1. **Linked list traversal**

#include <bits/stdc++.h>

using namespace std;

class Node {

public:

int data;

Node \*left;

Node \*right;

Node(int d) {

data = d;

left = NULL;

right = NULL;

}

};

class Solution {

public:

void preOrder(Node \*root) {

if( root == NULL )

return;

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

preOrder(root->left);

preOrder(root->right);

}

/\*

Node is defined as

class Node {

public:

int data;

Node \*left;

Node \*right;

Node(int d) {

data = d;

left = NULL;

right = NULL;

}

};

\*/

Node \* insert(Node \* root, int value) {

if(root==NULL) {

Node\* newNode;

newNode = (Node\*)malloc(sizeof(Node));

newNode->left = NULL;

newNode->right = NULL;

newNode->data = value;

return newNode;

}

if(value <= root->data)

root->left = insert(root->left, value);

else

root->right = insert(root->right, value);

return root;

}

};

int main() {

Solution myTree;

Node\* root = NULL;

root = myTree.insert(root, 10);

root = myTree.insert(root, 20);

root = myTree.insert(root, 1);

root = myTree.insert(root, 4);

root = myTree.insert(root, 67);

root = myTree.insert(root, 50);

myTree.preOrder(root);

return 0;

}

1. **Minimum cost**

// C++ code to find out minimum cost

// path to connect all the cities

#include <bits/stdc++.h>

using namespace std;

// Function to find out minimum valued node

// among the nodes which are not yet included in MST

int minnode(int n, int keyval[], bool mstset[]) {

int mini = numeric\_limits<int>::max();

int mini\_index;

// Loop through all the values of the nodes

// which are not yet included in MST and find

// the minimum valued one.

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

if (mstset[i] == false && keyval[i] < mini) {

mini = keyval[i], mini\_index = i;

}

}

return mini\_index;

}

// Function to find out the MST and

// the cost of the MST.

void findcost(int n, vector<vector<int>> city) {

// Array to store the parent node of a

// particular node.

int parent[n];

// Array to store key value of each node.

int keyval[n];

// Boolean Array to hold bool values whether

// a node is included in MST or not.

bool mstset[n];

// Set all the key values to infinite and

// none of the nodes is included in MST.

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

keyval[i] = numeric\_limits<int>::max();

mstset[i] = false;

}

// Start to find the MST from node 0.

// Parent of node 0 is none so set -1.

// key value or minimum cost to reach

// 0th node from 0th node is 0.

parent[0] = -1;

keyval[0] = 0;

// Find the rest n-1 nodes of MST.

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

// First find out the minimum node

// among the nodes which are not yet

// included in MST.

int u = minnode(n, keyval, mstset);

// Now the uth node is included in MST.

mstset[u] = true;

// Update the values of neighbor

// nodes of u which are not yet

// included in MST.

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

if (city[u][v] && mstset[v] == false &&

city[u][v] < keyval[v]) {

keyval[v] = city[u][v];

parent[v] = u;

}

}

}

// Find out the cost by adding

// the edge values of MST.

int cost = 0;

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

cost += city[parent[i]][i];

cout << cost << endl;

}

// Utility Program:

int main() {

// Input 1

int n1 = 5;

vector<vector<int>> city1 = {{0, 1, 2, 3, 4},

{1, 0, 5, 0, 7},

{2, 5, 0, 6, 0},

{3, 0, 6, 0, 0},

{4, 7, 0, 0, 0}};

findcost(n1, city1);

// Input 2

int n2 = 6;

vector<vector<int>> city2 = {{0, 1, 1, 100, 0, 0},

{1, 0, 1, 0, 0, 0},

{1, 1, 0, 0, 0, 0},

{100, 0, 0, 0, 2, 2},

{0, 0, 0, 2, 0, 2},

{0, 0, 0, 2, 2, 0}};

findcost(n2, city2);

return 0;

}

1. **Utility program**

#include <iostream>

#include <queue>

using namespace std ;

struct pqcmpr

{

bool operator () ( int a , int b )

{

return a > b ; // > not < , because default is greater !

