
      {x += a;};
   void y_move (float b)
      {y += b;};
   void reset ()
      { x = y = 0.0;};
};
```

```
point p1 (1.3, 3.7);

point p2 (53.3, 0.0);

point * p3 =
  new point(0.0, 0.0);

p1.x_move(9.3);

*p3 = p2;

p1.X = 3.4;
```

Queue (Code)

- insieme dinamico in cui l'elemento rimosso dall'operazione di cancellazione è predeterminato: "quello che per più tempo è rimasto nell'insieme"

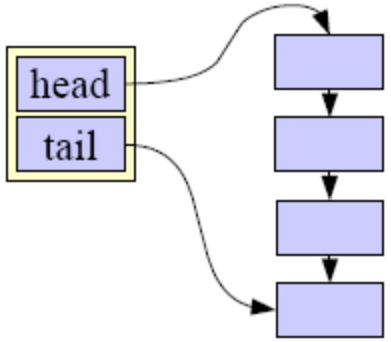
- La politica di funzionamento è "first in, first out" (FIFO)
- Operazioni previste
  - void enqueue(Item) // sinonimi: put, add, insert
  - Item dequeue() // sinonimi: removeFirst, extract
  - Item head() // non rimuove l'item; sinonimi get
  - boolean isEmpty()

Quanto costano le operazioni?

Code

- Esempi
  - Nei sistemi operativi, i processi in attesa di utilizzare una risorsa vengono gestiti tramite una coda
  - Prenotazione dei biglietti dei concerti in linea

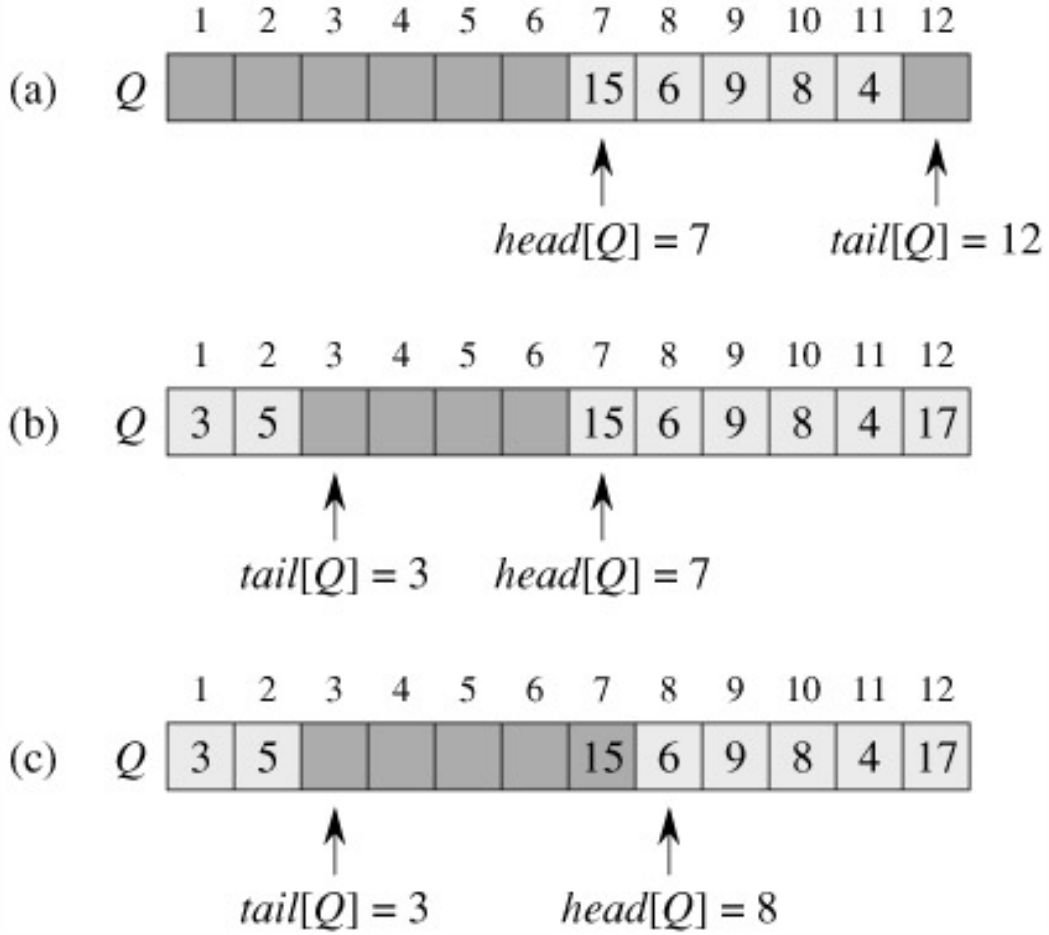
- Possibili implementazioni
  - Liste puntate semplici
  - Array



- Liste puntate semplici
  - Puntatore head (inizio della coda), per estrazione
  - puntatore tail (inizio della coda), per inserimento

- Tramite array circolari
  - Dimensione limitata, overhead più basso

Implementazione delle Code: Array



