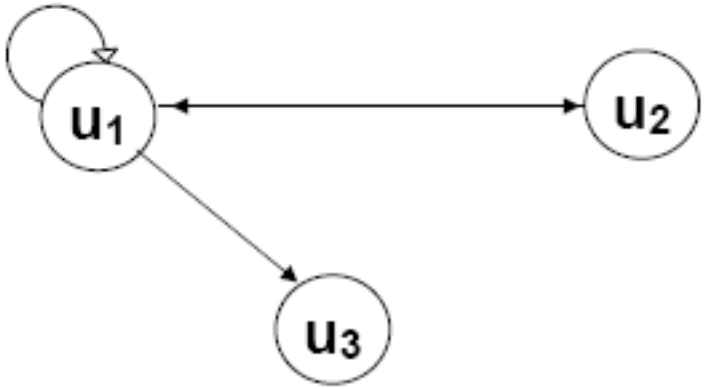
ADT *Albero binario*

I Grafi

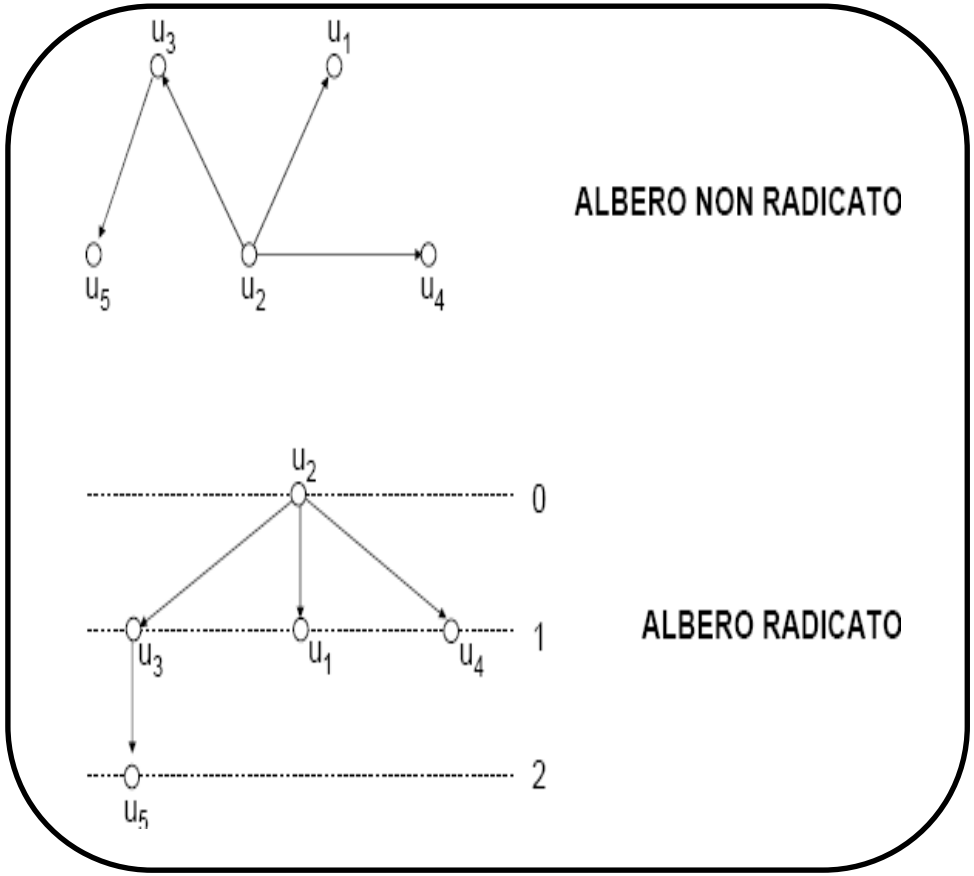
- Un grafo orientato G e' una coppia <N,A> dove N è un insieme finito non vuoto (insieme di nodi) e A $\subseteq$ NxN è un insieme finito di coppie ordinate di nodi, detti archi (o spigoli o linee).
- Se $< u_i, u_j > \in A$ nel grafo vi è un arco da u_i ad u_j
- Nell'esempio
  - $N = \{u_1,u_2,u_3\}$,
  - $A = \{(u_1,u_1),(u_1,u_2),(u_2,u_1),(u_1,u_3)\}$.