}

} ;

int main ()

{

int n , k ;

cin >> n >> k ;

priority\_queue < int , vector < int > , pqcmpr > q ; // vector < int > is imp

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

{

int no ;

cin >> no ;

q.push ( no ) ;

}

int counter = 0 ;

if ( q.empty () == true )

{

counter = - 1 ;

}

else

{

while ( 1 )

{

int element1 = q.top () ;

if ( element1 >= k )

{

break ;

}

q.pop () ;

if ( q.empty () == true )

{

counter = - 1 ;

break ;

}

int element2 = q.top () ;

q.pop () ;

q.push ( element1 + 2 \* element2 ) ;

counter ++ ;

}

}

cout << counter ;

}

1. **Threaded Binary tree**

#include <iostream>

#include <cstdlib>

#define MAX\_VALUE 65536

using namespace std;

class N { //node declaration

public:

int k;

N \*l, \*r;

bool leftTh, rightTh;

};

class ThreadedBinaryTree {

private:

N \*root;

public:

ThreadedBinaryTree() { //constructor to initialize the variables

root= new N();

root->r= root->l= root;

root->leftTh = true;

root->k = MAX\_VALUE;

}

void makeEmpty() { //clear tree

root= new N();

root->r = root->l = root;

root->leftTh = true;

root->k = MAX\_VALUE;

}

void insert(int key) {

N \*p = root;

for (;;) {

if (p->k< key) { / /move to right thread

if (p->rightTh)

break;

p = p->r;

} else if (p->k > key) { // move to left thread

if (p->leftTh)

break;

p = p->l;

} else {

return;

}

}

N \*temp = new N();

temp->k = key;

temp->rightTh= temp->leftTh= true;

if (p->k < key) {

temp->r = p->r;

temp->l= p;

p->r = temp;

p->rightTh= false;

} else {

temp->r = p;

temp->l = p->l;

p->l = temp;

p->leftTh = false;

}

}

bool search(int key) {

N \*temp = root->l;

for (;;) {

if (temp->k < key) { //search in left thread

if (temp->rightTh)

return false;

temp = temp->r;

} else if (temp->k > key) { //search in right thread

if (temp->leftTh)

return false;

temp = temp->l;

} else {

return true;

}

}

}

void Delete(int key) {

N \*dest = root->l, \*p = root;

for (;;) { //find Node and its parent.

if (dest->k < key) {

if (dest->rightTh)

return;

p = dest;

dest = dest->r;

} else if (dest->k > key) {

if (dest->leftTh)

return;

p = dest;

dest = dest->l;

} else {

break;

}

}

N \*target = dest;

if (!dest->rightTh && !dest->leftTh) {

p = dest; //has two children

target = dest->l; //largest node at left child

while (!target->rightTh) {

p = target;

target = target->r;

}

dest->k= target->k; //replace mode

}

if (p->k >= target->k) { //only left child

if (target->rightTh && target->leftTh) {

p->l = target->l;

p->leftTh = true;

} else if (target->rightTh) {

N\*largest = target->l;

while (!largest->rightTh) {

largest = largest->r;

}

largest->r = p;

p->l= target->l;

} else {

N \*smallest = target->r;

while (!smallest->leftTh) {

smallest = smallest->l;

}

smallest->l = target->l;

p->l = target->r;

}

} else {//only right child

if (target->rightTh && target->leftTh) {

p->r= target->r;

p->rightTh = true;

} else if (target->rightTh) {

N \*largest = target->l;

while (!largest->rightTh) {

largest = largest->r;

}

largest->r= target->r;

p->r = target->l;

} else {

N \*smallest = target->r;

while (!smallest->leftTh) {

smallest = smallest->l;

}

smallest->l= p;

p->r= target->r;

}

}

}

void displayTree() { //print the tree

node\* curr = root;

while(curr!=NULL)

