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

class TelephoneBook:

def \_\_init\_\_(self, name, tel\_no):

self.name = name

self.tel\_no = tel\_no

def Insertion\_QuadProbing():

hashtable=[None for i in range(10)]

num\_records = int(input("\nEnter number of records : "))

j=1

for i in range(num\_records):

n = input("Enter name : ")

t = int(input("Enter telephone no. : "))

hashValue = t%10 #hash function

if hashtable[hashValue] is None:

hashtable[hashValue] = TelephoneBook(n,t) #creating obj of class and inserting into hashtable

elif hashtable[hashValue] is not None:

hashValue = (hashValue + (j\*j)) % 10

hashtable[hashValue] = TelephoneBook(n,t)

j+=1

return hashtable

def Insertion\_DoubleHashing():

hashtable=[None for i in range(10)]

num\_records = int(input("\nEnter number of records : "))

j=2

for i in range(num\_records):

n = input("Enter name : ")

t = int(input("Enter telephone no. : "))

hashvalue = t%9 + 7-(t%7) #finding hashvalue using 2 hash functions 1) key%9

if hashtable[hashvalue] is None: # 2) 7-(key%7)

hashtable[hashvalue] = TelephoneBook(n,t)

elif hashtable[hashvalue] is not None:

hashvalue = t%9 + j\*(7-(t%7))

j+=1

return hashtable

def Display\_QP(hash1):

print("-------------------------------")

print("Index\tName\tTelephone No.")

print("-------------------------------")

for obj in hash1:

if(obj is None):

print("-\t-\t-")

if (obj is not None):

print(hash1.index(obj),"\t",obj.name,"\t", obj.tel\_no)

print("-------------------------------")

def Display\_DH(hash2):

print("-------------------------------")

print("Index\tName\tTelephone No.")

print("-------------------------------")

for obj in hash2:

if(obj is None):

print("-\t-\t-")

if (obj is not None):

print(hash2.index(obj),"\t",obj.name,"\t", obj.tel\_no)

print("-------------------------------")

def Search(hash1,hash2):

n = input("Enter name to search: ")

f1=0

f2=0

for obj in hash1:

if(obj is None):

continue

if obj.name == n:

print("\nFound in Hashtable-1 !")

print("-------------------------------")

print("Index\tName\tTelephone No.")

print("-------------------------------")

print(hash1.index(obj),"\t",obj.name,"\t", obj.tel\_no)

print("-------------------------------")

f1=1

for obj in hash2:

if(obj is None):

continue

if obj.name == n:

print("\nFound in Hashtable-2 !")

print("-------------------------------")

print("Index\tName\tTelephone No.")

print("-------------------------------")

print(hash2.index(obj),"\t",obj.name,"\t", obj.tel\_no)

print("-------------------------------")

f2=1

if f1==0 and f2==0:

print("\nNot found !!!\n")

def main():

# initialising hashtables to "None"

hash1=[None for i in range(10)]

hash2=[None for i in range(10)]

print("-------------------------------")

print(" Group-A Assignment-1")

while True:

print("-------------------------")

print("\t1.Insert Value")

print("\t2.Display")

print("\t3.Search")

print("\t4.Exit")

print("-------------------------")

ch = int(input("Enter choice : "))

if ch==1:

print("\nSelect collision method-")

print("\t1.Quadratic Probing")

print("\t2.Double Hashing")

c = int(input("Enter choice : "))

if c==1:

hash1=Insertion\_QuadProbing()

elif c==2:

hash2=Insertion\_DoubleHashing()

elif ch==2:

print("\t1.Display QP")

print("\t2.Display DH")

c1 = int(input("Enter choice : "))

if c1==1:

Display\_QP(hash1) #To display hashtable which uses quadratic probing collision method

else:

Display\_DH(hash2) #To display hashtable which uses double hashing collision method

elif ch==3:

Search(hash1,hash2)

elif ch==4:

quit()

else:

print("! Enter valid choice.")

main()

2.

setOne=[]

setTwo=[]

def addVal(Set):

val = int(input("Value to add:\t"))

if (val in Set): # Checking if value already exists in set

print(f"{val} already exists in the set.")

else: # Adding value if does not exist

Set.append(val)

print(f"Set is:\t{Set}")

def delVal(Set):

val = int(input("Value to remove:\t"))

if(val not in Set): # Checking if value is not there in set

print(f"{val} is not present in the set.")

else: # Deleting value if it exists in set

Set.remove(val)

print(f"Set is:\t{Set}")

def searchVal(Set):

val = int(input("Value to search:\t"))

if(val in Set): # Check if value is present in set

print(f"{val} is present in the set.")

else: # Print if value not present in set

print(f"{val} is not present in the set.")

def size(Set):

print(f"Size of set is:\t{len(Set)}")

def intersection(setA, setB):

intersectionSet = []

for i in setA:

if i in setB:

intersectionSet.append(i)

print(f"Intersection is:\t{intersectionSet}")

def iterator(set1):

a=iter(set1)

for i in range(0,len(set1)-1):

print(next(a),"->",end=' ')

print(next(a))

def union(setA, setB):

unionSet = []

for i in setA:

unionSet.append(i)

for j in setB:

if j not in setA:

unionSet.append(j)

print(f"Union is:\t{unionSet}")

def difference(setA, setB):

differenceSet = []

for i in setA:

if i not in setB:

differenceSet.append(i)

print(f"Difference is:\t{differenceSet}")

def subsetCheck(setA, setB):

for i in setB:

if i not in setA:

return False

return True

def subset(setA, setB):

if subsetCheck(setA,setB):

print("Set two is a subset of set one.")

