Esercitazioni di Prog. II (esercizi su alberi binari)

Chiara Petrioli

Alberi Binari

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

struct nodoalbero{
	struct nodoalbero *right;
	int elem;
	struct nodoalbero *left;
};

struct answer{
	int completo;
	int pieno;
	int altezza;
};
typedef struct nodoalbero TREENODE;
typedef TREENODE *ALBERO;
typedef struct answer RISPOSTA;
typedef RISPOSTA *RISPOSTAPTR;
```

Alberi Binari

```
/*Costruzione di un albero Binario inserito da input. Si assume che gli interi
 * contenuti nei nodi dell'albero siano inseriti da input in preordine
 * Il formato dell'input saranno coppie di interi in cui il primo valore
 * corrisponde al valore da inserire nel nodo, il secondo valore indica il
 * numero dei suoi figli (0 nessun figlio, 1 solo il figlio sinistro, -1 solo
 * il figlio destro, 2 entrambi i figli). I nodi sono inseriti i preordine (prima la
 * radice), quindi i nodi del sottoalbero sinistro, quindi i nodi del sottoalbero
 * destro. Ad esempio:
 * 3 2
 * 1 0
 * 4 2
 * 5 0
 * 7 0
 *
 * costruisce l'albero binario
 *    3
 *  1    4
 *     5   7
 *  La seguente procedura costruisce l'albero binario */

ALBERO creaalbero();

/*Verifica se l'albero e' vuoto*/
int alberovuoto(ALBERO);

/*stampa in inorder i valori contenuti nell'albero*/
void stampainalbero(ALBERO);

/*calcola il numero di nodi dell'albero al livello specificato*/
int numelemlivello (ALBERO, int, int);

/*calcola il numero di foglie dell'albero*/
int numfoglie(ALBERO);

/*stampa l'albero indentato*/
void stampaindentata(ALBERO, int);

/*verifica se un albero binario e' completo. Restituisce una struttura il cui campo completo contiene 1 se cio' e' vero
 * i campi pieno e altezza contengono rispettivamente 1/0 a seconda che l'albero sia pieno o meno, e l'altezza
 * dell'albero */
RISPOSTAPTR completo(ALBERO);

/*trasforma un albero nel suo speculare*/
void speculare (ALBERO);
```

Main

.....

```
main()
{
      ALBERO TT;
      int livello;
      RISPOSTAPTR Risp;
      TT=creaalbero();
      stampainalbero(TT);
      printf("inserisci livello:\n");
      scanf("%d",&livello);
      printf("stampa indentata dell'albero \n");
      stampaindentata(TT,0);
      printf("il numero di elementi dell'albero al livello %d e':%d
      \n",livello,numelemlivello(TT,livello,0));
      printf("il numero di foglie dell'albero e': %d \n", numfoglie(TT));
      Risp=completo(TT);
      if (Risp->completo)
            printf("l'albero e' completo \n");
      else
            printf("l'albero NON e' completo\n");
      if (Risp->pieno)
            printf("l'albero e' pieno \n");
      else
            printf("l'albero NON e' pieno \n");
      printf("l'altezza dell'albero e':%d\n",Risp->altezza);
      }
```

Esercizio 1

```
/*Pre: l'albero che si costruisce e' NON vuotoPost: costruisce in preordine un albero binario*/
ALBERO creaalbero()
{
        ALBERO T;
        int i,j;
        scanf("%d %d",&i,&j);
        T=malloc(sizeof(TREENODE));
        if (T==NULL)
                printf("memoria non allocata\n");
        else
        {
                T->elem=i;
                switch(j)
                {               case 0: T->right=NULL;
                                        T->left=NULL;
                                        break;
                                case 1: T->left=creaalbero();
                                        T->right=NULL;
                                        break;
                                case -1:T->left=NULL;
                                        T->right=creaalbero();
                                        break;
                                case 2: T->left=creaalbero();
                                        T->right=creaalbero();
                                        break; }
                return(T);
        }
}
```

Si scriva una funzione che prenda da Inout coppie <valore, numero di figli> E costruisca un Albero in preordine

Esercizio 2

