Alberi binari e alberi binari di ricerca

Violetta Lonati

Università degli studi di Milano
Dipartimento di Informatica

Laboratorio di algoritmi e strutture dati
Corso di laurea in Informatica

Alberi

- Un albero è una collezione non vuota di:
  - nodi con nome e informazioni contenute;
  - lati che collegano due nodi tra loro.
- Un cammino è una sequenza di nodi collegati da lati. La proprietà fondamentale degli alberi è che *esiste esattamente un cammino da un nodo ad una qualsiasi altro nodo* (altrimenti è un grafo).
- Negli alberi con radice si sceglie un nodo particolare come radice, che di solito è rappresentato in alto. Allora si usano espressioni come sopra, sotto, foglia, nodo interno, padre, figlio, antenato, discendente, ...
- Un sottoalbero è definito scegliendo un nodo interno e comprende tale nodo e tutti i suoi discendenti
- Nel caso degli alberi ordinati, i figli hanno un ordine (figlio destro, sinistro...)
- Definizione ricorsiva di albero: un albero è una foglia o una radice connessa ad un insieme di alberi.

Alberi binari

- Sono alberi (con radice) ordinati dove ogni nodo ha al più 2 figli (destro/sinistro)
- Definizione ricorsiva: un albero binario è una foglia oppure una radice connessa ad un albero binario destro e ad un albero binario sinistro.
- Proprietà numeriche:
  - un albero binario con N nodi ha $N - 1$ lati
  - un albero binario con N nodi ha altezza *circa* $log_2 N$

Rappresentazione di alberi binari in memoria

Sono strutture non monodimensionali (a differenza delle liste).
Ogni nodo può essere rappresentato come una struttura con un membro chiamato `item` (con le informazioni contenute nel nodo) e due link (al figlio destro e al figlio sinistro).

```
struct bit_node {
  Item item;
  struct bit_node *l, *r;
};

typedef struct bit_node *Bit_node;
```

`Item` può essere ad esempio `int` o un qualunque altro tipo, a seconda del genere di informazione contenuta nei nodi dell'albero.

Rappresentazione di alberi binari in memoria - continua

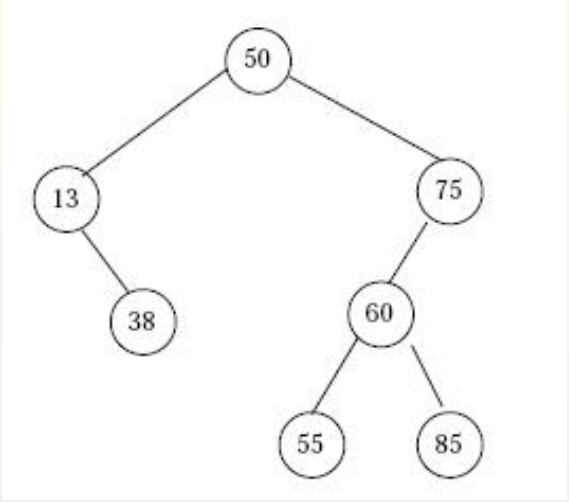
Se `n` è una variabile di tipo `Bit_node` e punta ad un certo nodo, allora

