**BELLFORD:**

**DSC:**

#include <stdio.h>

#include <iostream>

#include <fstream>

using namespace std;

#define MAX 20

struct Edge {

    int src, dest, weight;

};

struct Graph {

    int V, E;

    struct Edge\* edge;

};

struct Graph\* createGraph(int V, int E)

{

    struct Graph\* graph = new Graph;

    graph->V = V;

    graph->E = E;

    graph->edge = new Edge[E];

    return graph;

}

void printArr(int dist[], int n)

{

    printf("Vertex Distance from Src:\n");

    for (int i = 0; i < n; i++)

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

}

void BellmanFord(Graph\* graph, int src)

{

    int V = graph->V;

    int E = graph->E;

    int dist[MAX];

    for (int i = 0; i < V; i++)

    {

        dist[i] = INT\_MAX;

    }

    dist[src] = 0;

    for (int i = 1; i <= V - 1; i++)

    {

        for (int j = 0; j < E; j++)

        {

            int u = graph->edge[j].src;

            int v = graph->edge[j].dest;

            int weight = graph->edge[j].weight;

            if (dist[u] != INT\_MAX && dist[u] + weight < dist[v])

            {

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

            }

        }

    }

    for (int i = 1; i <= V - 1; i++)

    {

        for (int j = 0; j < E; j++)

        {

            int u = graph->edge[j].src;

            int v = graph->edge[j].dest;

            int weight = graph->edge[j].weight;

            if (dist[u] != INT\_MAX && dist[u] + weight < dist[v])

            {

                dist[v] = INT\_MAX;

                printf("Graph contains negative cycle\n");

                return;

            }

        }

    }

    printArr(dist, V);

}

int main()

{

    int V = 5;

    int E = 8;

    Graph\* graph = createGraph(V, E);

    graph->edge[0].src = 0;

    graph->edge[0].dest = 1;

    graph->edge[0].weight = -1;

    graph->edge[1].src = 0;

    graph->edge[1].dest = 2;

    graph->edge[1].weight = 4;

    graph->edge[2].src = 1;

    graph->edge[2].dest = 2;

    graph->edge[2].weight = 3;

    graph->edge[3].src = 1;

    graph->edge[3].dest = 3;

    graph->edge[3].weight = 2;

    graph->edge[4].src = 1;

    graph->edge[4].dest = 4;

    graph->edge[4].weight = 2;

    graph->edge[5].src = 3;

    graph->edge[5].dest = 2;

    graph->edge[5].weight = 5;

    graph->edge[6].src = 3;

    graph->edge[6].dest = 1;

    graph->edge[6].weight = 1;

    graph->edge[7].src = 4;

    graph->edge[7].dest = 3;

    graph->edge[7].weight = -3;

    BellmanFord(graph, 3);

    cin.get();

}

**MTK:**

#include <iostream>

#include <stdio.h>

#include <fstream>

#define MAX 20

//can't input matrix A from txt

using namespace std;

struct Node

{

    int info;

    Node\* next;

    Node(int x)

    {

        info = x;

        next = NULL;

    }

};

int A[MAX][MAX];

char vertex[MAX];

int n; //number of vertices

Node\* front, \*rear;

Node\* sp;

void initStack()

{

    sp = nullptr;

}

void push(int x)

{

    if (sp == NULL)

        sp = new Node(x);

    else

    {

        Node\* newNode = new Node(x);

        newNode->next = sp;

        sp = newNode;

    }

}

void pop()

{

    if (sp == nullptr)

        return;

    else

    {

        Node\* temp = sp;

        sp = sp->next;

        delete temp;

    }

}

int peekS()

{

    if (sp != NULL)

        return sp->info;

}

void initQueue()

{

    front = nullptr;

    rear = nullptr;

}

void enqueue(int x)

{

    if (front == NULL && rear == NULL)

    {

        front = new Node(x);

        rear = front;

    }

    else

    {

        rear->next = new Node(x);

        rear = rear->next;

    }

}

void dequeue()

{

    if (front == NULL && rear == NULL)

    {

        return;

    }

    else

    {

        if (front == rear)

        {

            delete front;

            front = nullptr;

            rear = nullptr;

        }

        else

        {

            Node\* temp = front;

            front = front->next;

            delete temp;

            temp = nullptr;

        }

    }

}

int peekQ()

{

    if (front != NULL && rear != NULL)

        return front->info;

}

void initGraph()

{

    n = 0;

}

void inputGraphFromTxt()

{

    string line;

    ifstream myfile("D:/OU 2ND YEAR/CTDL - TG 2/BAI TAP/CODE/26-11/mtts4.txt");

    if (myfile.is\_open())

    {

        myfile >> n;

        for (int i = 0; i < n; i++)

            myfile >> vertex[i];

        for (int i = 0; i < n; i++)

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

                myfile >> A[i][j];

        myfile.close();

    }

}

void inputGraph()

{

    cout << "Nhap so dinh cua do thi: "; cin >> n;

    for (int i = 0; i < n; i++)

    {

        cout << "Nhap ten cho dinh thu tu [" << i << "]: "; cin >> vertex[i];

        cout << "Nhap du lieu cho dong thu [" << i << "]: " << endl;

        for (int j = 0; j < n; j++)

        {

            cin >> A[i][j];

        }

    }

}

void output(int a[], int n)

{

    for (int i = 0; i < n; i++)

        cout << vertex[a[i]] << " ";

}

void outputGraph()

{

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

            cout << A[i][j] << " ";

        cout << "\n";

    }

}

//C is to check if the vertex is visited, if no then 1, yes then 0

int C[100], bfs[100];

int nbfs = 0;

void initC()

{

    for (int i = 0; i < n; i++)

    {

        C[i] = 1;

    }

}

void BFS(int v)

{

    int w, p;

    enqueue(v);

    C[v] = 0;

    while (front != NULL)

    {

        p = peekQ();

        dequeue();

        bfs[nbfs] = p;

        nbfs++;

        for (w = 0; w < n; w++)

        {

            if (C[w] && A[p][w] == 1)

            {

                enqueue(w);

                C[w] = 0;

            }

        }

    }

}

int dfs[100];

int ndfs = 0;

void DFS(int s)

{

    push(s);

    dfs[ndfs] = s;

    ndfs++;

    C[s] = 0;

    int u = s,

        v = -1;

    while (sp != nullptr)

    {

        if (v == n) //can not find any edge to the neighbour and the neighbour is not visited => have traversed to the deepest level

        {

            u = peekS(); //remember the top is always the "prev", so in other words, you are backtracking by poping into u

            pop();

            //after entering this block, the traversal is backtracking

            //so backtracking by using a stack and var tracking latest printed vertex

        }

        for (v = 0; v < n; v++)

        {

            if (A[u][v] != 0 && C[v] == 1) //v is to traverse u's neighbours and is the vertex to be printed

            {

                push(v); //so that later we can backtrack it

                push(u); //push prev of v twice so that you may revisit it twice or more

                dfs[ndfs] = v;

                ndfs++;

                C[v] = 0;

                u = v; //u keeps track of the vertex printed so far, which is at the furthest level, this line might be redundant when pushing only 1 vertex for each latest printed vertex

                break; //since you want to traverse to the next level's vertex, break makes sure you will not relax any other edge to other neighbour, since you want to keep moving to furthest level

            }

        }

    }

}

void searchByBFS(int x, int v)

{

    int w, p;

    enqueue(v);