Gli Alberi

- Il GRAFO è una struttura dati alla quale si possono ricondurre strutture più semplici: LISTE ed ALBERI

- L'ALBERO è una struttura informativa per rappresentare:
  - partizioni successive di un insieme in sottoinsiemi disgiunti
  - organizzazioni gerarchiche di dati
  - procedimenti decisionali enumerativi

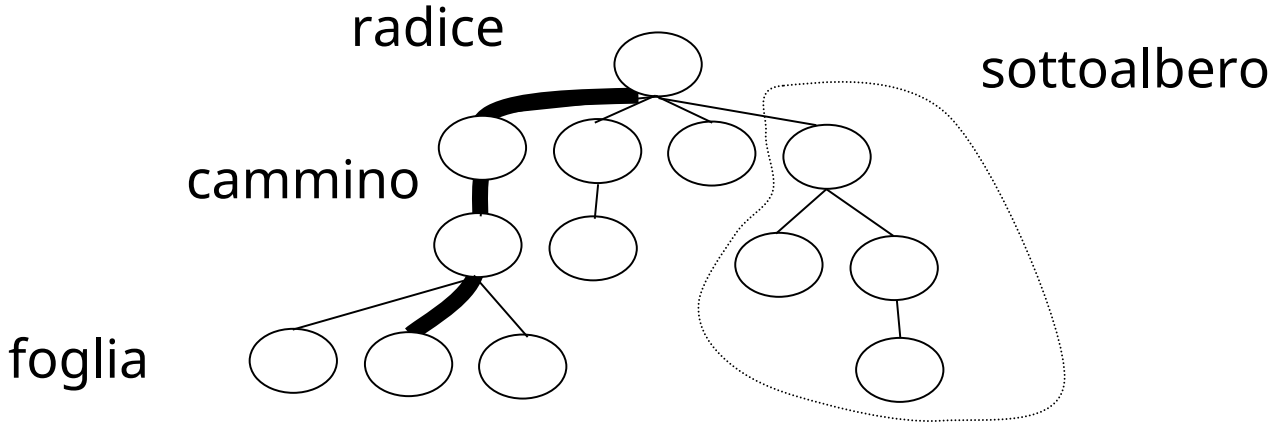


Strutture dati: Alberi

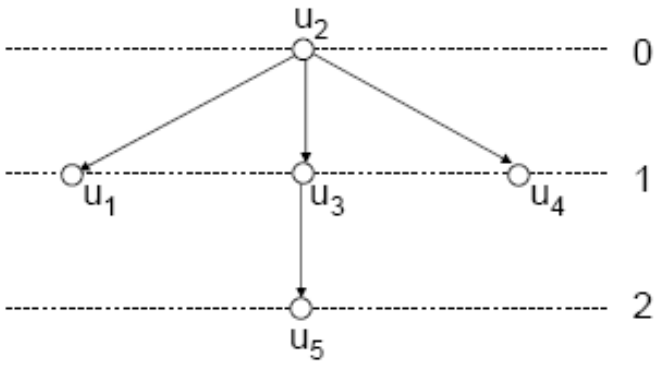
- Strutture gerarchiche di dati
- Esempi
  - Il file system di un sistema operativo
  - L'organigramma di un'azienda
- Alberi generali, alberi n-ari, alberi binari, …

Alberi

- Ogni nodo ha un unico arco entrante, tranne un nodo particolare, chiamato radice, che non ha archi entranti;
- Ogni nodo può avere zero o più archi uscenti
- Esiste un cammino fatto di archi che congiunge la radice con ciascuno dei nodi, e questo cammino è unico
- Non esistono cicli di archi
- I nodi senza archi uscenti sono detti foglie
- Un arco nell'albero induce una relazione padre-figlio
- A ciascun nodo è solitamente associato un valore, detto *etichetta* del nodo



Gli Alberi



ALBERO RADICATO ORDINATO

LA RADICE r E' A LIVELLO 0 E TUTTI I NODI u, ∃' < u, r > ∈ A, SONO FIGLI DI r E STANNO A LIVELLO 1 (r E' PADRE). NODI CON LO STESSO PADRE SONO FRATELLI. NODI TERMINALI SENZA FIGLI SONO DETTI FOGLIE.

