/\*

Lab Assignment 1

Consider telephone book database of N clients.

Make use of a hash table implementation to quickly look up client‘s telephone number.

Make use of two collision handling techniques and compare them

using number of comparisons required to find a set of telephone numbers

\*/

#include <iostream>

using namespace std;

// Store details : Node-> Key Name Telephone

class node{

private:

string name;

string telephone;

int key;

public:

node(){

key=0;

}

friend class hashing; // To access the private members of class node

};

// Hashng Fuction that generates different key value

// Sum of ascii value of each character in string

int ascii\_generator(string s){

int sum=0;

for (int i = 0; s[i] != '\0'; i++)

sum = sum + s[i];

return sum%100;

}

// Class -> Hashing

class hashing{

private:

node data[100]; // Size of directory -> 100

string n;

string tele;

int k, index;

int size=100;

public:

hashing(){

k=0;

}

// Function to create record

void create\_record(string n,string tele){

k=ascii\_generator(n); //using ascii value of string as key

index=k%size;

for (int j=0;j<size;j++){

if(data[index].key==0){

data[index].key=k;

data[index].name=n;

data[index].telephone=tele;

break;

}

else

index=(index+1)%size;

}

}

// Function to search for record based on name input

void search\_record(string name){

int index1,k,flag=0;

k=ascii\_generator(name);

index1=k%size;

for(int a=0;a<size;a++){

if(data[index1].key==k){

flag=1;

cout<<"\nRecord found\n";

cout<<"Name :: "<<data[index1].name<<endl;

cout<<"Telephone :: "<<data[index1].telephone<<endl;

break;

}

else

index1=(index1+1)%size;

}

if(flag==0)

cout<<"Record not found";

}

// Function to delete existing record

void delete\_record(string name){

int index1,key,flag=0;

key=ascii\_generator(name);

index1=key%size;

for(int a=0;a<size;a++){

if(data[index].key==key){

flag=1;

data[index1].key=0;

data[index1].name=" ";

data[index1].telephone=" ";

cout<<"\nRecord Deleted successfully"<<endl;

break;

}

else

index1=(index1+1)%size;

}

if(flag==0)

cout<<"\nRecord not found";

}

// Function to update existing record

void update\_record(string name){

int index1,key,flag=0;

key=ascii\_generator(name);

index1=key%size;

for(int a=0;a<size;a++){

if(data[index1].key==key){

flag=1;

break;

}

else

index1=(index1+1)%size;

}

if(flag==1){

cout<<"Enter the new telephone number :: ";

cin>>tele;

data[index1].telephone=tele;

cout<<"\nRecord Updated successfully";

}

}

// Function to display the directory

void display\_record(){

cout<<"\t Name \t\t Telephone";

for (int a = 0; a < size; a++) {

if(data[a].key!=0){

cout<<"\n\t"<<data[a].name<<" \t\t\t "<<data[a].telephone;

}

}

}

};

// Main Function

int main(){

hashing s;

string name;

string telephone;

int choice,x;

bool loop=1;

// Menu driven code

while(loop){

cout<<"\n-------------------------"<<endl

<<" Telephone book Database "<<endl

<<"-------------------------"<<endl

<<"1. Create Record"<<endl

<<"2. Display Record"<<endl

<<"3. Search Record"<<endl

<<"4. Update Record"<<endl

<<"5. Delete Record"<<endl

<<"6. Exit"<<endl

<<"Enter choice :: ";

cin>>choice;

switch (choice)

{

case 1:

cout<<"\nEnter name :: ";

cin>>name;

cout<<"Enter Telephone number :: ";

cin>>telephone;

s.create\_record(name,telephone);

break;

case 2:

s.display\_record();

break;

case 3:

cout<<"\nEnter the name :: ";

cin>>name;

s.search\_record(name);

break;

case 4:

cout<<"\nEnter the name :: ";

cin>>name;

s.update\_record(name);

break;

case 5:

cout<<"\nEnter name to Delete :: ";

cin>>name;

s.delete\_record(name);

break;

case 6:

loop=0;

break;

default:

cout<<"\nYou Entered something wrong!";

break;

}

}

return 0;

}

// ======================================================================== 2

// Name : ThreadedBinaryTree.cpp

// Author : Yash Sonar

// Title : Convert given binary tree into inordered and preordered threaded binary tree.

// Analyze time and space complexity of the algorithm.

