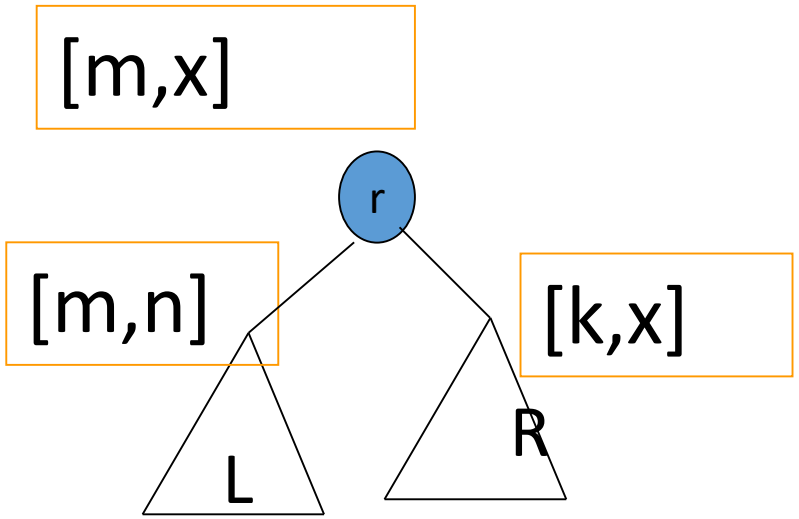
BST 2

esercizi avanzati

ricerca + efficiente

BST aumentato

aggiungere ad ogni nodo n l’intervallo [min, max] dei valori minimo e massimo dei nodi dell’albero radicato in n




```
struct vallo {int min, max; vallo(int a=0, int b=0){min=a; max=b;}};

i nodi dell'albero hanno tipo:

struct nodoA{int info; vallo v; nodoA * left, * right;

nodoA(int a=0, nodoA*b=0,nodoA*c=0){info=a;v.min=v.max=a;
left=b; right=c;}

};
```

calcolo degli intervalli

nei nodi

```
void F(nodo *R)
{
if(R)
 {
   F(R→left); F(R→right);
    R->v=vallo(R->info,R->info);
    if(R→left)
          R->v.min=R→left→v.min;
    if(R→right)
          R->v.max=R→right→v.max;
 }
```

la ricerca è + efficiente grazie agli intervalli

```
nodo * ric(nodo* r, int y)
{
if(!r || y<r->v.min || y>r->v.max)
return 0;
if(y==r->info) return r;
if(y<r->info)
        return ric(r->left,y);
else
        return ric(r->right,y);
}
```

è importante capire che

l’inserimento di un nuovo nodo deve poter modificare gli intervalli dei nodi lungo il cammino di inserimento

gli intervalli nei nodi devono essere corretti anche nel nuovo albero

```
nodoA* insert(nodoA *r, int y){
 if(!r) return new nodoA(y);
 if(r→info >=y)
 {
        if((r->v). min > y)
                (r->v).min=y;
        r→left=insert(r→left, y);
 }
 else
 {
   if((r->v).max<y)
      (r->v).max=y;
    r→right=insert(r→right, y);
  }
return r; }
```

predecessore di un nodo

trovare il predecessore di un nodo con dato campo info

basta percorrere l'albero in modo infisso, restituendo sempre l'ultimo nodo del sottoalbero percorso

PRE=(albero(r) ben formato, vex=ex)

nodo* pred(nodo*r, int y, nodo*& ex)

POST=(se c'è un nodo con info=y restituisce il suo predecessore e se esso è il minimo nodo di albero(r), restituisce vex come predecessore) &&(se non c'è nodo con info=y, restituisce 0 e ex è l'ultimo nodo di albero(r))

```
nodo* pred(nodo*r, int y, nodo*& ex)
{
  if(!r) return 0;
  if(r->info >= y)
            {
               nodo*z=pred(r->left,y,ex);
               if(z) return z;
               if(r->info==y)
                  return ex;
                else
                  return 0;
             }
  else
    {ex=r; return pred(r->right,y, ex);}
}
```

esercizio:

scrivere la funzione successore per trovare il successore di un nodo con info=y

potete seguire la stessa idea usata per precedessore.

Esercizio: cancellazione da un BST di un nodo con info=y (se c'è)

l'operazione deve restituire ancora un BST:

2 casi :

i) il nodo da cancellare ha al più un figlio

ii) il nodo da cancellare ha 2 figli, allora cerchiamo il max del sottoalbero sinistro e lo «sostituiamo» al nodo da cancellare

eliminazione del massimo di un sottoalbero:

PRE=(alb(r) ben formato e non vuoto, valb(r)=alb(r))

```
int elimMax(nodo*&r)
{
  if(r->right) return elimMax(r->right);
     else
   {
     nodo*x=r;
     r=r->left;
     int z=x->info;
     delete x;
     return z;
   }
}
```

POST=(alb(r) è valb(r) meno il nodo max ed è BST, restituisce il suo info)

```
void cercaXcanc(nodo*&r, int y)
{
  if(r){
  if(r->info==y)
  { if(r->left && r->right)
            r->info=elimMax(r->left);
     else
        { nodo*z=r;
         if(!r->left) r=r->right;
            else r=r->left;
         delete z;
        }
  } else
      if(r->info > y) cercaXcanc(r->left,y) ;
         else cercaXcanc(r->right,y);
   }
}
```

cancellazione di un nodo con info=y da un albero