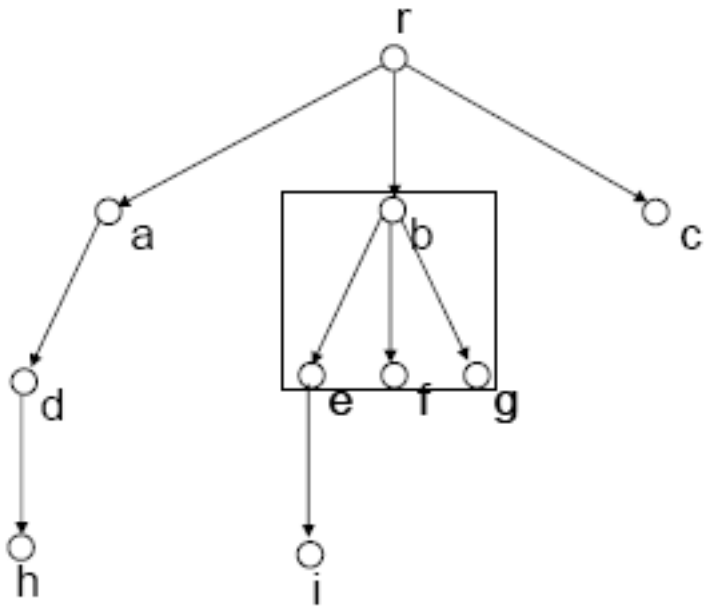
UN ALBERO ORDINATO E' OTTENUTO DA UNO RADICATO STABILENDO UN ORDINAMENTO TRA NODI ALLO STESSO LIVELLO.

Alberi: Alcuni concetti

- Grado di un nodo: numero di figli del nodo
- Cammino: sequenza di nodi $<n_0, n_1, \ldots, n_k>$ dove il nodo n_i è padre del nodo n_{i+1}, per $0 \leq i < k$
  - La lunghezza del cammino è k
  - Dato un nodo, esiste un unico cammino dalla radice dell'albero al nodo
- Livello di un nodo: lunghezza del cammino dalla radice al nodo
  - Definizione ricorsiva: il livello della radice è 0, il livello di un nodo non radice è 1 + il livello del padre
- Altezza dell'albero: la lunghezza del più lungo cammino nell'albero
  - Parte dalla radice e termina in una foglia

Gli Alberi

DEFINIAMO ALBERO DI ORDINE K UN ALBERO IN CUI OGNI NODO HA AL MASSIMO K FIGLI



ALBERO TERNARIO

Sottoalberi

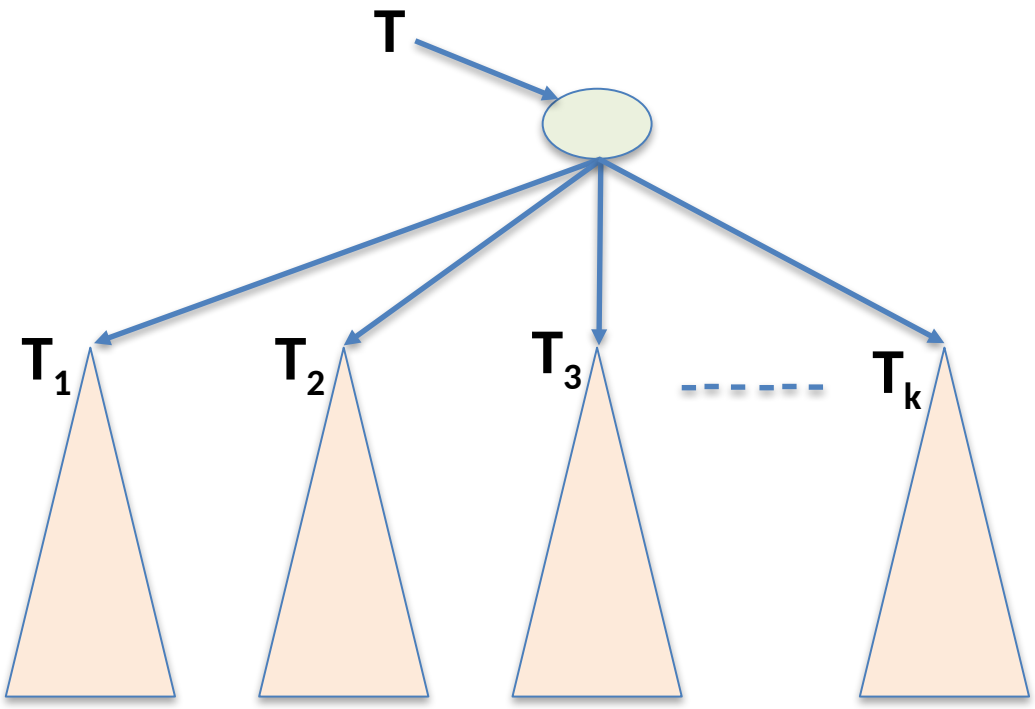
- dato un albero ed un suo nodo *u*, i suoi discendenti costituiscono un albero detto sottoalbero di radice *u*