    C[v] = 0;

    while (front != NULL)

    {

        p = peekQ();

        dequeue();

        if (x == p)

        {

            cout << "Tim thay x = " << x << endl;

            return;

        }

        for (w = 0; w < n; w++)

        {

            if (C[w] && A[p][w] == 1)

            {

                enqueue(w);

                C[w] = 0;

            }

        }

    }

}

void searchByDFS(int x, int s)

{

    push(s);

    dfs[ndfs] = s;

    ndfs++;

    C[s] = 0;

    int u = s, v = -1;

    while (sp != nullptr)

    {

        if (v == n)

        {

            u = peekS();

            pop();

        }

        if (x == u)

        {

            cout << "Tim thay x = " << x << endl;

            return;

        }

        for (v = 0; v < n; v++)

        {

            if (A[u][v] != 0 && C[v] == 1)

            {

                push(v);

                push(u);

                dfs[ndfs] = v;

                ndfs++;

                C[v] = 0;

                u = v;

                break;

            }

        }

    }

}

int dist[MAX];

int previous[MAX];

void initBell()

{

    for (int i = 0; i < MAX; i++)

    {

        dist[i] = INT\_MAX;

        previous[i] = 0;

    }

}

void BellmanFord(int src) {

    dist[src] = 0;

    for (int k = 0; k < n - 1; k++)

    {

        for (int i = 0; i < n; i++) //go through each edge of graph

        {

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

            {

                if (dist[i] != INT\_MAX && A[i][j] && dist[j] > dist[i] + A[i][j])

                {

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

                    previous[j] = i;

                }

            }

        }

    }

    for (int k = 0; k < n - 1; k++)

    {

        for (int i = 0; i < n; i++)

        {

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

            {

                if (A[i][j] && dist[j] > (dist[i] + A[i][j]))

                {

                    dist[j] = INT\_MAX;

                    previous[j] = -1;

                    cout << "Graph contains negative weight cycle" << endl;

                    return;

                }

            }

        }

    }

    cout << "Vertex: \t\t";

    for (int i = 0; i < n; i++)

        cout << vertex[i] << "\t";

    cout << "\nDistance From source: \t";

    for (int i = 0; i < n; i++)

        cout << dist[i] << "\t";

    cout << "\n";

    return;

}

int main()

{

    int option, x;

    system("cls");

    cout << "----------BIEU DIEN DO THI-------------\n";

    cout << "1. Khoi tao ma tran ke rong\n";

    cout << "2. Nhap ma tran ke tu file text\n";

    cout << "3. Nhap tay ma tran ke\n";

    cout << "4. Xuat ma tran ke\n";

    cout << "5. BFS" << endl;

    cout << "6. DFS" << endl;

    cout << "7. Search by BFS: " << endl;

    cout << "8. Search by BFS: " << endl;

    cout << "9. Duong di ngan nhat BellFord: " << endl;

    do

    {

        cout << "Nhap lua chon: "; cin >> option;

        switch (option)

        {

        case 1:

            initGraph();

            break;

        case 2:

            inputGraphFromTxt();

            break;

        case 3:

            inputGraph();

            break;

        case 4:

            outputGraph();

            break;

        case 5:

            initQueue();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            nbfs = 0;

            BFS(x);

            cout << "Thu tu dinh sau khi duyet BFS: ";

            output(bfs, n);

            cout << "\n";

            break;

        case 6:

            initStack();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            ndfs = 0;

            DFS(x);

            cout << "Thu tu dinh sau khi duyet DFS: " << endl;

            output(dfs, n);

            cout << "\n";

            break;

        case 7:

            initQueue();

            initC();

            nbfs = 0;

            cout << "Vui long nhap dinh can tim: ";

            cin >> x;

            searchByBFS(x, 0);

            cout << "\n";

            break;

        case 8:

            initStack();

            initC();

            ndfs = 0;

            cout << "Vui long nhap dinh can tim: ";

            cin >> x;

            searchByDFS(x, 0);

            cout << "\n";

            break;

        case 9:

            cout << "Vui long nhap dinh: "; cin >> x;

            initBell();

            BellmanFord(x);

            break;

        default:

            break;

        }

    } while (option <= 9 && option >= 1);

    cin.get();

}

**DSK:**

//dtk

//unfinished: haven't finished coding

#include <iostream>

#include <fstream>

#include <sstream>

#define MAX 100

using namespace std;

struct GNode

{

    int info; //neighbour

    GNode\* next;

    int weight; //weight of edge to the neighbour

    GNode(int x, int \_weight)

    {

        info = x;

        weight = \_weight;

        next = NULL;

    }

};

struct Node

{

    int info;

    Node\* next;

    Node(int x)

    {

        info = x;

        next = NULL;

    }

};

GNode\* First[MAX];

char vertex[MAX];

int n;

Node\* front, \*rear;

Node\* sp;

void initStack()

{

    sp = nullptr;

}

void push(int x)

{

    if (sp == NULL)

        sp = new Node(x);

    else

    {

        Node\* newNode = new Node(x);

        newNode->next = sp;

        sp = newNode;

    }

}

void pop()

{

    if (sp == nullptr)

        return;

    else

    {

        Node\* temp = sp;

        sp = sp->next;

        delete temp;

    }

}

int peekS()

{

    if (sp != NULL)

        return sp->info;

}

void initQueue()

{

    front = nullptr;

    rear = nullptr;

}

void enqueue(int x)

{

    if (front == NULL && rear == NULL)

    {

        front = new Node(x);

        rear = front;

    }

    else

    {

        rear->next = new Node(x);

        rear = rear->next;

    }

}

void dequeue()

{

    if (front == NULL && rear == NULL)

    {

        return;

    }

    else

    {

        if (front == rear)

        {

            delete front;

            front = nullptr;

            rear = nullptr;

        }

        else

        {

            Node\* temp = front;

            front = front->next;

            delete temp;

            temp = nullptr;

        }

    }

}

int peekQ()

{

    if (front != NULL && rear != NULL)

        return front->info;

}

void insertLast(GNode\*& first, GNode\* p)

{

    if (first == NULL)

        first = p;

    else

    {

        GNode\* tmp = first;

        while (tmp->next != NULL)

        {

            tmp = tmp->next;

        }

        tmp->next = p;

    }

}

void initGraph()

{

    n = 0;

    for (int i = 0; i < MAX; i++)

        First[i] = NULL;

}

void inputGraphFromTxt()

{

    ifstream myfile("D:/OU 2ND YEAR/CTDL - TG 2/BAI TAP/CODE/26-11/mtts4.txt");

    if (myfile.is\_open())

    {

        string str;

        myfile >> n;

        for (int i = 0; i < n; i++)

            myfile >> vertex[i];

        myfile.ignore();

        int i = 0;

        while (getline(myfile, str))

        {

            int weight, j = 0;

            istringstream sstr(str);

            while (sstr >> weight)

            {

                if (weight != 0)

                    insertLast(First[i], new GNode(j, weight));

                j++;

            }

            i++;

        }

        myfile.close();

    }

}

void inputGraph()

{

    cout << "Nhap so nut trong do thi: "; cin >> n;

    for (int i = 0; i < n; i++)

    {

        int dinhke, weight;

        cout << "Nhap ten cua nut: "; cin >> vertex[i];

        cout << "Nhap noi dung dong [" << i << "] (-1 neu muon thoat): \n";

        do

        {

            cin >> dinhke;

            cin >> weight;

            if (dinhke != -1)

                insertLast(First[i], new GNode(dinhke, weight));