{

printf("%d ",curr->key);

if(curr->left!=NULL)

curr=curr->left;

else if(curr->is\_rchild==1)

curr=curr->right;

else

{

while(curr->right!=NULL && curr->is\_rchild==0) //right thread exists

curr=curr->right;

if(curr->right == NULL) //last node

break;

else

curr=curr->right;

}

}

}

cout<<endl;

}

};

int main() {

ThreadedBinaryTree tbt;

cout<<"ThreadedBinaryTree

";

char ch;

int c, v;

while(1) {

cout<<"1. Insert "<<endl;

cout<<"2. Delete"<<endl;

cout<<"3. Search"<<endl;

cout<<"4. Clear"<<endl;

cout<<"5. Display"<<endl;

cout<<"6. Exit"<<endl;

cout<<"Enter Your Choice: ";

cin>>c;

//perform switch operation

switch (c) {

case 1 :

cout<<"Enter integer element to insert: ";

cin>>v;

tbt.insert(v);

break;

case 2 :

cout<<"Enter integer element to delete: ";

cin>>v;

tbt.Delete(v);

break;

case 3 :

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

cin>>v;

if (tbt.search(v) == true)

cout<<"Element "<<v<<" found in the tree"<<endl;

else

cout<<"Element "<<v<<" not found in the tree"<<endl;

break;

case 4 :

cout<<"

Tree Cleared

";

tbt.makeEmpty();

break;

case 5:

cout<<"Display tree:

";

tbt.displayTree();

break;

case 6:

exit(1);

default:

cout<<"

Invalid type!

";

}

}

cout<<"

";

return 0;

}



// C++ program for the above approach

#include <bits/stdc++.h>

using namespace std;

// DSU data structure

// path compression + rank by union

class DSU {

int\* parent;

int\* rank;

public:

DSU(int n)

{

parent = new int[n];

rank = new int[n];

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

parent[i] = -1;

rank[i] = 1;

}

}

// Find function

int find(int i)

{

if (parent[i] == -1)

return i;

return parent[i] = find(parent[i]);

}

// Union function

void unite(int x, int y)

{

int s1 = find(x);

int s2 = find(y);

if (s1 != s2) {

if (rank[s1] < rank[s2]) {

parent[s1] = s2;

}

else if (rank[s1] > rank[s2]) {

parent[s2] = s1;

}

else {

parent[s2] = s1;

rank[s1] += 1;

}

}

}

};

class Graph {

vector<vector<int> > edgelist;

int V;

public:

Graph(int V) { this->V = V; }

void addEdge(int x, int y, int w)

{

edgelist.push\_back({ w, x, y });

}

void kruskals\_mst()

{

// 1. Sort all edges

sort(edgelist.begin(), edgelist.end());

// Initialize the DSU

DSU s(V);

int ans = 0;

cout << "Following are the edges in the "

"constructed MST"

<< endl;

for (auto edge : edgelist) {

int w = edge[0];

int x = edge[1];

int y = edge[2];

// Take this edge in MST if it does

// not forms a cycle

if (s.find(x) != s.find(y)) {

s.unite(x, y);

ans += w;

cout << x << " -- " << y << " == " << w

<< endl;

}

}

cout << "Minimum Cost Spanning Tree: " << ans;

}

};

// Driver's code

int main()

{

/\* Let us create following weighted graph

10

0--------1

| \ |

6| 5\ |15

| \ |

2--------3

4 \*/

Graph g(4);

g.addEdge(0, 1, 10);

g.addEdge(1, 3, 15);

g.addEdge(2, 3, 4);

g.addEdge(2, 0, 6);

g.addEdge(0, 3, 5);

// Function call

g.kruskals\_mst();

return 0;

}

1. **Employee Bio Data**

#include <bits/stdc++.h>

#define max 20

using namespace std;

struct employee {

string name;

long int code;

string designation;

int exp;

int age;