//======

#include <iostream>

using namespace std;

class TBT;

class node

{

node \*left,\*right;

int data;

bool rbit,lbit;

public:

node()

{

left=NULL;

right=NULL;

rbit=lbit=0;

}

node(int d)

{

left=NULL;

right=NULL;

rbit=lbit=0;

data=d;

}

friend class TBT;

};

class TBT

{

node \*root; //acts as a dummy node

public:

TBT() //dummy node initialization

{

root=new node(9999);

root->left=root;

root->rbit=1;

root->lbit=0;

root->right=root;

}

void create();

void insert(int data);

node \*inorder\_suc(node \*);

void inorder\_traversal();

node \* preorder\_suc(node \*c);

void preorder\_traversal();

};

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

void TBT::preorder\_traversal()

{

node \*c=root->left;

while(c!=root)

{

cout<<" "<<c->data;

c=preorder\_suc(c);

}

}

void TBT::inorder\_traversal()

{

node \*c=root->left;

while(c->lbit==1)

c=c->left;

while(c!=root)

{

cout<<" "<<c->data;

c=inorder\_suc(c);

}

}

node\* TBT::inorder\_suc(node \*c)

{

if(c->rbit==0)

return c->right;

else

c=c->right;

while(c->lbit==1)

{

c=c->left;

}

return c;

}

node \*TBT::preorder\_suc(node \*c)

{

if(c->lbit==1)

{

return c->left;

}

while(c->rbit==0)

{

c=c->right;

}

return c->right;

}

//-------- Create Method

void TBT::create()

{

int n;

if(root->left==root&&root->right==root)

{

cout<<"\nEnter number of nodes:";

cin>>n;

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

{

int info;

cout<<"\nEnter data: ";

cin>>info;

this->insert(info);

}

}

else

{

cout<<"\nTree is Already created.\n";

}

}

void TBT::insert(int data)

{

if(root->left==root&&root->right==root) //no node in tree

{

node \*p=new node(data);

p->left=root->left;

p->lbit=root->lbit; //0

p->rbit=0;

p->right=root->right;

root->left=p;

root->lbit=1;

cout<<"\nInserted start"<<data;

return;

}

node \*cur=new node;

cur=root->left;

while(1)

{

if(cur->data<data) //insert right

{

node \*p=new node(data);

if(cur->rbit==0)

{

p->right=cur->right;

p->rbit=cur->rbit;

p->lbit=0;

p->left=cur;

cur->rbit=1;

cur->right=p;

cout<<"\nInserted right "<<data;

return;

}

else

cur=cur->right;

}

if(cur->data>data) //insert left

{

node \*p=new node(data);

if(cur->lbit==0)

{

p->left=cur->left;

p->lbit=cur->lbit;

p->rbit=0;

p->right=cur; //successor

cur->lbit=1;

cur->left=p;

cout<<"\nInserted left"<<data;

return;

}

else

cur=cur->left;

}

}

}

int main() {

TBT t1;

int value;

int choice;

do

{

cout<<"\n1.Create Tree\n2.Insert into tree\n3.Preorder\n4.Inorder\n0.Exit\nEnter your choice: ";

cin>>choice;

switch(choice)

{

case 1:

t1.create();

break;

case 2:

cout<<"\nEnter Number(data): ";

cin>>value;

t1.insert(value);

break;

case 3:

cout<<"\nPreorder traversal of TBT\n";

t1.preorder\_traversal();

break;

case 4:

cout<<"\nInoder Traversal of TBT\n";

t1.inorder\_traversal();

break;

default:

cout<<"\nWrong choice";

}

}while(choice!=0);

return 0;

}

/\*

Experiment 3 : Represent a given grapg using adjacency matrix/list to perform DFS and using adjacency list to perform BFS.

Use the map of the area around the college as a graph. Identify the prominent land marks as nodes and perform DFS and BFS on that.

Adjacency Matrix : using adj matrix -BFS(Que)

\*/

#include <iostream>

#include <stdlib.h>

using namespace std;

int cost[10][10], i, j, k, n, qu[10], front, rear, v, visit[10], visited[10];

int stk[10], top, visit1[10], visited1[10];

int main()

{

int m;

cout << "Enter number of vertices : ";

cin >> n;

cout << "Enter number of edges : ";

cin >> m;

cout << "\nEDGES :\n";

for (k = 1; k <= m; k++)

{

cin >> i >> j;

cost[i][j] = 1;

cost[j][i] = 1;

}

//display function

cout << "The adjacency matrix of the graph is : " << endl;

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

cout << " " << cost[i][j];

}

cout << endl;

}

cout << "Enter initial vertex : ";

cin >> v;

cout << "The BFS of the Graph is\n";

cout << v<<endl;

visited[v] = 1;

k = 1;

while (k < n)

{

for (j = 1; j <= n; j++)

if (cost[v][j] != 0 && visited[j] != 1 && visit[j] != 1)

{

visit[j] = 1;

qu[rear++] = j;

}

v = qu[front++];

cout << v << " ";

k++;

visit[v] = 0;

visited[v] = 1;

}

cout <<endl<<"Enter initial vertex : ";

cin >> v;

cout << "The DFS of the Graph is\n";

cout << v<<endl;

visited[v] = 1;

k = 1;

while (k < n)

{

for (j = n; j >= 1; j--)

if (cost[v][j] != 0 && visited1[j] != 1 && visit1[j] != 1)

{

visit1[j] = 1;

stk[top] = j;

top++;

}

v = stk[--top];

cout << v << " ";

k++;

visit1[v] = 0;

visited1[v] = 1;

}

return 0;

}

/\* LAB 4

Author: Parag Ghorpade

PROBLEM STATEMENT:

Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry.