Gli Alberi

- La natura **ricorsiva** degli alberi
  - Un albero può essere definito ricorsivamente
  - Un albero è un insieme di nodi ai quali sono associate delle informazioni
  - Un albero può essere vuoto
  - Tra i nodi di un albero non vuoto esiste un nodo particolare che è la radice (livello 0)
  - Gli altri nodi sono partizionati in sottoinsiemi che sono a loro volta alberi (livelli successivi):
    - *Vuoto o costituito da un solo nodo (detto radice)*
    - *Oppure è una radice cui sono connessi altri alberi*

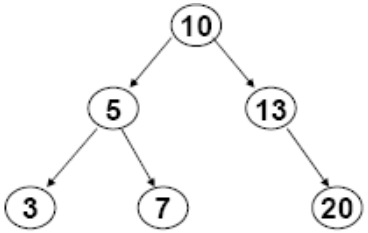
Gli Alberi

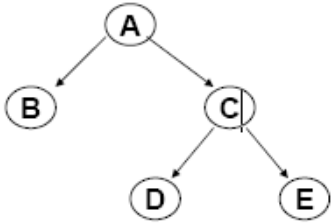
- La natura **ricorsiva** degli alberi

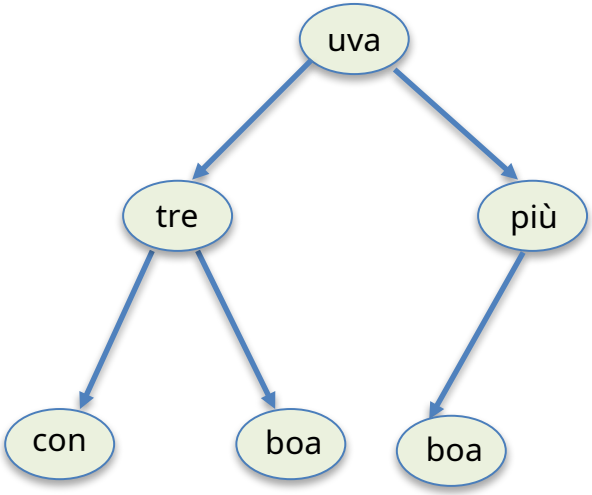


Gli Alberi binari

- Alberi Binari
  - Sono particolari alberi ordinati in cui ogni nodo ha al più due figli e si fa sempre distinzione tra il figlio sinistro, che viene prima nell'ordinamento e il figlio destro. Nell'esempio gli alberi sono etichettati con interi e con caratteri
  - Un albero binario è un grafo orientato che o è vuoto o è costituito da un solo nodo o è formato da un nodo n (detto radice) e da due sottoalberi binari, chiamati rispettivamente sottoalbero (o figlio) sinistro e sottoalbero (o figlio) destro







Alberi binari

- Particolari alberi n-ari con caratteristiche molto importanti
- Ogni nodo può avere al più due figli
  - sottoalbero sinistro e sottoalbero destro
- Definizione ricorsiva:
  - un albero senza nodi (albero vuoto) è un albero binario
  - una terna (r, s, d), dove r è un nodo (la radice), s e d sono alberi binari (sottoalberi sinistro e destro) è un albero binario
- Alberi binari semplificati
  - Costruttore bottom-up
  - Operatori di selezione
  - Operatori di visita

Gli Alberi Binari

SPECIFICA SINTATTICA

TIPI: *ALBEROBIN, BOOLEAN, NODO, ITEM*

OPERATORI:

newBtree : ( ) → ALBEROBIN
emptyBtree : (ALBEROBIN) → BOOLEAN
getRoot : (ALBEROBIN) → NODO
figlioSX : (ALBEROBIN) → ALBEROBIN
figlioDX : (ALBEROBIN) → ALBEROBIN
consBtree : (ITEM, ALBEROBIN, ALBEROBIN) → ALBEROBIN

Gli Alberi Binari

SPECIFICA SEMANTICA

TIPI:

ALBEROBIN = insieme degli alberi binari, dove:
$\Lambda \in$ ALBEROBIN (albero vuoto)
se N $\in$ NODO, T1 e T2 $\in$ ALBEROBIN
allora <N, T1, T2> $\in$ ALBEROBIN
BOOLEAN = {vero, falso}

NODO è un qualsiasi insieme non vuoto

ITEM è un qualsiasi insieme non vuoto disgiunto da NODO

Gli Alberi Binari

SPECIFICA SEMANTICA

OPERATORI:

newBtree ( ) = T
pre:
post: T = ∧

emptyBtree (T) = v
pre:
post: se T è vuoto, allora v = vero, altrimenti v = falso

getRoot (T) = N'
pre: T = <N, Tsx, Tdx> non è l'albero vuoto
post: N = N'

figlioSX (T) = T'
pre: T = <N, Tsx, Tdx> non è l'albero vuoto
post: T' = Tsx

Gli Alberi Binari

SPECIFICA SEMANTICA

OPERATORI:

figlioDX (T) = T'
pre: T = <N, Tsx, Tdx> non è l'albero vuoto
post: T' = Tdx

consBtree (elem, T1, T2) = T'
pre: elem ≠ NULLITEM
post: T' = <N, T1, T2>
N è un nodo con etichetta elem

Alberi binari: realizzazione

- Realizzazione più diffusa: struttura a puntatori con nodi doppiamente concatenati
- Ogni nodo è una struttura con 3 componenti:
  - Puntatore alla radice del sottoalbero sinistro
  - Puntatore alla radice del sottoalbero destro
  - Etichetta (useremo il tipo generico ITEM per questo campo)
- Un albero binario è definito come puntatore ad un nodo:
  - Se l'albero binario è vuoto, puntatore nullo
  - Se l'albero binario non è vuoto, puntatore al nodo radice

… e adesso un po’ di codice C

Dichiarazione del tipo nodo

- Per usare una lista concatenata serve una struttura che rappresenti i nodi
- La struttura conterrà i dati necessari (un intero nel seguente esempio) ed un puntatore al prossimo elemento della lista:

```
struct node {
  item value;                  /* etichetta del nodo*/
  struct node *left;    /* puntatore al sottoalbero sinistro*/
  struct node *right;    /* puntatore al sottoalbero destro*/
};
```

Dichiarazione del tipo Btree

- Il passo successivo è quello di dichiarare il tipo Btree

```
typedef struct node *Btree;
```

- una variabile di tipo Btree punterà al nodo radice dell'albero
- Assegnare a T il valore NULL indica che l'albero è inizialmente vuoto

```
Btree T = NULL;
```

Costruire un albero binario

- Un albero binario viene costruito in maniera bottom-up
- Man mano che costruiamo l'albero, creiamo dei nuovi nodi da aggiungere come nodo radice
- I passi per creare un nodo sono:
  1. Allocare la memoria necessaria
  2. Memorizzare i dati nel nodo
  3. Collegare il sottoalbero sinistro e il sottoalbero destro, già costruiti in precedenza

Realizzare il modulo *Btree*:
header file Btree.h

```
// file Btree.h


typedef struct node *Btree;

// prototipi

Btree newBtree(void);

int emptyBtree(Btree T);

Btree figlioSX(Btree T);

Btree figlioDX(Btree T);

Btree consBtree(item val, Btree sx, Btree dx);

node *getRoot (Btree T);
```

L'ADT Btree è indipendente dal tipo degli elementi contenuti.

Utilizziamo un tipo generico item definite in un file item.h

Realizzazione di Btree: file Btree.c

```
#include <stdio.h>
#include <stdlib.h>
#include “item.h”
#include “Btree.h”


struct node {
    item value;
    struct node *left;
    struct node *right;
};

item getItem(node *N);
void setItem(node *N, item el);

item getItem(node *N)
{
     if (N == NULL)  return NULLITEM;
     return N->value;
}

void setItem(node *N, item el)
{
    if (N==NULL) return;
    N->value = el;  // correttezza di =
}                   // dipende dal tipo item
```

```
Btree newBtree(void)
{
    return NULL;
}

int emptyBtree(Btree T)
{
    return T == NULL;
}

node *getRoot(Btree T)
{
    return T;
}

Btree consBtree(item val, Btree sx, Btree dx)
{
     struct node *nuovo;
     nuovo = malloc (sizeof(struct node));
     if (nuovo != NULL) {
            setItem(nuovo, val);
            nuovo->left = sx;
            nuovo->right = dx;
     }
     return nuovo;
}
```