};

int num;

void showMenu();

employee emp[max], tempemp[max],

sortemp[max], sortemp1[max];

void build()

{

cout << "Build The Table\n";

cout << "Maximum Entries can be "

<< max << "\n";

cout << "Enter the number of "

<< "Entries required";

cin >> num;

if (num > 20) {

cout << "Maximum number of "

<< "Entries are 20\n";

num = 20;

}

cout << "Enter the following data:\n";

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

cout << "Name ";

cin >> emp[i].name;

cout << "Employee ID ";

cin >> emp[i].code;

cout << "Designation ";

cin >> emp[i].designation;

cout << "Experience ";

cin >> emp[i].exp;

cout << "Age ";

cin >> emp[i].age;

}

showMenu();

}

void insert()

{

if (num < max) {

int i = num;

num++;

cout << "Enter the information "

<< "of the Employee\n";

cout << "Name ";

cin >> emp[i].name;

cout << "Employee ID ";

cin >> emp[i].code;

cout << "Designation ";

cin >> emp[i].designation;

cout << "Experience ";

cin >> emp[i].exp;

cout << "Age ";

cin >> emp[i].age;

}

else {

cout << "Employee Table Full\n";

}

showMenu();

}

void deleteIndex(int i)

{

for (int j = i; j < num - 1; j++) {

emp[j].name = emp[j + 1].name;

emp[j].code = emp[j + 1].code;

emp[j].designation

= emp[j + 1].designation;

emp[j].exp = emp[j + 1].exp;

emp[j].age = emp[j + 1].age;

}

return;

}

void deleteRecord()

{

cout << "Enter the Employee ID "

<< "to Delete Record";

int code;

cin >> code;

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

if (emp[i].code == code) {

deleteIndex(i);

num--;

break;

}

}

showMenu();

}

void searchRecord()

{

cout << "Enter the Employee"

<< " ID to Search Record";

int code;

cin >> code;

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

if (emp[i].code == code) {

cout << "Name "

<< emp[i].name << "\n";

cout << "Employee ID "

<< emp[i].code << "\n";

cout << "Designation "

<< emp[i].designation << "\n";

cout << "Experience "

<< emp[i].exp << "\n";

cout << "Age "

<< emp[i].age << "\n";

break;

}

}

showMenu();

}

void showMenu()

{

cout << "-------------------------"

<< "GeeksforGeeks Employee"

<< " Management System"

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

cout << "Available Options:\n\n";

cout << "Build Table (1)\n";

cout << "Insert New Entry (2)\n";

cout << "Delete Entry (3)\n";

cout << "Search a Record (4)\n";

cout << "Exit (5)\n";

int option;

cin >> option;

if (option == 1) {

build();

}

else if (option == 2) {

insert();

}

else if (option == 3) {

deleteRecord();

}

else if (option == 4) {

searchRecord();

}

else if (option == 5) {

return;

}

else {

cout << "Expected Options"

<< " are 1/2/3/4/5";

showMenu();

}

}

int main()

{

showMenu();

return 0;

}

1. **AVL tree insertion**

// C++ program to insert a node in AVL tree

#include<bits/stdc++.h>

using namespace std;

// An AVL tree node

class Node

{

public:

int key;

Node \*left;

Node \*right;

int height;

};

// A utility function to get the

// height of the tree

int height(Node \*N)

{

if (N == NULL)

return 0;

return N->height;

}

// A utility function to get maximum

// of two integers

int max(int a, int b)

{

return (a > b)? a : b;

}

/\* Helper function that allocates a

new node with the given key and

NULL left and right pointers. \*/

Node\* newNode(int key)

{

Node\* node = new Node();

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1; // new node is initially

// added at leaf

return(node);

}

// A utility function to right

// rotate subtree rooted with y

// See the diagram given above.

