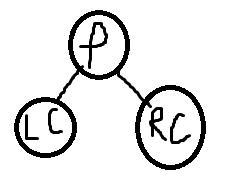
A binary tree (BT) is a binary search tree (BST) if for any family



(value of lc) < (value of p) < (value of rc)

Ex: insert name of days of week in BST

Sun Mon Tue Wed Thr Fri Sat

Insert Sun (Sun)

/\

Mon (mon)

(sun)

/\

Tue (mon) (tue)

\

Wed / (wed)

/

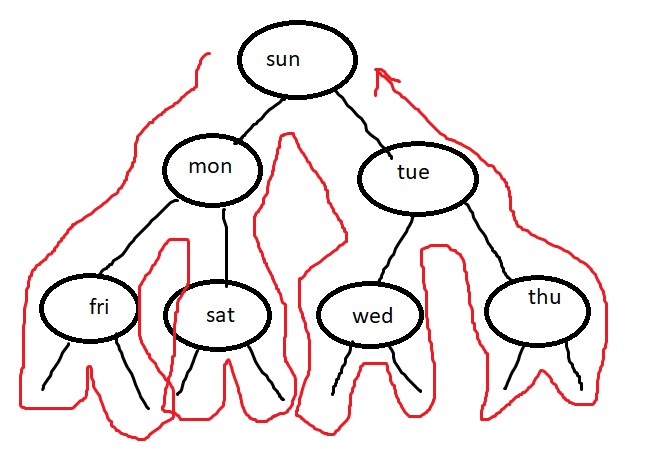
Thu (thu)

//on Monday (mon)

/\

fri (fri)

sat (sat)



Preoder(1) //order seen for first time

Sun mon fri sat tue thu wed

Inorder: (2) //seen the 2nd time, the inorder sorts it

//with binary you eliminate half of it by determining the greater than or less than makes searching faster

This makes runtime = O(log2 ^n) //log base 2

Example you have

log2 ^2000 = log2 ^2\*1000

=log~~2 ^2~~ + log2 ^1000

=1+10

Writing the program for binary search tree

#include <iostream>

#include <string>

using namespace std;

class BTREE

{

private:

struct node

{

string day;

node \*left;

node \*right;

}//end struct node

node \*root; //we want root to point to new node

public:

//set root to null and we use constructor

BTREE() {rppt=NULL;}

//insert x in BTREE

void insert(node\* & r, string x)

{

//create a new node

node \*p = new(node);

p->day=x;

p->left=NULL;

p->right=NULL;

if(r==NULL)

{r=p;}//endif

else

{

if(x < r->day) insert(r->left, x)

if(x > r->day) insert(r->right, x);

}//endelse

}//end insert

//display the tree using inorder traversal

void displayInOrder(node \*&r)

{

//the nodes you see for the second time

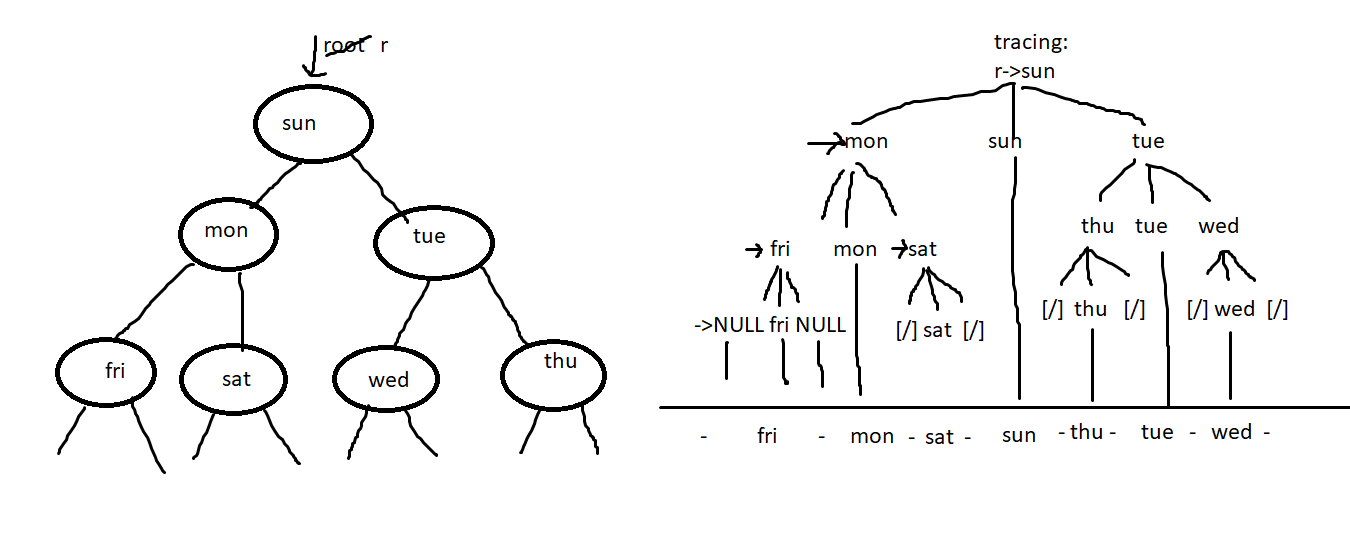
//inorder (LC)(P)(RC)

