*Programmazione*

# *Esercizi svolti in aula*

*(25-05-2015)*

# /* Esercizio 1 */

/*

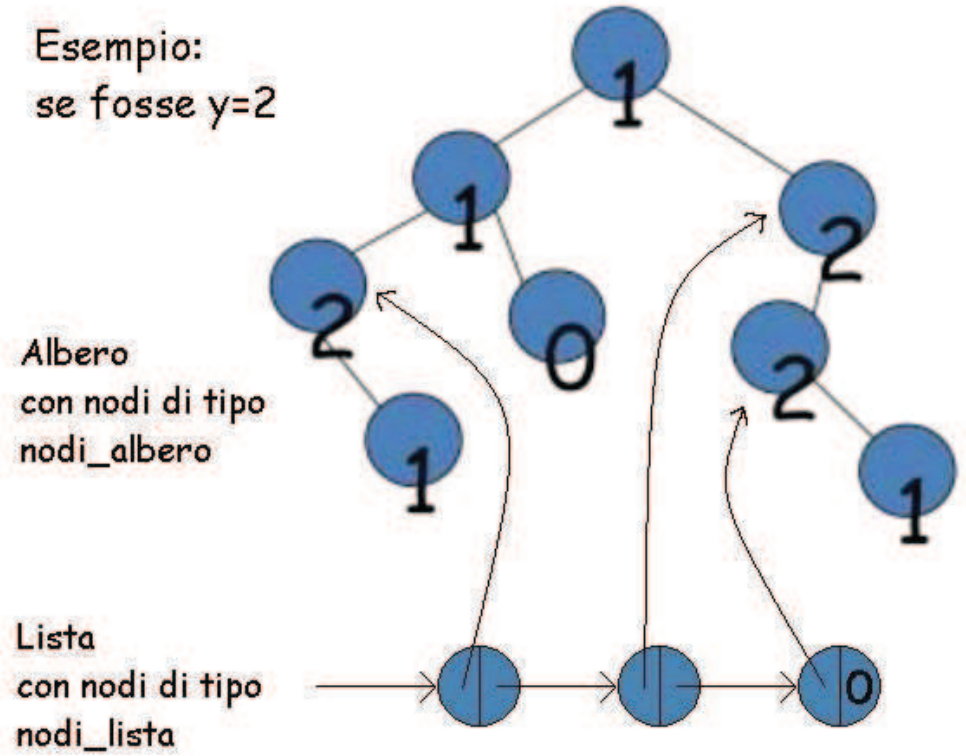
dato un intero y e un albero(r) lo si deve percorrere (in ordine infisso) per trovarne tutti i nodi che abbiano campo info==y.

si vuole restituire una lista di nodi i cui campi info siano, nell'ordine, i puntatori ai nodi trovati.

(possibili varianti: percorrere l'albero in ordine infisso e/o postfisso).

il codice che trovate a seguire è disponibile anche all'indirizzo: http://pastebin.com/ExgeMhkX