        } while (dinhke != -1);

    }

}

void output(int a[], int n)

{

    for (int i = 0; i < n; i++)

    {

        cout << vertex[a[i]] << " ";

    }

}

void outputGraph()

{

    for (int i = 0; i < n; i++)

    {

        cout << "First[" << i << "] (" << vertex[i] << "): ";

        GNode\* tmp = First[i];

        while (tmp != NULL)

        {

            cout << tmp->info << " ";

            tmp = tmp->next;

        }

        cout << "\n";

    }

}

int C[100], bfs[100];

int nbfs = 0;

void initC()

{

    for (int i = 0; i < n; i++)

    {

        C[i] = 1;

    }

}

void BFS(int v)

{

    int p;

    GNode\* w;

    enqueue(v);

    C[v] = 0;

    while (front != NULL)

    {

        p = peekQ();

        dequeue();

        bfs[nbfs] = p;

        nbfs++;

        w = First[p];

        while (w != NULL)

        {

            if (C[w->info])

            {

                enqueue(w->info);

                C[w->info] = 0;

            }

            w = w->next;

        }

    }

}

int dfs[100];

int ndfs = 0;

void DFS(int s)

{

    push(s);

    dfs[ndfs] = s;

    ndfs++;

    C[s] = 0;

    int u = s;

    GNode\* v;

    while (sp != nullptr)

    {

        v = First[u];

        while (v != NULL)

        {

            if (C[v->info] == 1)

            {

                push(v->info);

                push(u);

                dfs[ndfs] = v->info;

                ndfs++;

                C[v->info] = 0;

                u = v->info;

                break;

            }

            v = v->next;

        }

        if (v == NULL)

        {

            u = peekS();

            pop();

        }

    }

}

void searchByBFS(int x, int v)

{

    int p;

    GNode\* w;

    enqueue(v);

    C[v] = 0;

    while (front != NULL)

    {

        p = peekQ();

        dequeue();

        if (x == p)

        {

            cout << "Tim thay x = " << x << endl;

            return;

        }

        w = First[p];

        while (w != NULL)

        {

            if (C[w->info])

            {

                enqueue(w->info);

                C[w->info] = 0;

            }

            w = w->next;

        }

    }

}

int dist[MAX];

int previous[MAX];

void initBell()

{

    for (int i = 0; i < MAX; i++)

    {

        dist[i] = INT\_MAX;

        previous[i] = 0;

    }

}

void BellmanFord(int src) {

    dist[src] = 0;

    for (int k = 0; k < n - 1; k++)

    {

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

            {

                GNode\* to = First[j];

                while (to != NULL)

                {

                    if (dist[j] != INT\_MAX && dist[to->info] > dist[j] + to->weight)

                    {

                        dist[to->info] = dist[j] + to->weight;

                        previous[to->info] = j;

                    }

                    to = to->next;

                }

            }

    }

    for (int k = 0; k < n - 1; k++)

    {

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

            {

                GNode\* to = First[j];

                while (to != NULL)

                {

                    if (dist[j] != INT\_MAX && dist[to->info] > dist[j] + to->weight)

                    {

                        dist[to->info] = INT\_MAX;

                        previous[to->info] = -1;

                        cout << "Graph contains negative weight cycle" << endl;

                        return;

                    }

                    to = to->next;

                }

            }

    }

    cout << "Vertex: \t\t";

    for (int i = 0; i < n; i++)

        cout << vertex[i] << "\t";

    cout << "\nDistance From source: \t";

    for (int i = 0; i < n; i++)

        cout << dist[i] << "\t";

    cout << "\n";

    return;

}

int main()

{

    int chon, x;

    cout << "1. Nhap tu file" << endl;

    cout << "2. BFS" << endl;

    cout << "3. DFS" << endl;

    cout << "4. Search by BFS: " << endl;

    cout << "5. SSSP BellFord: " << endl;

    do

    {

        cout << "Chon: ";  cin >> chon;

        switch (chon)

        {

        case 1:

            inputGraphFromTxt();

            break;

        case 2:

            initQueue();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            nbfs = 0;

            BFS(x);

            cout << "Thu tu dinh sau khi duyet BFS: ";

            output(bfs, n);

            cout << "\n";

            break;

        case 3:

            initStack();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            ndfs = 0;

            DFS(x);

            cout << "Thu tu dinh sau khi duyet DFS: " << endl;

            output(dfs, n);

            cout << "\n";

            break;

        case 4:

            initQueue();

            initC();

            nbfs = 0;

            cout << "Vui long nhap dinh can tim: ";

            cin >> x;

            searchByBFS(x, 0);

            cout << "\n";

            break;

        case 5:

            initBell();

            cout << "Vui long nhap dinh bat dau: ";

            cin >> x;

            BellmanFord(x);

            cout << "\n";

            break;

        default:

            break;

        }

    } while (chon >= 1 && chon <= 4);

    cin.get();

}

**BFS\_DFS**

**DSK:**

//dtk

#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

#define MAX 100

using namespace std;

struct Node

{

    int info;

    Node\* next;

    Node(int x)

    {

        info = x;

        next = NULL;

    }

};

Node\* First[MAX];

char vertex[MAX];

int n;

Node\* front, \*rear;

Node\* sp;

void initStack()

{

    sp = nullptr;

}

void push(int x)

{

    if (sp == NULL)

        sp = new Node(x);

    else

    {

        Node\* newNode = new Node(x);

        newNode->next = sp;

        sp = newNode;

    }

}

void pop()

{

    if (sp == nullptr)

        return;

    else

    {

        Node\* temp = sp;

        sp = sp->next;

        delete temp;

    }

}

int peekS()

{

    if (sp != NULL)

        return sp->info;

}

void initQueue()

{

    front = nullptr;

    rear = nullptr;

}

void enqueue(int x)

{

    if (front == NULL && rear == NULL)

    {

        front = new Node(x);

        rear = front;

    }

    else

    {

        rear->next = new Node(x);

        rear = rear->next;

    }

}

void dequeue()

{

    if (front == NULL && rear == NULL)

    {

        return;

    }

    else

    {

        if (front == rear)

        {

            delete front;

            front = nullptr;

            rear = nullptr;

        }

        else

        {

            Node\* temp = front;

            front = front->next;

            delete temp;

            temp = nullptr;

        }

    }

}

int peekQ()

{

    if (front != NULL && rear != NULL)

        return front->info;

}

void insertLast(Node\*& first, Node\* p)

{

    if (first == NULL)

        first = p;

    else

    {

        Node\* tmp = first;

        while (tmp->next != NULL)

        {

            tmp = tmp->next;

        }

        tmp->next = p;

    }

}

void initGraph()

{

    n = 0;

    for (int i = 0; i < MAX; i++)

        First[i] = NULL;

}

void inputGraphFromTxt()

{

    ifstream myfile("D:/OU 2ND YEAR/CTDL - TG 2/BAI TAP/CODE/12-11/12-11/dsk.txt");

    if (myfile.is\_open())

    {

        string str;

        myfile >> n;

        for (int i = 0; i < n; i++)

            myfile >> vertex[i];

        myfile.ignore();

        int i = 0;

        while (getline(myfile, str))

        {

            int u;

            istringstream sstr(str);

            while (sstr >> u)

            {

                insertLast(First[i], new Node(u));

            }

            i++;

        }

        myfile.close();

    }

}

void inputGraph()