if(r != NULL)

{

displayInOrder(r->left); //LC

cout<<r->day<<” “; //P

 displayInOrder(r->right); //RC

}//endif

}//end displayInOrder

void insert(string x)

{

insert(root, x); //pass the root and the x

}//end insert overiddenfx

void displayTree()

{

displayInOrder(root); //dummy function to give access to public //to access the private

}//end displayTree

}//end class btree

int main()

{

//we need a pointer to point ot the tree

BTREE t; //object

//we want to insert the name of the day

string days[7]={ “Sun” ,“Mon”, “Tue”, “Wed”, “Thu”, “Fri”, “Sat” };

for(int i=0; i<7; ++i)

{

//we don’t have access to root so we create a temp dummy member to call the insert

t.insert(days[i];//this is not a call to the top insert

}//endfor

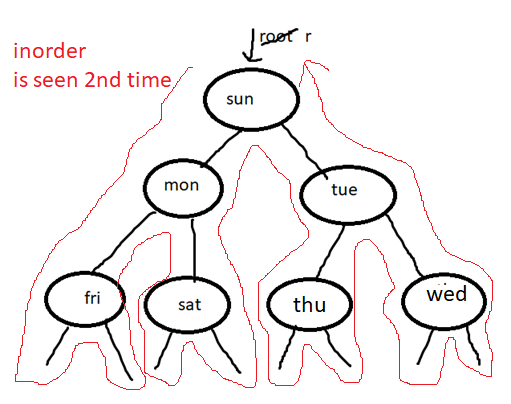
//display the tree, we can’t call it bec we need root and root is in private section

t.displayTree();

system(“pause”);

return 0;

}//end main



//in preorder u see parent first time

//Inorder you see parent 2nd time

//Postorder you see last

void displayInOrder(node\* r)

{

if(r!=NULL)

{

//LC P RC

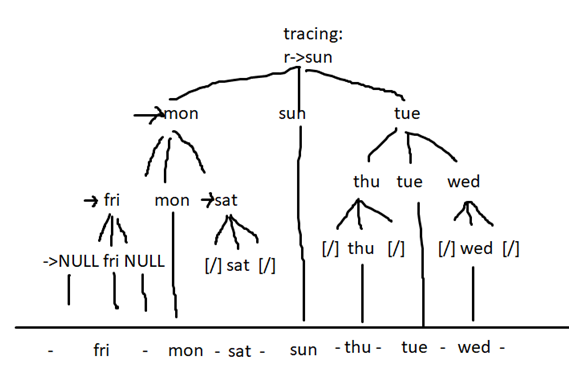
displayInOrder(r->left);

cout<<r->info<<” “;

displayInOrder(r->right);

}

}

//above function is this 

//displays in order in alphabetical order

PREORDER TRAVERSAL

void displayPreOrder(node\* r)

{

if(r!=NULL)

{

// P LC RC

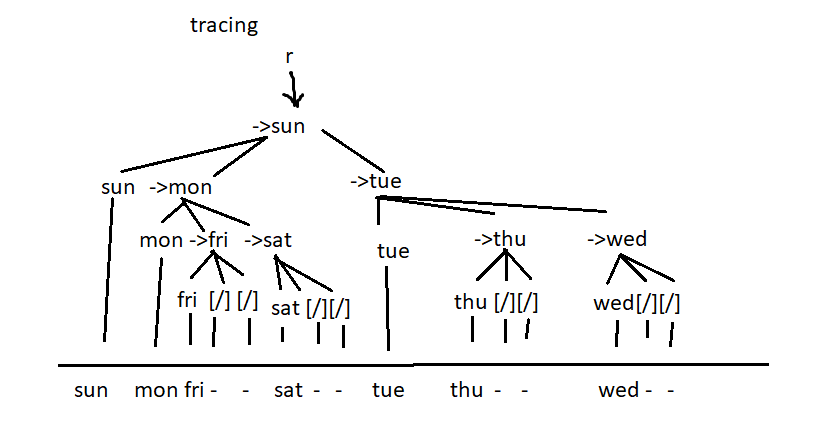
cout<<r->info<<” “;

displayInOrder(r->left);

displayInOrder(r->right);

