Lab a

Q1

#include <iostream>

#include <vector>

int cc = 0;

using namespace std;

template <typename t>

class queue

{

public:

t data;

queue<t> \*next;

queue(t d)

{

this->data = d;

this->next = NULL;

}

void push(queue<t> \*&q, t d)

{

queue<t> \*n = new queue<t>(d);

if (q == NULL)

{

q = n;

return;

}

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

void pop(queue<t> \*&q)

{

if (q != NULL)

{

queue<t> \*to = q;

q = q->next;

delete to;

}

}

t top(queue<t> \*&q)

{

if (q != NULL)

return q->data;

return 0;

}

bool isempty(queue<t> \*&q)

{

return q == NULL ? true : false;

}

};

class node

{

node \*insert\_into\_bst(node \*root, int d)

{

if (root == NULL)

return new node(d);

if (d < root->data)

root->left = insert\_into\_bst(root->left, d);

else if (d > root->data)

root->right = insert\_into\_bst(root->right, d);

else

{

cout << "\nduplicate data\n";

return NULL;

}

return root;

}

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

void insert(node \*&root)

{

// int d;

// cout << "enter the data (-1 to end) : ";

// cin >> d;

int c = 2000;

while (c--)

{

int r=rand()/10;

cout<<r<<" ";

root = insert\_into\_bst(root, r);

// cout << "enter the data (-1 to end) : ";

// cin >> d;

}

}

void inorder(node \*root, vector<int> &a)

{

if (root == NULL)

return;

inorder(root->left, a);

a.push\_back(root->data);

inorder(root->right, a);

}

void search(node \*root, int key)

{

if (root == NULL)

return;

if (root->data == key)

return;

search(root->left, key);

search(root->right, key);

}

void height\_cal(node \*root, int h, int &ans)

{

if (root == NULL)

{

ans = max(ans, h);

return;

}

height\_cal(root->left, h + 1, ans);

height\_cal(root->right, h + 1, ans);

}

int height(node \*root)

{

int ans = 0;

height\_cal(root, 0, ans);

return ans;

}

};

int main()

{

srand(1002);

int k = 100;

while(k--)

{

node \* root = NULL;

root->insert(root);

cout << root->height(root)<<"\*\*\n ";

cc += root->height(root);

}

cout << cc;

}

// insertion

// c=1000 avg 3846

// c=2000 avg 7727

// c=3000 avg 11577

// c=4000 avg 15500

// searching

// c=1000 15

// c=2000 16

// c=3000 18

// c=4000 19

// height

// c=1000 9.69

// c=2000 16.68

// c=3000 17.04

// c=4000 16.70

Lab b

Q1

#include <iostream>

#include<vector>

using namespace std;

template <typename t>

class queue

{

public:

t data;

queue<t> \*next;

queue(t d)

{

this->data = d;

this->next = NULL;

}

void push(queue<t> \*&q, t d)

{

queue<t> \*n = new queue<t>(d);

if (q == NULL)

{

q = n;

return;

}

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

void pop(queue<t> \*&q)

{

if (q != NULL)

{

queue<t> \*to = q;

q = q->next;

delete to;

}

}

t top(queue<t> \*&q)

{

if (q != NULL)

return q->data;

return 0;

}

bool isempty(queue<t> \*&q)

{

return q == NULL ? true : false;

}

};

class node

{

node \*insert\_into\_bst(node \*root, int d)

{

if (root == NULL)

return new node(d);

if (d < root->data)

root->left = insert\_into\_bst(root->left, d);

else if (d > root->data)

root->right = insert\_into\_bst(root->right, d);

else

{

cout << "\nduplicate data\n";

return NULL;

}

return root;

}

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

void insert(node \*&root)

{

int d;

cout << "enter the data (-1 to end) : ";

cin >> d;

while (d != -1)

{

root = insert\_into\_bst(root, d);

cout << "enter the data (-1 to end) : ";

cin >> d;

}

}

void inorder(node \* root,vector<int> & a)

{

if(root==NULL) return;

inorder(root->left,a);

a.push\_back(root->data);

inorder(root->right,a);

}

};

int main()

{

// queue<int> \* q=NULL;

// for(int i=0;i<10;i++)

// {

// q->push(q,i+1);

// }

// while(!q->isempty(q))

// {

// cout<<q->top(q)<<" ";

// q->pop(q);

// }

node \* root=NULL;

root->insert(root);

vector<int> a;

root->inorder(root,a);

int k;

cout<<"enter the value of k : ";

cin>>k;

if(k>a.size())

{

cout<<"k elemnts are not present : ";

}

else{

cout<<"kth largest element is "<<a[a.size()-k];

}

}

Q2

#include<iostream>

using namespace std;

template<typename t>

class queue

{

public:

t data;

queue<t> \* next;

queue(t d)

{

this->data=d;

this->next=NULL;

}

void push(queue<t> \* & r, t d)

{

queue<t> \* n=new queue<t> (d);

if(r==NULL)

{

r=n;

return ;

}

queue<t> \* temp=r;

while(temp->next!=NULL) temp=temp->next;

temp->next=n;

return;