else:

print("Set two is not a subset of set one.")

def main():

while (True):

print("--- MAIN MENU ---")

print("1 -> Add value to set")

print("2 -> Remove value from set")

print("3 -> Search value in set")

print("4 -> Show size of set")

print("5 -> Iterator (implementation pending)")

print("6 -> Intersection of two sets")

print("7 -> Union of two sets")

print("8 -> Difference of two sets")

print("9 -> Subset of two sets")

print("10 -> Exit")

optn = int(input("Choose an option (1-10):\t"))

if (optn == 1):

setSel = int(input("Which set to operate on?\n1. Set one\n2. Set two\nSet 1/2:\t"))

total = int(input("Total values to add:\t"))

for i in range(total):

if (setSel == 1):

addVal(setOne)

elif (setSel == 2):

addVal(setTwo)

else:

print("\nPlease choose a valid option.\n")

elif (optn == 2):

setSel = int(input("Which set to operate on?\n1. Set one\n2. Set two\nSet 1/2:\t"))

if (setSel == 1):

delVal(setOne)

elif (setSel == 2):

delVal(setTwo)

else:

print("\nPlease choose a valid option.\n")

elif (optn == 3):

setSel = int(input("Which set to operate on?\n1. Set one\n2. Set two\nSet 1/2:\t"))

if (setSel == 1):

searchVal(setOne)

elif (setSel == 2):

searchVal(setTwo)

else:

print("\nPlease choose a valid option.\n")

elif (optn == 4):

setSel = int(input("Which set to operate on?\n1. Set one\n2. Set two\nSet 1/2:\t"))

if (setSel == 1):

size(setOne)

elif (setSel == 2):

size(setTwo)

else:

print("\nPlease choose a valid option.\n")

elif (optn == 5):

setSel = int(input("Which set to operate on?\n1. Set one\n2. Set two\nSet 1/2:\t"))

a=None

if (setSel == 1):

iterator(setOne)

elif (setSel == 2):

iterator(setTwo)

else:

print("\nPlease choose a valid option.\n")

elif (optn == 6):

intersection(setOne, setTwo)

elif (optn == 7):

union(setOne, setTwo)

elif (optn == 8):

difference(setOne, setTwo)

elif (optn == 9):

subset(setOne, setTwo)

elif (optn == 10):

print("\n\n## END OF CODE\n\n")

exit(1)

else:

print("Please choose a valid option (1-10).")

main() # Calling the main function

3.

# include <iostream>

# include <cstdlib>

# include <string.h>

using namespace std;

struct node

{

char label[10];

int ch\_count;

struct node \*child[10];

}\*root;

/\*

\* Class Declaration

\*/

class GT

{

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>>root->label;

cout<<"Enter no. of chapters in book";

cin>>tchapters;

root->ch\_count = tchapters;

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

{

root->child[i] = new node;

cout<<"Enter Chapter name\n";

cin>>root->child[i]->label;

cout<<"Enter no. 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 Section "<<j+1<<"name\n";

cin>>root->child[i]->child[j]->label;

//cout<<"Enter no. of subsections in "<<r1->child[i]->child[j]->label;

//cin>>r1->child[i]->ch\_count;

}

}

}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<<"\n Chapter "<<i+1;

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

cout<<"\n Sections";

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

{

//cin>>r1->child[i]->child[j]->label;

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

}

}

}

}

int main()

{

int choice;

GT gt;

while (1)

{

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

cout<<"Book Tree Creation\n1.Create\n2.Display\n3.Quit"<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:gt.create\_tree();

case 2: gt.display(root);

break;

case 3:exit(1);

default:

cout<<"Wrong choice"<<endl;

}

}

}

**4.**

#include <iostream>

using namespace std;

struct node

{

int data;

node \*L;

node \*R;

};

node \*root,\*temp;

int count,key;

class bst

{

public:

void create();

void insert(node\*,node\*);

void disin(node\*);

void dispre(node\*);

void dispost(node\*);

void search(node\*,int);

int height(node\*);

void mirror(node\*);

void min(node\*);

bst()

{

root=NULL;

count=0;

}

};

void bst::create()

{

char ans;

do

{

temp=new node;

cout<<"Enter the data : ";

cin>>temp->data;

temp->L=NULL;

temp->R=NULL;

if(root==NULL)

{

root=temp;

}

else

insert(root,temp);

cout<<"Do you want to insert more value :";

cin>>ans;

count++;

cout<<endl;

}while(ans=='y');

cout<<"The Total no.of nodes are:"<<count;

}

void bst::insert(node \*root,node\* temp)

{

if(temp->data>root->data)

{

if(root->R==NULL)

{

root->R=temp;

}

else

insert(root->R,temp);

}

else

{

if(root->L==NULL)

{

root->L=temp;

}

else

insert(root->L,temp);

}

}

void bst::disin(node \*root)

{

if(root!=NULL)

{

disin(root->L);

cout<<root->data<<"\t";

disin(root->R);

count++;

}

}

void bst::dispre(node \*root)

{

if(root!=NULL)

{

cout<<root->data<<"\t";

dispre(root->L);

dispre(root->R);

}

}

void bst::dispost(node \*root)