Realizzazione di Btree: file Btree.c

```
Btree figlioSX(Btree T)
{
     if (T != NULL)
          return  T->left;
     else
          return NULL;
}
```

```
Btree figlioDX(Btree T)
{
     if (T != NULL)
          return  T->right;
     else
          return NULL;
}
```

Alcune note sull'ADT Btree

- L'insieme degli operatori così definiti costituisce l'insieme degli operatori di base (il minimo insieme di operatori) di un albero binario
- Ogni altro operatore che si volesse aggiungere all'ADT albero binario potrebbe essere implementato utilizzando gli operatori dell'insieme di base
- E' frequente la pratica di arricchire il tipo Btree con l'aggiunta di operatori per inserire o cancellare nodi in determinate posizioni dell'albero

Algoritmi di Visita

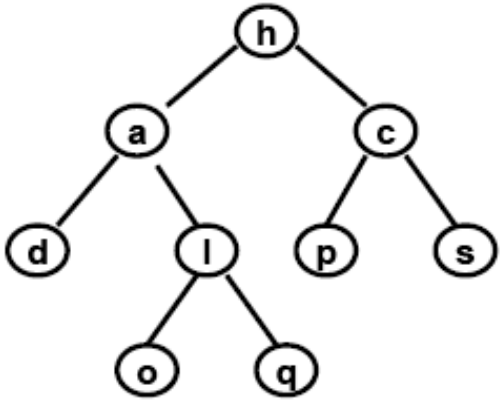
- La visita di un albero non vuoto consiste nel seguire una rotta di viaggio che consenta di esaminare ogni nodo dell'albero esattamente una volta.

  - Visita in **pre-ordine**: richiede dapprima l'analisi della radice dell'albero e, poi, la visita, effettuata con lo stesso metodo, dei due sottoalberi, prima il sinistro, poi il destro
  - Visita in **post-ordine**: richiede dapprima la visita, effettuata con lo stesso metodo, dei sottoalberi, prima il sinistro e poi il destro, e, in seguito, l'analisi della radice dell'albero
  - Visita **simmetrica**: richiede prima la visita del sottoalbero sinistro (effettuata sempre con lo stesso metodo), poi l'analisi della radice, e poi la visita del sottoalbero destro

Algoritmi di Visita

ESEMPIO:

SIA UN ALBERO BINARIO CHE HA DEI CARATTERI NEI NODI



LA VISITA IN PREORDINE: h a d l o q c p s

LA VISITA IN POSTORDINE: d o q l a p s c h

LA VISITA SIMMETRICA: d a o l q h p c s

Algoritmi di Visita

visita in preordine l'albero binario t

{

se l'albero non è vuoto

allora

    visita la radice di t

    visita in preordine il sottoalbero sinistro di t

    visita in preordine il sottoalbero destro di t

fine

}

Algoritmi di Visita

```
void inorder (Btree T)
{
    if (emptyBtree(T)) return;
    inorder(figlioSX(T));
    output_item(getItem(getRoot(T)));
    inorder(figlioDX(T));
}
```

Esercizi

- Realizzare una funzione inputBtree() per costruire ricorsivamente un albero binario
- Realizzare delle funzioni per determinare l'altezza e il numero di nodi di un albero binario
- Realizzare le tre visite dell'albero binario in maniera iterativa, con l'uso di uno stack, e in maniera ricorsiva
- Realizzare una visita per livelli di un albero binario (avanzato)

- Realizzare una funzione inputBtree() per costruire ricorsivamente un albero binario

l'albero è vuoto? // caso di base

ritorna newBtree()

inserisci la radice el

T1 = inputBtree() // costruisci sottoalbero SX

T2 = inputBtree() // costruisci sottoalbero DX

//

ricorsivamente

ritorna consBtree(el, T1, T2)

Realizzare una funzione inputBtree() per costruire ricorsivamente un albero binario

```
Btree  inputBtree(void)
{
        Btree T1, T2;
        int ris;
        item el;
        printf("\nL'albero è vuoto? (1/0): ");
        scanf("%d", &ris);
        if (ris)  return newBtree();
        printf("\nInserisci la radice: ");
        input_item(&el);
        printf ("costruisco il sottoalbero SX\n");
        T1 = inputBtree();
        printf ("costruisco il sottoalbero DX\n");
        T2 = inputBtree();
        return consBtree(el, T1, T2);
}
```