}

void pop(queue<t> \* &r)

{

if(r!=NULL)

{

queue <t> \* to=r;

r=r->next;

delete to;

}

}

t top(queue<t> \* & r)

{

if(r!=NULL) return r->data;

return 0;

}

bool isempty(queue<t> \* &r)

{

return r==NULL ?true: false;

}

};

class node

{

public:

node \*left;

node \* right;

int data;

node(int d)

{

this->data=d;

this->left=NULL;

this->right=NULL;

}

node \* insert()

{

int d;

cout<<"enter the data :(-1 for NULL) ";

cin>>d;

if(d==-1) return NULL;

node \* root=new node (d);

cout<<"for the left of "<<d<<" ";

root->left=insert();

cout<<"for the right of "<<d<<" ";

root->right=insert();

return root;

}

void level\_order(node \* root)

{

queue<node \*> \* q=NULL;

q->push(q,root);

q->push(q,NULL);

while(!q->isempty(q))

{

node \* f=q->top(q);

q->pop(q);

if(f==NULL)

{

if(!q->isempty(q))

{

q->push(q,NULL);

}

cout<<endl;

}

else

{

cout<<f->data<<" ";

if(f->left)

{

q->push(q,f->left);

}

if(f->right)

{

q->push(q,f->right);

}

}

}

}

void mirror(node\* & root)

{

if(root==NULL) return;

node \* temp=root->left;

root->left=root->right;

root->right=temp;

mirror(root->left);

mirror(root->right);

}

};

int main()

{

node \* root=NULL;

root=root->insert();

cout<<"\n level order : \n";

root->level\_order(root);

cout<<"\nlevel order of mirror tree : \n";

root->mirror(root);

root->level\_order(root);

}

// 1 3 -1 -1 2 5 -1 -1 4 -1 -1

Q3

#include<iostream>

using namespace std;

#include<vector>

template<typename t>

class queue

{

public:

t data;

queue<t> \* next;

queue(t d)

{

this->data=d;

this->next=NULL;

}

void push(queue<t> \* & r, t d)

{

queue<t> \* n=new queue<t> (d);

if(r==NULL)

{

r=n;

return ;

}

queue<t> \* temp=r;

while(temp->next!=NULL) temp=temp->next;

temp->next=n;

return;

}

void pop(queue<t> \* &r)

{

if(r!=NULL)

{

queue <t> \* to=r;

r=r->next;

delete to;

}

}

t top(queue<t> \* & r)

{

if(r!=NULL) return r->data;

return 0;

}

bool isempty(queue<t> \* &r)

{

return r==NULL ?true: false;

}

};

class node

{

public:

node \*left;

node \* right;

int data;

node(int d)

{

this->data=d;

this->left=NULL;

this->right=NULL;

}

node \*insert\_into\_bst(node \*root, int d)

{

if (root == NULL)

return new node(d);

if (d < root->data)

root->left = insert\_into\_bst(root->left, d);

else if (d > root->data)

root->right = insert\_into\_bst(root->right, d);

else

{

cout << "\nduplicate data\n";

return NULL;

}

return root;

}

void insert(node \* &root)

{

int d;

cout << "enter the data (-1 to end) : ";

cin >> d;

while (d != -1)

{

root = insert\_into\_bst(root, d);

cout << "enter the data (-1 to end) : ";

cin >> d;

}

}

void level\_order(node \* root)

{

queue<node \*> \* q=NULL;

q->push(q,root);

q->push(q,NULL);

while(!q->isempty(q))

{

node \* f=q->top(q);

q->pop(q);

if(f==NULL)

{

if(!q->isempty(q))

{

q->push(q,NULL);

}

cout<<endl;

}

else

{

cout<<f->data<<" ";

if(f->left)

{

q->push(q,f->left);

}

if(f->right)

{

q->push(q,f->right);

}

}

}

}

void inorder(node \* root ,vector<int> & a)

{

if(root==NULL)

{

return;

}

inorder(root->left,a);

a.push\_back(root->data);

inorder(root->right,a);

}

};

int main()

{

node \* root1=NULL;

cout<<"insert in bst 1 : \n";

root1->insert(root1);

cout<<"\n level order bst 1 : \n";

root1->level\_order(root1);

node \* root2=NULL;

cout<<"insert in bst 2 : \n";

root2->insert(root2);

cout<<"\n level order bst 2 : \n";

root2->level\_order(root2);

vector<int> a1;

vector<int> a2;

root1->inorder(root1,a1);

root2->inorder(root2,a2);

vector<int> ans(a1.size()+a2.size());

int m=max(a1[a1.size()-1],a2[a2.size()-1])+1;

a1.push\_back(m);

a2.push\_back(m);

int k=0;

int i=0;

int j=0;

while(k<a1.size()+a2.size()-2)

{

if(a1[i]<a2[j]) ans[k++]=a1[i++];

else ans[k++]=a2[j++];

cout<<ans[k-1]<<" ";

}

cout<<"\nmerge sorted array : ";

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

{

cout<<ans[i]<<" ";

}

}

// 5 3 6 2 4 -1

// 2 1 3 7 6 -1