*/



```
struct nodo_albero
{
  //membri - campi dati
  int info;
  nodo_albero *left, *right;

  //costruttore
  nodo_albero(int a=0, nodo_albero* b=0, nodo_albero*c=0)
  { info=a;
    left=b;
    right=c;
  }
};


struct nodo_lista
{
  //membri - campi dati
  nodo_albero *p;
  nodo_lista *next;

  //costruttore
  nodo_lista(nodo_albero *a=0, nodo_lista* b=0)
  { p=a;
    next=b;
  }
};
```

```
28. nodo_lista *conc(nodo_lista *L1, nodo_lista *L2)
29. {  //PRE=(lista(L1) e lista(L2) sono corrette).
30.   if(!L1)  //se L1 è vuota, ritorno semplicemente L2
31.     return L2;
32.   else  //altrimenti scorro L1 fino in fondo, al ritorno riattacco tutto
33.   {
34.     L1->next=conc(L1->next, L2);  //fa un sacco di "riagganci" inutili
35.     return L1;
36.   }
37. }  //POST=(detto c il puntatore restituito, allora lista(c)=lista(a)@lista(b)).
38.
39.
40. void conc_rif(nodo_lista *&L1, nodo_lista *L2)
41. {  //PRE=(lista(L1) e lista(L2) sono corrette, lista(vL1)=lista(L1) e lista(vL2)=lista(L2)).
42.   if(!L1)
43.     L1=L2;  //se L1 è vuota "restituisco" L2 tramite side-effect su L1
44.   else
45.     conc_rif(L1->next, L2);
46. }  //POST=(al ritorno di questa funzione sarà: lista(L1)==lista(vL1)@lista(L2) e lista(L2==lista(vL2)).
47.
48.
```

```
49. nodo_lista *f(nodo_albero *r, int y)
50. {  //PRE=(albero(r) corretto, albero(vr)=albero(r), y definito).
51.   if(!r)  //caso base: albero vuoto
52.     return 0;  //ritorno una lista vuota
53.   else
54.   {
55.     nodo_lista *L1=f(r->left, y);  //ottengo il risultato del sottoalbero sx
56.     nodo_lista *L2=f(r->right, y);  //ottengo il risultato del sottoalbero dx
57.
58.     if(r->info==y)  //valutazione nodo corrente
59.       L2=new nodo_lista(r, L2);  //attacco in testa ad L2 il nodo corrente
60.
61.     return conc(L1, L2);  //se volessi usare conc_rif dovrei sostituire questa riga con: conc_rif(L1, L2); return L1;
62.   }
63. }  //POST=(ritorna una lista (corretta) di puntatori ai nodi di albero(r) che hanno campo info==y in ordine infisso, albero(r)==albero(vr) cioè non è stato modificato).
```

```
/* VARIANTI:

- variante 0 - ordine INFISSO
quella soprastante, r (nodo attuale) se r->info==y devo generare L1@r::L2

- variante 1 - ordine PREFISSO
r (nodo attuale) se r->info==y andrebbe prima di tutto, cioè in testa ad L1, dovrei generare: r::L1@L2
quindi dopo le invocazioni ricorsive su sotto-albero sx e dx avrei:
  if(r->info==y)  //valutazione nodo corrente
    L1=new nodo_lista(r, L1);  //attacco in testa ad L1 il nodo corrente
  return conc(L1, L2);  //se volessi usare conc_rif dovrei sostituire questa riga con: conc_rif(L1, L2); return L1;
Oppure, sempre dopo le invocazioni ricorsive:
  if(r->info==y)
    return new nodo_lista(r, conc(L1, L2));  //se volessi usare conc_rif dovrei sostituire questa riga con: conc_rif(L1, L2); return new nodo_lista(r, L1);
  else
    return conc(L1, L2);  //se volessi usare conc_rif dovrei sostituire questa riga con: conc_rif(L1, L2); return L1;

- variante 2 - ordine POSTFISSO
r (nodo attuale) se r->info==y andrebbe in fondo, cioè in coda ad L2, dovrei generare: L1@L2@r
quindi dopo le invocazioni ricorsive su sotto-albero sx e dx avrei:
  if(r->info==y)  //valutazione nodo corrente
  {
    nodo_lista L3=new nodo_lista(r, 0);
    L2=conc(L2, L3);  //se volessi usare conc_rif dovrei sostituire questa riga con: conc_rif(L2, L3);
  }     //o semplicemente: L2=conc(L2, new nodo_lista(r, 0));  //se volessi usare conc_rif dovrei sostituire questa riga con: conc_rif(L2, new nodo_lista(r, 0));
  return conc(L1, L2);  //se volessi usare conc_rif dovrei sostituire questa riga con: conc_rif(L1, L2); return L1;
*/
```

# /* Esercizio 2 */

```
/*
l'esercizio 2 è composto di 3 step successivi:
I) si chiede solo la profondità della foglia minima
II) oltre alla profondità si chiede anche di restituire un puntatore alla foglia minima
III) come il II) ma si chiede di ottimizzare il codice in modo tale da fare meno "esplorazioni possibili"
*/
```

```
1.  //nel seguito viene utilizzata la seguente struttura, per "manipolare" al contempo un puntatore e una profondità
2.  struct foglia
3.  {
4.    //membri - campi dati
5.    nodo *fo;
6.    int prof;
7.
8.    //costruttore
9.    foglia(nodo *a=0, int b=INT_MAX)
10.   { fo=a;
11.     prof=b;
12.   }
13. };
```

trovare profondità minima tra le foglie

e poi vogliamo anche una  foglia a profondità minima

usiamo:

```
bool leaf(nodo *n)
{return (!n->left && !n->right);}
```

//PRE=(x albero corretto non vuoto, prof def.)

```
int  prof_min(nodo*x, int prof)
```

//POST=(restituisce k t.c. k-prof è profondità minima di una foglia in x)

ATTENZIONE: PRE richiede di fermarci prima di esaurire l'albero !!

```
int  prof_min(nodo*x, int prof)
{if(leaf(x)) return prof;
 int a=INT_MAX,b=INT_MAX;
 if(x->left)
   a=prof_min(x->left,prof+1);
 if(x->right)
   b=prof_min(x->right,prof+1);
   if(a <= b)
     return a;
   else
     return b;
}
```

vogliamo anche il puntatore al nodo:

la funzione restituisce un valore:

struct foglia{nodo* fo; int prof;};

PRE=(x punta ad albero corretto anche vuoto, prof è definito)

non dobbiamo preoccuparci di esaurire l'albero

```
foglia  prof_min(nodo*x, int prof)
{if(x )
        if(leaf(x))
                return foglia(x,prof);
        else
        {foglia a = prof_min(x->left,prof+1);
        foglia b=prof_min(x->right,prof+1);
        if(a.prof>b.prof) return b;
        else
          return a;
        }
  return foglia(0,INT_MAX);
}
```

NOTARE:
niente
allocazione
dinamica
PROBLEMI?

altra soluzione più efficiente:
inutile cercare a profondità k se abbiamo già trovato una foglia a profondità minore o uguale di k

passaggio per riferimento

```
void f(nodo*x,int prof, foglia & m)
{ if(prof>=m.prof) return;
if(x)
        {
         if(leaf(x))
              {m.fo=x; m.prof=prof; return;}
          else
              {
                    f(x->left,prof+1,m);
                    f(x->right,prof+1,m);
              }

        }
}
```

invocazione:

```
foglia p(0,INT_MAX);
f(root,0,p);
```