Also,display whole data sorted in ascending or descending order.

Find how many maximum comparisons may require for finding any keyword. Use height balance tree and find the complexity for finding a keyword.

\*/

#include<iostream>

#include<string.h>

using namespace std;

class dict

{

dict \*root,\*node,\*left,\*right,\*tree1;

string s1,s2;

int flag,flag1,flag2,flag3,cmp;

public:

dict()

{

flag=0,flag1=0,flag2=0,flag3=0,cmp=0;

root=NULL;

}

void input();

void create\_root(dict\*,dict\*);

void check\_same(dict\*,dict\*);

void input\_display();

void display(dict\*);

void input\_remove();

dict\* remove(dict\*,string);

dict\* findmin(dict\*);

void input\_find();

dict\* find(dict\*,string);

void input\_update();

dict\* update(dict\*,string);

};

void dict::input()

{

node=new dict;

cout<<"\nEnter the keyword:\n";

cin>>node->s1;

cout<<"Enter the meaning of the keyword:\n";

cin.ignore();

getline(cin,node->s2);

create\_root(root,node);

}

void dict::create\_root(dict \*tree,dict \*node1)

{

int i=0,result;

char a[20],b[20];

if(root==NULL)

{

root=new dict;

root=node1;

root->left=NULL;

root->right=NULL;

cout<<"\nRoot node created successfully"<<endl;

return;

}

for(i=0;node1->s1[i]!='\0';i++)

{

a[i]=node1->s1[i];

}

for(i=0;tree->s1[i]!='\0';i++)

{

b[i]=tree->s1[i];

}

result=strcmp(b,a);

check\_same(tree,node1);

if(flag==1)

{

cout<<"The word you entered already exists.\n";

flag=0;

}

else

{

if(result>0)

{

if(tree->left!=NULL)

{

create\_root(tree->left,node1);

}

else

{

tree->left=node1;

(tree->left)->left=NULL;

(tree->left)->right=NULL;

cout<<"Node added to left of "<<tree->s1<<"\n";

return;

}

}

else if(result<0)

{

if(tree->right!=NULL)

{

create\_root(tree->right,node1);

}

else

{

tree->right=node1;

(tree->right)->left=NULL;

(tree->right)->right=NULL;

cout<<"Node added to right of "<<tree->s1<<"\n";

return;

}

}

}

}

void dict::check\_same(dict \*tree,dict \*node1)

{

if(tree->s1==node1->s1)

{

flag=1;

return;

}

else if(tree->s1>node1->s1)

{

if(tree->left!=NULL)

{

check\_same(tree->left,node1);

}

}

else if(tree->s1<node1->s1)

{

if(tree->right!=NULL)

{

check\_same(tree->right,node1);

}

}

}

void dict::input\_display()

{

if(root!=NULL)

{

cout<<"The words entered in the dictionary are:\n\n";

display(root);

}

else

{

cout<<"\nThere are no words in the dictionary.\n";

}

}

void dict::display(dict \*tree)

{

if(tree->left==NULL&&tree->right==NULL)

{

cout<<tree->s1<<" = "<<tree->s2<<"\n\n";

}

else

{

if(tree->left!=NULL)

{

display(tree->left);

}

cout<<tree->s1<<" = "<<tree->s2<<"\n\n";

if(tree->right!=NULL)

{

display(tree->right);

}

}

}

void dict::input\_remove()

{

char t;

if(root!=NULL)

{

cout<<"\nEnter a keyword to be deleted:\n";

cin>>s1;

remove(root,s1);

if(flag1==0)

{

cout<<"\nThe word '"<<s1<<"' has been deleted.\n";

}

flag1=0;

}

else

{

cout<<"\nThere are no words in the dictionary.\n";

}

}

dict\* dict::remove(dict \*tree,string s3)

{

dict \*temp;

if(tree==NULL)

{

cout<<"\nWord not found.\n";

flag1=1;

return tree;

}

else if(tree->s1>s3)

{

tree->left=remove(tree->left,s3);

return tree;

}

else if(tree->s1<s3)

{

tree->right=remove(tree->right,s3);

return tree;

}

else

{

if(tree->left==NULL&&tree->right==NULL)

{

delete tree;

tree=NULL;

}

else if(tree->left==NULL)

{

temp=tree;

tree=tree->right;

delete temp;

}

else if(tree->right==NULL)

{

temp=tree;

tree=tree->left;

delete temp;

}

else

{

temp=findmin(tree->right);

tree=temp;

tree->right=remove(tree->right,temp->s1);

}

}

return tree;

}

dict\* dict::findmin(dict \*tree)

{

while(tree->left!=NULL)