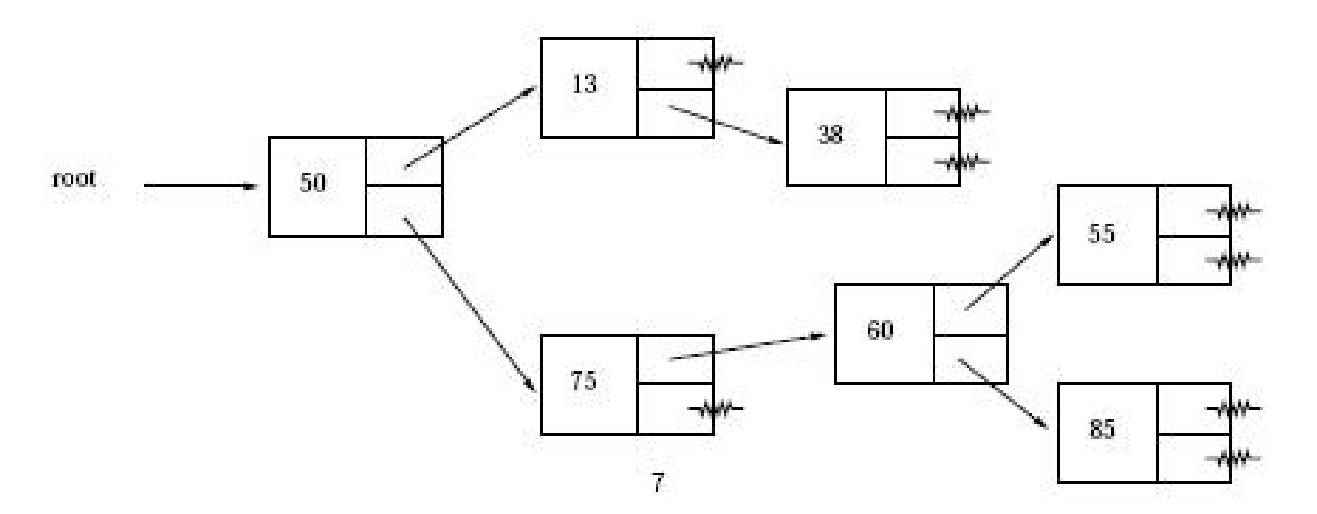
- `n -> l` punta al suo figlio sinistro;
- l'assegnamento `n = n -> r` fa in modo che `n` si sposti sul figlio destro.

NB: questa rappresentazione è comoda per attraversare l'albero dalla radice verso le foglie ma non viceversa: si potrebbe aggiungere un ulteriore membro `struct bit_node *up` (come per le liste bidirezionali, concatenate doppie)

Esempio: albero contenente interi

Nel seguente esempio, costruiamo manualmente un albero di interi, quindi `Item` è definito come `int`.



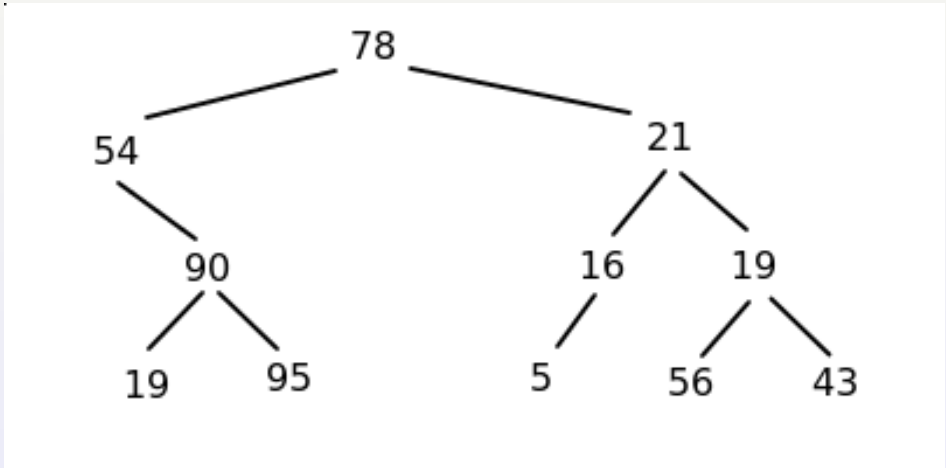


Esempio: albero contenente interi

```
Bit_node root; /* radice dell'albero */
root = malloc( sizeof(struct bit_node) );
root -> item = 50;
root -> l = malloc( sizeof(struct bit_node) );
root -> r = malloc( sizeof(struct bit_node) );
root -> l -> item = 13;
root -> l -> l = NULL;
root -> l -> r = malloc( sizeof(struct bit_node) );
root -> r -> item = 75;
root -> r -> l = malloc( sizeof(struct bit_node) );
root -> r -> r = NULL;
root -> l -> r -> item = 38;
```