{

    cout << "Nhap so nut trong do thi: "; cin >> n;

    for (int i = 0; i < n; i++)

    {

        cout << "Nhap ten cua nut: "; cin >> vertex[i]; cin.ignore();

        cout << "Nhap noi dung dong [" << i << "] (-1 neu muon thoat): \n";

        string str; int dinhke;

        getline(cin, str);

        istringstream sstream(str);

        while (sstream >> dinhke)

        {

            insertLast(First[i], new Node(dinhke));

        }

    }

}

void output(int a[], int n)

{

    for (int i = 0; i < n; i++)

    {

        cout << vertex[a[i]] << " ";

    }

}

void outputGraph()

{

    for (int i = 0; i < n; i++)

    {

        cout << "First[" << i << "] (" << vertex[i] << "): ";

        Node\* tmp = First[i];

        while (tmp != NULL)

        {

            cout << tmp->info << " ";

            tmp = tmp->next;

        }

        cout << "\n";

    }

}

int C[MAX]; //visited array in DS page, any vertex (pointed by the index) that is processed will be marked as 0 (basically means "Unavailable")

int bfs[MAX]; //the array is a depiction of an order in which a vertex is processed

int nbfs = 0; //basically, the actual size of bfs

void initC()

{

    for (int i = 0; i < n; i++)

    {

        C[i] = 1;

    }

}

void BFS(int v)

{

    int p;

    Node\* w;

    enqueue(v);

    C[v] = 0; //vertex v is visted

    while (front != NULL)

    {

        p = peekQ();

        dequeue();

        bfs[nbfs] = p;

        w = First[p];

        nbfs++;

        while (w != NULL)

        {

            if (C[w->info])

            {

                enqueue(w->info);

                C[w->info] = 0; //normally, this statement would be after the vertex is "processed",

            }

            w = w->next;

        }

    }

}

int dfs[MAX];

int ndfs = 0;

void DFS(int s)

{

    push(s);

    dfs[ndfs] = s;

    ndfs++;

    C[s] = 0;

    int u = s;

    Node\* v = nullptr;

    while (sp != nullptr)

    {

        if (v == NULL)

        {

            u = peekS();

            pop();

        }

        v = First[u];

        while (v != NULL)

        {

            if (C[v->info] == 1)

            {

                push(u);

                push(v->info);

                dfs[ndfs] = v->info;

                ndfs++;

                C[v->info] = 0;

                u = v->info;

                break;

            }

            v = v->next;

        }

    }

}

void searchByBFS(char x, int v)

{

    int p;

    Node\* w;

    enqueue(v);

    C[v] = 0;

    while (front != NULL)

    {

        p = peekQ();

        dequeue();

        if (x == vertex[p] )

        {

            cout << "Tim thay x = " << vertex[p] << endl;

            return;

        }

        w = First[p];

        while (w != NULL)

        {

            if (C[w->info])

            {

                enqueue(w->info);

                C[w->info] = 0;

            }

            w = w->next;

        }

    }

    cout << "Khong tim thay " << x << "\n";

}

void searchByDFS(char x, int s)

{

    push(s);

    dfs[ndfs] = s;

    ndfs++;

    C[s] = 0;

    int u = s;

    Node\* v = nullptr;

    while (sp != nullptr)

    {

        if (v == NULL)

        {

            u = peekS();

            pop();

            if (x == vertex[u])

            {

                cout << "Tim thay x = " << vertex[u] << endl;

                return;

            }

        }

        v = First[u];

        while (v != NULL)

        {

            if (C[v->info] == 1)

            {

                push(u);

                push(v->info);

                dfs[ndfs] = v->info;

                ndfs++;

                C[v->info] = 0;

                u = v->info;

                break;

            }

            v = v->next;

        }

    }

}

int main()

{

    int chon, x;

    char letter;

    cout << "1. Nhap tu file" << endl;

    cout << "2. BFS" << endl;

    cout << "3. DFS" << endl;

    cout << "4. Search by BFS: " << endl;

    cout << "5. Search by DFS: " << endl;

    cout << "6. Nhap tay" << endl;

    cout << "7. Xuat do thi" << endl;

    do

    {

        cout << "Chon: ";  cin >> chon;

        switch (chon)

        {

        case 1:

            inputGraphFromTxt();

            break;

        case 2:

            initQueue();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            nbfs = 0;

            BFS(x);

            cout << "Thu tu dinh sau khi duyet BFS: ";

            output(bfs, n);

            cout << "\n";

            break;

        case 3:

            initStack();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            ndfs = 0;

            DFS(x);

            cout << "Thu tu dinh sau khi duyet DFS: " << endl;

            output(dfs, n);

            cout << "\n";

            break;

        case 4:

            initQueue();

            initC();

            nbfs = 0;

            cout << "Vui long nhap dinh can tim: ";

            cin >> letter;

            searchByBFS(letter, 0);

            cout << "\n";

            break;

        case 5:

            initQueue();

            initC();

            nbfs = 0;

            cout << "Vui long nhap dinh can tim: ";

            cin >> letter;

            searchByDFS(letter, 0);

            cout << "\n";

            break;

        case 6:

            inputGraph();

            break;

        case 7:

            outputGraph();

            break;

        default:

            break;

        }

    } while (chon >= 1 && chon <= 7);

    cin.get();

}

**MTK:**

#include <iostream>

#include <stdio.h>

#include <fstream>

#define MAX 20

//can't input matrix A from txt

using namespace std;

struct Node

{

    int info;

    Node\* next;

    Node(int x)

    {

        info = x;

        next = NULL;

    }

};

int A[MAX][MAX];

char vertex[MAX];

int n;

Node\* front, \*rear;

Node\* sp;

void initStack()

{

    sp = nullptr;

}

void push(int x)

{

    if (sp == NULL)

        sp = new Node(x);

    else

    {

        Node\* newNode = new Node(x);

        newNode->next = sp;

        sp = newNode;

    }

}

void pop()

{

    if (sp == nullptr)

        return;

    else

    {

        Node\* temp = sp;

        sp = sp->next;

        delete temp;

    }

}

int peekS()

{

    if (sp != NULL)

        return sp->info;

}

void initQueue()

{

    front = nullptr;

    rear = nullptr;

}

void enqueue(int x)

{

    if (front == NULL && rear == NULL)

    {

        front = new Node(x);

        rear = front;

    }

    else

    {

        rear->next = new Node(x);

        rear = rear->next;

    }

}

void dequeue()

{

    if (front == NULL && rear == NULL)

    {

        return;

    }

    else

    {

        if (front == rear)

        {

            delete front;

            front = nullptr;

            rear = nullptr;

        }

        else

        {

            Node\* temp = front;

            front = front->next;

            delete temp;

            temp = nullptr;

        }

    }

}

int peekQ()

{

    if (front != NULL && rear != NULL)

        return front->info;

}

void initGraph()

{

    n = 0;

}

void inputGraphFromTxt()

{

    string line;

    ifstream myfile("D:/OU 2ND YEAR/CTDL - TG 2/BAI TAP/CODE/12-11/12-11/dsk.txt");

    if (myfile.is\_open())

    {

        myfile >> n;

        for (int i = 0; i < n; i++)

            myfile >> vertex[i];

        for (int i = 0; i < n; i++)

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

                myfile >> A[i][j];

        myfile.close();

    }

}

void inputGraph()

{

    cout << "Nhap so dinh cua do thi: "; cin >> n;

    for (int i = 0; i < n; i++)

