

esercizi su alberi n-ari

Alberi N-ari

```
#include <stdio.h>
#include<stdlib.h>

struct nodoalberon{
    struct nodoalberon *fratello;
    int elem;
    struct nodoalberon *primofiglio;
};

typedef struct nodoalberon TREENODEN;
typedef TREENODEN *ALBERONARIO;
```

Alberi N-ari

```
/*costruzione di un albero nario inserito da
input*/
ALBERONARIO creaalbero();

/*Verifica se l'albero e' vuoto*/
int alberonvuoto(ALBERONARIO);

/*stampa in inorder i valori contenuti
nell'albero*/
void stampapreordinealberon(ALBERONARIO);

/*calcola il massimo valore contenuto in un
albero n-ario non vuoto*/
int maxalberon(ALBERONARIO);

/*verifica se un valore intero specificato
sia o meno presente nell'albero */
int findalberon(ALBERONARIO, int);
```

Main

```
main()
{
    ALBERONARIO TT;
    int i;
    TT=creaalbero();
    stampapreordinealberon(TT);
    printf("inserisci valore da ricercare \n");
    scanf("%d",&i);
    if (findalberon(TT,i))
        printf("valore %d presente nell'albero\n",i);
    else
        printf("valore %d non presente nell'albero \n",i);
    if (TT!=NULL)
        printf("valore massimo nell'albero e':%d
\n",maxalberon(TT));
}
```

Esercizio 1

Si scriva una funzione che prende da Input coppie <valore, numero di figli> e costruisce un albero in preordine

```
/*Pre: l'albero che si costruisce e'  
NON vuoto*/  
/*Post: costruisce in preordine un  
albero n-ario*/
```

Esercizio 1b

```
ALBERONARIO creaalbero()  
{  
    ALBERONARIO T, temp;  
    int i,j,k;  
    scanf("%d %d",&i,&j);  
    T=malloc(sizeof(TREENODEN));  
    if (T==NULL)  
        printf("memoria non allocata\n");  
    else  
    {  
        T->elem=i;  
        T->fratello=NULL;  
        if (j==0)  
            {  
                T->primofiglio=NULL;  
            }  
        else  
            T->primofiglio=creaalbero();  
        temp=T->primofiglio;  
        for (k=0;k<j-1;k++)  
            {  
                /* printf("entrato nel ciclo con i pari a %d\n",i);*/  
                temp->fratello=creaalbero();  
                /*printf("valore mio e del fratello %d %d\n",temp->elem, temp->fratello->elem);*/  
                temp=temp->fratello;  
            }  
    }  
    return T;  
}
```

Esercizio 2

Si scriva una procedura che dato un albero n-ario lo stampi in preordine

```
/*Post: stampa in preorder (prima la
       radice quindi il primo dei suoi
       sottoalberi,
       quindi il secondo dei suoi sottoalberi
       etc... i valori contenuti nei nodi
       dell'albero*/
```

Esercizio 2b

```
void stampapreordinealberon(ALBERONARIO T)
{
    ALBERONARIO temp;
    if (T!=NULL)
    {
        printf("%d\n",T->elem);
        temp=T->primofiglio;
        while (temp!=NULL)
            { stampapreordinealberon(temp);
              temp=temp->fratello;
            }
    }
    return;
}
```

Esercizio 3

Si scriva una funzione che dato un albero n-ario calcoli il valore massimo contenuto nell'albero

```
/* Pre: albero non vuoto
Post: restituisce il valore massimo
contenuto nell'albero */
```

Esercizio 3b

```
int maxalberon(ALBERONARIO T)
{
    ALBERONARIO temp=T->primofiglio;
    int massimo = T->elem;
    int currmax;

    while (temp!=NULL)
    {
        currmax=maxalberon(temp);
        if (currmax>massimo)
            massimo=currmax;
        temp=temp->fratello;
    }
    return massimo;
}
```

Esercizio 4

Si scriva una funzione che dato un albero n-ario ed un intero x verifichi se x compare tra i valori contenuti nell'albero

```
/*Post: restituisce 1 se il valore x
e' contenuto nell'albero, 0
altrimenti */
```

Esercizio 4b

```
int findalberon(ALBERONARIO T, int x)
{
    ALBERONARIO temp;
    int trovato;
    if (T==NULL) return 0;
    if (T->elem==x) return 1;
    temp=T->primofiglio;
    trovato=0;
    while ((temp!=NULL)&& !trovato)
    {
        if (findalberon(temp,x))
            trovato=1;
        temp=temp->fratello;
    }
    return trovato;
}
```

Esercizio 5

Si scriva una funzione che dato un albero n-ario verifichi se tutti i valori dei suoi nodi sono pari

```
/* verifico se tutti gli elementi di
   un albero n-ario sono pari*/
int tutti_pari_P (Nodo * albero)
{
  if (!albero) return 1;
  if (albero->valore % 2) return 0;
  else
    return (tutti_pari_P(albero->figlio)
            && tutti_pari_P(albero->fratello));
}
```

Esercizio 6

Si scriva una funzione che dato un albero n-ario calcoli la sua altezza

```
/* Post: restituisce l'altezza dell'albero generico */
int altezza(ALBERONARIO T)
{
  ALBERONARIO temp;
  int altezza=0;
  int curralt;
  if (T==NULL) return 0;
  temp=T->primofiglio;

  while (temp!=NULL)
  {
    curralt=altezza(temp);
    if (curralt>altezza) altezza=curralt;
    temp=temp->fratello;
  }
  return (altezza+1);
}
```

Esercizio 7

Si scriva una funzione che dato un albero n-ario calcoli il numero delle sue foglie

```
/* calcola il numero di foglie dell'albero
   generico*/
int numfoglie (ALBERONARIO T)
{
    if (T==NULL) return 0;
    if (T->primofiglio ==NULL)
        return 1+ numfoglie(T->fratello);
    else
        return (numfoglie (T->primofiglio)+
                numfoglie(T->fratello));
}
```

Esercizio 8

Si scriva una funzione che dato un albero n-ario calcoli il numero di nodi con k figli

```
/* calcola il numero di nodi con k figli*/
int nodiKfigli (ALBERONARIO T, int k)
{
    int k1=0;
    ALBERONARIO temp;
    if (T == NULL) return 0;
    else
    {
        temp=T->primofiglio;
        while (temp!=NULL)
        {
            k1++;
            temp = temp->fratello;
        }
        return ((k==k1)?(1+nodiKfigli(T->fratello)+
                                nodiKfigli(T->primofiglio)):
                (nodiKfigli(T->fratello)+nodiKfigli(T->primofiglio)));
    }
}
```

Esercizio 9

Scrivere funzione che dato un albero N-ario verifichi se ogni nodo pari ha solo figli dispari

```
/* verifica: ogni nodo pari ha solo figli dispari */
int paridispari(ALBERONARIO T)
{
    ALBERONARIO temp;
    int ris=1;
    if (T==NULL)
        return 1;
    if (T->elem%2) /*caso in cui il nodo e' dispari*/
    {
        temp=T->primofiglio;
        while (temp!=NULL)
        {
            ris=ris&&paridispari(temp);
            temp=temp->fratello;
        }
        return ris;
    }
    else /*caso in cui il nodo radice e' pari*/
    {
        temp=T->primofiglio;
        while (temp!=NULL)
        {
            ris=ris&&(temp->elem%2)&&paridispari(temp);
            temp=temp->fratello;
        }
        return ris;
    }
}
```