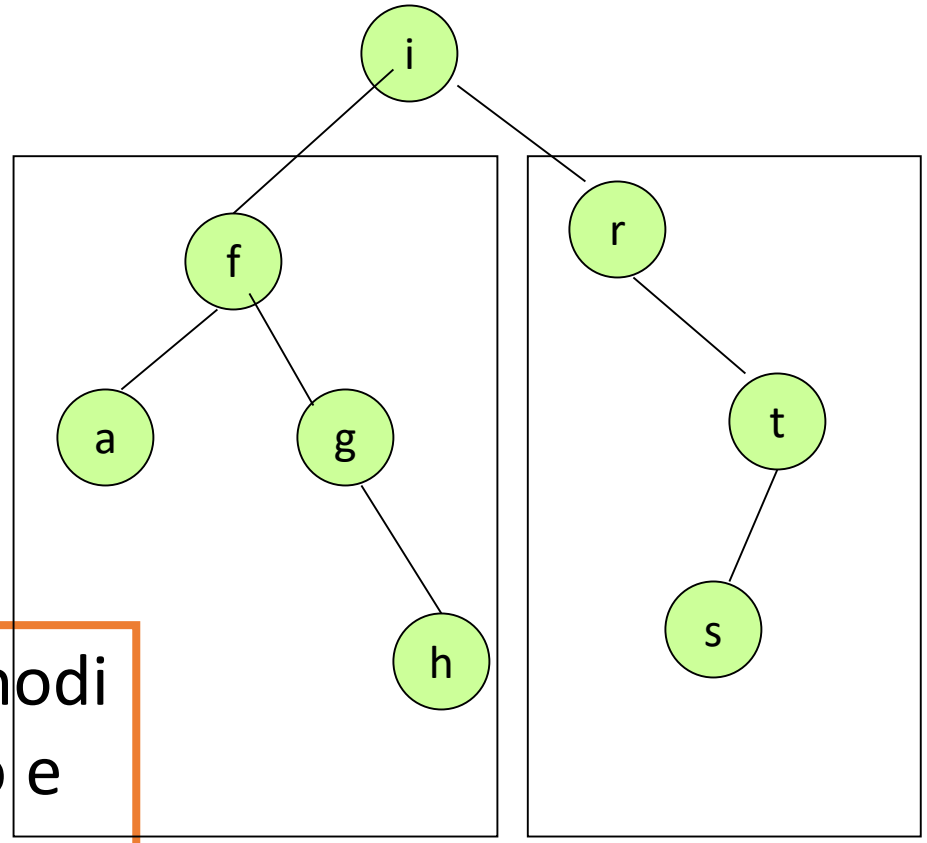
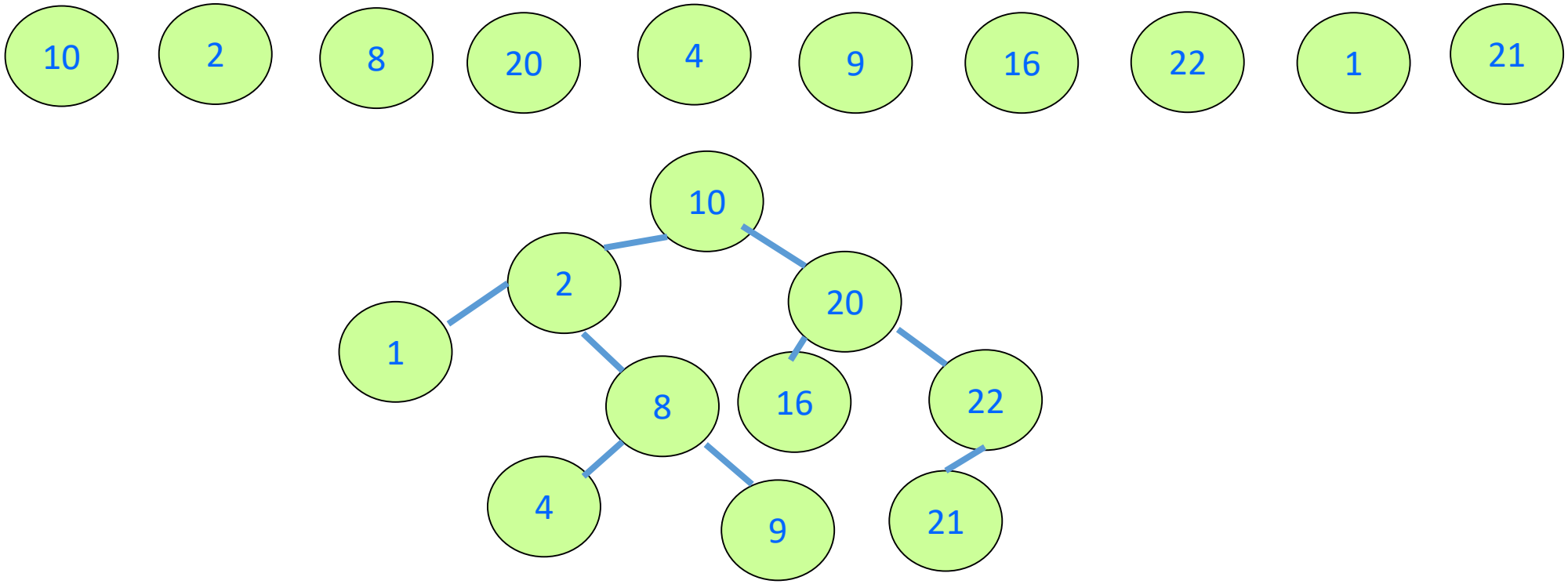
binary search trees
BST