Node \*rightRotate(Node \*y)

{

Node \*x = y->left;

Node \*T2 = x->right;

// Perform rotation

x->right = y;

y->left = T2;

// Update heights

y->height = max(height(y->left),

height(y->right)) + 1;

x->height = max(height(x->left),

height(x->right)) + 1;

// Return new root

return x;

}

// A utility function to left

// rotate subtree rooted with x

// See the diagram given above.

Node \*leftRotate(Node \*x)

{

Node \*y = x->right;

Node \*T2 = y->left;

// Perform rotation

y->left = x;

x->right = T2;

// Update heights

x->height = max(height(x->left),

height(x->right)) + 1;

y->height = max(height(y->left),

height(y->right)) + 1;

// Return new root

return y;

}

// Get Balance factor of node N

int getBalance(Node \*N)

{

if (N == NULL)

return 0;

return height(N->left) - height(N->right);

}

// Recursive function to insert a key

// in the subtree rooted with node and

// returns the new root of the subtree.

Node\* insert(Node\* node, int key)

{

/\* 1. Perform the normal BST insertion \*/

if (node == NULL)

return(newNode(key));

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

else // Equal keys are not allowed in BST

return node;

/\* 2. Update height of this ancestor node \*/

node->height = 1 + max(height(node->left),

height(node->right));

/\* 3. Get the balance factor of this ancestor

node to check whether this node became

unbalanced \*/

int balance = getBalance(node);

// If this node becomes unbalanced, then

// there are 4 cases

// Left Left Case

if (balance > 1 && key < node->left->key)

return rightRotate(node);

// Right Right Case

if (balance < -1 && key > node->right->key)

return leftRotate(node);

// Left Right Case

if (balance > 1 && key > node->left->key)

{

node->left = leftRotate(node->left);

return rightRotate(node);

}

// Right Left Case

if (balance < -1 && key < node->right->key)

{

node->right = rightRotate(node->right);

return leftRotate(node);

}

/\* return the (unchanged) node pointer \*/

return node;

}

// A utility function to print preorder

// traversal of the tree.

// The function also prints height

// of every node

void preOrder(Node \*root)

{

if(root != NULL)

{

cout << root->key << " ";

preOrder(root->left);

preOrder(root->right);

}

}

// Driver Code

int main()

{

Node \*root = NULL;

/\* Constructing tree given in

the above figure \*/

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 30);

root = insert(root, 40);

root = insert(root, 50);

root = insert(root, 25);

/\* The constructed AVL Tree would be

30

/ \

20 40

/ \ \

10 25 50

\*/

cout << "Preorder traversal of the "

"constructed AVL tree is \n";

preOrder(root);

return 0;

}

// This code is contributed by

// rathbhupendra

1. **Heap**

// C++ program for implement deletion in Heaps

#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is

// an index of arr[] and n is the size of heap

void heapify(int arr[], int n, int i)

{

int largest = i; // Initialize largest as root

int l = 2 \* i + 1; // left = 2\*i + 1

int r = 2 \* i + 2; // right = 2\*i + 2

// If left child is larger than root

if (l < n && arr[l] > arr[largest])

largest = l;

// If right child is larger than largest so far

if (r < n && arr[r] > arr[largest])

largest = r;

// If largest is not root

if (largest != i) {

swap(arr[i], arr[largest]);

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

// Function to delete the root from Heap

void deleteRoot(int arr[], int& n)

{

// Get the last element

int lastElement = arr[n - 1];

// Replace root with last element

arr[0] = lastElement;

// Decrease size of heap by 1

n = n - 1;

// heapify the root node

heapify(arr, n, 0);

}

void printArray(int arr[], int n)

{

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

cout << arr[i] << " ";

cout << "\n";

}

int main()

{

int arr[] = { 10, 5, 3, 2, 4 };

int n = sizeof(arr) / sizeof(arr[0]);

deleteRoot(arr, n);

printArray(arr, n);

return 0;

}

1. **BFS**

#include<stdio.h>

int a[20][20],n, visited[10],q[10],f=-1,r=-1;

void bfs(int v)