{

if(root!=NULL)

{

dispost(root->L);

dispost(root->R);

cout<<root->data<<"\t";

}

}

void bst::search(node \* root,int key)

{

int flag=0;

cout<<"\nEnter your key : ";

cin>>key;

temp=root;

while(temp!=NULL)

{

if(key==temp->data)

{

cout<<" KEY FOUND \n";

flag=1;

break;

}

node \*parent=temp;

if(key>parent->data)

{

temp=temp->R;

}

else

{

temp=temp->L;

}

}

if(flag==0)

{

cout<<" KEY NOT FOUND "<<endl;

}

}

int bst::height(node \*root)

{

int hl,hr; if(root==NULL)

{

return 0;

} else if(root->L==NULL && root->R==NULL)

{

return 0;

}

cout<<endl;

hr=height(root->R);

hl=height(root->L);

if(hr>hl)

{

return(1+hr);

}

else

{

return(1+hl);

}

}

void bst::min(node \*root)

{

temp=root;

cout<<endl;

while(temp->L!=NULL)

{

temp=temp->L;

}

cout<<temp->data;

}

void bst::mirror(node \*root)

{

temp=root;

if(root!=NULL)

{

mirror(root->L);

mirror(root->R);

temp=root->L;

root->L=root->R;

root->R=temp;

}

}

int main()

{

bst t;

int ch;

char ans;

do

{

cout<<"\n1) Insert new node \n2)number of nodes in longest path \n3) minimum \n4) mirror \n5) search \n6) inorder \n7) preorder \n8) postorder"<<endl;

cin>>ch;

switch(ch)

{

case 1:

t.create();

break;

case 2:

cout<<"\n Number of nodes in longest path:"<<(1+(t.height(root)));

break;

case 3:

cout<<"\nThe min element is:";

t.min(root);

break;

case 4:

t.mirror(root);

cout<<"\nThe mirror of tree is: ";

t.disin(root);

break;

case 5:

t.search(root,key);

break;

case 6:

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

t.disin(root);

break;

case 7:

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PREORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

t.dispre(root);

break;

case 8:

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*POSTORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

t.dispost(root);

break;

}

cout<<"\nDo you want to continue : "; cin>>ans;

}while(ans=='y');

return 0;

}

**4.OR**

#include <iostream>

using namespace std;

//Structre to create node

struct Node{

int data;

Node\* right;

Node\* left;

};

//Function to create node

Node\* create(int data){

Node\* temp=new Node();

temp->data=data;

temp->left=temp->right=NULL;

return temp;

}

//Function to insert node

void insert(Node\* &root,int data){

if(root==NULL){

root=create(data); //say 5 root

} //Next element n

else if(root->data > data){ //if (5 > n)

insert(root->left,data); //n will go to left side

}

else{ //else

insert(root->right,data); //n will go to right side

}

}

//Preorder

void displayPre(Node\* root){

if(root!=NULL){

cout<<root->data<<" "; //PARENT

displayPre(root->left); //LEFT

displayPre(root->right); //RIGHT

}

}

//Inorder

void displayIn(Node\* root){

if(root!=NULL){

displayIn(root->left); //LEFT

cout<<root->data<<" "; //PARENT

displayIn(root->right); //RIGHT

}

}

//Postorder

void displayPost(Node\* root){

if(root!=NULL){

displayPost(root->left); //LEFT

displayPost(root->right); //RIGHT

cout<<root->data<<" "; //PARENT

}

}

//Function to calculate Height

int height(Node\* root){

if(root==NULL){

return 0;

}

else{

int l\_h=height(root->left);

int r\_h=height(root->right);

if(l\_h>=r\_h){

return l\_h+1;

}

else{

return r\_h+1;

}

}

}

//Function to seach for value

void search(Node\* root,int value){

if(root!=NULL){

if(root->data>value){

search(root->left,value);

}

else if(root->data<value){

search(root->right,value);

}

else if(root->data==value){

cout<<"\nelement FOUND";

}

}

else{

cout<<"\nelement NOT found";

}

}

//Function to find smallest element i.e display extreme left

void smallest(Node\* root){

if(root->left!=NULL){

smallest(root->left);

}

else{

cout<<"Smallest :: "<<root->data;

}

}

//Function to display largest element i.e display extreme right

void largest(Node\* root){

if(root->right!=NULL){

largest(root->right);

}

else{

cout<<"\nlargest :: "<<root->data;

}

}

//Function mirror the tree

void mirror(Node\* root){

if(root==NULL){

return;

}

mirror(root->left);

mirror(root->right);

swap(root->left,root->right);

}

int main(){

bool loop=1;

Node \* root=NULL;

int ch,n,num;

while(loop==1){

//Menu

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

<<"1. create BST"<<endl

<<"2. preorder"<<endl

<<"3. inorder"<<endl

<<"4. postorder"<<endl

<<"5. height"<<endl

<<"6. search"<<endl

<<"7. smallest"<<endl

<<"8. largest"<<endl

<<"9. mirror"<<endl

<<"10. exit"<<endl

<<"ENTER :: ";

cin>>ch;

switch (ch)

{

case 1:

{

cout<<"\nEnter the number of elements :: ";

cin>>n;

cout<<"Enter the numbers :: ";

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

cin>>num;

insert(root,num);

}

break;

}

case 2:

{

cout<<"\nPRE ORDER : ";

displayPre(root);

break;