a) Coda iniziale

b) Dopo aver svolto ENQUEUE(Q, 17), ENQUEUE(Q, 3), e ENQUEUE(Q, 5)

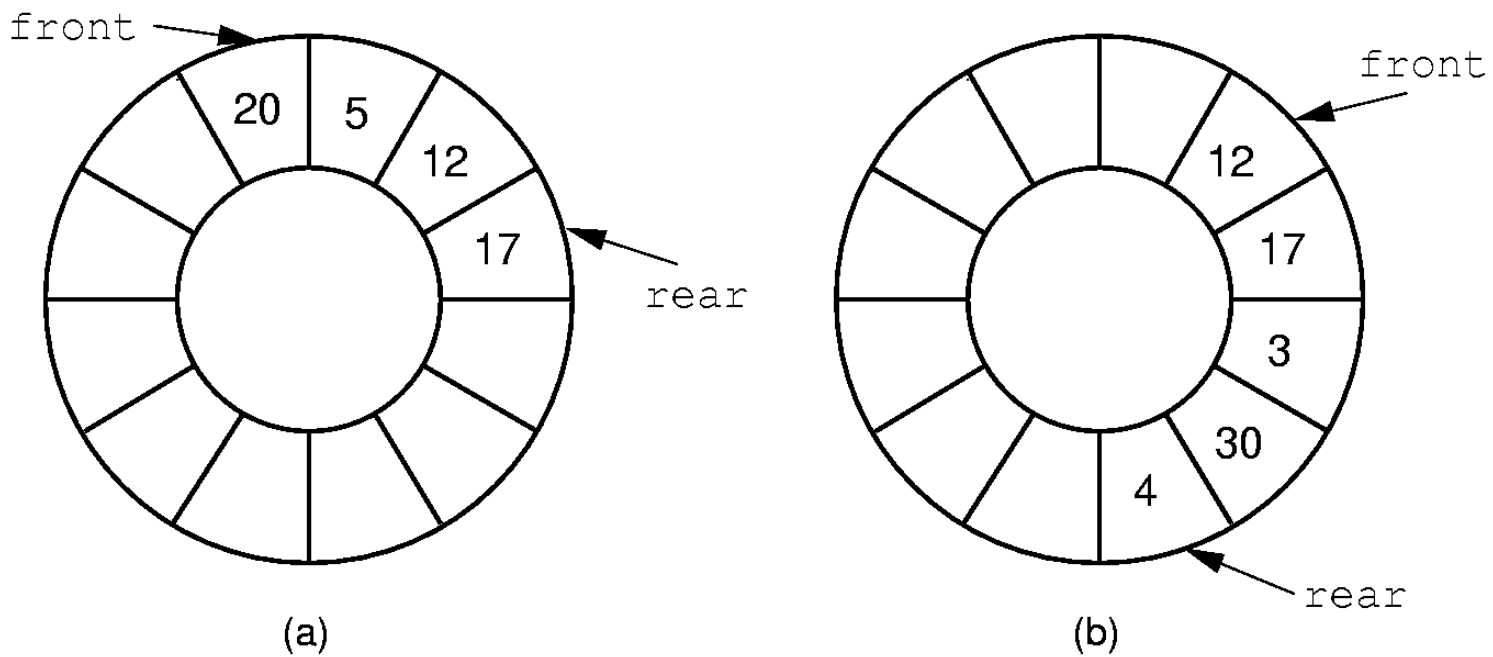
c) Dopo aver svolto DEQUEUE(*Q*) che ritorna il valore 15

Implementazione delle Code: Array


```
ENQUEUE(Q, x)
1  Q[tail[Q]] ← x
2  if tail[Q] = length[Q]
3     then tail[Q] ← 1
4  else
5     tail[Q] ← tail[Q]+1
```

```
DEQUEUE(Q)
1  x ← Q[head[Q]]
2  if head[Q] = length[Q]
3     then head[Q] ← 1
4  else
5     head[Q] ← head[Q]+1
6  return x
```

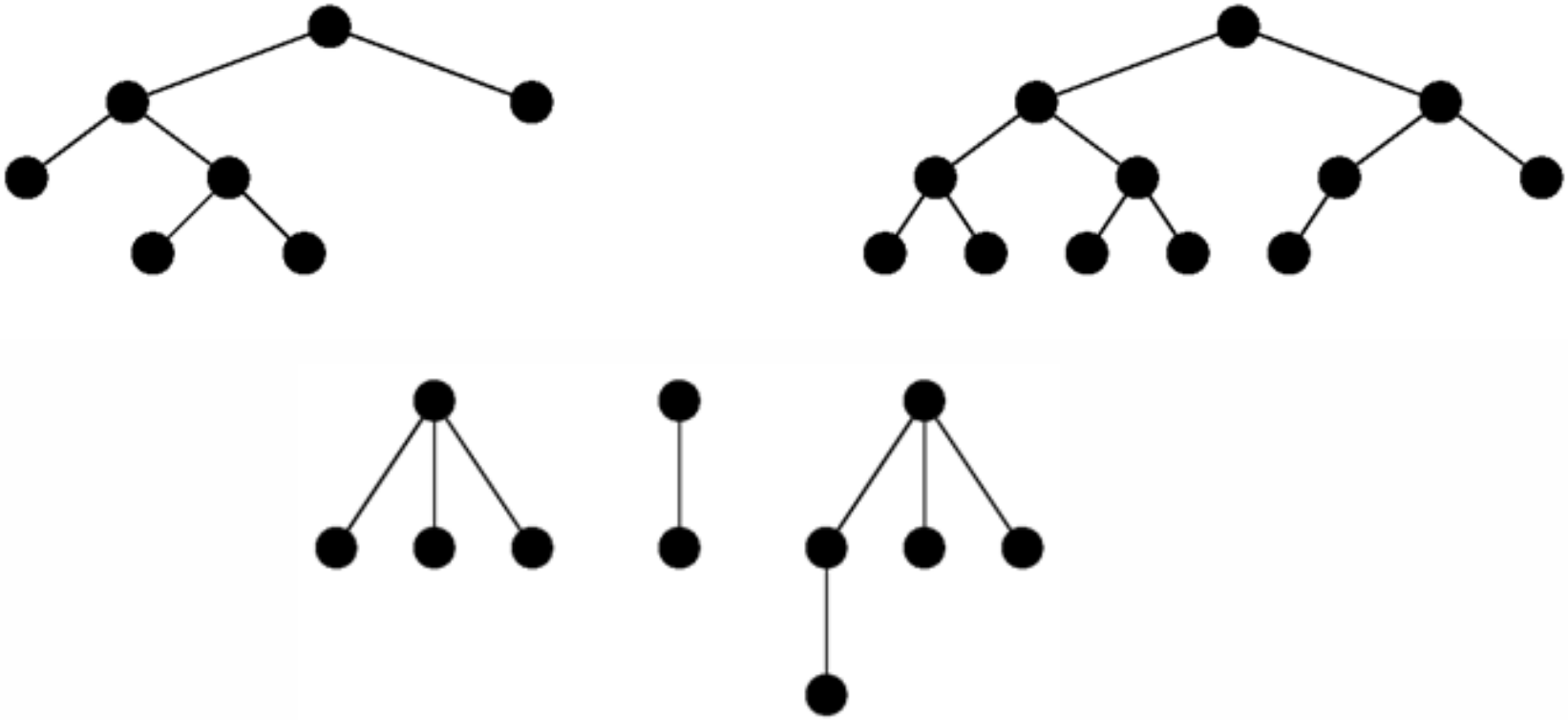
Implementazione delle Code: Array Circolari



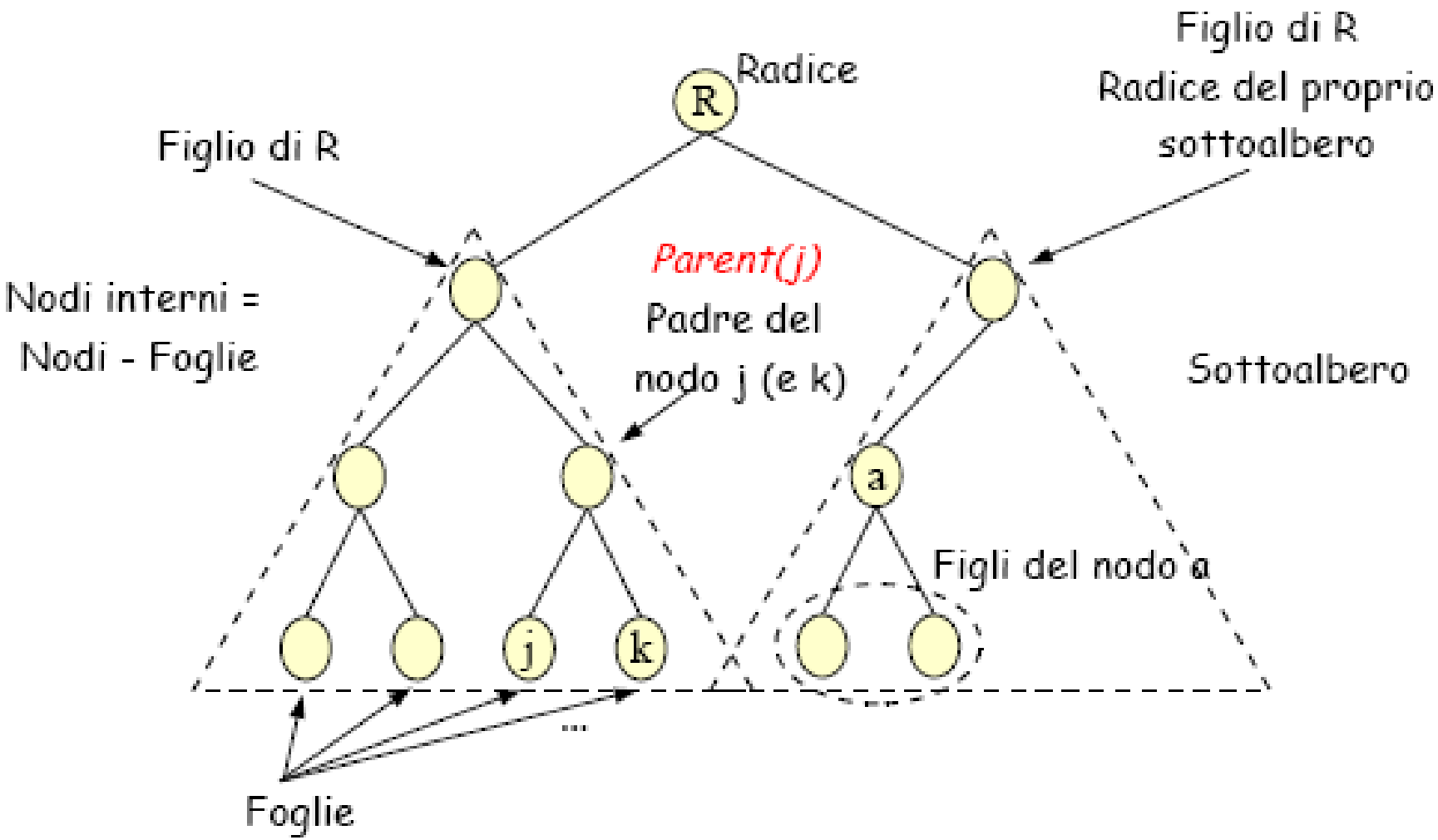
Alberi radicati...