{

int i;

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

{

if(a[v][i]!=0 && visited[i]==0)

{

r=r+1;;

q[r]=i;

visited[i]=1;

printf("%d",i);

}

}

f++;

if(f<=1)

bfs(q[f]);

}

int main()

{

int i,j,v;

printf("Enter the number of vertices :");

scanf("%d",&n);

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

visited[i]=0;

printf("Enter the data of vertices :");

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

{

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

{

scanf("%d",&a[i][j]);

}}

printf("Enter the starting vertex :");

scanf("%d",&v);

printf("DFS Traversal is :");

visited[v]=1;

//printf("%d",v);

bfs(v);

}

/\*0 1 0 0 1

1 0 1 1 1

0 1 0 1 0

0 1 1 0 1

1 1 0 1 0\*/

1. **BST**

#include<bits/stdc++.h>

using namespace std;

class Node

{

public :

int data;

Node\* left = NULL;

Node\* right = NULL;

Node (int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

};

Node\* InsertIntoBst(Node\* &root, int d)

{

if(root == NULL)

{

root = new Node(d);

return root;

}

if(root->data<d)

root->right = InsertIntoBst(root->right , d);

else if (root ->data>d)

root->left = InsertIntoBst(root->left , d);

else

return root;

}

void TakeInput(Node\* root)

{

int data;

cin>>data;

while(data!=-1)

{

InsertIntoBst(root,data);

cin>>data;

}

}

void inorder(Node\*root)

{

if(root==NULL)

return;

inorder(root ->left);

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

inorder(root->right);

}

int main()

{

Node\* root = NULL;

cout<<"Enter the data to create BST :"<<endl;

TakeInput(root);

cout<<"Printing the Inorder"<<endl;

inorder(root);

return 0;

}

1. **Circular Linked List**

// C++ program for the above methods

#include <bits/stdc++.h>

using namespace std;

struct Node {

int data;

struct Node\* next;

};

struct Node\* addToEmpty(struct Node\* last, int data)

{

// This function is only for empty list

if (last != NULL)

return last;

// Creating a node dynamically.

struct Node\* temp

= (struct Node\*)malloc(sizeof(struct Node));

// Assigning the data.

temp->data = data;

last = temp;

// Creating the link.

last->next = last;

return last;

}

struct Node\* addBegin(struct Node\* last, int data)

{

if (last == NULL)

return addToEmpty(last, data);

struct Node\* temp

= (struct Node\*)malloc(sizeof(struct Node));

temp->data = data;

temp->next = last->next;

last->next = temp;

return last;

}

struct Node\* addEnd(struct Node\* last, int data)

{

if (last == NULL)

return addToEmpty(last, data);

struct Node\* temp

= (struct Node\*)malloc(sizeof(struct Node));

temp->data = data;

temp->next = last->next;

last->next = temp;

last = temp;

return last;

}

struct Node\* addAfter(struct Node\* last, int data, int item)

{

if (last == NULL)

return NULL;

struct Node \*temp, \*p;

p = last->next;

do {

if (p->data == item) {

temp

= (struct Node\*)malloc(sizeof(struct Node));

temp->data = data;

temp->next = p->next;

p->next = temp;

if (p == last)

last = temp;

return last;

}

p = p->next;

} while (p != last->next);

cout << item << " not present in the list." << endl;

return last;

}

void traverse(struct Node\* last)

{

struct Node\* p;

// If list is empty, return.

if (last == NULL) {

cout << "List is empty." << endl;

return;

}

// Pointing to first Node of the list.

p = last->next;

// Traversing the list.

do {

cout << p->data << " ";

p = p->next;

} while (p != last->next);

}

// Driver code

int main()

{

struct Node\* last = NULL;

last = addToEmpty(last, 6);

last = addBegin(last, 4);

last = addBegin(last, 2);

last = addEnd(last, 8);

last = addEnd(last, 12);

last = addAfter(last, 10, 8);