{

tree=tree->left;

}

return tree;

}

void dict::input\_find()

{

flag2=0,cmp=0;

if(root!=NULL)

{

cout<<"\nEnter the keyword to be searched:\n";

cin>>s1;

find(root,s1);

if(flag2==0)

{

cout<<"Number of comparisons needed: "<<cmp<<"\n";

cmp=0;

}

}

else

{

cout<<"\nThere are no words in the dictionary.\n";

}

}

dict\* dict::find(dict \*tree,string s3)

{

if(tree==NULL)

{

cout<<"\nWord not found.\n";

flag2=1;

flag3=1;

cmp=0;

}

else

{

if(tree->s1==s3)

{

cmp++;

cout<<"\nWord found.\n";

cout<<tree->s1<<": "<<tree->s2<<"\n";

tree1=tree;

return tree;

}

else if(tree->s1>s3)

{

cmp++;

find(tree->left,s3);

}

else if(tree->s1<s3)

{

cmp++;

find(tree->right,s3);

}

}

return tree;

}

void dict::input\_update()

{

if(root!=NULL)

{

cout<<"\nEnter the keyword to be updated:\n";

cin>>s1;

update(root,s1);

}

else

{

cout<<"\nThere are no words in the dictionary.\n";

}

}

dict\* dict::update(dict \*tree,string s3)

{

flag3=0;

find(tree,s3);

if(flag3==0)

{

cout<<"\nEnter the updated meaning of the keyword:\n";

cin.ignore();

getline(cin,tree1->s2);

cout<<"\nThe meaning of '"<<s3<<"' has been updated.\n";

}

return tree;

}

int main()

{

int ch;

dict d;

do

{

cout<<"\n==========================================\n"

"\n\*\*\*\*\*\*\*\*DICTIONARY\*\*\*\*\*\*\*\*\*\*\*:\n"

"\nEnter your choice:\n"

"1.Add new keyword.\n"

"2.Display the contents of the Dictionary.\n"

"3.Delete a keyword.\n"

"4.Find a keyword.\n"

"5.Update the meaning of a keyword.\n"

"6.Exit.\n"

"===============================================\n";

cin>>ch;

switch(ch)

{

case 1:d.input();

break;

case 2:d.input\_display();

break;

case 3:d.input\_remove();

break;

case 4:d.input\_find();

break;

case 5:d.input\_update();

break;

default:cout<<"\nPlease enter a valid option!\n";

break;

}

}while(ch!=6);

return 0;

}

/\* LAB 6

Department maintains a student information. The file contains roll number, name, division

and address. Allow user to add, delete information of student. Display information of

particular employee. If record of student does not exist an appropriate message is displayed.

If it is, then the system displays the student details. Use sequential file to main the data.

\*/

#include<iostream>

#include<fstream>

#include<string.h>

using namespace std;

class student

{

typedef struct stud

{

int roll;

char name[10];

char div;

char add[10];

}stud;

stud rec;

public:

void create();

void display();

int search();

void Delete();

};

void student::create()

{

char ans;

ofstream fout;

fout.open("stud.dat",ios::out|ios::binary);

do

{

cout<<"\n\tEnter Roll No of Student : ";

cin>>rec.roll;

cout<<"\n\tEnter a Name of Student : ";

cin>>rec.name;

cout<<"\n\tEnter a Division of Student : ";

cin>>rec.div;

cout<<"\n\tEnter a Address of Student : ";

cin>>rec.add;

fout.write((char \*)&rec,sizeof(stud))<<flush;

cout<<"\n\tDo You Want to Add More Records: ";

cin>>ans;

}while(ans=='y'||ans=='Y');

fout.close();

}

void student::display()

{

ifstream fin;

fin.open("stud.dat",ios::in|ios::binary);

fin.seekg(0,ios::beg);

cout<<"\n\tThe Content of File are:\n";

cout<<"\n\tRoll\tName\tDiv\tAddress";

while(fin.read((char \*)&rec,sizeof(stud)))

{

if(rec.roll!=-1)

cout<<"\n\t"<<rec.roll<<"\t"<<rec.name<<"\t"<<rec.div<<"\t"<<rec.add;

}

fin.close();

}

int student::search()

{

int r,i=0;

ifstream fin;

fin.open("stud.dat",ios::in|ios::binary);

fin.seekg(0,ios::beg);

cout<<"\n\tEnter a Roll No: ";

cin>>r;

while(fin.read((char \*)&rec,sizeof(stud)))

{

if(rec.roll==r)

{

cout<<"\n\tRecord Found...\n";

cout<<"\n\tRoll\tName\tDiv\tAddress";

cout<<"\n\t"<<rec.roll<<"\t"<<rec.name<<"\t"<<rec.div<<"\t"<<rec.add;

return i;

}

i++;

}

fin.close();

return 0;

}

void student::Delete()