Alberi Radicati

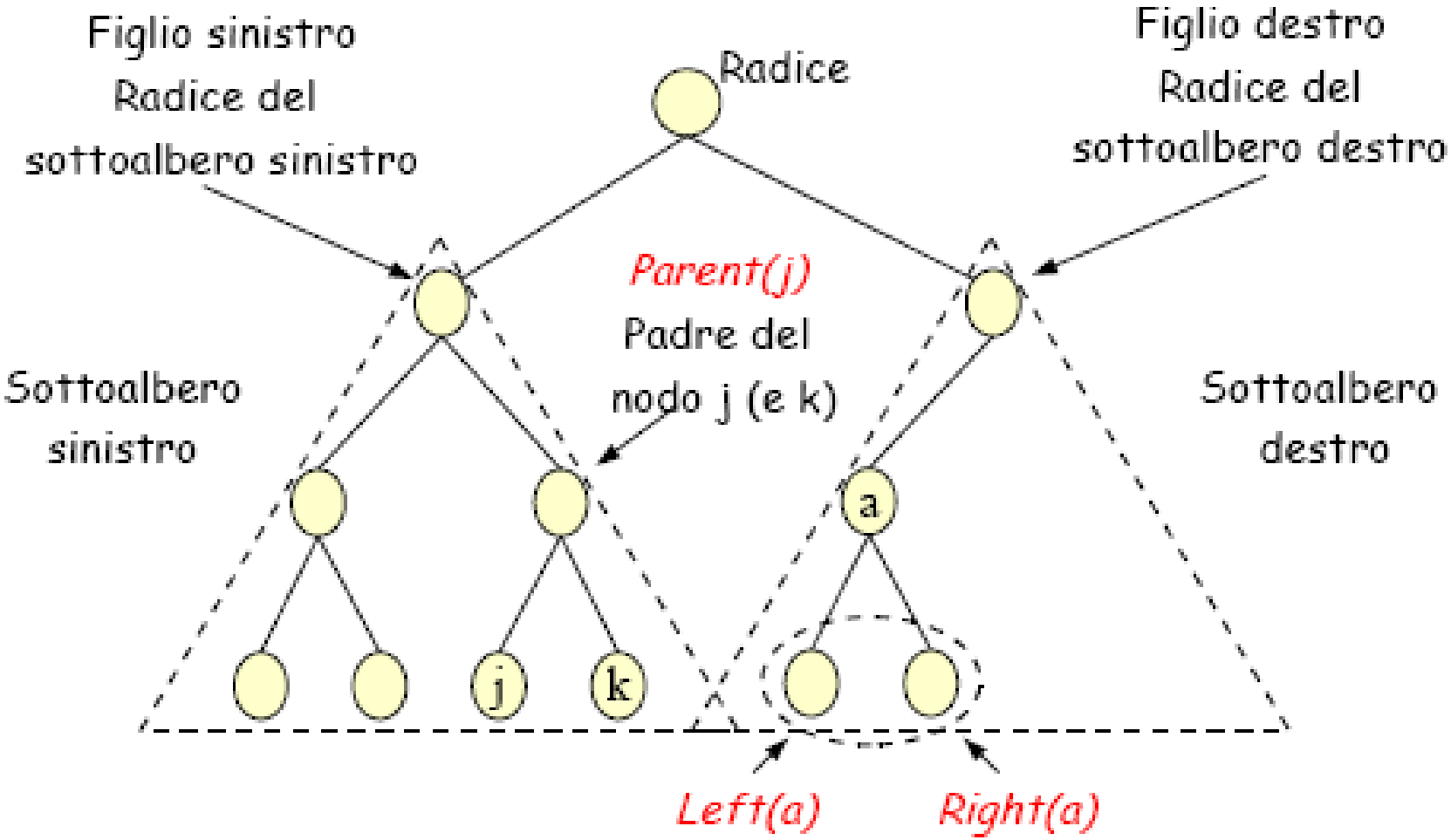
- Albero radicato
  - Un insieme vuoto di nodi
  - Un nodo radice R collegato a 0 o più alberi (sottoalberi)



Alberi Radicati

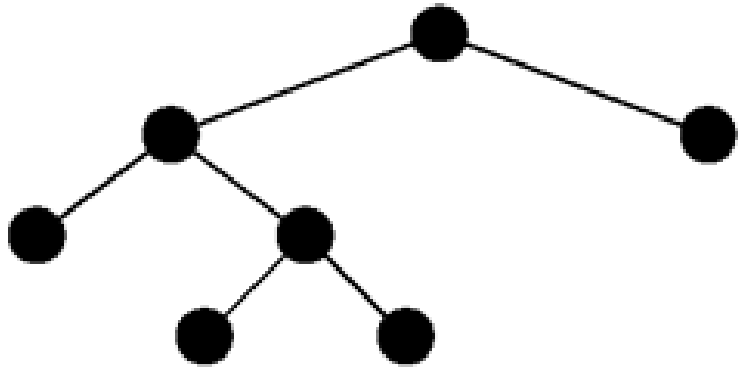


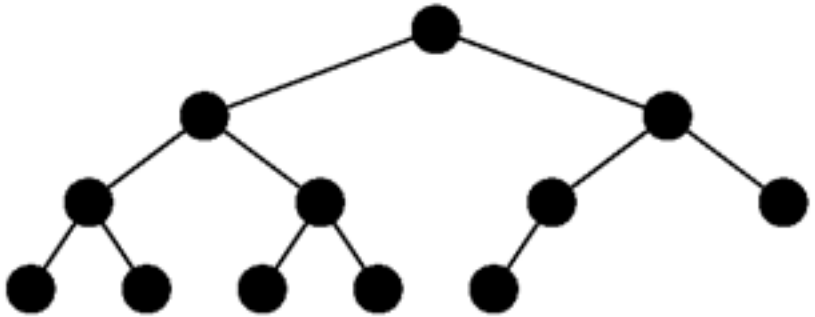
Alberi Radicati Binari



Alberi Radicati

- Profondità di un nodo: la lunghezza del percorso dalla radice al nodo (numero archi attraversati)
- Livello: l'insieme dei nodi alla stessa profondità
- Altezza dell'albero: massima profondità+1
- Grado di un nodo: numero dei figli

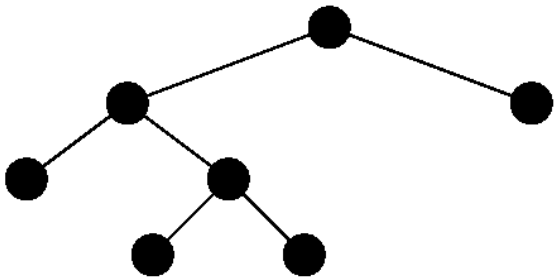