binary search trees (BST)



ogni nodo è maggiore dei nodi nel suo sottoalbero sinistro e minore di quelli del suo sottoalbero destro

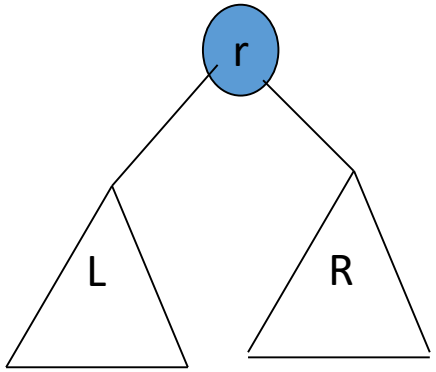
inseriamo questi nodi partendo dall'albero vuoto in modo da soddisfare la proprietà BST ad ogni passo



in un BST è facile (efficiente) cercare un nodo con un certo campo info y

e restituire il puntatore a quel nodo se lo troviamo e 0 altrimenti

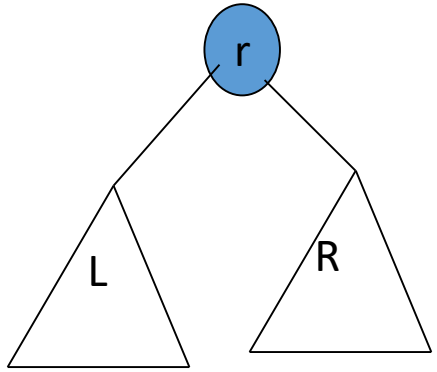
invece per non BST:



controllo r, se no, cerco in L e, se no, cerco in R

insomma se non c'è devo visitare tutti i nodi !!

in un BST la cosa è più semplice ed efficiente:



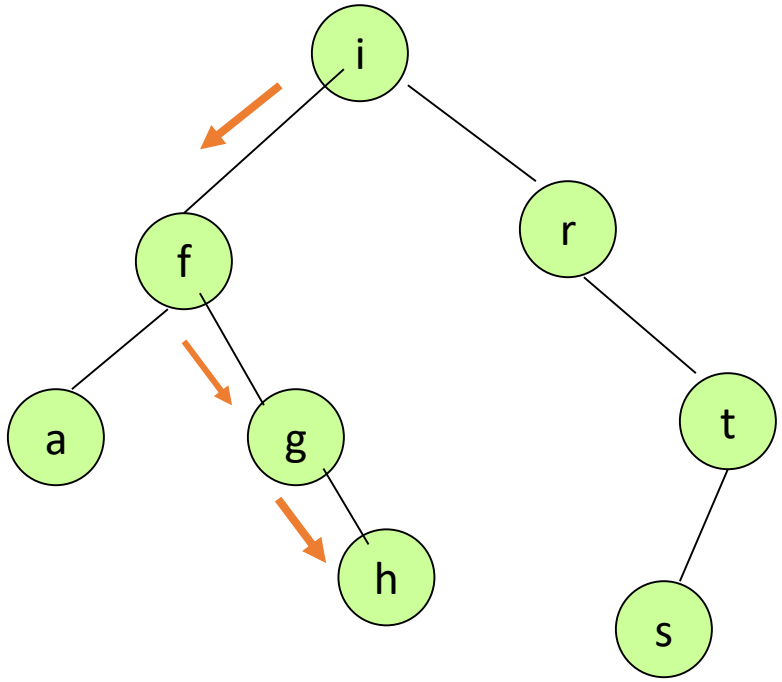
1. r→info == y restituisco r, altrimenti:
2. se r→info > y ➔ cerco solo in L altrimenti cerco solo in R