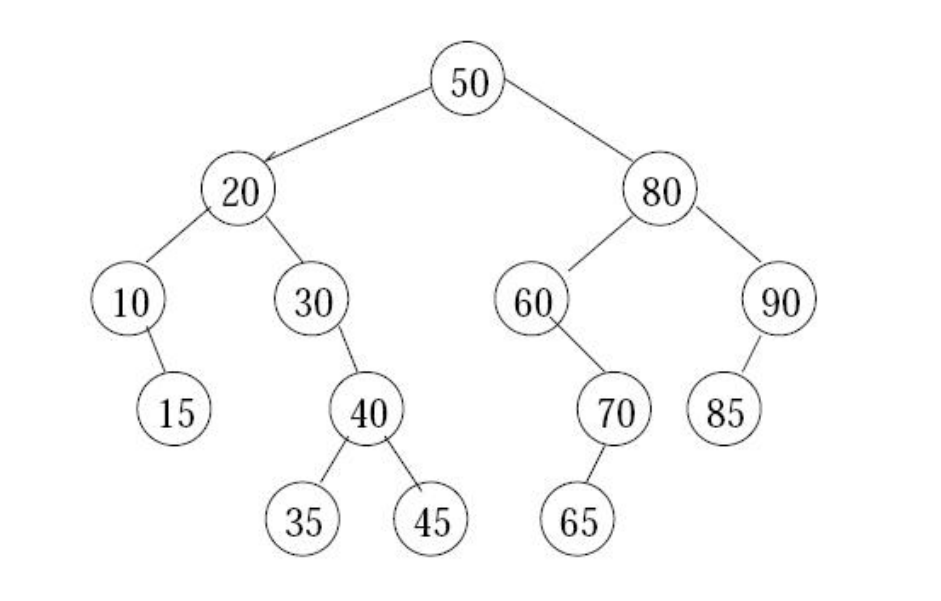
Esercizio: stampa di alberi *a sommario*

Scrivete quindi una funzione che stampi un albero binario nella rappresentazione usata nei sommari dei libri, oppure in un file browser, come nel seguente esempio:




```
*78
  *54
    *
    *90
      *19
      *95
  *21
    *16
      *5
      *
    *19
      *56
      *43
```

Attraversamento di alberi



Preorder: 50 20 10 15 30 40 35 45 80 60 70 65 90 85
Inorder: 10 15 20 30 35 40 45 50 60 65 70 80 85 90
Postorder: 15 10 35 45 40 30 20 65 70 60 85 90 80 50

Attraversamento di alberi

Attraversamento in ordine simmetrico (inorder): prima il sottoalbero di sinistra, poi la radice, infine il sottoalbero di destra:

```
void bit_inorder( Bit_node p ){
  if ( p ) {
    bit_inorder( p -> l );
    bit_printnode( p );
    bit_inorder( p -> r );
  }
}
```

Attraversamento di alberi

Attraversamento in ordine anticipato (preorder): prima la radice, poi il sottoalbero di sinistra, infine il sottoalbero di destra:

```
void bit_preorder( Bit_node p ){
  if ( p ) {
    bit_printnode( p );
    bit_preorder( p -> l );
    bit_preorder( p -> r );
  }
}
```

Attraversamento di alberi

Attraversamento in ordine differito (postorder): prima il sottoalbero di sinistra, poi il sottoalbero di destra, infine la radice:

```
void bit_postorder( Bit_node p ){
  if ( p ) {
    bit_postorder( p -> l );
    bit_postorder( p -> r );
    bit_printnode( p );
  }
}
```

Esercizio: dal vettore all'albero

Scrivete una funzione

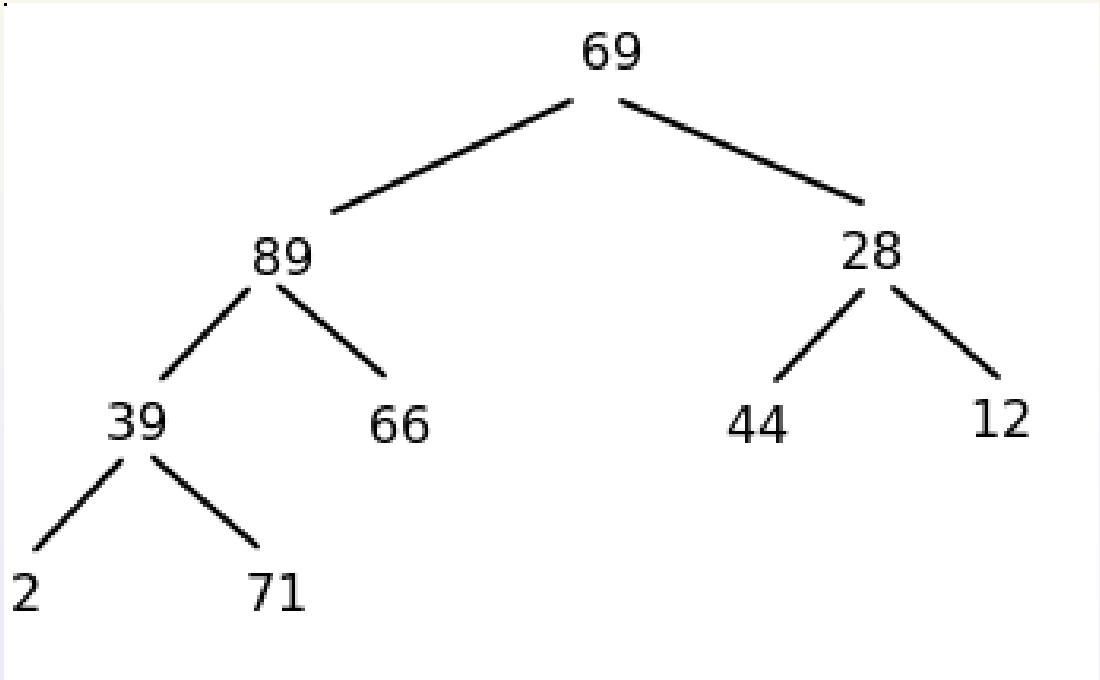
```
Bitnode bit_arr2tree( Item a[], int size, int j)
```