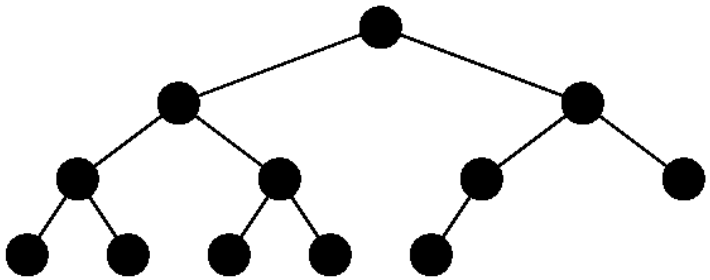
Alberi Binari: Pieni e Completi

- Alberi binari pieni: Ogni nodo è una foglia oppure è un nodo interno con esattamente due figli non vuoti

- Alberi binari completi: se l'altezza dell'albero è d, allora tutte le foglie eccetto possibilmente il libello d sono completamente piene. L'ultimo livello ha tutti i nodi sul lato sinistro



(a)



(b)

Alberi Binari Pieni

- Teorema: Il numero di foglie in un albero binario pieno non vuoto è uno di più del numero di nodi interni

- Dimostrazione: per induzione sul numero di nodi.

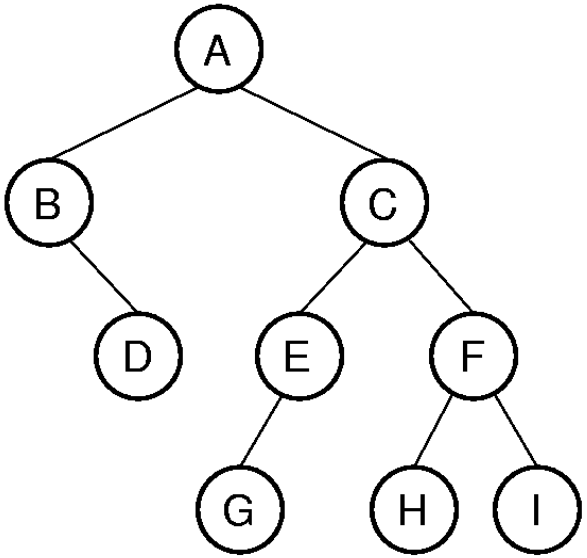
  Caso base: un albero binario pieno con 1 nodo interno deve avere due foglie

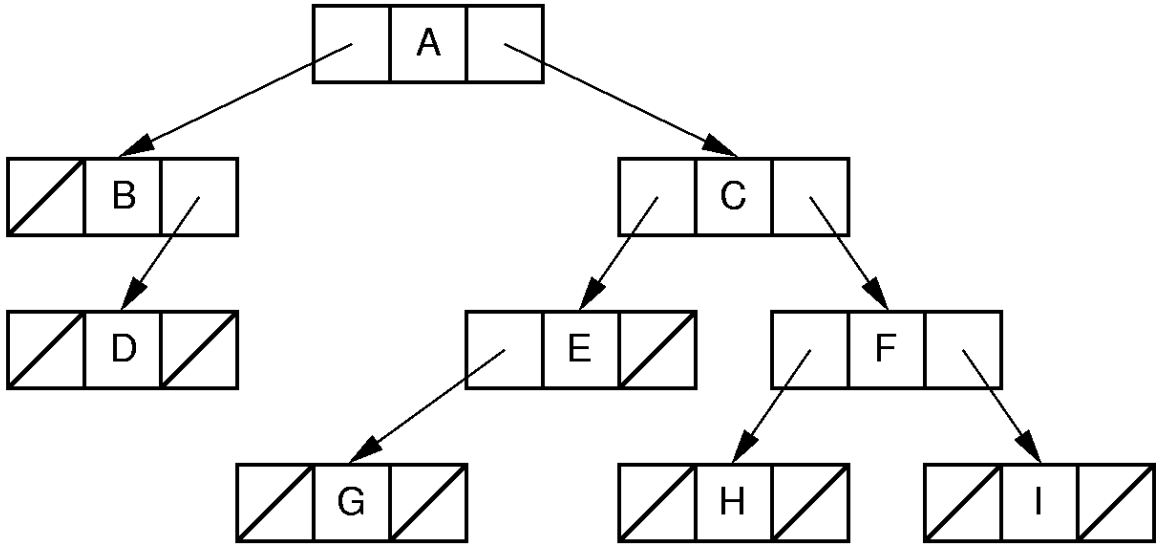
  Ipotesi induttiva: assumiamo che un albero binario T con n-1 nodi interni abbia n foglie

  Passo induttivo: dato un albero T con n nodi interni, prendiamo un nodo interno con due figli, rimuovendo I due figli otteniamo l'albero T' (che ha n foglie)

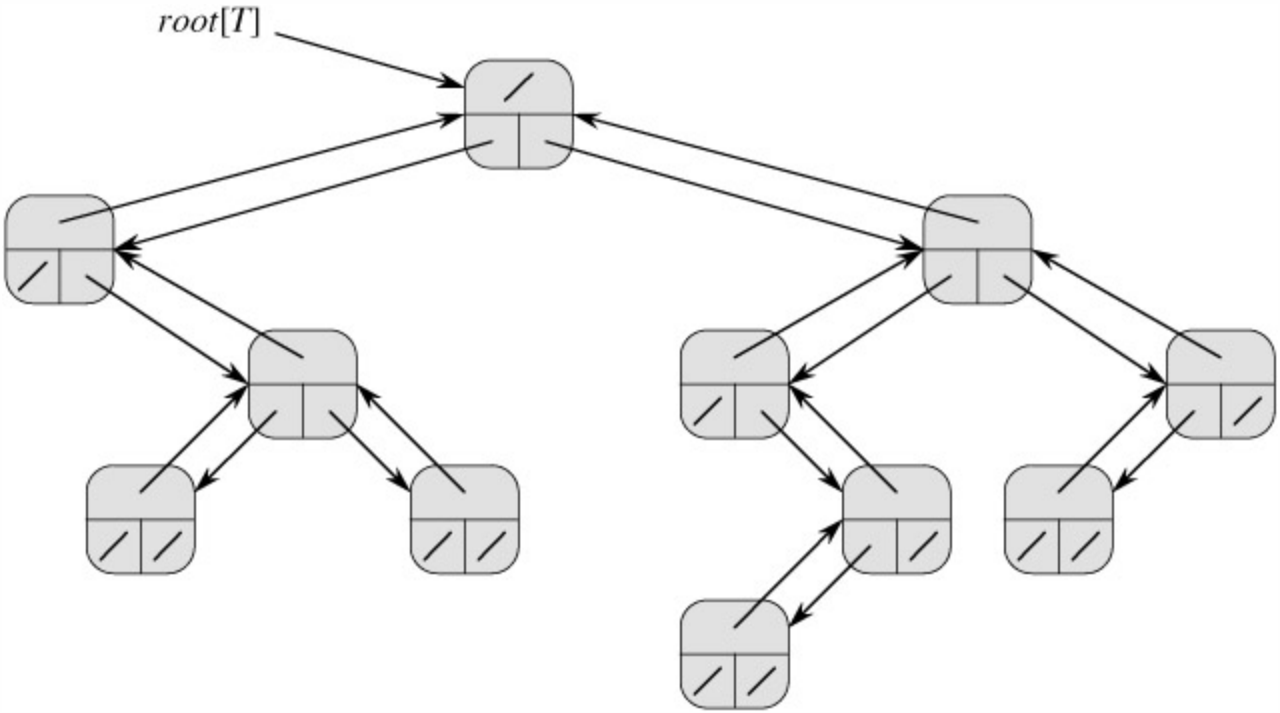
  Il numero di nodi interni è aumentato di un per raggiungere e il numero di foglie è anche aumentato di uno.

Rappresentazione degli Alberi Binari

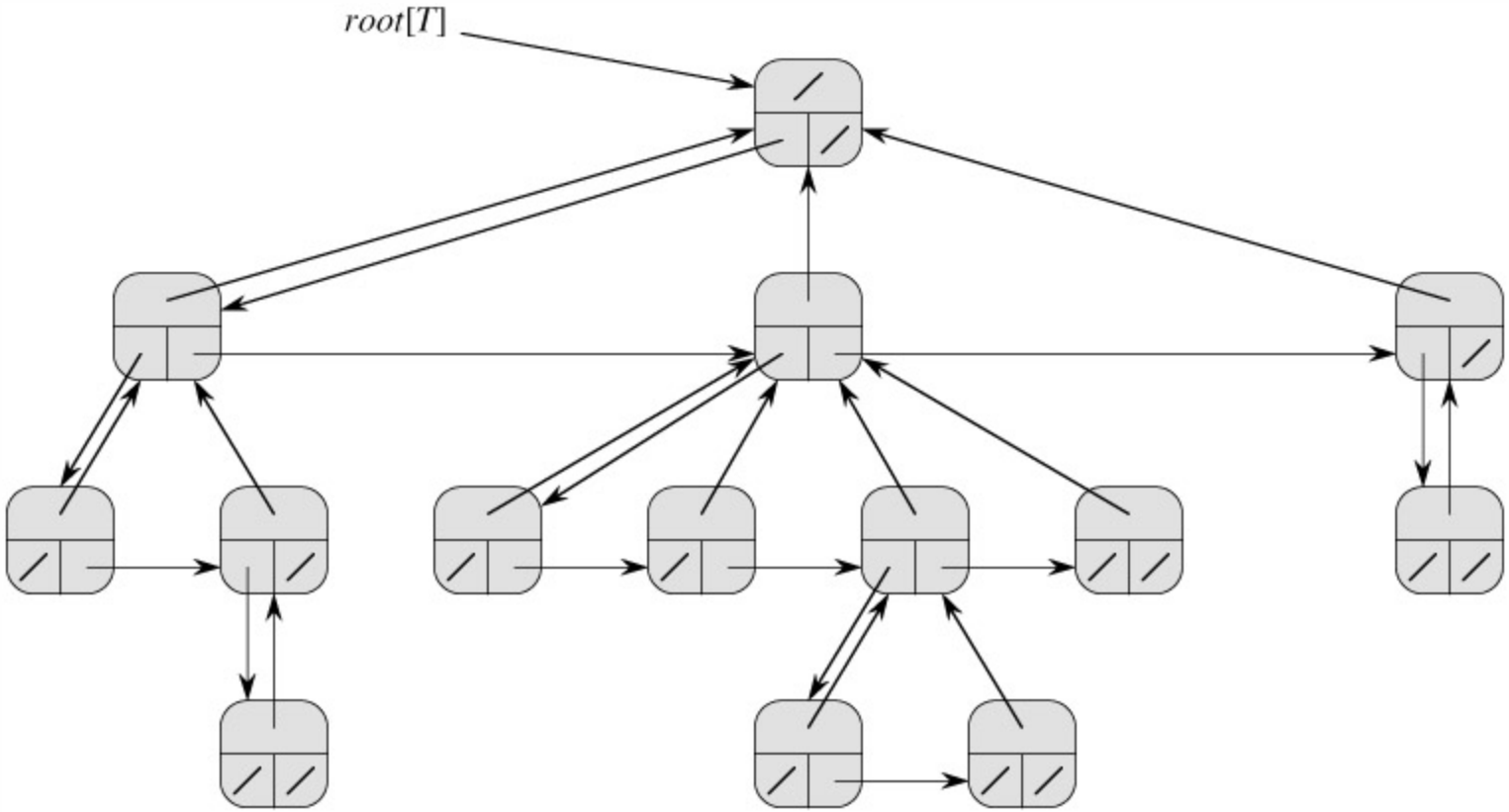




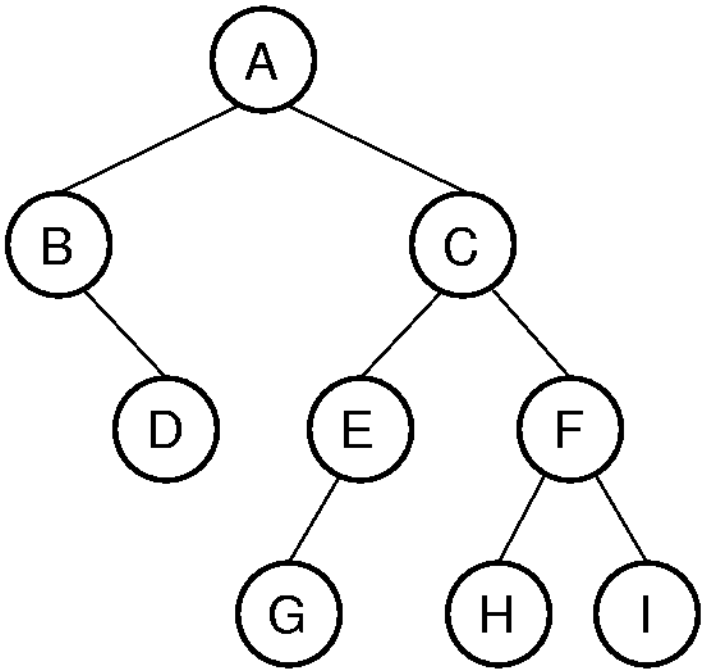
Altre Rappresentazioni: Con puntatore al parente



Altre Rappresentazioni: Con puntatori ai nodi dello stesso livello



Altre Rappresentazioni: Array



	Left	Key	Right	Par
0	1	A	3	-1
1	-1	B	2	0
2	-1	D	-1	1
3	4	C	6	0
4	5	E	-1	3
5	-1	G	-1	4
6	7	F	8	3
7	-1	H	-1	6
8	-1	I	-1	6

Tipo di Dati Astratti per Alberi Binari (1)


```
// Binary tree node class
template <class Elem>
class BinNodePtr : public BinNode<Elem> {
private:
  Elem it;          // The node's value
  BinNodePtr* lc;   // Pointer to left child
  BinNodePtr* rc;   // Pointer to right child
public:
  BinNodePtr() { lc = rc = NULL; }
  BinNodePtr(Elem e, BinNodePtr* l =NULL,
                     BinNodePtr* r =NULL)
    { it = e; lc = l; rc = r; }
```

Tipo di Dati Astratti per Alberi Binari (2)


```
 Elem& val() { return it; }
  void setVal(const Elem& e) { it = e; }
  inline BinNode<Elem>* left() const
    { return lc; }
  void setLeft(BinNode<Elem>* b)
    { lc = (BinNodePtr*)b; }
  inline BinNode<Elem>* right() const
    { return rc; }
  void setRight(BinNode<Elem>* b)
    { rc = (BinNodePtr*)b; }
  bool isLeaf()
    { return (lc == NULL) && (rc == NULL); }
};
```

Algoritmi di visita degli alberi

- Visita (o attraversamento) di un albero:
  - Algoritmo per "visitare" tutti i nodi di un albero
- In profondità (depth-first search, a scandaglio): DFS
  - Vengono visitati i rami, uno dopo l'altro
  - Tre varianti: pre-ordine, post-ordine, in-ordine
- In ampiezza (breadth-first search, a ventaglio): BFS
  - A livelli, partendo dalla radice

Visita in Profondità: Implementazioni

```
template <class Elem> // Pre-ordine
void preorder(BinNode<Elem>* subroot) {