// Function call

traverse(last);

return 0;

}

1. **DFS**

#include<stdio.h>

int a[20][20],n, visited[10];

void dfs(int v)

{

int i;

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

{

if(a[v][i]!=0 && visited[i]==0)

{

visited[i]=1;

printf("%d",i);

dfs(i);

}

}

}

int main()

{

int i,j,v;

printf("Enter the number of vertices :");

scanf("%d",&n);

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

visited[i]=0;

printf("Enter the data of vertices :");

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

{

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

{

scanf("%d",&a[i][j]);

}}

printf("Enter the starting vertex :");

scanf("%d",&v);

printf("DFS Traversal is :");

visited[v]=1;

//printf("%d",v);

dfs(v);

}

/\*0 1 0 0 1

1 0 1 1 1

0 1 0 1 0

0 1 1 0 1

1 1 0 1 0\*/

1. **Dijkstras Algo**

// C program for Dijkstra's single source shortest path

// algorithm. The program is for adjacency matrix

// representation of the graph

#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

// Number of vertices in the graph

#define V 9

// A utility function to find the vertex with minimum

// distance value, from the set of vertices not yet included

// in shortest path tree

int minDistance(int dist[], bool sptSet[])

{

// Initialize min value

int min = INT\_MAX, min\_index;

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

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

// A utility function to print the constructed distance

// array

void printSolution(int dist[])

{

printf("Vertex \t\t Distance from Source\n");

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

printf("%d \t\t\t\t %d\n", i, dist[i]);

}

// Function that implements Dijkstra's single source

// shortest path algorithm for a graph represented using

// adjacency matrix representation

void dijkstra(int graph[V][V], int src)

{

int dist[V]; // The output array. dist[i] will hold the

// shortest

// distance from src to i

bool sptSet[V]; // sptSet[i] will be true if vertex i is

// included in shortest

// path tree or shortest distance from src to i is

// finalized

// Initialize all distances as INFINITE and stpSet[] as

// false

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

dist[i] = INT\_MAX, sptSet[i] = false;

// Distance of source vertex from itself is always 0

dist[src] = 0;

// Find shortest path for all vertices

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

// Pick the minimum distance vertex from the set of

// vertices not yet processed. u is always equal to

// src in the first iteration.

int u = minDistance(dist, sptSet);

// Mark the picked vertex as processed

sptSet[u] = true;

// Update dist value of the adjacent vertices of the

// picked vertex.

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

// Update dist[v] only if is not in sptSet,

// there is an edge from u to v, and total

// weight of path from src to v through u is

// smaller than current value of dist[v]

if (!sptSet[v] && graph[u][v]

&& dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

// print the constructed distance array

printSolution(dist);

}

// driver's code

int main()

{

/\* Let us create the example graph discussed above \*/

int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

// Function call

dijkstra(graph, 0);

return 0;

}

1. **Sweetness of cookie**

#include <stdio.h>

#define N 100

int k,arr[N],i,j,temp,n,sum;

void sort(int [],int n);

void dis(int [],int n);

void mitha();

int main() {

printf("Input k : ");

scanf("%d",&k);

printf("Enter the number of cookies : ");

scanf("%d",&n);

printf("Enter the sweetness of cookies : ");

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

{

scanf("%d",&arr[i]);

}

repeat:

sort(arr,n);

dis(arr,n);

mitha();

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

{

arr[i]=arr[i+1];

}

dis(arr,(n-1));

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

{

if(arr[i]<=k)

{

n=n-1;

goto repeat;

}

else

{

n=n-1;

printf("Final sweetness of all cookies : ");

dis(arr,n);

}

}

return 0;

}

void sort(int arr[],int n)

{

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

{

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

{

if(arr[i]>arr[j])

{

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

}

void dis(int arr[],int n)

{

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

{

printf(" %d",arr[i]);

}

printf("\n");

}

void mitha()

{

sum=arr[0]+arr[1];

arr[0]=sum;

}