che, dato un array `a` di lunghezza `size` ed un indice `j`, costruisca ricorsivamente l'albero binario (completo)

- con radice contenente l'Item `a[0]`
- e tale che valga la seguente proprietà: se un nodo è etichettato con `a[i]`, allora il suo figlio sinistro è etichettato con `a[2*i+1]` e il suo figlio destro è etichettato con `a[2*i+2]`.

Esercizio: dal vettore all'albero - continua

Ad esempio, dato a = {69, 89, 28, 39, 66, 44, 12, 2, 71}, la funzione deve costruire l'albero



Alberi binari di ricerca

Sono una struttura dati che consente di rappresentare un insieme (dinamico) totalmente ordinato. Le operazioni previste sono:

- ricerca di un elemento nell'insieme
- verifica dell'appartenenza di un elemento all'insieme
- inserimento di un elemento nell'insieme
- cancellazione di un elemento dall'insieme
- ricerca del minimo e massimo dell'insieme
- successore e predecessore di un elemento nell'insieme

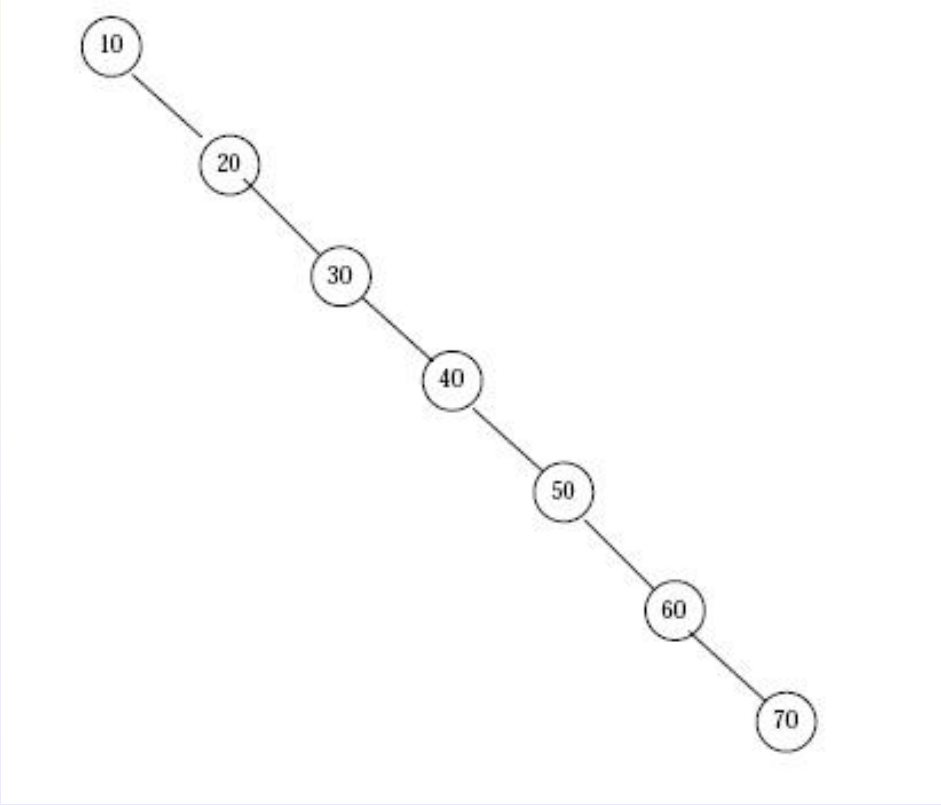
Rappresentazione di insiemi ordinati con alberi binari di ricerca