  if (subroot == NULL) return;  // Empty
  visit(subroot);  // Perform some action
  preorder(subroot->left());
  preorder(subroot->right());
}

template <class Elem> // In-ordine
void inorder(BinNode<Elem>* subroot) {
  if (subroot == NULL) return;  // Empty
  preorder(subroot->left());
  visit(subroot);  // Perform some action
  preorder(subroot->right());
}
```

Visita in Profondità: Implementazioni

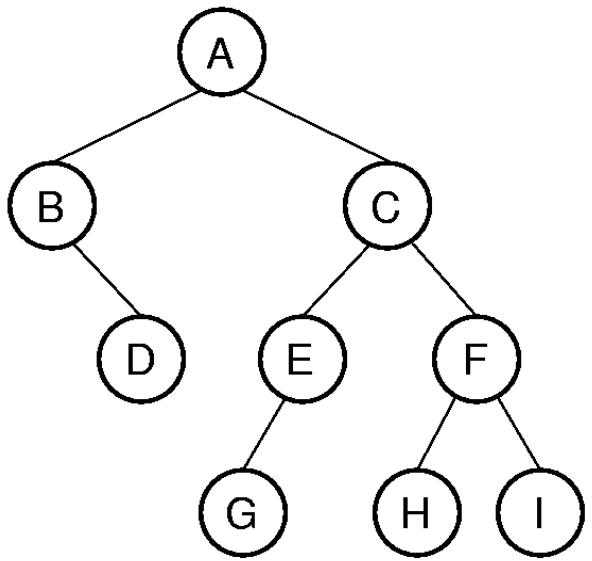
```
template <class Elem> // Post-ordine
void preorder(BinNode<Elem>* subroot) {
  if (subroot == NULL) return;  // Empty
  preorder(subroot->left());
  preorder(subroot->right());
  visit(subroot);  // Perform some action
}
```

Visita in Ampiezza


```
VisitaAmpiezza(T)
   q = new Queue()
   q.insert(T)
   while not q.empty() do
      p := q.dequeue()
      visita p
      q.enqueue(p.left())
      q.enqueue(p.right())
```

Esercizi

- Stampare il risultato della
  - Visita in pre-ordine
  - Visita in-ordine
  - Visita in post-ordine
  - Visita in ampiezza



- Scrivere un algoritmo per
  - Calcolare l'altezza di un albero binario T
  - Calcolare il nume di nodi di un albero binario T
  - Stampare tutti i nodi di profondità h di un albero binario T

Sommario

Sommario

- Tipo di dato astratto = collezione di valori + operazioni ammesse su questi valori
- Strutture dati vs Implementazione
- Incapsula i dettagli dell'implementazione
  - Elencare le proprietà degli oggetti
  - Incorporare al loro interno gli attributi (le caratteristiche) che i metodi (il comportamento)
  - Oggetti/Classi/Interfaccia
  - Costruttori/Distruttori
- Strutture Dati
  - lineari/non lineari
  - statiche/dinamiche (variazione di dimensione, contenuto)
  - omogenee/disomogenee (dati contenuti)