    {

        cout << "Nhap ten cho dinh thu tu [" << i << "]: "; cin >> vertex[i];

        cout << "Nhap du lieu cho dong thu [" << i << "]: " << endl;

        for (int j = 0; j < n; j++)

        {

            cin >> A[i][j];

        }

    }

}

void output(int a[], int n)

{

    for (int i = 0; i < n; i++)

        cout << vertex[a[i]] << " ";

}

void outputGraph()

{

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

            cout << A[i][j] << " ";

        cout << "\n";

    }

}

int C[100], bfs[100];

int nbfs = 0;

void initC()

{

    for (int i = 0; i < n; i++)

    {

        C[i] = 1;

    }

}

void BFS(int v)

{

    int w, p;

    enqueue(v);

    C[v] = 0;

    while (front != NULL)

    {

        p = peekQ();

        dequeue();

        bfs[nbfs] = p;

        nbfs++;

        for (w = 0; w < n; w++)

        {

            if (C[w] && A[p][w] != 0)

            {

                enqueue(w);

                C[w] = 0;

            }

        }

    }

}

int dfs[100];

int ndfs = 0;

void DFS(int s)

{

    push(s);

    dfs[ndfs] = s;

    ndfs++;

    C[s] = 0;

    int u = s, v = -1; //u la dinh dang visit

    while (sp != nullptr)

    {

        if (v == n) //khong tim duoc them dinh ke cua u

        {

            u = peekS(); //stack top luc nao cung la dinh di truoc, nen pop no ra la bat dau backtrack

            pop();

        }

        for (v = 0; v < n; v++)

        {

            if (A[u][v] != 0 && C[v] == 1)

            {

                push(v); //de backtrack

                push(u); //de backtrack 1 dinh hon 2 lan

                dfs[ndfs] = v;

                ndfs++;

                C[v] = 0;

                u = v; //dinh tiep theo can visit la v, vi v o level tiep theo

                break; //de chi di theo huong cua v

            }

        }

    }

}

void searchByBFS(char x, int v)

{

    int w, p;

    enqueue(v);

    C[v] = 0;

    while (front != NULL)

    {

        p = peekQ();

        dequeue();

        if (x == vertex[p])

        {

            cout << "Tim thay x = " << vertex[p] << endl;

            return;

        }

        for (w = 0; w < n; w++)

        {

            if (C[w] && A[p][w] != 0)

            {

                enqueue(w);

                C[w] = 0;

            }

        }

    }

}

void searchByDFS(char x, int s)

{

    push(s);

    dfs[ndfs] = s;

    ndfs++;

    C[s] = 0;

    int u = s, v = -1;

    while (sp != nullptr)

    {

        if (v == n)

        {

            u = peekS();

            pop();

            if (x == vertex[u])

            {

                cout << "Tim thay x = " << vertex[u] << endl;

                return;

            }

        }

        for (v = 0; v < n; v++)

        {

            if (A[u][v] != 0 && C[v] == 1)

            {

                push(v);

                push(u);

                dfs[ndfs] = v;

                ndfs++;

                C[v] = 0;

                u = v;

                break;

            }

        }

    }

}

int main()

{

    int option, x;

    char letter;

    system("cls");

    cout << "----------BIEU DIEN DO THI-------------\n";

    cout << "1. Khoi tao ma tran ke rong\n";

    cout << "2. Nhap ma tran ke tu file text\n";

    cout << "3. Nhap tay ma tran ke\n";

    cout << "4. Xuat ma tran ke\n";

    cout << "5. BFS" << endl;

    cout << "6. DFS" << endl;

    cout << "7. Search by BFS: " << endl;

    cout << "8. Search by DFS: " << endl;

    do

    {

        cout << "Nhap lua chon: "; cin >> option;

        switch (option)

        {

        case 1:

            initGraph();

            break;

        case 2:

            inputGraphFromTxt();

            break;

        case 3:

            inputGraph();

            break;

        case 4:

            outputGraph();

            break;

        case 5:

            initQueue();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            nbfs = 0;

            BFS(x);

            cout << "Thu tu dinh sau khi duyet BFS: ";

            output(bfs, n);

            cout << "\n";

            break;

        case 6:

            initStack();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            ndfs = 0;

            DFS(x);

            cout << "Thu tu dinh sau khi duyet DFS: " << endl;

            output(dfs, n);

            cout << "\n";

            break;

        case 7:

            initQueue();

            initC();

            nbfs = 0;

            cout << "Vui long nhap dinh can tim: ";

            cin >> letter;

            searchByBFS(letter, 0);

            cout << "\n";

            break;

        case 8:

            initStack();

            initC();

            ndfs = 0;

            cout << "Vui long nhap dinh can tim: ";

            cin >> letter;

            searchByDFS(letter, 0);

            cout << "\n";

            break;

        default:

            break;

        }

    } while (option <= 8 && option >= 1);

    cin.get();

}

**BUCKET**

#include <iostream>

#include <iomanip>

using namespace std;

#define NARRAY 7

#define NBUCKET 6

#define INTERVAL 10

struct Node {

    int data;

    Node\* next;

    Node(int x)

    {

        data = x;

        next = NULL;

    }

};

void BucketSort(int arr[]);

struct Node\* InsertionSort(struct Node\* list);

void print(int arr[]);

void printBuckets(struct Node\* list);

int getBucketIndex(int value);

void BucketSort(int arr[])

{

    int i, j;

    Node\*\* buckets;

    buckets = new Node\*[NBUCKET];

    for (i = 0; i < NBUCKET; i++)

    {

        buckets[i] = nullptr;

    }

    for (i = 0; i < NARRAY; i++)

    {

        Node\* current;

        int pos = getBucketIndex(arr[i]);

        current = new Node(arr[i]);

        current->next = buckets[pos];

        buckets[pos] = current;

    }

    for (i = 0; i < NBUCKET; i++)

    {

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

        printBuckets(buckets[i]);

        cout << endl;

    }

    for (i = 0; i < NBUCKET; i++)

    {

        buckets[i] = InsertionSort(buckets[i]);

    }

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

    cout << "Buckets after sorted" << endl;

    for (i = 0; i < NBUCKET; i++)

    {

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

        printBuckets(buckets[i]);

        cout << endl;

    }

    for (j = 0, i = 0; i < NBUCKET; i++)

    {

        Node\* node;

        node = buckets[i];

        while (node) {

            arr[j++] = node->data;

            node = node->next;

        }

    }

    for (i = 0; i < NBUCKET; i++)

    {

        Node\* node;

        node = buckets[i];

        while (node) {

            Node\* tmp;

            tmp = node;

            node = node->next;

            delete tmp;

        }

    }

    delete[] buckets;

    return;

}

struct Node\* InsertionSort(struct Node\* list)

{

    Node\* k, \*nodeList;

    if (list == 0 || list->next == 0) {

        return list;

    }

    nodeList = list;

    k = list->next;

    nodeList->next = 0;

    while (k != 0)

    {

        Node\* ptr;

        if (nodeList->data > k->data) {

            Node\* tmp;

            tmp = k;

            k = k->next;

            tmp->next = nodeList;

            nodeList = tmp;

            continue;

        }

        for (ptr = nodeList; ptr->next != 0; ptr = ptr->next)

        {

            if (ptr->next->data > k->data)

                break;

        }

        if (ptr->next != 0)