{

int pos;

pos=search();

fstream f;

f.open("stud.dat",ios::in|ios::out|ios::binary);

f.seekg(0,ios::beg);

if(pos==0)

{

cout<<"\n\tRecord Not Found";

return;

}

int offset=pos\*sizeof(stud);

f.seekp(offset);

rec.roll=-1;

strcpy(rec.name,"NULL");

rec.div='N';

strcpy(rec.add,"NULL");

f.write((char \*)&rec,sizeof(stud));

f.seekg(0);

f.close();

cout<<"\n\tRecord Deleted";

}

int main()

{

student obj;

int ch,key;

char ans;

do

{

cout<<"\n\t\*\*\*\*\* Student Information \*\*\*\*\*";

cout<<"\n\t1. Create\n\t2. Display\n\t3. Delete\n\t4. Search\n\t5. Exit";

cout<<"\n\t..... Enter Your Choice: ";

cin>>ch;

switch(ch)

{

case 1: obj.create();

break;

case 2: obj.display();

break;

case 3: obj.Delete();

break;

case 4: key=obj.search();

if(key==0)

cout<<"\n\tRecord Not Found...\n";

break;

case 5:

break;

}

cout<<"\n\t..... Do You Want to Continue in Main Menu: ";

cin>>ans;

}while(ans=='y'||ans=='Y');

return 1;

}

// Problem Statement: 7 Represent a given graph using adjacency matrix to find length of shortest

path between every pair of vertices. Use Floyd Warshall&#39;s algorithm to implement it.//

#include <iostream>

using namespace std;

#define V 4

#define INF 99999

void printSolution(int dist[][V]);

void floydWarshall(int dist[][V])

{

int i, j, k;

for (k = 0; k < V; k++) {

// Pick all vertices as source one by one

for (i = 0; i < V; i++) {

// Pick all vertices as destination for the

// above picked source

for (j = 0; j < V; j++) {

// If vertex k is on the shortest path from

// i to j, then update the value of

// dist[i][j]

if (dist[i][j] > (dist[i][k] + dist[k][j])

&& (dist[k][j] != INF

&& dist[i][k] != INF))

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

// Print the shortest distance matrix

printSolution(dist);

}

/\* A utility function to print solution \*/

void printSolution(int dist[][V])

{

cout << "The following matrix shows the shortest "

"distances"

" between every pair of vertices \n";

for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {

if (dist[i][j] == INF)

cout << "INF"

<< " ";

else

cout << dist[i][j] << " ";

}

cout << endl;

}

}

// Driver's code

int main()

{

int graph[V][V] = { { 0, 5, INF, 10 },

{ INF, 0, 3, INF },

{ INF, INF, 0, 1 },

{ INF, INF, INF, 0 } };

// Function call

floydWarshall(graph);

return 0;

}

// This code is contributed by Mythri J L

KIMI MAM LAB 1

/\*

Author: Parag Ghorpade

PROBLEM STATEMENT:

To create ADT that implements the SET concept.

a. Add (newElement) -Place a value into the set

b. Remove (element) Remove the value

c. Contains (element) Return true if element is in collection

d. Size () Return number of values in collection Iterator () Return an iterator used to loop over collection

e. Intersection of two sets

f. Union of two sets

g. Difference between two sets

h.Subset

\*/

#include <iostream>

#include<list>

#include<cstdlib>

using namespace std;

class set{

private:

int num,flag=1;

public:

list<int>l,l1,u,I,d;

list<int>::iterator t,t1,t2,t3,t4;

void add();

void delete1(int);

void search(int);

void searchB(int);

void display();

void union1();

void Intersection();

void insert();

void Differerence();

};

void set::insert()

{

int n,m;

cout<<"\nSET A:\n";

cout<<"How many Elements You want Add in Set A:\n";

cin>>n;

cout<<"Enter Elements\n";

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

{

cin>>num;

l.push\_back(num);

}

cout<<"\nSET B:\n";

cout<<"How many Elements You want Add in Set B:\n";

cin>>m;

cout<<"Enter Elements\n";

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

{

cin>>num;

l1.push\_back(num);

}

}

void set::add()

{

char c;

cout<<"In Which Set do you want Add Element (A/B)\n";

cin>>c;

if(c=='A' ||c=='a')

{

cout<<"Enter Elements\n";

cin>>num;

l.push\_back(num);

cout<<"\nElement Inserted\n";

}

else if(c=='B' ||c=='b')

{

cout<<"Enter Elements\n";

cin>>num;

l1.push\_back(num);

cout<<"\nElement Inserted\n";

}

else

cout<<"Invalid Set!!!";

}

void set::display()

{

cout<<"The Elements for Set A:\n{\t";

for(t=l.begin();t!=l.end();t++)

{

cout<<\*t<<"\t";

}

cout<<"}";

cout<<"\n\n";

cout<<"The Elements for Set B:\n{\t";

for(t1=l1.begin();t1!=l1.end();t1++)