cerchiamo ‘h’

h<i andiamo a sinistra

h>f destra

h>g destra



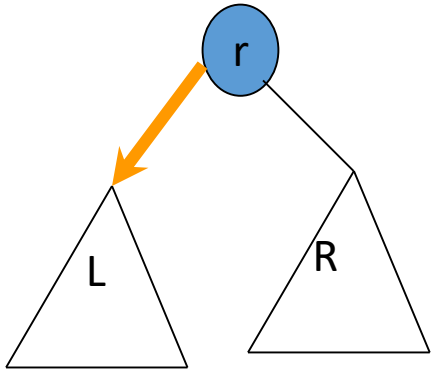
trovato !!!

ricerca in un BST:

```
nodo *search(nodo *x,char y)
{   if(!x) return 0;
 if(x->info==y) return x;
 if(x->info>y)
  return search(x->left,y);
else
   return search(x->right,y);
}
```

ricorsione terminale → iterazione esercizio

seguo un solo cammino : al massimo farò tante invocazioni quant'è l'altezza dell'albero

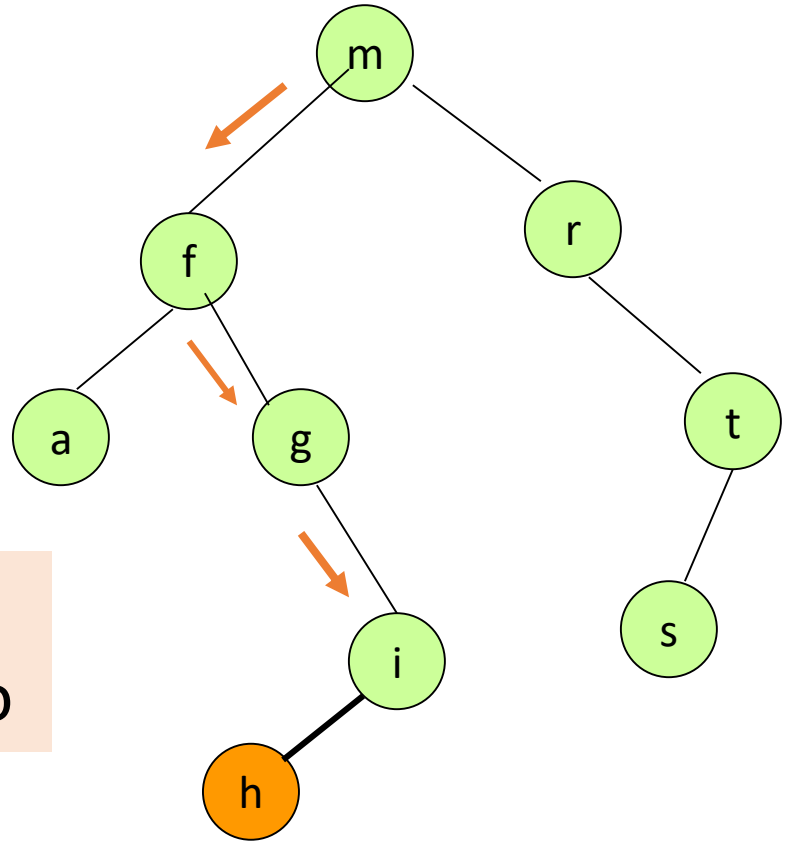


se l'albero è equilibrato, altezza = log n

dove n è il numero dei nodi dell'albero

una bella differenza tra n e log n !!!

se h non ci fosse?



idea per l’inserimento: lo inseriamo dove lo cerchiamo

inserimento in un BST:

```
nodo * insert(nodo *r, int y)
{
        if(!r) return new nodo(y);
        if(r→info >= y)
                r→left=insert(r→left, y);
        else
                r→right=insert(r→right, y);
        return r;
}
```

questa è la soluzione del tipo 1

soluzione di tipo 2

PRE=(albero(r) ben formato e non vuoto)

```
void insert(nodo *r, int y)
{
    if(r->info >=y)
        if(r→left )
            insert(r→left, y);
        else
            r->left=new nodo(y);
    else
        if(r->right)
            insert(r->right,y);
        else
            r->right=new nodo(y);
}
```

