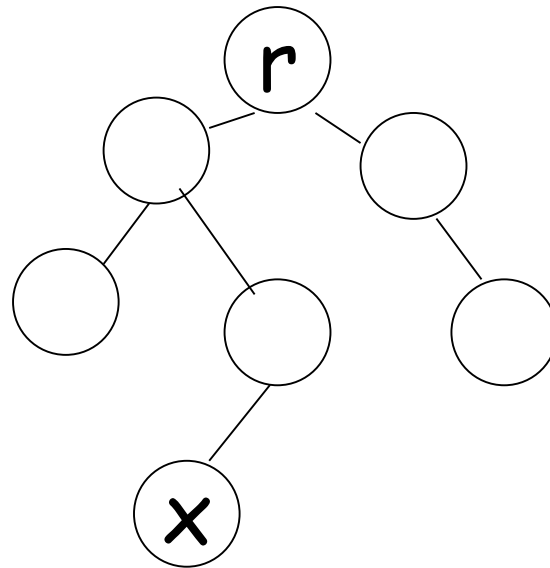


alberi binari 2

un cammino di un albero = sequenza di 0 e 1

0=sinistra 1=destra

array int C[] e il valore lung indica la lunghezza della sequenza:



cammino da r a x:

C=[0,1,0] lung=3

cammino da r a r C=[] e lung =0

Problema

dato un array C che contiene una sequenza di lung 0 e 1 e un albero ben formato, restituire il nodo corrispondente, se c'è

invocazione: nodo *z= trova(root, C, lung);

PRE=(albero(x) ben formato, lung ≥ 0 , C[0,..,lung-1]
def e 0/1)

nodo * trova(nodo *x, int* C, int lung)

{

if(!x) return 0; // fallito

if(lung==0) return x; //trovato

if(*C==0) return trova(x->left, C+1, lung-1);

else

return trova(x->right,C+1, lung-1);

}

POST=(restituisce puntatore al nodo collegato alla
radice dal cammino C[0..lung-1], se c'è, e altrimenti
0)

Esercizio: trovare il nodo di un albero con massimo campo info. In caso di più nodi con campo info massimo, vogliamo restituire il primo rispetto all'ordine infisso:

PRE=(albero(r) è corretto)

nodo* max(nodo*r)

POST=(restituisce il puntatore al primo nodo con info max rispetto all'ordine infisso e 0 se r=0)

```
nodo* max(nodo*r)
{
  if(!r) return 0;
  nodo*a=max(r->left);
  nodo*b=max(r->right);
  if(a&&b)
  {nodo*c=maggiore(a,r); return maggiore(c,b); }
  else
  if(a)
    return maggiore(a,r);
  else
    if(b) return maggiore(r,b);
  else
    return r;
}
```

```
nodo* maggiore(nodo*a, nodo*b)
if(a->info>=b->info)
    return a;
else
    return b;
}
```

si osservi che la funzione in caso di =
restituisce il primo dei 2 nodi

in max viene invocata sempre in modo che il
primo parametro sia sempre minore del
secondo rispetto all'ordine infisso.

max segue lo schema di soluzione I:
riceve in input un albero e ci restituisce il
risultato. Tutto per valore

Funziona seguendo il seguente principio: per
risolvere il problema per l'intero albero, lo
risolviamo sui 2 sottoalberi (left e right) e poi
combiniamo le risposte con la radice r per dare la
risposta per l'intero albero

Si tratta di un modello molto generale per le
funzioni ricorsive

contare i nodi con esattamente un figlio

```
int cncuf(nodo *x)
{ if(x)
    if(!x->left && x->right || x->left && !x->right)
        return 1+ cncuf(x->left)+cncuf(x->right);
    else
        return cncuf(x->left)+cncuf(x->right);
else
    return 0;
}
```

come riconoscere un nodo di profondità k ?

..parto dalla radice con k e lo diminuisco ad ogni livello finchè non diventa 0

..quale cammino seguo?

..è arbitrario purchè si sia in grado di percorrerli tutti

..non appena troviamo un nodo a profondità k ,
interrompiamo la ricorsione e ritorniamo il nodo

PRE=(albero(r) corretto, $k \geq 0$)

nodo * prof_k(nodo * r, int k)

```
{  
    if(!r ) return 0;  
    if(k==0) return r;  
    nodo * p=prof_k(r→left,k-1);  
    if(p) return p;  
    return prof_k(r→right,k-1);  
}
```

POST=(restituisce nodo a prof k se c'è e 0 altrimenti)

Esercizio: trovare nodo minimo a profondità k

PRE=(albero(r) corretto, $k \geq 0$)

nodo* prof(nodo*r, int k)

POST=(restituisce un nodo a prof k con campo info minimo tra i nodi a prof. k e se non ci sono nodi a prof. k, restituisce 0)

```
nodo* prof(nodo*r, int k)
{
    if(r)
    {
        if(k==0)
            return r;
        nodo*a=prof(r->left,k-1);
        nodo*b=prof(r->right,k-1);
        if(a && b)
            if(a->info <= b->info)
                return a;
            else
                return b;
    }
}
```

```
if(a)
    return a;
else
    return b;
}
else //vuoto
    return 0;
}
```

Esercizio: trovare la profondità minima tra le foglie

usiamo:

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

//PRE=(albero(x) ben formato e prof ≥ 0)

int prof_min(nodo*x, int prof)

//POST=(se albero(x) è non vuoto, restituisce k t.c.
k-prof è profondità minima di una foglia in albero(x),
altrimenti restituisce -1)

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


Variante: vogliamo anche il puntatore alla foglia a profondità minima

la funzione restituisce un valore:

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

PRE=(albero(x) ben formato, prof è definito)

```

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== -1 || b.prof== -1)
              if(a.prof== -1) return b;
              else return a;
          else
              if(a.prof>b.prof) return b;
              else
                  return a;}
return foglia(0,-1);
}

```

NOTARE: i
valori foglia
sono
automatici

altra soluzione più efficiente:

inutile cercare a profondità k se abbiamo già trovato una foglia a profondità minore o uguale di k

usiamo il passaggio per riferimento per fare in modo che tutte le funzioni invocate condividano una variabile di tipo foglia che, in ogni momento del calcolo, contenga la migliore foglia incontrata fino a quel momento:

```
void prof_min(nodo*r, int prof, foglia & m)
```