}

case 3:

{

cout<<"\nIN ORDER : ";

displayIn(root);

break;

}

case 4:

{

cout<<"\nPOST ORDER : ";

displayPost(root);

break;

}

case 5:

{

int h=height(root);

cout<<"\nHeight of BST :: "<<h;

break;

}

case 6:

{

int value;

cout<<"Enter the element to search :: ";

cin>>value;

search(root, value);

break;

}

case 7:

{

smallest(root);

break;

}

case 8:

{

largest(root);

break;

}

case 9:

{

cout<<"\nBEFORE MIRROR"

<<"\nInorder :: ";

displayIn(root);

mirror(root);

cout<<"\nAFTER MIRROR"

<<"\nInorder :: ";

displayIn(root);

break;

}

case 10:

{

loop=0;

break;

}

}

}

return 0;

}

5.

#include<iostream>

using namespace std;

class Node

{

public:

char data;

Node \*left,\*right;

};

class Tree

{

public:

Node \*root;

Tree()

{

root=NULL;

}

void inorder\_recursive(Node \*);

void preorder\_recursive(Node \*);

void postorder\_recursive(Node \*);

void inorder\_nonrecursive(Node \*);

void preorder\_nonrecursive(Node \*);

void postorder\_nonrecursive(Node \*);

Node \*expression();

};

Node \* Tree::expression()

{

Node \*temp;

int i,top=-1;

char exp[20];

Node \*stack[10];

int flag[10];

cout<<"Enter the expression:";

cin>>exp;

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

{

if(exp[i]>='a' && exp[i]<='z')

{

temp=new Node;

temp->data=exp[i];

temp->left=temp->right=NULL;

stack[++top]=temp;

}

else

{

root=new Node;

root->data=exp[i];

root->left=root->right=NULL; // root creation for operator

root->right=stack[top--];

root->left=stack[top--];

stack[++top]=root;

}

}

root=stack[top--];

return root;

}

void Tree:: inorder\_recursive(Node \*root)

{

if(root!=NULL)

{

inorder\_recursive(root->left);

cout<<root->data;

inorder\_recursive(root->right);

}

}

void Tree:: preorder\_recursive(Node \*root)

{

if(root!=NULL)

{

cout<<root->data;

preorder\_recursive(root->left);

preorder\_recursive(root->right);

}

}

void Tree:: postorder\_recursive(Node \*root)

{

if(root!=NULL)

{

postorder\_recursive(root->left);

postorder\_recursive(root->right);

cout<<root->data;

}

}

void Tree::inorder\_nonrecursive(Node \*root)

{

struct Node \*stack[20];

int top=-1;

while(root!=NULL||top!=-1)

{

if(root!=NULL)

{

stack[++top]=root;

root=root->left;

}

else

{

root=stack[top--];

cout<<root->data;

root=root->right;

}

}

}

void Tree::preorder\_nonrecursive(Node \*root)

{

struct Node \*stack[20];

int top=-1;

stack[++top]=root;

while(top!=-1)

{

root=stack[top--];

if(root!=NULL)

{

cout<<root->data;

stack[++top]=root->right;

stack[++top]=root->left;

}

}

}

void Tree::postorder\_nonrecursive(Node \*root)

{

struct Node \*stack[20];

int top=-1;

int flag[10];

while(top!=-1||root!=NULL)

{

if(root!=NULL)

{

stack[++top]=root;

flag[top]=0;

root=root->left;

}

else

{

root=stack[top];

if(flag[top]==1)

{

cout<<root->data;

top--;

root=NULL;

}

else

{

flag[top]=1;

root=root->right;

}

}

}

}

int main()

{

Tree s;

s.root=s.expression();

int ch,ans;

cout<<"\n\*\*\*\*\*MENU\*\*\*\*\*";

cout<<"\n1.Recursive function for inorder";

cout<<"\n2.Recursive function for preorder";

cout<<"\n3.Recursive function for postorder";

cout<<"\n4.Nonrecursive function for inorder";

cout<<"\n5.Nonrecursive function for preorder";

cout<<"\n6.Nonrecursive function forpostorder";

do

{

cout<<"\nENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1:

cout<<"\n Inorder Exp with Recursive =>";

s.inorder\_recursive(s.root);

break;

case 2:

cout<<"\n preorder Exp with Recursive =>";

s.preorder\_recursive(s.root);

break;

case 3:

cout<<"\n postorder Exp with Recursive =>";

s.postorder\_recursive(s.root);

break;

case 4:

cout<<"\n Inorder Exp with Non\_Recursive =>";

s.inorder\_nonrecursive(s.root);

break;

case 5:

cout<<"\n preorder Exp with Non\_Recursive =>";

s.preorder\_nonrecursive(s.root);

break;

case 6:

cout<<"\n postorder Exp with Non\_Recursive =>";

s.postorder\_nonrecursive(s.root);

break;

}

cout<<"\n do you want to continue";

cin>>ans;

}while(ans==1);

return 0;

}

6.

Graph Example :

1)

FC Clg 6

^

|

|

MCOE Clg 5

^

|

|

Pune------>MCOE--------->JM-------->COEP

City bus corner ground

1 stop 3 4

2

2)

4

^

|

|

1----->2----->3

3)

A

/ | \

B C D

/ \

E F

\*/

#include <iostream>

#include <string>

#include <map>

#include <list>

using namespace std;

class Graph

{

public:

int V;

string places[10];

map<string, bool> visited;

map<string, list<string>> adj;

Graph(int v,string place[10]){

V=v;

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

places[i]=place[i];

}

void addEdge(string v, string w);

void set\_default(){

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

visited[places[i]]=false;

}

}

void DFS(string v);

void BFS(string v);

};

void Graph::addEdge(string v, string w)

{

adj[v].push\_back(w);

}

void Graph::DFS(string v)

{

visited[v] = true;

cout << v << " ";

list<string>::iterator i;

for (i = adj[v].begin(); i != adj[v].end(); ++i)

if (!visited[\*i])

DFS(\*i);

}

void Graph::BFS(string s)

{

visited[s]=false;

list<string> queue;

visited[s] = true;

queue.push\_back(s);

list<string>::iterator i;

while(!queue.empty())

{

s = queue.front();

cout << s << " ";

queue.pop\_front();

for (i = adj[s].begin(); i != adj[s].end(); ++i)

{

if (!visited[\*i])

{

visited[\*i] = true;

queue.push\_back(\*i);

}

}

}

}

int main(){

int n;

cout<<"\nEnter the number of elements in the graph - ";

cin>>n;

int mat[n][n];

string name;

string name\_list[n];

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

{

cout<<"Enter the name of vertice "<<i+1<<" - ";

cin>>name;

name\_list[i] = name;

}

Graph g(n,name\_list);

char connected;

cout<<"\nY-YES & N-NO\n";

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

{

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

{

cout<<"\n"<<name\_list[i]<<" connected "<<name\_list[j]<<"--->";

cin >> connected;

if (connected=='y' || connected=='Y'){

g.addEdge(name\_list[i],name\_list[j]);

}

else {

continue;

}

}

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

}

char loop='y';

while(loop=='y' || loop=='Y'){

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

cout<<"Enter place :: ";

cin>>name;

cout<<"\nDFS :: ";

g.set\_default();

g.DFS(name);

cout<<"\nBFS :: ";

g.set\_default();

g.BFS(name);

cout<<"\nYou want to continue (y/n) :: ";

cin>>loop;

}

return 0;

}

7.

#include<iostream>

using namespace std;

int main()

{

int n, i, j, k, row, col, mincost=0, min;

char op;

cout<<"Enter no. of vertices: ";

cin>>n;

int cost[n][n];

int visit[n];

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

visit[i] = 0;

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

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

cost[i][j] = -1;

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

{

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

{

cout<<"Do you want an edge between "<<i+1<<" and "<<j+1<<": ";

//use 'i' & 'j' if your vertices start from 0

cin>>op;

if(op=='y' || op=='Y')

{

cout<<"Enter weight: ";

cin>>cost[i][j];

cost[j][i] = cost[i][j];

}

}

}

visit[0] = 1;

for(k=0; k<n-1; k++)

{

min = 999;

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

{

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

{

if(visit[i] == 1 && visit[j] == 0)

{

if(cost[i][j] != -1 && min>cost[i][j])

{

min = cost[i][j];

row = i;

col = j;

}

}

}

}

}

mincost += min;

visit[col] = 1;

cost[row][col] = cost[col][row] = -1;

cout<<row+1<<"->"<<col+1<<endl;

}

//use 'row' & 'col' if your vertices start from 0

8.

/\*

Lab Assignment 8

Given sequence k = k1 <k2 < … <kn of n sorted keys, with a search probability pi for each key ki.

Build the Binary search tree that has the least search cost given the access probability for each key.

\*/

#include <iostream>

#include <limits.h>

using namespace std;

int sum(int freq[], int i, int j)

{

int s = 0;

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

s += freq[k];

return s;

}

int optCost(int keys[], int freq[], int n) {

int cost[n][n];

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

cost[i][i] = freq[i];

for (int length=2; length<=n; length++)

{

for (int i=0; i<=n-length+1; i++)

{

int j = i+length-1;

cost[i][j] = INT\_MAX;

for (int r=i; r<=j; r++)

{

int c = ((r > i)?cost[i][r-1]:0)+((r < j)?cost[r+1][j]:0)+sum(freq, i, j);

if (c < cost[i][j])

cost[i][j] = c;

}

}

}

return cost[0][n-1];

}

int main()

{

int n;

cout<<"Enter the number of keys :: ";

cin>>n;

int keys[10],freq[10];

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

cout<<"Key["<<i<<"] :: ";

cin>>keys[i];

cout<<"Freq["<<i<<"] :: ";

cin>>freq[i];

}

cout << "Cost of Optimal BST is "

<< optCost(keys, freq, n);

return 0;

}

//Input

//10 - 34

//12 - 8

//20 - 50

//Expected output

//142

9.

#include<iostream>

#include<string>

using namespace std;

class dictionary;

class avlnode

{

string keyword;

string meaning;

avlnode \*left,\*right;

int bf;

public:

avlnode()

{

keyword='\0';

meaning='\0';

left=right=NULL;

bf=0;

}

avlnode(string k,string m)

{

keyword=k;

meaning=m;

left=right=NULL;

bf=0;

}

friend class dictionary;

};

class dictionary

{

avlnode \*par,\*loc;

public:

avlnode \*root;

dictionary()

{

root=NULL;

par=loc=NULL;

}

void accept();

void insert(string key,string mean);

void LLrotation(avlnode\*,avlnode\*);

void RRrotation(avlnode\*,avlnode\*);

void inorder(avlnode \*root);

void deletekey(string key);

void descending(avlnode \*);

void search(string);

void update(string,string);

};

void dictionary::descending(avlnode \*root)

{

if(root)

{

descending(root->right);

cout<<root->keyword<<" "<<root->meaning<<endl;

descending(root->left);

}

}

void dictionary::accept()

{

string key,mean;

cout<<"Enter keyword "<<endl;

cin>>key;

cout<<"Enter meaning "<<endl;

cin>>mean;

insert(key,mean);

}

void dictionary::LLrotation(avlnode \*a,avlnode \*b)

{

cout<<"LL rotation"<<endl;

a->left=b->right;

b->right=a;

a->bf=b->bf=0;

}

void dictionary::RRrotation(avlnode \*a,avlnode \*b)

{

cout<<"RR rotation"<<endl;

a->right=b->left;

b->left=a;

a->bf=b->bf=0;

}

void dictionary::insert(string key,string mean)

{

//cout<<"IN Insert \n";

if(!root)

{

//create new root

root=new avlnode(key,mean);

cout<<"ROOT CREATED \n";

return;

}

// else

// {

avlnode \*a,\*pa,\*p,\*pp;

pa=NULL;

p=a=root;

pp=NULL;

while(p)

{

if(p->bf)

{

a=p;

pa=pp;

}

if(key<p->keyword)

{

pp=p;

p=p->left;

}

else if(key>p->keyword)

{

pp=p;

p=p->right;

}

else

{

cout<<"Already exist \n";

return;

}

}

avlnode \*y=new avlnode(key,mean);

if(key < pp->keyword)

{

pp->left=y;

}

else

pp->right=y;

cout<<"KEY INSERTED \n";

int d;

avlnode \*b,\*c;

b=c=NULL;

if(key>a->keyword)

{

cout<<"KEY >A->KEYWORD \n";

b=p=a->right;

d=-1;

cout<<" RIGHT HEAVY \n";

}

else

{

cout<<"KEY < A->KEYWORD \n";

b=p=a->left;

d=1;

cout<<" LEFT HEAVY \n";

}

while(p!=y)

{

if(key>p->keyword)

{

p->bf=-1;

p=p->right;

}

else

{

p->bf=1;

p=p->left;

}

}

cout<<" DONE ADJUSTING INTERMEDIATE NODES \n";

if(!(a->bf)||!(a->bf+d))

{

a->bf+=d;

return;

}

if(d==1)

{

if(b->bf==1)

{

LLrotation(a,b);

}

else

{

cout<<"LR rotation"<<endl;

c=b->right;

b->right=c->left;

a->left=c->right;

c->left=b;

c->right=a;

switch(c->bf)

{

case 1:

{

a->bf=-1;

b->bf=0;

break;

}

case -1:

{

a->bf=0;

b->bf=1;

break;

}

case 0:

{

a->bf=0;

b->bf=0;

break;

}

}

c->bf=0;

b=c;

}

}

if(d==-1)

{

if(b->bf==-1)

{

RRrotation(a,b);

}

else

{

c=b->left;

a->right=c->left;

b->left=c->right;

c->left=a;

c->right=b;

switch(c->bf)

{

case 1:

{

a->bf=0;

b->bf=-1;

break;

}

case -1:

{

a->bf=1;

b->bf=0;

break;

}

case 0:

{

a->bf=0;

b->bf=0;

break;

}

}

c->bf=0;

b=c; //b is new root

}

//else

//cout<<"Balanced \n";

}

//}

if(!pa)

root=b;

else if(a==pa->left)

pa->left=b;

else

pa->right=b;

cout<<"AVL tree created!! \n";

//cout<<"AVL \n";

//inorder(root);

}

void dictionary::search(string key)

{

cout<<"ENTER SEARCH \n";

loc=NULL;

par=NULL;

if(root==NULL)

{

cout<<"Tree not created "<<endl;

// root=key;

loc=NULL;

par=NULL;

}

//par=NULL;loc=NULL;

avlnode \*ptr;

ptr=root;

while(ptr!=NULL)

{

if(ptr->keyword==key)

{

//flag=1;

loc=ptr;

break;

}

else if(key<ptr->keyword)

{

par=ptr;

ptr=ptr->left;

}

else

{

par=ptr;

ptr=ptr->right;

}

}

if(loc==NULL)

{

cout<<"Not found "<<endl;

}

}

void dictionary::update(string oldkey,string newmean)

{

search(oldkey);

loc->meaning=newmean;

cout<<"UPDATE SUCCESSFUL \n";

}

void dictionary::inorder(avlnode \*root)

{

if(root)

{

inorder(root->left);

cout<<root->keyword<<" "<<root->meaning<<endl;

inorder(root->right);

}

}

int main()

{

string k,m;

dictionary d;

int ch;

string key,mean;

do

{

cout<<"1.Insert \n2.Update \n3.Ascending \n4.Descending \n5.Display \n6.Quit \n";

cin>>ch;

switch(ch)

{

case 1:

{

d.accept();

break;

}

case 2:

{

cout<<"Enter key whose meaning to update \n";

cin>>key;

cout<<"Enter new meaning\n";

cin>>mean;

d.update(key,mean);

break;

}

case 3:

d.inorder(d.root);

break;

case 4:

cout<<"Descending \n";

d.descending(d.root);

break;

case 5:

d.inorder(d.root);

break;

default:

break;

}

}while(ch!=6); /\*cout<<"Enter word and its meaning"<<endl;

cin>>k>>m;

d.insert(k,m);\*/

// d.accept();

//cout<<"Enter another word and its meaning \n";

// cin>>k>>m;

// d.insert(k,m);

//cout<<"MAIN \n";

return 0;

}

10.

#include<iostream>

#include<string>

//#define N 20

using namespace std;

string Q[10];

int Pr[10];

int r = -1,f = -1,n;

void enqueue(string data,int p)//Enqueue function to insert data and its priority in queue

{

int i;

if((f==0)&&(r==n-1)) //Check if Queue is full

cout<<"Queue is full";

else {

if(f==-1) { //if Queue is empty

f = r = 0;

Q[r] = data;

Pr[r] = p;

}

else {

for(i = r;i>=f;i--)

{

if(p>Pr[i]) {

Q[i+1] = Q[i];

Pr[i+1] = Pr[i];

}

else break;

}

Q[i+1] = data;

Pr[i+1] = p;

r++;

}

}

}

void print() { //print the data of Queue

int i;

for(i=f;i<=r;i++) {

cout << "Patient's Name - "<<Q[i];

switch(Pr[i]) {

case 1:

cout << " Priority - 'Checkup' " << endl;

break;

case 2:

cout << " Priority - 'Non-serious' " << endl;

break;

case 3:

cout << " Priority - 'Serious' " << endl;

break;

default:

cout << "Priority not found" << endl;

}

}

}

void dequeue() { //remove the data from front

if(f == -1) {

cout<<"Queue is Empty";

}

else {

cout<<"deleted Element ="<<Q[f]<<endl;

cout<<"Its Priority = "<<Pr[f]<<endl;

if(f==r) f = r = -1;

else f++;

}

}

int main() {

string data;

int opt,i,p;

cout<<"Enter Your Choice:-"<<endl;

do {

cout << "1 for Insert the Data in Queue" << endl << "2 for show the Data in Queue " << endl << "3 for Delete the data from the Queue" << endl << "0 for Exit"<< endl;

cin >> opt;

switch(opt) {

case 1:

cout << "Enter the number of patinent" << endl;

cin >> n;

i = 0;

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

{

cout << "Enter your name of the patient : ";

cin >> data;

cout << "Enter your Prioritys (3: serious, 2: non-serious, 1: genral checkup) : ";

cin >> p;

enqueue(data,p);

}

break;

case 2:

print();

break;

case 3:

dequeue();

break;

}

}while(opt!=0);

return 0;

}

11.

#include<iostream>

#include<fstream>

#include<cstring>

using namespace std;

class tel

{

public:

int rollNo,roll1;

char name[10];

char div;

char address[20];

void accept()

{

cout<<"\n\tEnter Roll Number : ";

cin>>rollNo;

cout<<"\n\tEnter the Name : ";

cin>>name;

cout<<"\n\tEnter the Division:";

cin>>div;

cout<<"\n\tEnter the Address:";

cin>>address;

}

void accept2()

{

cout<<"\n\tEnter the Roll No. to modify : ";

cin>>rollNo;

}

void accept3()

{

cout<<"\n\tEnter the name to modify : ";

cin>>name;

}

int getRollNo()

{

return rollNo;

}

void show()

{

cout<<"\n\t"<<rollNo<<"\t\t"<<name<<"\t\t"<<div<<"\t\t"<<address;

}

};

int main()

{

int

i,n,ch,ch1,rec,start,count,add,n1,add2,start2,n2,y,a,b,on,oname,add3,start3,n3,y1,add4,start4,

n4;

char name[20],name2[20];

tel t1;

count=0;

fstream g,f;

do

{

cout<<"\n>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<";

cout<<"\n1.Insert and overwrite\n2.Show\n3.Search & Edit(number)\n4.Search &

Edit(name)\n5.Search & Edit(onlynumber)\n6.Search & edit(only name)\n 7.Delete a Student

Record\n 8.Exit\n\tEnter the Choice\t:";

cin>>ch;

switch(ch)

{

case 1:

f.open("StuRecord.txt",ios::out);

x:t1.accept();

f.write((char\*) &t1,(sizeof(t1)));

cout<<"\nDo you want to enter more records?\n1.Yes\n2.No";

cin>>ch1;

if(ch1==1)

goto x;

else

{

f.close();

break;

}

case 2:

f.open("StuRecord.txt",ios::in);

f.read((char\*) &t1,(sizeof(t1)));

//cout<<"\n\tRoll No.\t\tName \t\t Division \t\t Address";

while(f)

{

t1.show();

f.read((char\*) &t1,(sizeof(t1)));

}

f.close();

break;

case 3:

cout<<"\nEnter the roll number you want to find";

cin>>rec;

f.open("StuRecord.txt",ios::in|ios::out);

f.read((char\*)&t1,(sizeof(t1)));

while(f)

{

if(rec==t1.rollNo)

{

cout<<"\nRecord found";

add=f.tellg();

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

start=f.tellg();

n1=(add-start)/(sizeof(t1));

f.seekp((n1-1)\*sizeof(t1),ios::beg);

t1.accept();

f.write((char\*) &t1,(sizeof(t1)));

f.close();

count++;

break;

}

f.read((char\*)&t1,(sizeof(t1)));