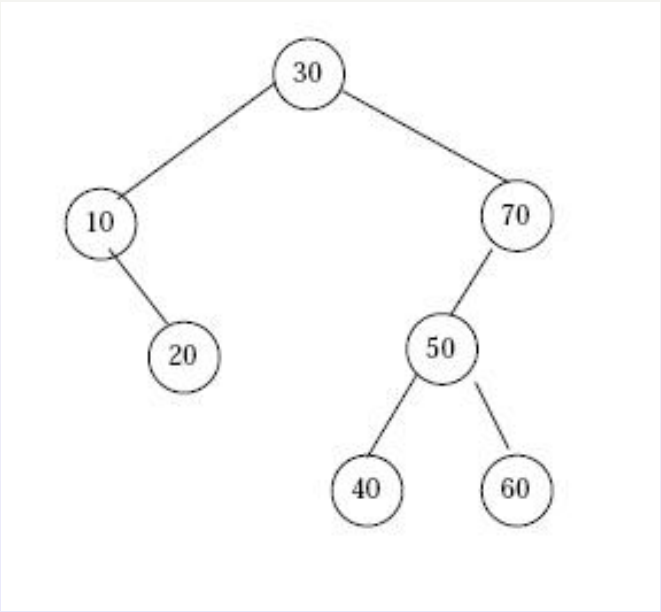
Se S è un insieme totalmente ordinato, lo rappresento come l'albero binario T avente per etichette gli elementi di S e tale che:

- per ogni $a \in S$, esiste un unico nodo v con etichetta a;
- per ogni nodo v di T:
  - se u appartiene al sottoalbero di sinistra di v, allora l'etichetta di u è minore dell'etichetta di v;
  - se u appartiene al sottoalbero di destra di v, allora l'etichetta di u è maggiore dell'etichetta di v.

La forma dell'albero dipende dall'ordine di inserimento degli elementi!

Differenza tra l'albero ottenuto inserendo gli elementi in ordine, oppure in questa sequenza: 30 10 70 20 50 40 60





Un insieme ordinato è rappresentabile con vari alberi binari di ricerca!!

Rappresentazione di insiemi ordinati con alberi binari di ricerca

Se S è un insieme totalmente ordinato, lo rappresento come un albero binario T avente per etichette gli elementi di S e tale che:

- per ogni $a \in S$, esiste un unico nodo v con etichetta a;
- per ogni nodo v di T:
  - se u appartiene al sottoalbero di sinistra di v, allora l'etichetta di u è minore dell'etichetta di v;
  - se u appartiene al sottoalbero di destra di v, allora l'etichetta di u è maggiore dell'etichetta di v.

Struttura dei nodi di un albero binario di ricerca

E' uguale a quella degli alberi binari, ma in questo caso assumiamo che il tipo `Item` sia dotato di una chiave, attraverso la quale si stabilisce l'ordine totale.

Prototipi di funzione

Alcune funzioni servono per la gestione generica di alberi binari e le riconosciamo dal prefisso **bit** (**bi**nary **t**ree).

```
Bit_node bit_new( Item );
void bit_destroy( Bit_node );

Item bit_item( Bit_node );
Bit_node bit_left( Bit_node );
Bit_node bit_right( Bit_node );
void bit_printnode( Bit_node );

void bit_preorder( Bit_node );
void bit_inorder( Bit_node );
void bit_postorder( Bit_node );

/* vedi esercizi: */
void bit_printassummary( Bit_node p, int spaces );
Bit_node bit_arr2tree( Item *a, int size, int i );
```

Prototipi di funzione

Caratterizziamo le funzioni specifiche degli alberi binari di ricerca col prefisso **bist** (**bi**nary **s**earch **t**ree).

```
Item bist_min( Bit_node p );
Item bist_max( Bit_node p );

/* stampa in ordine inverso: */
void bist_orderprint( Bit_node p );

/* stampa in ordine inverso: */
void bist_invorderprint( Bit_node p );

Item bist_search ( Bit_node r, Key k );
void bist_insert( Bit_node *q, Item item );
int bist_delete( Bit_node *r, Key k );
```