}

}



binay searching makes search faster because it halves searching process by 2

void search(node\* r, string x)

{

if(r==NULL) //if we look for something we don’t find it points to NULL..dont exist

{cout<<x<<” does not exist\n”; return; //returns this statement and exits the function}

else if (x==r->info)

{cout<<x<<” is found\n”; return;}

else

{

if(x<r->info) //call function again and start pointer

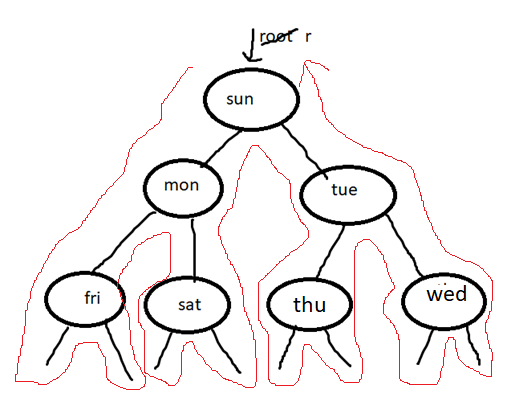
{search(r->left, x);}

if(x>r->info)

{search(r->right, x);} //recursive functions to search whether in tree or not

}

}



//sample Saturday

is r == to null? no

is satu.. trace it

and try searching for wd

to find out the algo

of how it doesn’t exist

//--------------------------------------------------------

//practice

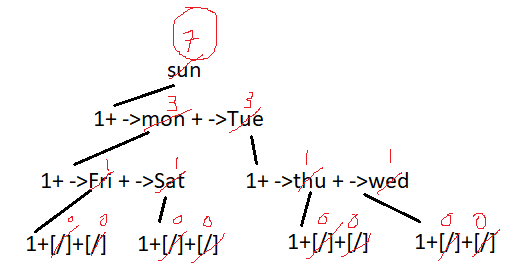
int f(node\* r)

{

if(r==NULL) return 0;

else return 1+f(r->left)+f(r->right);

}



//practice---------------------------------------------------------

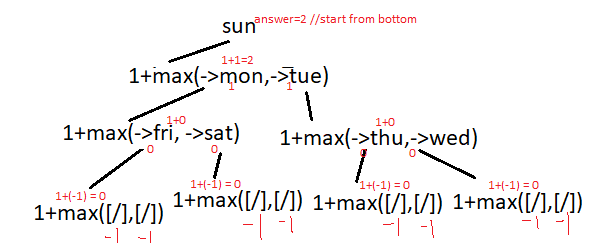
int f(node\* r)

{

if(r==NULL) return -1;

else return 1+max(f(r->left), f(r->right));

}



//ABOVE IS .. this is how you find how tall the tree is

int treeHeight(node\* r)

{

if(r==NULL) return -1;

else return 1+max(treeHeight (r->left), treeHeight (r->right));

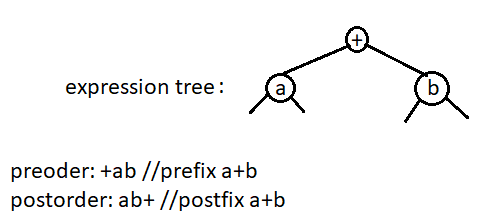
}

APPLICATIONS OF PREODER AND POSTORDER---------------------------------------------

Given expression a+b

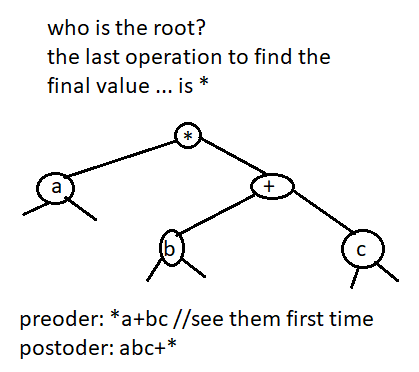
//to find the value of this expression we want to add a to b

the operation becomes the root and a is left child b is right child



ex: convert a\*(b+c) to prefix and postfix

//first construct expression tree



//-------------------------------------

convert a\*(b-c\*d)