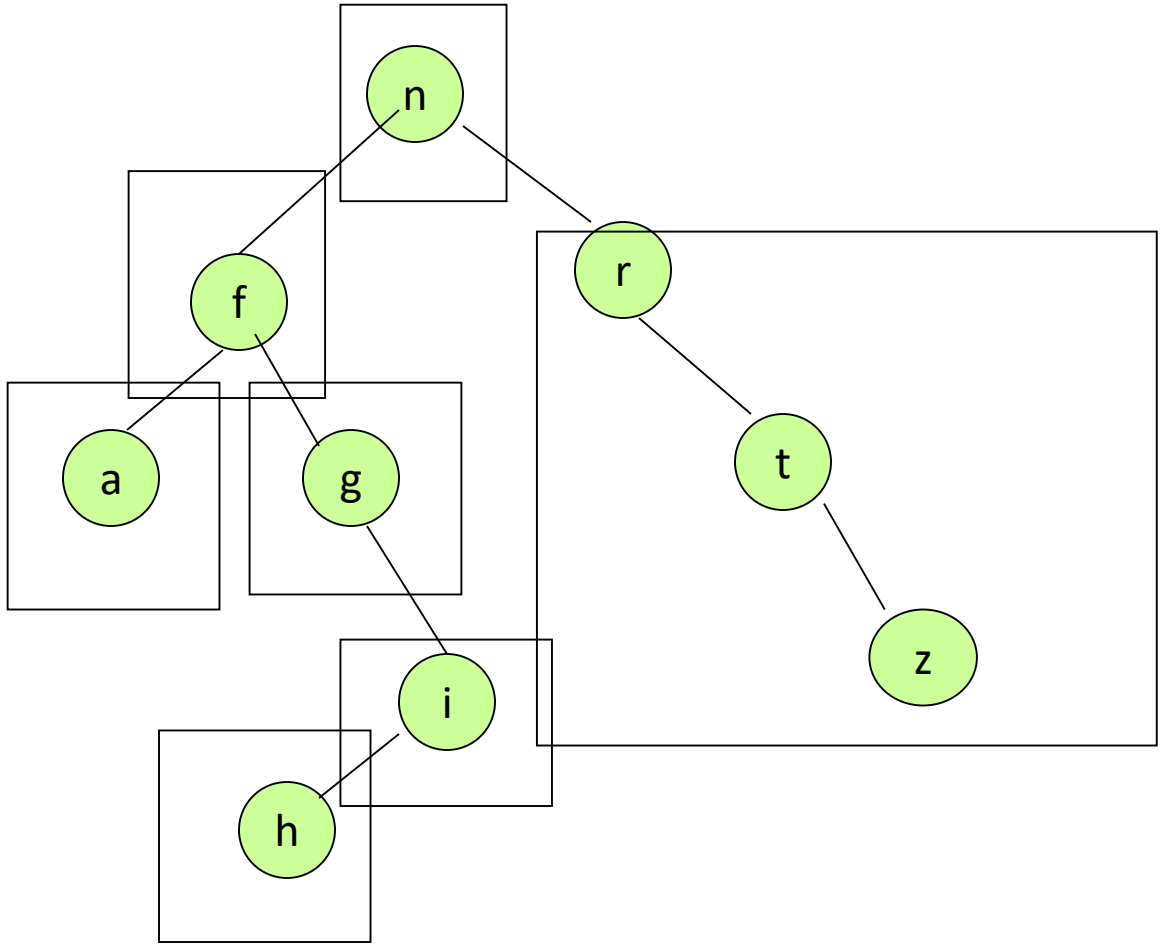
è ricorsiva terminale

iterativa

```
PRE=(albero(r)  ben formato e non vuoto)
nodo * insert(nodo *r, int y)
{
        bool continua=true;
        while (continua)
        {
            if(r→info >= y)
                if(r→left)
                        r=r->left;
                else
                {r→left=new nodo(y);  continua=false;}
            else
                if(r→right)
                        r=r->right;
                else
                        {r→right=new nodo(y); continua=false;}
}
```

esercizio: soluzione ricorsiva di tipo 3

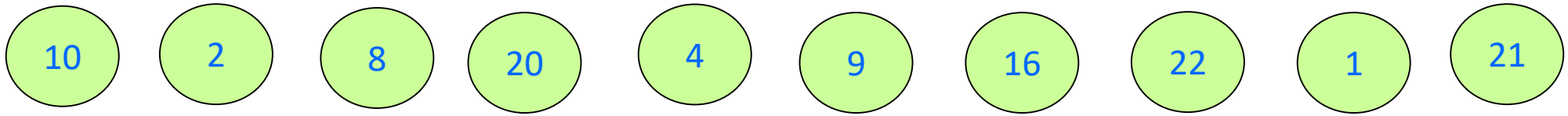
percorso infisso



a f g h i n r t z

sono in ordine !!

inseriamo



partendo dall’albero vuoto: