# Title: Hash Table Implementation

'''

Problem Statement: Consider the telephone book database of N clients. Make use of a hash table

implementation to quickly look up a client's telephone number. Make use of two collision handling

techniques and compare them using the number of comparisons required to find a set of telephone numbers.

'''

class hashtable:

    def \_\_init\_\_(self):

        self.m = int(input("enter size of hash table : "))

        self.hashTable = [None] \* self.m

        self.elecount = 0

        self.comparisms = 0

        print(self.hashTable)

    def hashFunction(self, key):

        return key % self.m

    def isfull(self):

        if self.elecount == self.m:

            return True

        else:

            return False

    def linearprobr(self, key, data):

        index = self.hashFunction(key)

        compare = 0

        while self.hashTable[index] != None:

            index = index + 1

            compare = compare + 1

            if index == self.m:

                index = 0

        self.hashTable[index] = [key, data]

        self.elecount += 1

        print("data inserted at ", index)

        print(self.hashTable)

        print("no of comparisons : ", compare)

    def getlinear(self, key, data):

        index = self.hashFunction(key)

        while self.hashTable[index] is not None:

            if self.hashTable[index] == [key, data]:

                return index

            index = (index + 1) % self.m

        return None

    def quadraticprobr(self, key, data):

        index = self.hashFunction(key)

        compare = 0

        i = 0

        while self.hashTable[index] != None:

            index = (index + i\*i) % self.m

            compare = compare + 1

            i = i + 1

        self.hashTable[index] = [key, data]

        self.elecount += 1

        print("data inserted at ", index)

        print(self.hashTable)

        print("no of comparisons : ", compare)

    def getQuadratic(self, key, data):

        index = self.hashFunction(key)

        i = 0

        while self.hashTable[index] is not None:

            if self.hashTable[index] == [key, data]:

                return index

            i = i + 1

            index = (index + i\*i) % self.m

        return None

    def insertvialinear(self, key, data):

        if self.isfull():

            print("table is full")

            return False

        index = self.hashFunction(key)

        if self.hashTable[index] == None:

            self.hashTable[index] = [key, data]

            self.elecount += 1

            print("data inserted at ", index)

            print(self.hashTable)

        else:

            print("collision occurred, applying Linear method")

            self.linearprobr(key, data)

    def insertviaQuadratic(self, key, data):

        if self.isfull():

            print("table is full")

            return False

        index = self.hashFunction(key)

        if self.hashTable[index] == None:

            self.hashTable[index] = [key, data]

            self.elecount += 1

            print("data inserted at ", index)

            print(self.hashTable)

        else:

            print("collision occurred, applying quadratic method")

            self.quadraticprobr(key, data)

def menu():

    obj = hashtable()

    ch = 0

    while ch != 3:

        print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        print("1. Linear Probe    \*")

        print("2. Quadratic Probe  \*")

        print("3.Exit")

        print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

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

        if ch == 1:

            ch2 = 0

            while ch2 != 3:

                print("\*\* 1.Insert \*\*")

                print("\*\* 2.Search \*\*")

                print("\*\* 3.Exit \*\*")

                ch2 = int(input("enter your choice : "))

                if ch2 == 1:

                    a = int(input("enter phone number : "))

                    b = str(input("enter name : "))

                    obj.insertvialinear(a, b)

                elif ch2 == 2:

                    k = int(input("enter key to be searched : "))

                    b = str(input("enter name : "))

                    f = obj.getlinear(k, b)

                    if f == None:

                        print("Key not found")

                    else:

                        print("key found at", f)

        elif ch == 2:

            ch2 = 0

            obj1 = hashtable()

            while ch2 != 3:

                print("\*\* 1.Insert \*\*")

                print("\*\* 2.Search \*\*")

                print("\*\* 3.Exit \*\*")

                ch2 = int(input("enter your choice : "))

                if ch2 == 1:

                    a = int(input("enter phone number : "))

                    b = str(input("enter name : "))

                    obj1.insertviaQuadratic(a, b)

                elif ch2 == 2:

                    k = int(input("enter key to be searched : "))

                    b = str(input("enter name : "))

                    f = obj1.getQuadratic(k, b)

                    if f == None:

                        print("Key not found")

                    else:

                        print("key found at", f)

menu()

# Title: To create ADT that implements the "set" concept.

'''

Problem Statement: Implement all the functions of a dictionary (ADT) using hashing and handle collisions

using chaining with / without replacement. Data: Set of (key, value) pairs, Keys are mapped to values, Keys

must be comparable, Keys must be unique. Standard Operations: Insert(key, value), Find(key), Delete(key)

'''

setOne=[]

setTwo=[]

def addVal(Set):

# Function to add value to 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):

# Function to delete value from 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):

# Function to search value in 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):

# Function to print size (length) of set

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

def iterator(setA):

    a = iter(setA) # iter is a built-in function

    for i in  range(0,len(setA)-1):

        print(next(a),"->",end=' ')

    print(next(a))

def intersection(setA, setB):

# Function to perform intersection of two sets

    intersectionSet = []

    for i in setA:

        if i in setB:

            intersectionSet.append(i)

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

def union(setA, setB):

# Function to perform union of two sets

    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):

# Function to perform difference of two sets

    differenceSet = []

    for i in setA:

        if i not in setB:

            differenceSet.append(i)

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

def subsetCheck(setA, setB):

# Function to check if two sets are subsets, called in subset()

    for i in setB:

        if i not in setA:

            return False

    return True

def subset(setA, setB):

# Function to print if two sets are subsets

    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():

# Function for main menu

    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 -> Iterate")

        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()

// Title: Implementation of General Tree

/\*

Problem Statement: 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;

    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;

}

// Title:Implementation of Binary Search Tree

/\*

Problem Statement:Beginning with an empty binary search tree, Construct a 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 <climits>

#include <stack>

using namespace std;

class TreeNode

{

public:

    int data;

    TreeNode \*left;

    TreeNode \*right;

    TreeNode(int data){

        left = NULL;

        right = NULL;

        this->data = data;

    }

};

class Tree{

public:

    TreeNode \*root;

    Tree(){

        root = NULL;

    }

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

    TreeNode \*create();

    void PrintInorder(TreeNode \*root);

    void PrintPostorder(TreeNode \*root);

    void PrintPreorder(TreeNode \*root);

    void search(TreeNode \*root, int key);

    int longestPath(TreeNode \*root);

    int MinInTree(TreeNode \*root);

    void Mirror(TreeNode \*root);

    void PrintInorderNonRecursive(TreeNode \*root);

    void PrintPreorderNonRecursive(TreeNode \*root);

    void PrintPostorderNonRecursive(TreeNode \*root);

    int Max(int a, int b);

    int Min(int a, int b);

};

int Tree::Max(int a, int b)

{

    int max = (a > b) ? a : b;

    return max;

}

int Tree::Min(int a, int b)

{

    int min = (a < b) ? a : b;

    return min;

}

void Tree::insert(TreeNode \*root, TreeNode \*node)

{

    if (root == NULL)

    {

        return;

    }

    if (root->data > node->data)

    {

        if (root->left == NULL)

            root->left = node;

        else

        {

            insert(root->left, node);

            return;

        }

    }

    else

    {

        if (root->right == NULL)

            root->right = node;

        else

        {

            insert(root->right, node);

            return;

        }

    }

}

TreeNode\* Tree::create()

{

    char ch;

    ch = 'y';

    do

    {

        cout << "Enter the data for the node: " << endl;

        int newdata;

        cin >> newdata;

        TreeNode \*newNode = new TreeNode(newdata);

        if (root == NULL)

            root = newNode;

        else

        {

            insert(root, newNode);

        }

        cout << "Do you want to continue?(y|n) :  ";

        cin >> ch;

    } while (ch != 'n');

    return this->root;

}

void Tree::PrintInorder(TreeNode \*root)

{

    if (root == NULL)

        return;

    PrintInorder(root->left);

    cout << root->data << "\t";

    PrintInorder(root->right);

}

void Tree::PrintPostorder(TreeNode \*root)

{

    if (root == NULL)

        return;

    PrintPostorder(root->left);

    PrintPostorder(root->right);

    cout << root->data << "\t";

}

void Tree::PrintPreorder(TreeNode \*root)

{

    if (root == NULL)

        return;

    cout << root->data << "\t";

    PrintPreorder(root->left);

    PrintPreorder(root->right);

}

void Tree::search(TreeNode \*root, int key)

{

    if (root == NULL)

    {

        cout << "key is not present\n";

        return;

    }

    if (key < root->data) search(root->left, key);

    else if (key > root->data)  search(root->right, key);

    else{

        cout << "key is present\n";

        return;

    }

}

int Tree::longestPath(TreeNode \*root)

{

    if (root == NULL)

        return 0;

    int leftPath = longestPath(root->left);

    int rightPath = longestPath(root->right);

    return Max(leftPath, rightPath) + 1;

}

int Tree::MinInTree(TreeNode \*root)

{

    if (root == NULL)

        return INT\_MAX;

    int smallestInLeft = MinInTree(root->left);

    int smallestInRight = MinInTree(root->right);

    return Min(root->data, Min(smallestInLeft, smallestInRight));

}

void Tree::Mirror(TreeNode \*root)

{

    if (root == NULL)

        return;

    TreeNode \*temp = root->left;

    root->left = root->right;

    root->right = temp;

    Mirror(root->left);

    Mirror(root->right);

}

void Tree::PrintInorderNonRecursive(TreeNode \*root)

{

    stack<TreeNode\*> s;

    TreeNode\* curr = root;

    while(curr!=NULL || s.empty() == false)

    {

        while(curr!=NULL)

        {

            s.push(curr);

            curr=curr->left;

        }

        curr=s.top();

        s.pop();

        cout<<curr->data<<"\t";

        curr=curr->right;

    }

}

void Tree::PrintPreorderNonRecursive(TreeNode \*root)

{

    if(root==NULL)

        return;

    stack<TreeNode\*> s;

    s.push(root);

    while(s.empty() == false)

    {

        TreeNode\* curr = s.top();

        cout<<curr->data<<"\t";

        s.pop();

        if(curr->right)

            s.push(curr->right);

        if(curr->left)

            s.push(curr->left);

    }

}

void Tree::PrintPostorderNonRecursive(TreeNode \*root)

{

    if(root==NULL)

        return;

    stack<TreeNode\*> s;

    TreeNode\* prev = NULL;

    while(root || !s.empty())

    {

        while(root)

        {

            s.push(root);

            root=root->left;

        }

        root = s.top();

        if(!root->right ||root->right == prev)

        {

            cout<<root->data<<"\t";

            s.pop();

            prev=root;

            root=NULL;

        }

        else

        {

            root=root->right;

        }

    }

}

int main()

{

    char ch;

    cout<<".....MENU.....\n";

    cout<<"1. Create a tree\n";

    cout<<"2. Display Tree\n";

    cout<<"3. Search a value\n";

    cout<<"4. To find the number of nodes in the Longest Path\n";

    cout<<"5. To find Node with Minimum data\n";

    cout<<"6. To Mirror the Tree\n";

    cout<<"7. Exit\n";

    cout<<"Enter Your Choice : ";

    cin >> ch;

    TreeNode \*root;

    Tree Mytree;

    while (ch != '7')

    {

        if (ch == '1')

        {

            root = Mytree.create();

        }

        else if (ch == '2')

        {

            cout<<"\nPreorder Tree(Recursive):";cout<<"\t";

            Mytree.PrintPreorder(root);cout<<endl;

            cout<<"\nPreorder Tree(Iterative):";cout<<"\t";

            Mytree.PrintPreorderNonRecursive(root);cout<<endl;

            cout<<"\nInorder Tree(Recursive):";cout<<"\t";

            Mytree.PrintInorder(root);cout<<endl;

            cout<<"\nInorder Tree(Iterative):";cout<<"\t";

            Mytree.PrintInorderNonRecursive(root);cout<<endl;

            cout<<"\nPostorder Tree(Recursive):";cout<<"\t";

            Mytree.PrintPostorder(root);cout<<endl;

            cout<<"\nPostorder Tree(Iterative):";cout<<"\t";

            Mytree.PrintPostorderNonRecursive(root);cout<<endl;

        }

        else if (ch == '3')

        {

            cout << "Enter the key that want to find in the Tree\n";

            int key;

            cin >> key;

            Mytree.search(root, key);

        }

        else if (ch == '4')

        {

            cout << Mytree.longestPath(root);

        }

        else if (ch == '5')

        {

            cout << Mytree.MinInTree(root);

        }

        else if (ch == '6')

        {

            Mytree.Mirror(root);

            cout<<"Mirroring The Inorder Tree:";

            Mytree.PrintInorder(root);cout<<endl;

            cout<<"\nMirroring The Postorder Tree:";

            Mytree.PrintPostorder(root);cout<<endl;

            cout<<"\nMirroring The Preorder Tree:";

            Mytree.PrintPreorder(root);cout<<endl;

        }

        cout<<"\n1. Create a tree\n";

        cout<<"2. Display Tree\n";

        cout<<"3. Search a value\n";

        cout<<"4. To find the number of nodes in the Longest Path\n";

        cout<<"5. To find Node with Minimum data\n";

        cout<<"6. To Mirror the Tree\n";

        cout<<"7. Exit\n";

        cout<<"Enter Your Choice : ";

        cin >> ch;

    }

    cout << "Thank You!!\n";

}

//Title:Implementation of Expression Tree from Given Prefix Expression

/\*

Problem Statement : Construct an expression tree from the given prefix expression eg. +--a\*bc/def and

traverse it using post order traversal (non recursive) and then delete the entire tree.

Aim:To make the use of Binary Tree and stack Data structure to construct the Expression Tree from the given

prefix expression.

\*/

#include <iostream>

#include <string.h>

using namespace std;

struct node

{

    char data;

    node \*left;

    node \*right;

};

class tree

{

    char prefix[20];

public:

    node \*top;

    void expression(char[]);

    void display(node \*);

    void non\_rec\_postorder(node \*);

    void del(node \*);

};

class stack1

{

    node \*data[30];

    int top;

public:

    stack1()

    {

        top = -1;

    }

    int empty()

    {

        if (top == -1)

            return 1;

        return 0;

    }

    void push(node \*p)

    {

        data[++top] = p;

    }

    node \*pop()

    {

        return (data[top--]);

    }

};

void tree::expression(char prefix[])

{

    //char c;

    stack1 s;

    node \*t1, \*t2;

    int len, i;

    len = strlen(prefix);

    for (i = len - 1; i >= 0; i--)

    {

        top = new node;

        top->left = NULL;

        top->right = NULL;

        if (isalpha(prefix[i]))

        {

            top->data = prefix[i];

            s.push(top);

        }

        else if (prefix[i] == '+' || prefix[i] == '\*' || prefix[i] == '-' || prefix[i] == '/')

        {

            t2 = s.pop();

            t1 = s.pop();

            top->data = prefix[i];

            top->left = t2;

            top->right = t1;

            s.push(top);

        }

    }

    top = s.pop();

}

void tree::display(node \*root)

{

    if (root != NULL)

    {

        cout << root->data;

        display(root->left);

        display(root->right);

    }

}

void tree::non\_rec\_postorder(node \*top)

{

    stack1 s1, s2; /\*stack s1 is being used for flag . A NULL data implies that the right subtree has not been visited \*/

    node \*T = top;

    cout << "\n";

    s1.push(T);

    while (!s1.empty())

    {

        T = s1.pop();

        s2.push(T);

        if (T->left != NULL)

            s1.push(T->left);

        if (T->right != NULL)

            s1.push(T->right);

    }

    while (!s2.empty())

    {

        top = s2.pop();

        cout << top->data;

    }

}

void tree::del(node \*node)

{

    if (node == NULL)

        return;

    /\* first delete both subtrees \*/

    del(node->left);

    del(node->right);

    /\* then delete the node \*/

    cout <<endl<<"Deleting node : " << node->data<<endl;

    free(node);

}

int main()

{

    char expr[20];

    tree t;

    cout <<"Enter prefix Expression : ";

    cin >> expr;

    cout << expr;

    t.expression(expr);

    //t.display(t.top);

    //cout<<endl;

    t.non\_rec\_postorder(t.top);

    t.del(t.top);

    // t.display(t.top);

}

/\*

Title:Implementation of Graph Data Structure

Problem Statement: Represent a given graph using adjacency matrix/list to perform DFS and using

adjacency list to perform BFS. Use the map of the area around the college as the graph. Identify the

prominent land marks as nodes and perform DFS and BFS on that.

\*/

#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;

}

/\*

Title:Implementation of Minimum Spanning Tree using Prim’s Algorithm.

Problem Statement: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 a minimum total cost. Solve the problem by

suggesting appropriate data structure

\*/

#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;

}

/\*

Title: Implementation of an Optimal Binary search tree Data Structure

Problem Statement: 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>

using namespace std;

#define MAX 10

int find(int,int);

void print(int,int);

int p[MAX],q[MAX],w[10][10],c[10][10],r[10][10],i,j,k,n,m;

char idnt[7][10];

int main()

{

    cout<<"enter a number of identifiers : ";

    cin>>n;

    cout<<"enter idntifiers : ";

    for(i=1;i<=n;i++)

    cin>>idnt[i];

    cout<<"enter success probability for identifiers : ";

    for(i=1;i<=n;i++)

        cin>>p[i];

    cout<<"enter failure probability  for identifiers : ";

        for(i=0;i<=n;i++)

        cin>>q[i];

           cout<<"\n Weight        Cost          Root \n";

        for(i=0;i<=n;i++)

        {

            w[i][i]=q[i];

            c[i][i]=r[i][i]=0;

            cout<<"\n"<<w[i][i]<<"          "<<c[i][i]<<"             "<<r[i][i];

        }

            for(i=0;i<n;i++)

            {

            j=i+1;

            w[i][j]=q[i]+q[j]+p[j];

            c[i][j]=q[i]+c[i][j-1]+c[j][j];

            r[i][j]=j;

            cout<<"\n"<<w[i][j]<<"           "<<c[i][j]<<"              "<<r[i][j];

            }

        for(m=2;m<=n;m++)

        {

            for(i=0;i<=n-m;i++)

            {

                j=i+m;

                w[i][j]=w[i][j-1]+p[i]+q[j];

                k=find(i,j);

                r[i][j]=k;

                c[i][j]=w[i][j]+c[i][k-1]+c[k][j];

                cout<<"\n"<<w[i][j]<<"            "<<c[i][j]<<"           "<<r[i][j];

            }

        }

        cout<<"\n THE FINAL OBST IS : \n ";

       print(0,n);

        return 0;

}

int find(int i,int j)

{

    int min=2000,m,l;//c[i][j];

    for(m=i+1;m<=j;m++)

        if(c[i][m-1]+c[m][j]<min)

        {

            min=c[i][m-1]+c[m][j];

            l=m;

        }

    return l;

}

void print(int i,int j)

{

    if(i<j)

        cout<<"\n"<<idnt[r[i][j]];

    else

        return;

    print(i,r[i][j]-1);

    print(r[i][j],j);

}

/\*

Title: Implementation of a Priority Queue as ADT.

Problem Statement: Consider a scenario for Hospital to cater services to different kinds of patients as

Serious (top priority), b) non-serious (medium priority), c) General Check-up (Least priority). Implement the

priority queue to cater services to the patients

\*/

#include<iostream>

#define N 20

#define SERIOUS 10

#define NONSERIOUS 5

#define CHECKUP 1

using namespace std;

string Q[N];

int Pr[N];

int r = -1, f = -1;

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 if(r == N-1) { //if there there is some elemets in Queue

            for(i=f;i<=r;i++) {

                Q[i-f] = Q[i];

                Pr[i-f] = Pr[i];

                r = r-f;

                f = 0;

                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++;

                }

            }

        }

        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;

    if(f == -1){

        cout<<"No records found\n";

        return;

    }

    for(i=f;i<=r;i++) {

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

        switch(Pr[i]) {

            case 1:

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

            break;

            case 5:

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

            break;

            case 10:

                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 = 0, p;

    while(opt != 4){

        cout<<"----- PRIORITY QUEUE -----\n";

        cout<<"1. Insert data\n2. Display data\n3. Delete data\n4. Exit\n";

        cout<<"Enter your choice: ";

        cin>>opt;

        switch(opt){

            case 1:

                cout<<"Enter patient name: ";

                cin>>data;

                cout<<"Enter priority of patient(1 - Serious, 2 - Non-serious, 3 - General checkup): ";

                cin>>p;

                switch (p){

                    case 1:

                        enqueue(data, SERIOUS);

                        break;

                    case 2:

                        enqueue(data, NONSERIOUS);

                        break;

                    case 3:

                        enqueue(data, CHECKUP);

                        break;

                    default:

                        cout<<"Enter valid priority value!\n";

                        break;

                }

                break;

            case 2:

                print();

                break;

            case 3:

                dequeue();

                break;

            case 4:

                cout<<"\*\*\*\*\* Exited \*\*\*\*\*\n";

                break;

            default:

                cout<<"Enter valid option!\n";

                break;

        }

    }

    return 0;

}

/\*

Title: Implementation of sequential file organization concept using cpp .

Problem Statement: Department maintains student information. The file contains roll number, name,

division and address. Allow users to add, delete information about students. Display information of a

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

then the system displays the student details. Use a sequential file to maintain 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("student.txt",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 0;

  }