}

if(count==0)

cout<<"\nRecord not found";

f.close();

break;

case 4:

cout<<"\nEnter the name you want to find and edit";

cin>>name;

f.open("StuRecord.txt",ios::in|ios::out);

f.read((char\*)&t1,(sizeof(t1)));

while(f)

{

y=(strcmp(name,t1.name));

if(y==0)

{

cout<<"\nName found";

add2=f.tellg();

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

start2=f.tellg();

n2=(add2-start2)/(sizeof(t1));

f.seekp((n2-1)\*sizeof(t1),ios::beg);

t1.accept();

f.write((char\*) &t1,(sizeof(t1)));

f.close();

break;

}

f.read((char\*)&t1,(sizeof(t1)));

}

break;

case 5:

cout<<"\n\tEnter the roll number you want to modify";

cin>>on;

f.open("StuRecord.txt",ios::in|ios::out);

f.read((char\*) &t1,(sizeof(t1)));

while(f)

{

if(on==t1.rollNo)

{

cout<<"\n\tNumber found";

add3=f.tellg();

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

start3=f.tellg();

n3=(add3-start3)/(sizeof(t1));

f.seekp((n3-1)\*(sizeof(t1)),ios::beg);

t1.accept2();

f.write((char\*)&t1,(sizeof(t1)));

f.close();

break;

}

f.read((char\*)&t1,(sizeof(t1)));

}

break;

case 6:

cout<<"\nEnter the name you want to find and edit";

cin>>name2;

f.open("StuRecord.txt",ios::in|ios::out);

f.read((char\*)&t1,(sizeof(t1)));

while(f)

{

y1=(strcmp(name2,t1.name));

if(y1==0)

{

cout<<"\nName found";

add4=f.tellg();

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

start4=f.tellg();

n4=(add4-start4)/(sizeof(t1));

f.seekp((n4-1)\*sizeof(t1),ios::beg);

t1.accept3();

f.write((char\*) &t1,(sizeof(t1)));

f.close();

break;

}

f.read((char\*)&t1,(sizeof(t1)));

}

break;

case 7:

int roll;

cout<<"Please Enter the Roll No. of Student Whose Info You Want to Delete: ";

cin>>roll;

f.open("StuRecord.txt",ios::in);

g.open("temp.txt",ios::out);

f.read((char \*)&t1,sizeof(t1));

while(!f.eof())

{

if (t1.getRollNo() != roll)

g.write((char \*)&t1,sizeof(t1));

f.read((char \*)&t1,sizeof(t1));

}

cout << "The record with the roll no. " << roll << " has been deleted " << endl;

f.close();

g.close();

remove("StuRecord.txt");

rename("temp.txt","StuRecord.txt");

break;

case 8:

cout<<"\n\tThank you";

break;

}

}while(ch!=8);

}

11.OR

#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;

}

12.

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Arrays;

import java.util.Random;

public class Main

{

static int N = 10; // size of the file in disk

static int M = 5; // max items the memory buffer can hold

public static void externalSort(String fileName)

{

String tfile = "run";

int[] buffer = new int[M < N ? M : N];

try

{

FileReader fr = new FileReader(fileName);

BufferedReader br = new BufferedReader(fr);

int slices = (int) Math.ceil((double) N/M);

int i, j;

i = j = 0;

// Iterate through the elements in the file

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

{

// Read M-element chunk at a time from the file

for (j = 0; j < (M < N ? M : N); j++)

{

String t = br.readLine();

if (t != null)

buffer[j] = Integer.parseInt(t);

else

break;

}

// Sort M elements

Arrays.sort(buffer);

// Write the sorted numbers to temp file

FileWriter fw = new FileWriter(tfile + Integer.toString(i) + ".txt");

System.out.println("Run"+i+" Generated");

PrintWriter pw = new PrintWriter(fw);

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

pw.println(buffer[k]);

pw.close();

fw.close();

}

br.close();

fr.close();

// Now open each file and merge them, then write back to disk

int[] topNums = new int[slices];

BufferedReader[] brs = new BufferedReader[slices];

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

{

brs[i] = new BufferedReader(new FileReader(tfile + Integer.toString(i) + ".txt"));

String t = brs[i].readLine();

if (t != null)

topNums[i] = Integer.parseInt(t);

else

topNums[i] = Integer.MAX\_VALUE;

}

FileWriter fw = new FileWriter("externalsort.txt");

PrintWriter pw = new PrintWriter(fw);

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

{

int min = topNums[0];

int minFile = 0;

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

{

if (min > topNums[j])

{

min = topNums[j];

minFile = j;

}

}

pw.println(min);

String t = brs[minFile].readLine();

if (t != null)

topNums[minFile] = Integer.parseInt(t);

else

topNums[minFile] = Integer.MAX\_VALUE;

}

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

brs[i].close();

pw.close();

fw.close();

}

catch (FileNotFoundException e)

{

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

static String generateInput(int n)

{

String fileName = "InputData.txt";

Random rand = new Random();

try

{

FileWriter fw = new FileWriter(fileName);

PrintWriter pw = new PrintWriter(fw);

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

pw.println(rand.nextInt(101));

pw.close();

}

catch (IOException e)

{

e.printStackTrace();

}

return fileName;

}

public static void main(String[] args)

{

String fileName = generateInput(10);

System.out.println("Random Input Generated in InputData.txt");

externalSort(fileName);

System.out.println("Sorted data stored in externalsort.txt");

}

}