{

cout<<\*t1<<"\t";

}

cout<<"}";

}

void set::search(int key)

{

for(t=l.begin(),t1=l1.begin();t!=l.end();t++,t1++)

{

if(\*t==key | \*t1==key)

{

cout<<"The Element is Present\n";

flag=1;

break;

}

else

flag=0;

}

if(flag==0)

{

cout<<"The Element is not Present\n";

}

}

void set::delete1(int key)

{

if(l.empty()&& l1.empty())

{

cout<<"The Set A & Set B is Empty\n";

}

else

{

search(key);

if(flag==1)

{

l.remove(key);

l1.remove(key);

cout<<"Element Deleted\n";

}

else

cout<<"Element not Deleted\n";

}

}

void set::union1()

{

int flag=0;

for(t=l.begin();t!=l.end();t++)

{

u.push\_back(\*t);

}

for(t1=l1.begin();t1!=l1.end();t1++)

{

for(t2=u.begin();t2!=u.end();t2++)

{

if(\*t1==\*t2)

{

flag=0;

break;

}

else

flag=1;

}

if(flag==1)

{

u.push\_back(\*t1);

}

}

cout<<"The Union Set of A & B is : {\t";

for(t2=u.begin();t2!=u.end();t2++)

{

cout<<\*t2<<"\t";

}

cout<<"}";

}

void set::Intersection()

{

for(t=l.begin();t!=l.end();t++)

{

for(t1=l1.begin();t1!=l1.end();t1++)

{

if(\*t==\*t1)

{

I.push\_back(\*t);

break;

}

}

}

if(I.empty())

{

cout<<"There is no Common element in Set A & Set B\n";

}

else

{

cout<<"The Intersection Set of A & B is : {\t";

for(t3=I.begin();t3!=I.end();t3++)

{

cout<<\*t3<<"\t";

}

cout<<"}";

}

}

void set::Differerence()

{

int flag=0;

for(t=l.begin();t!=l.end();t++)

{

for(t1=l1.begin();t1!=l1.end();t1++)

{

if(\*t==\*t1)

{

flag=0;

break;

}

else

flag=1;

}

if(flag==1)

{

d.push\_back(\*t);

}

}

if(d.empty())

{

cout<<"The Set A & Set B are Equal\n";

}

else

{

cout<<"The Difference Set of A & B is : {\t";

for(t4=d.begin();t4!=d.end();t4++)

{

cout<<\*t4<<"\t";

}

cout<<"}";

}

}

int main()

{

set s;

int ch,key;

s.insert();

while(1)

{

cout<<"\n\n-----------------------------\n";

cout<<"\nSet Theory\n";

cout<<"\n\n-----------------------------\n";

cout<<"1.Add Element\n";

cout<<"2.Delete Element\n";

cout<<"3.Search Element\n";

cout<<"4.Display\n";

cout<<"5.Union\n";

cout<<"6.Intersection\n";

cout<<"7.Difference\n";

cout<<"8.Exit\n";

cout<<"Enter Your Choice: ";

cin>>ch;

switch(ch)

{

case 1:

s.add();

break;

case 2:

cout<<"Enter which Element to Deleted: ";

cin>>key;

s.delete1(key);

break;

case 3:

cout<<"Enter the Element to be Searched : ";

cin>>key;

s.search(key);

break;

case 4:

cout<<endl;

s.display();

break;

case 5:

s.union1();

break;

case 6:

s.Intersection();

break;

case 7:

s.Differerence();

break;

case 8:

cout<<"Exiting...";

exit(1);

break;

default:

cout<<"Invalid Choice";

}

}

return 0;

}

/\*

Experiment 2 : C++ Program To read details of a book consists of chapters, chapters consist of sections and sections

consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.

\*/

#include <iostream>

#include <string.h>

using namespace std;

struct node // Node Declaration

{

string label;

//char label[10];

int ch\_count;

struct node \*child[10];

} \* root;

class GT // Class Declaration

{

public:

void create\_tree();

void display(node \*r1);

GT()

{

root = NULL;

}

};

void GT::create\_tree()

{

int tbooks, tchapters, i, j, k;

root = new node;

cout << "Enter name of book : ";

cin.get();

getline(cin, root->label);

cout << "Enter number of chapters in book : ";

cin >> tchapters;

root->ch\_count = tchapters;

for (i = 0; i < tchapters; i++)

{

root->child[i] = new node;

cout << "Enter the name of Chapter " << i + 1 << " : ";

cin.get();

getline(cin, root->child[i]->label);

cout << "Enter number of sections in Chapter : " << root->child[i]->label << " : ";

cin >> root->child[i]->ch\_count;

for (j = 0; j < root->child[i]->ch\_count; j++)

{

root->child[i]->child[j] = new node;

cout << "Enter Name of Section " << j + 1 << " : ";

cin.get();

getline(cin, root->child[i]->child[j]->label);

}

}

}

void GT::display(node \*r1)

