preparazione II compitino

terza puntata

Esercizio 2 del 4/3/2014

Abbiamo un albero binario r un valore intero y e k>=0 e vogliamo sapere se r contiene un **cammino di r che si estende dalla radice fino ad una foglia** che contiene esattamente k nodi con campo info uguale a y.

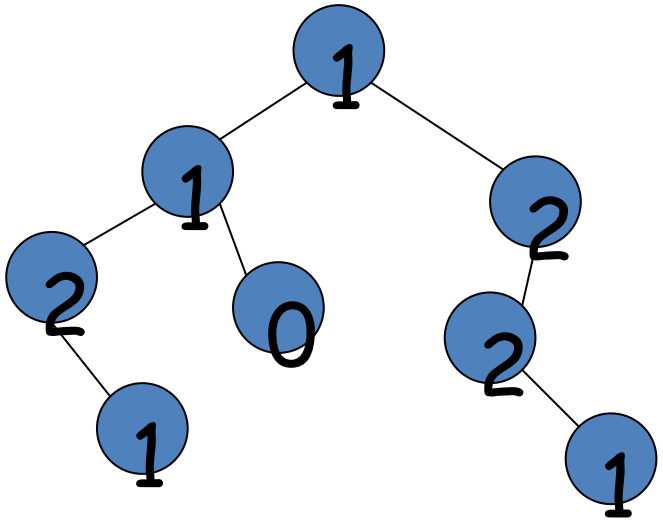
C'è anche il problema di creare un albero da una sequenza di interi letti da "input". Seguiremo l'idea di un esempio contenuto nelle slide della lezione 18 sugli alberi binari. Leggeremo degli interi da "input" e costruiremo un albero aggiungendo, dopo la lettura di x da "input", un nuovo nodo con campo info x, all'albero in costruzione, inserendolo nel sotto-albero sinistro o in quello destro a seconda di quale dei due ha meno nodi. Il nuovo nodo va inserito nel sotto-albero con meno nodi (l'idea è di creare un albero bilanciato) e naturalmente questo deve essere fatto ricorsivamente ad ogni livello dell'albero fino a trovare un punto in cui sia possibile inserire il nuovo nodo. In caso sotto-albero sinistro e destro abbiano lo stesso numero di nodi, la funzione deve inserire il nuovo nodo nel sotto-albero sinistro.

Questa funzione si chiama crea_a ed ha il prototipo

void crea_a(nodo*&r, int dim, ifstream & INP) e soddisfa le seguenti pre- e post-condizioni:
PRE=(r è un albero corretto, dim>=0, r=vr)
POST=(r è vr a cui sono stati aggiunti dim nodi con campi info letti da "input")
Possiamo anche aggiungere che se r è inizialmente vuoto (come è nel nostro programma) allora per ogni chiamata ricorsiva r ha la proprietà che **per ogni suo nodo** c'è al massimo una differenza di 1 tra il numero di nodi del suo sotto-albero sinistro e di quello destro.

Chiamiamo tali alberi : **quasi bilanciati**

Esempio: se il nostro albero r fosse il seguente:



e k=2 e y=1 allora un cammino con esattamente 2 uni sarebbe il cammino 01. C'è anche un altro cammino con esattamente 2 uni ed è il cammino 101. Invece il cammino 00, nonostante abbia 2 uni non andrebbe bene perché non arriva ad una foglia e portandolo a raggiungere una foglia aggiungeremmo un extra uno che violerebbe la proprietà. Se invece k=1 e y=2 il cammino 001 soddisferebbe la richiesta. E per k=3 e y=2, nessun cammino andrebbe bene.

Per risolvere il problema appena descritto, si chiede di realizzare una funzione:

bool cerca_cam(nodo*r, int k, int y) che soddisfa la seguente pre- e post-condizione:

PRE_cerca=(r albero corretto, k>=0 e y valore qualsiasi)

POST_cerca=(restituisce true sse in r esiste un cammino con esattamente k nodi con campo info=y).

Output da produrre: il programma deve scrivere su "output" la rappresentazione lineare dell'albero prodotto da crea_a, seguito dal risultato della funzione cerca_cam (che sarà o 0 o 1).

Correttezza: fare la prova induttiva di correttezza di cerca_cam.

costruzione albero

```
int conta_n(nodo*r)
{
  if(!r) return 0;
  return conta_n(r->left)+conta_n(r->right)+1;
}
nodo*&  agg(nodo* &r, int x)
{
  if(!r) return r;
   else
{   if(conta_n(r->left)<= conta_n(r->right))
        return agg(r->left,x);
     else
         return agg(r->right,x);
}}
```

PRE=(r=vr albero corretto quasi bilanciato, INP contiene dim valori)

```
void crea_a(nodo*&r, int dim, ifstream & INP)
{
int x;
   if(dim)
    {
      INP>>x;
      agg(r,x)=new nodo(x,0);
      crea_a(r,dim-1,INP);
    }
```

} POST=(r è vr + dim nuovi nodi ed è ancora quasi bilanciato)

```
bool cerca_cam(nodo*r, int k, int y)
{
 if(!r) return false;
 if(foglia(r))
  {
   if((r->info ==y && k==1) || (r->info!=y && k==0))
     {return true;}
   else
    return false;
  }
 else
 {
   if(r->info==y)
    {
     if(k==0)
       return false;
     else
        k--;
     }
    return cerca_cam(r->left,k,y) ||
           cerca_cam(r->right,k,y);

  } }
```

PRE_cerca=(r albero corretto, k>=0 e y valore qualsiasi)

POST_cerca=(restituisce true sse in r esiste un cammino con esattamente k nodi con campo info=y).

CASI BASE

```
if(!r) return false;


 if(foglia(r))
  {
    if((r->info ==y && k==1) || (r->info!=y && k==0))
      {return true;}
    else
     return false;
  }
```

altro caso base:

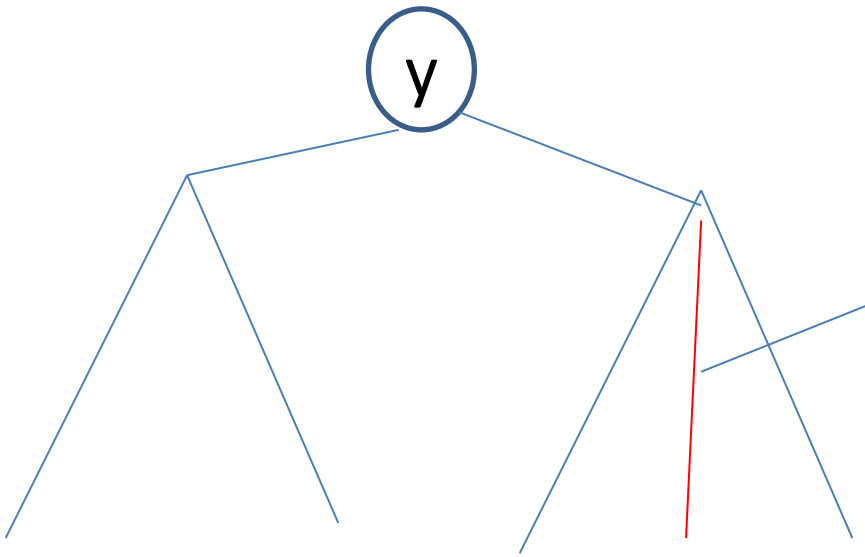
```
if(r->info==y)
      {
       if(k==0)
         return false;
```

passo induttivo:

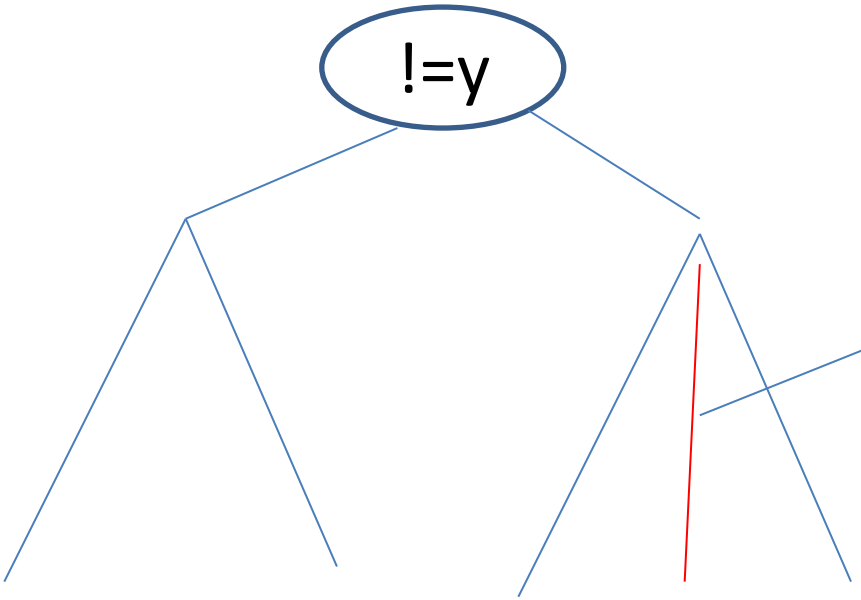
a) r->info==y

k--;

return cerca_cam(r->left,k,y) ||
cerca_cam(r->right,k,y);



sse c'è cammino con k-1 y allora OK



sse c’è cammino con
k y allora OK

esercizio 3: vogliamo anche il cammino (minimo) che contiene k y

PRE_cerca=(r albero corretto, C ha almeno tanti elementi quant'è l'altezza di r, lung>=0, vlung=lung)

bool cerca_cam(nodo*r, int*C, int lung&, int k, int y)

POST_cerca=(restituisce true sse in r esiste un cammino con esattamente k nodi con campo info=y, se restituisce true allora C[vlung..lung-1] contiene il cammino minimo che soddisfa questa proprietà, se restituisce false allora lung=vlung).

```
bool cerca_cam(nodo*r, int*C, int & lung, int k, int y)
{  if(!r) return false;
  if(foglia(r))
   {
     if((r->info ==y && k==1) || (r->info!=y && k==0))
       {return true;}
     else
       return false;
   }
  else
   {
     if(r->info==y)
      {
       if(k==0)
        return false;
       else
        k--;
```

casi base come prima

```
}
    C[lung]=0;
    lung++;
     bool x=cerca_cam(r->left, C, lung, k,y);
     if(x)
        return true;
     else
    {
      C[lung-1]=1;
     bool x=cerca_cam(r->right,C,lung,k,y);
     if(x)
       return true;
     else
      {lung--; return false;}
    }
  }
}
```

passo ricorsivo
costruzione cammino

se ricerca fallisce rimetto
lung a posto