```
/*Post:Restituisce 1 se l'albero e' vuoto, 0 altrimenti*/
int alberovuoto(ALBERO T)
{
        return(T==NULL);
}
/*Post: stampa in inorder i valori contenuti nei nodi dell'albero*/
void stampainalbero(ALBERO T)
{
        if (T!=NULL)
        {
                stampainalbero(T->left);
                printf("%d\n",T->elem);
                stampainalbero(T->right);
        }
        return;
}
```

Si scriva una procedura
Che dato un
Albero lo stampi
In inordine

Esercizio 3

```
/* OSSERVAZIONE: La funzione e' chiamata dal main con l (livello del nodo corrente) ==0 e sul nodo radice
 * Post: calcola il numero di nodi di livello h nell'albero. Se l'albero e' vuoto
 * o h e' maggiore dell'altezza dell'albero restituisce 0 */
int numelemlivello(ALBERO T, int h, int l)
{
	if ((T==NULL)||(h<0))
		return 0;
	else
	{
		if (l==h)
			return 1;
		else
			return (numelemlivello(T->left,h,l+1)
			numelemlivello(T->right,h,l+1));
	}
}
```

Si scriva una funzione Che dato un Albero binario Ed un livello h Calcoli il numero di Nodi al livello h Dell'albero

Esercizio 4

```
/*Post: restituisce il numero di foglie dell'albero binario. 0 se l'albero e' vuoto*/
int numfoglie(ALBERO T)
{
    if (T==NULL)
        return 0;
    else
    {
        if ((T->left ==NULL) && (T->right ==NULL))
            return 1;
        else
            return (numfoglie(T->left)+numfoglie(T->right));
    }
}
```

Si scriva una funzione Che dato un Albero binario Calcoli il numero di Foglie Dell'albero

Esercizio 5

```
/*OSSERVAZIONE: nel main chiamata con h a zero*/
/*Post: stampa i valori dell'albero indentati*/
void stampaindentata (ALBERO T, int h)
{
      int i;
      if (T!=NULL)
      {
            stampaindentata(T->right,h+1);
            for (i=1;i<=h;i++)
                  printf("\t");
            printf("%d \n", T->elem);
            stampaindentata(T->left,h+1);
      }

}
```

Si scriva una procedura
Che dato un
Albero binario
Lo stampi in
Inordine indentando
I vari valori un num
Di \tab pari al
Loro livello

Esercizio 6 (1/2)

```
/*Post: restituisce una struttura contenente info sul fatto che l'albero sia
 * o meno completo, pieno e sulla sua altezza*/
RISPOSTAPTR completo(ALBERO T)
{
    RISPOSTAPTR temp, temp1, temp2;
    if (T==NULL)
    {
        temp=malloc(sizeof(RISPOSTA));
        if (temp==NULL)
        {
            printf("memoria non allocata \n");
            return NULL;
        }
        else
        {
            temp->completo=1;
            temp->pieno=1;
            temp->altezza =0;
            return temp;
        }
    }
```

Esercizio 6 (2/2)

```
else
        {
                temp1=completo(T->left);
                temp2=completo(T->right);
                temp=malloc(sizeof(RISPOSTA));
                if(temp==NULL)
                {
                        printf("memoria non allocata \n");
                        return NULL;
                }
                else
                {
                        if(((temp1->completo)&&(temp2->completo))
                        &&(((temp1->altezza-temp2->altezza==0)&&(temp1->pieno))||
                        ((temp1->altezza-temp2->altezza==1)
                                &&(temp2->pieno))))
                                temp->completo=1;
                        else
                                temp->completo=0;
                        temp->pieno=((temp1->pieno)&&(temp2->pieno)&&
                        (temp1->altezza==temp2->altezza));
                        temp->altezza=(temp1->altezza > temp2->altezza)?(temp1->altezza+1):
                        (temp2->altezza+1);
                        return temp;
                }
        }
}
```

Esercizio 7

```
/*Post:trasforma un albero nel suo speculare*/
void speculare(ALBERO T)
{
    ALBERO temp;
    if (T!=NULL)
    {
        temp=T->left;
        T->left=T->right;
        T->right=temp;
        speculare(T->left);
        speculare(T->right);
    }
}
```

Si scriva una procedura
Che dato un
Albero binario
Lo trasformi nel
Suo speculare