{

int i, j, k, tchapters;

if (r1 != NULL)

{

cout << "\n-----Book Hierarchy---";

cout << "\n Book title : " << r1->label;

tchapters = r1->ch\_count;

for (i = 0; i < tchapters; i++)

{

cout << "\nChapter " << i + 1;

cout << " : " << r1->child[i]->label;

cout << "\nSections : ";

for (j = 0; j < r1->child[i]->ch\_count; j++)

{

cout << "\n"<< r1->child[i]->child[j]->label;

}

}

}

cout << endl;

}

int main()

{

int choice;

GT gt;

while (1)

{

cout << "-----------------" << endl;

cout << "Book Tree Creation" << endl;

cout << "-----------------" << endl;

cout << "1.Create" << endl;

cout << "2.Display" << endl;

cout << "3.Quit" << endl;

cout << "Enter your choice : ";

cin >> choice;

switch (choice)

{

case 1:

gt.create\_tree();

case 2:

gt.display(root);

break;

case 3:

cout << "Thanks for using this program!!!";

exit(1);

default:

cout << "Wrong choice!!!" << endl;

}

}

return 0;

}

Problem Statement:3

Beginning with an empty binary search tree, construct binary search tree by inserting the values in

the order given. After constructing a binary tree

i. Insert new node

ii. Find number of nodes in longest path from root

iii. Minimum data value found in the tree

iv. Change a tree so that the roles of the left and right pointers are swapped at every node

v. Search a value

#include<iostream>

#include<math.h>

using namespace std;

struct Bstnode

{

int data;

Bstnode \*left=NULL;

Bstnode \*right=NULL;

};

class Btree

{

int n,x,flag;

public:

Bstnode \*root;

Btree()

{

root=NULL;

}

Bstnode \*GetNewNode(int in\_data)

{

Bstnode \*ptr=new Bstnode();

ptr->data=in\_data;

ptr->left=NULL;

ptr->right=NULL;

return ptr;

}

Bstnode \*insert(Bstnode \*temp,int in\_data)

{

if(temp==NULL)

{

temp=GetNewNode(in\_data);

}

else if(temp->data > in\_data)

{

temp->left=insert(temp->left,in\_data);

}

else

{

temp->right=insert(temp->right,in\_data);

}

return temp;

}

void input()

{

cout<<"ENTER NUMBER OF ELEMENTS IN THE BST: ";

cin>>n;

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

{

cout<<"NUMBER= ";

cin>>x;

root=insert(root,x);

}

}

int search(Bstnode \*temp,int in\_data)

{

if(temp!=NULL)

{

if(temp->data==in\_data)

{

cout<<":-- RECORD FOUND --:"<<endl;

return 1;

}

else if(in\_data < temp->data)

{

this->search(temp->left,in\_data);

}

else if(in\_data > temp->data)

{

this->search(temp->right,in\_data);

}

}

else

{

return 0;

}

}

void minvalue(Bstnode \*temp)

{

while(temp->left!=NULL)

{

temp=temp->left;

}

cout<<"MINIMUM VALUE = "<<temp->data<<endl;

}

void maxvalue(Bstnode \*temp)

{

while(temp->right!=NULL)

{

temp=temp->right;

}

cout<<"MAXIMUM VALUE = "<<temp->data<<endl;

}

void mirror(Bstnode \*temp)

{

if(temp==NULL)

{

return;

}

else

{

Bstnode \*ptr;

mirror(temp->left);

mirror(temp->right);

ptr=temp->left;

temp->left=temp->right;

temp->right=ptr;

}

}

void display()

{

cout<<endl<<"---- INORDER TRAVERSAL ----"<<endl;

inorder(root);

cout<<endl;

}

void inorder(Bstnode \*temp)

{

if(temp!=NULL)

{

inorder(temp->left);

cout<<temp->data<<" ";

inorder(temp->right);

}

}

int depth(Bstnode \*temp)

{

if(temp!=NULL)

//return 0;

return(max((depth(temp->left)),(depth(temp->right)))+1);

}

};

int main()

{

int ch,val,cnt;

int a=0;

Btree obj;

obj.input();

obj.display();

cout<<" 1.Number of nodes in longest path \n 2.Minimum Data Value found in the tree";

cout<<"\n 3.Maximum Data Value found in the tree";

cout<<"\n 4.Change a tree so that the roles of the left and right pointers are swapped at every node";

cout<<"\n 5.Search a Value \n 6.Inorder Traversal \n 7.Preorder Traversal \n 8.Postorder Traversal \n 9.Exit";

do

{

cout<<"\n Enter a Choice: "<<endl;

cin>>ch;

switch(ch)

{

case 1:

cout<<endl<<obj.depth(obj.root)<<endl;

break;

case 2:

obj.minvalue(obj.root);

break;

case 3:

obj.maxvalue(obj.root);

break;

case 4:

obj.mirror(obj.root);

obj.display();

break;

case 5:

cout<<"Enter value to be search: ";

cin>>val;

a=obj.search(obj.root,val);

if(a==0)

{

cout<<"ELEMENT NOT FOUND"<<endl;

}

else

cout<<"ELEMENT FOUND"<<endl;

cout<<endl<<a<<endl;

break;

case 6:

cout<<endl<<"---- INORDER TRAVERSAL ----"<<endl;

obj.inorder(obj.root);

cout<<endl;

break;

case 7:

exit(0);

}

cout<<"Do you want to continue(yes=1 and no=0): "<<endl;

cin>>cnt;

}while(cnt==1);

return 0;

}

/\*

Author: Parag Ghorpade

PROBLEM STATEMENT: 4

You have a business with several offices; you want to lease phone lines to connect them up

with each other; and the phone company charges different amounts of money to connect

different pairs of cities. You want a set of lines that connects all your offices with minimum total cost.

Solve the problem by suggesting appropriate data structures.

\*/

#include<iostream>

using namespace std;

class tree

{

int a[20][20],l,u,w,i,j,v,e,visited[20];

public:

void input();

void display();

void minimum();

};

void tree::input()

{

cout<<"Enter the no. of branches: ";

cin>>v;

for(i=0;i<v;i++)

{

visited[i]=0;

for(j=0;j<v;j++)

{

a[i][j]=999;

}

}

cout<<"\nEnter the no. of connections: ";

cin>>e;

for(i=0;i<e;i++)

{

cout<<"Enter the end branches of connections: "<<endl;

cin>>l>>u;

cout<<"Enter the phone company charges for this connection: ";

cin>>w;

a[l-1][u-1]=a[u-1][l-1]=w;

}

}

void tree::display()

{

cout<<"\nAdjacency matrix:";

for(i=0;i<v;i++)

{

cout<<endl;

for(j=0;j<v;j++)

{

cout<<a[i][j]<<" ";

}

cout<<endl;

}

}

void tree::minimum()

{

int p=0,q=0,total=0,min;

visited[0]=1;

for(int count=0;count<(v-1);count++)

{

min=999;

for(i=0;i<v;i++)

{

if(visited[i]==1)

{

for(j=0;j<v;j++)

{

if(visited[j]!=1)

{

if(min > a[i][j])

{

min=a[i][j];

p=i;

q=j;

}

}

}

}

}

visited[p]=1;

visited[q]=1;

total=total+min;

cout<<"Minimum cost connection is"<<(p+1)<<" -> "<<(q+1)<<" with charge : "<<min<< endl;

}

cout<<"The minimum total cost of connections of all branches is: "<<total<<endl;

}

int main()

{

int ch;

tree t;

do

{

cout<<"==========PRIM'S ALGORITHM================="<<endl;

cout<<"\n1.INPUT\n \n2.DISPLAY\n \n3.MINIMUM\n"<<endl;

cout<<"Enter your choice :"<<endl;

cin>>ch;

switch(ch)

{

case 1: cout<<"\*\*\*\*\*\*\*INPUT YOUR VALUES\*\*\*\*\*\*\*"<<endl;

t.input();

break;

case 2: cout<<"\*\*\*\*\*\*\*DISPLAY THE CONTENTS\*\*\*\*\*\*\*\*"<<endl;

t.display();

break;

case 3: cout<<"\*\*\*\*\*\*\*\*\*MINIMUM\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

t.minimum();

break;

}

}while(ch!=4);

return 0;

}

/\*PROBLEM STATEMENT 5 Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm.\*/

#include<iostream>

using namespace std;

class Heap

{

int n;

int \*minheap,\*maxheap;

public:

void get();

void displayMin(){cout<<"Minimum marks are :"<<minheap[0]<<endl;}

void displayMax(){cout<<"Maximum marks are :"<<maxheap[0]<<endl;}

void upAdjust(bool,int);

};

void Heap::get()

{

cout<<"Enter number of students."<<endl;

cin>>n;

int k;

minheap=new int[n];

maxheap=new int[n];

cout<<"Enter marks of students."<<endl;

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

{

cin>>k;

minheap[i]=k;

upAdjust(0,i);

maxheap[i]=k;

upAdjust(1,i);

}

}

void Heap::upAdjust(bool m,int l)

{

int s;

if(!m)

{

while(minheap[(l-1)/2]<minheap[l])

{

s=minheap[l];

minheap[l]=minheap[(l-1)/2];

minheap[(l-1)/2]=s;

l=(l-1)/2;

if(l==-1)

break;

}

}

else

{

while(maxheap[(l-1)/2]>maxheap[l])

{

s=maxheap[l];

maxheap[l]=maxheap[(l-1)/2];

maxheap[(l-1)/2]=s;

l=(l-1)/2;

if(l==-1)

break;

}

}

}

main()

{

Heap H;

H.get();

H.displayMin();

H.displayMax();

return(0);

}