        {

            Node\* tmp;

            tmp = k;

            k = k->next;

            tmp->next = ptr->next;

            ptr->next = tmp;

            continue;

        }

        else {

            ptr->next = k;

            k = k->next;

            ptr->next->next = 0;

            continue;

        }

    }

    return nodeList;

}

void print(int arr[])

{

    int i;

    for (i = 0; i < NARRAY; i++)

    {

        cout << setw(3) << arr[i];

    }

    cout << endl;

}

void printBuckets(struct Node\* list) {

    Node\* cur = list;

    while (cur) {

        cout << setw(3) << cur->data;

        cur = cur->next;

    }

}

int getBucketIndex(int value) {

    return value / INTERVAL;

}

int main()

{

    int array[NARRAY] = { 42, 32, 33, 52, 37, 47, 51 };

    cout << "Initial array: " << endl;

    print(array);

    cout << "========================" << endl;

    BucketSort(array);

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

    cout << "Sorted array: " << endl;

    print(array);

    cin.get();

}

**DIJKSTRA**

**MTK:**

#include <iostream>

#include <stdio.h>

#include <fstream>

#define MAX 20

//can't input matrix A from txt

using namespace std;

struct Node

{

    int info;

    Node\* next;

    Node(int x)

    {

        info = x;

        next = NULL;

    }

};

int A[MAX][MAX];

char vertex[MAX];

int n;

Node\* front, \*rear;

Node\* sp;

void initStack()

{

    sp = nullptr;

}

void push(int x)

{

    if (sp == NULL)

        sp = new Node(x);

    else

    {

        Node\* newNode = new Node(x);

        newNode->next = sp;

        sp = newNode;

    }

}

void pop()

{

    if (sp == nullptr)

        return;

    else

    {

        Node\* temp = sp;

        sp = sp->next;

        delete temp;

    }

}

int peekS()

{

    if (sp != NULL)

        return sp->info;

}

void initQueue()

{

    front = nullptr;

    rear = nullptr;

}

void enqueue(int x)

{

    if (front == NULL && rear == NULL)

    {

        front = new Node(x);

        rear = front;

    }

    else

    {

        rear->next = new Node(x);

        rear = rear->next;

    }

}

void dequeue()

{

    if (front == NULL && rear == NULL)

    {

        return;

    }

    else

    {

        if (front == rear)

        {

            delete front;

            front = nullptr;

            rear = nullptr;

        }

        else

        {

            Node\* temp = front;

            front = front->next;

            delete temp;

            temp = nullptr;

        }

    }

}

int peekQ()

{

    if (front != NULL && rear != NULL)

        return front->info;

}

void initGraph()

{

    n = 0;

}

void inputGraphFromTxt()

{

    string line;

    ifstream myfile("C:/Users/admin/Downloads/26-11/mtts5.txt");

    if (myfile.is\_open())

    {

        myfile >> n;

        for (int i = 0; i < n; i++)

            myfile >> vertex[i];

        for (int i = 0; i < n; i++)

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

                myfile >> A[i][j];

        myfile.close();

    }

}

void inputGraph()

{

    cout << "Nhap so dinh cua do thi: "; cin >> n;

    for (int i = 0; i < n; i++)

    {

        cout << "Nhap ten cho dinh thu tu [" << i << "]: "; cin >> vertex[i];

        cout << "Nhap du lieu cho dong thu [" << i << "]: " << endl;

        for (int j = 0; j < n; j++)

        {

            cin >> A[i][j];

        }

    }

}

void output(int a[], int n)

{

    for (int i = 0; i < n; i++)

        cout << vertex[a[i]] << " ";

}

void outputGraph()

{

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

            cout << A[i][j] << " ";

        cout << "\n";

    }

}

int C[100], bfs[100];

int nbfs = 0;

void initC()

{

    for (int i = 0; i < n; i++)

    {

        C[i] = 1;

    }

}

void BFS(int v)

{

    int w, p;

    enqueue(v);

    C[v] = 0;

    while (front != NULL) //check q rổng

    {

        p = peekQ();

        dequeue();

        bfs[nbfs] = p;

        nbfs++;

        for (w = 0; w < n; w++)

        {

            if (C[w] && A[p][w] != 1)

            {

                enqueue(w);

                C[w] = 0;

            }

        }

    }

}

int dfs[100];

int ndfs = 0;

void DFS(int s)

{

    push(s);

    dfs[ndfs] = s;

    ndfs++;

    C[s] = 0;

    int u = s, v = -1;

    while (sp != nullptr)

    {

        if (v == n)

        {

            u = peekS();

            pop();

        }

        for (v = 0; v < n; v++)

        {

            if (A[u][v] != 0 && C[v] == 1)

            {

                push(v);

                push(u);

                dfs[ndfs] = v;

                ndfs++;

                C[v] = 0;

                u = v;

                break;

            }

        }

    }

}

void searchByBFS(int x, int v)

{

    int w, p;

    enqueue(v);

    C[v] = 0;

    while (front != NULL)

    {

        p = peekQ();

        dequeue();

        if (x == p)

        {

            cout << "Tim thay x = " << x << endl;

            return;

        }

        for (w = 0; w < n; w++)

        {

            if (C[w] && A[p][w] == 1)

            {

                enqueue(w);

                C[w] = 0;

            }

        }

    }

}

void searchByDFS(int x, int s)

{

    push(s);

    dfs[ndfs] = s;

    ndfs++;

    C[s] = 0;

    int u = s, v = -1;

    while (sp != nullptr)

    {

        if (v == n)

        {

            u = peekS();

            pop();

        }

        if (x == u)

        {

            cout << "Tim thay x = " << x << endl;

            return;

        }

        for (v = 0; v < n; v++)

        {

            if (A[u][v] != 0 && C[v] == 1)

            {

                push(v);

                push(u);

                dfs[ndfs] = v;

                ndfs++;

                C[v] = 0;

                u = v;

                break;

            }

        }

    }

}

int dist[MAX];

int previous[MAX];

void initBell()

{

    for (int i = 0; i < MAX; i++)

    {

        dist[i] = INT\_MAX;

        previous[i] = 0;

    }

}

void BellmanFord(int src) {

    dist[src] = 0;

    for (int k = 0; k < n - 1; k++)

    {

        for (int i = 0; i < n; i++)

        {

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

            {

                if (dist[i] != INT\_MAX && A[i][j] && dist[j] > (dist[i] + A[i][j]))

                {

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

                    previous[j] = i;

                }

            }

        }

    }

    for (int k = 0; k < n - 1; k++)

    {

        for (int i = 0; i < n; i++)

        {

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

            {

                if (A[i][j] && dist[j] > (dist[i] + A[i][j]))

                {

                    dist[j] = INT\_MAX;

                    previous[j] = -1;

                    cout << "Graph contains negative weight cycle" << endl;

                    return;

                }

            }

        }

    }

    cout << "Vertex: \t\t";

    for (int i = 0; i < n; i++)

        cout << vertex[i] << "\t";

    cout << "\nDistance From source: \t";

    for (int i = 0; i < n; i++)

        cout << dist[i] << "\t";

    cout << "\n";

    return;

}

int minimumDist(int dist[], bool Tset[])

{

    int min = INT\_MAX, index;

    for (int i = 0; i < 6; i++)

    {

        if (Tset[i] == false && dist[i] <= min)

        {

            min = dist[i];

            index = i;

        }

    }

    return index;

}

void Dijkstra(int src)

{

    int dist[MAX];

    dist[src] = 0;

    bool Tset[MAX]; //visited array

    for (int i = 0; i < n; i++)

    {

        dist[i] = INT\_MAX;

        Tset[i] = false;

    }

    dist[src] = 0;

    //priority queue is replaced with minimumDist() and dist[]

    for (int i = 0; i < n; i++)

    {

        int m = minimumDist(dist, Tset);

        Tset[m] = true;

        for (int i = 0; i < n; i++)

        {

            if (!Tset[i] && A[m][i] && dist[m] != INT\_MAX && dist[m] + A[m][i] < dist[i])

                dist[i] = dist[m] + A[m][i];

        }

    }

    cout << "Vertex\t\tDistance from source" << endl;

    for (int i = 0; i < n; i++)

    {

        cout << vertex[i] << "\t\t\t" << dist[i] << endl;

    }

}

int main()

{

    int option, x;

    system("cls");

    cout << "----------BIEU DIEN DO THI-------------\n";

    cout << "1. Khoi tao ma tran ke rong\n";

    cout << "2. Nhap ma tran ke tu file text\n";

    cout << "3. Nhap tay ma tran ke\n";

    cout << "4. Xuat ma tran ke\n";

    cout << "5. BFS" << endl;

    cout << "6. DFS" << endl;

    cout << "7. Search by BFS: " << endl;

    cout << "8. Search by BFS: " << endl;

    cout << "9. Duong di ngan nhat BellFord: " << endl;

    cout << "10. Duong di ngan nhat Dijkstra: " << endl;

    do

    {

        cout << "Nhap lua chon: "; cin >> option;

        switch (option)

        {

        case 1:

            initGraph();

            break;

        case 2:

            inputGraphFromTxt();

            break;

        case 3:

            inputGraph();

            break;

        case 4:

            outputGraph();

            break;

        case 5:

            initQueue();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            nbfs = 0;

            BFS(x);

            cout << "Thu tu dinh sau khi duyet BFS: ";

            output(bfs, n);

            cout << "\n";

            break;

        case 6:

            initStack();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            ndfs = 0;

            DFS(x);

            cout << "Thu tu dinh sau khi duyet DFS: " << endl;

            output(dfs, n);

            cout << "\n";

            break;

        case 7:

            initQueue();

            initC();

            nbfs = 0;

            cout << "Vui long nhap dinh can tim: ";

            cin >> x;

            searchByBFS(x, 0);

            cout << "\n";

            break;

        case 8:

            initStack();

            initC();

            ndfs = 0;

            cout << "Vui long nhap dinh can tim: ";

            cin >> x;

            searchByDFS(x, 0);

            cout << "\n";

            break;

        case 9:

            cout << "Vui long nhap dinh: "; cin >> x;

            initBell();

            BellmanFord(x);

            break;

        case 10:

            cout << "Vui long nhap dinh: "; cin >> x;

            Dijkstra(x);

            break;

        default:

            break;

        }

    } while (option <= 10 && option >= 1);

    cin.get();

}

**HEAPSORT**

#include <stdio.h>

#include <iostream>

#include <ctime>

#define MAX 100000

int a[MAX];

int n = 0;

using namespace std;

void swap(int& a, int& b)

{

    int tmp = a;

    a = b;

    b = tmp;

}

void init()

{

    do {

        cout << "Nhap so phan tu: "; cin >> n;

        if (n < 1 || n > MAX)

        {

            cout << "Nhap so n < " << MAX << endl;

        }

    } while (n < 1 || n > MAX);

    srand(time(0));

    for (int i = 0; i < n; i++)

    {

        a[i] = rand() % 20 + 1;

    }

    cout << "Mang sau khi nhap ngau nhien: " << endl;

    for (int i = 0; i < n; i++)

    {

        cout << a[i] << " ";

    }

}

void input()

{

    do {

        cout << "Nhap so phan tu: "; cin >> n;

        if (n < 1 || n > MAX)

        {

            cout << "Nhap so n < " << MAX << endl;

        }

    } while (n < 1 || n > MAX);

    for (int i = 0; i < n; i++)

    {

        cout << "Nhap phan tu a[" << i << "]: "; cin >> a[i];

    }

    cout << "Mang sau khi nhap: " << endl;

    for (int i = 0; i < n; i++)

    {

        cout << a[i] << " ";

    }

}

void output()

{

    if (n == 0) return;

    for (int i = 0; i < n; i++)

    {

        cout << a[i] << " ";

    }

}

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

{

    int largest = i;

    int l = 2 \* i + 1;

    int r = 2 \* i + 2;

    if (l < n && a[l] > a[largest])

    {

        largest = l;

    }

    if (r < n && a[r] > a[largest])

    {

        largest = r;

    }

    if (largest != i)

    {

        swap(a[i], a[largest]);

        heapify(a, n, largest); //largest gio co the khong theo luat cua heap nen goi tiep heapify cho largest

    }

}

//Heap sort that works:

void heapSort2()

{

    for (int i = n / 2 - 1; i >= 0; i--) //tao heap lan dau tien khi sap xep

    {

        heapify(a, n, i);

    }

    for (int i = n - 1; i >= 0; i--)

    {

        swap(a[0], a[i]);

        heapify(a, i, 0); //tao heap cac lan tiep theo

    }

}

void shift(int a[], int i, int n)

{

    int j = 2 \* i + 1;

    if (j >= n)

    {

        return;

    }

    if (j + 1 < n)

        if (a[j] < a[j + 1])

            j++;

    if (a[i] >= a[j])

        return;

    else

    {

        int x = a[i];

        a[i] = a[j];

        a[j] = x;

        shift(a, j, n);

    }

}

void HeapSort(int a[], int n)

{

    int i = (n / 2) - 1;

    while (i >= 0)

    {

        shift(a, i, n);

        i--;

    }

    int r = n - 1;

    while (r > 0)

    {

        swap(a[0], a[r]);

        r--;

        if (r > 0)

            shift(a, 0, r);

    }

}

void selectionSort(int a[], int n)

{

    int i, j, min\_pos;

    for (i = 0; i < n - 1; i++)

    {

        min\_pos = i;

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

        {

            if (a[min\_pos] > a[j])

            {

                min\_pos = j;

                swap(a[min\_pos], a[j]);

            }

        }

    }

}

void insertionSort(int a[], int n)

{

    for (int i = 1; i < n; i++)

    {

        int x = a[i];

        int j = i - 1;

        while (j >= 0 && a[j] > x)

        {

            a[j + 1] = a[j];

            j--;

        }

        a[j] = x;

    }

}

void copyArray(int a[], int b[], int n)

{

    for (int i = 0; i < n; i++)

    {

        b[i] = a[i];

    }

}

int main()

{

    int choice;

    clock\_t start;

    double duration;

    int b[MAX];

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

    cout << "1. Nhap mang ngau nhien." << endl;

    cout << "2. Tu nhap mang " << endl;

    cout << "3. Xuat mang" << endl;

    cout << "4. Heap Sort 1" << endl;

    cout << "5. Selection Sort" << endl;

    cout << "6. Insertion Sort" << endl;

    cout << "7. Heap Sort 2" << endl;

    cout << "9. Thoat" << endl;

    do {

        cout << "Chon: "; cin >> choice;

        switch (choice)

        {

        case 1:

            init();

            cout << endl;

            break;

        case 2:

            input();

            cout << endl;

            break;

        case 3:

            output();

            cout << endl;

            break;

        case 4:

            HeapSort(b, n);

            break;

        case 5:

            copyArray(a, b, n);

            start = clock();

            selectionSort(b, n);

            duration = (clock() - start) / (double)CLOCKS\_PER\_SEC;

            cout << "Selection sort ton: " << duration \* pow(10.0, 6.0) << "ms" << endl;

            break;

        case 6:

            copyArray(a, b, n);

            start = clock();

            insertionSort(b, n);

            duration = (clock() - start) / (double)CLOCKS\_PER\_SEC;

            cout << "Insertion sort ton: " << duration \* pow(10.0, 6.0) << "ms" << endl;

            break;

        case 7:

            copyArray(a, b, n);

            start = clock();

            heapSort2();

            duration = (clock() - start) / (double)CLOCKS\_PER\_SEC;

            cout << "Heap sort ton: " << duration \* pow(10.0, 6.0) << "ms" << endl;

            break;

        default:

            break;

        }

    } while (choice != 9);

    system("pause");

}

**QUICKSORT**

#include <stdio.h>

#include <iostream>

#include <ctime>

#define MAX 100000

int a[MAX];

int n = 0;

using namespace std;

void swap(int& a, int& b)

{

    int tmp = a;

    a = b;

    b = tmp;

}

void init()

{

    do {

        cout << "Nhap so phan tu: "; cin >> n;

        if (n < 1 || n > MAX)

        {

            cout << "Nhap so n < " << MAX << endl;

        }

    } while (n < 1 || n > MAX);

    srand(time(0));

    for (int i = 0; i < n; i++)

    {

        a[i] = rand() % 20 + 1;

    }

    cout << "Mang sau khi nhap ngau nhien: " << endl;

    for (int i = 0; i < n; i++)

    {

        cout << a[i] << " ";

    }

}

void input()

{

    do {

        cout << "Nhap so phan tu: "; cin >> n;

        if (n < 1 || n > MAX)

        {

            cout << "Nhap so n < " << MAX << endl;

        }

    } while (n < 1 || n > MAX);

    for (int i = 0; i < n; i++)

    {

        cout << "Nhap phan tu a[" << i << "]: "; cin >> a[i];

    }

    cout << "Mang sau khi nhap: " << endl;

    for (int i = 0; i < n; i++)

    {

        cout << a[i] << " ";

    }

}

void output()

{

    if (n == 0) return;

    for (int i = 0; i < n; i++)

    {

        cout << a[i] << " ";

    }

}

void QuickSort1(int a[], int l, int r)

{

    int x = a[(l + r) / 2];

    int i = l;

    int j = r;

    while (i < j)

    {

        while (a[i] < x)

        {

            i++;

        }

        while (a[j] > x)

        {

            j--;

        }

        if (i <= j)

        {

            swap(a[i], a[j]);

            i++;

            j--;

        }

    }

    if (l < j) QuickSort1(a, l, j);

    if (i < r) QuickSort1(a, i, r);

}

int partition(int a[], int low, int high)

{

    int pivot = a[high];

    int left = low;

    int right = high - 1;

    while (true)

    {

        while (left <= right && a[left] < pivot) left++;

        while (right >= left && a[right] > pivot) right--;

        if (left >= right)

            break;

        swap(a[left], a[right]);

        left++;

        right--;

    }

    swap(a[left], a[high]);

    return left;

}

void QuickSort2(int a[], int low, int high)

{

    if (low < high)

    {

        int pi = partition(a, low, high);

        QuickSort2(a, low, pi - 1);

        QuickSort2(a, pi + 1, high);

    }

}

int Partition(int A[], int start, int end)

{

    int pivot = end;

    int pindex = start;

    for (int i = start; i < end; i++)

    {

        if (A[i] <= A[pivot])

        {

            swap(A[i], A[pindex]);

            pindex++;

        }

    }

    swap(A[pivot], A[pindex]);

    return pindex;

}

void QuickSort3(int a[], int l, int r)

{

    if (l < r)

    {

        int pivot = Partition(a, l, r);

        QuickSort3(a, l, pivot - 1);

        QuickSort3(a, pivot + 1, r);

    }

}

void selectionSort(int a[], int n)

{

    int i, j, min\_pos;

    for (i = 0; i < n - 1; i++)

    {

        min\_pos = i;

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

        {

            if (a[min\_pos] > a[j])

            {

                min\_pos = j;

                swap(a[min\_pos], a[j]);

            }

        }

    }

}

void insertionSort(int a[], int n)

{

    for (int i = 1; i < n; i++)

    {

        int x = a[i];

        int j = i - 1;

        while (j >= 0 && a[j] > x)

        {

            a[j + 1] = a[j];

            j--;

        }

        a[j] = x;

    }

}

void copyArray(int a[], int b[], int n)

{

    for (int i = 0; i < n; i++)

    {

        b[i] = a[i];

    }

}

int main()

{

    int choice;

    clock\_t start;

    double duration;

    int b[MAX];

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

    cout << "1. Nhap mang ngau nhien." << endl;

    cout << "2. Tu nhap mang " << endl;

    cout << "3. Xuat mang" << endl;

    cout << "4. Quick Sort 1" << endl;

    cout << "5. Selection Sort" << endl;

    cout << "6. Insertion Sort" << endl;

    cout << "7. Quick Sort 2" << endl;

    cout << "8. Quick Sort 3" << endl;

    cout << "9. Thoat" << endl;

    do {

        cout << "Chon: "; cin >> choice;

        switch (choice)

        {

        case 1:

            init();

            cout << endl;

            break;

        case 2:

            input();

            cout << endl;

            break;

        case 3:

            output();

            cout << endl;

            break;

        case 4:

            copyArray(a, b, n);

            start = clock();

            QuickSort1(a, 0, n - 1);

            duration = (clock() - start) / (double)CLOCKS\_PER\_SEC;

            cout << "Quick sort 1 ton: " << duration \* pow(10.0, 6.0) << "ms" << endl;

            break;

        case 5:

            copyArray(a, b, n);

            start = clock();

            selectionSort(b, n);

            duration = (clock() - start) / (double)CLOCKS\_PER\_SEC;

            cout << "Selection sort ton: " << duration \* pow(10.0, 6.0) << "ms" << endl;

            break;

        case 6:

            copyArray(a, b, n);

            start = clock();

            insertionSort(b, n);

            duration = (clock() - start) / (double)CLOCKS\_PER\_SEC;

            cout << "Insertion sort ton: " << duration \* pow(10.0, 6.0) << "ms" << endl;

            break;

        case 7:

            copyArray(a, b, n);

            start = clock();

            QuickSort2(a, 0, n - 1);

            duration = (clock() - start) / (double)CLOCKS\_PER\_SEC;

            cout << "Quick sort 2 ton: " << duration \* pow(10.0, 6.0) << "ms" << endl;

            break;

        case 8:

            copyArray(a, b, n);

            start = clock();

            QuickSort3(a, 0, n - 1);

            duration = (clock() - start) / (double)CLOCKS\_PER\_SEC;

            cout << "Quick sort 3 ton: " << duration \* pow(10.0, 6.0) << "ms" << endl;

            break;

        default:

            break;

        }

    } while (choice != 9);

    system("pause");

}

**KRUSKAL**

#include <iostream>

#include <stdio.h>

#include <fstream>

#define MAX 20

//can't input matrix A from txt

using namespace std;

struct Node

{

    int info;

    Node\* next;

    Node(int x)

    {

        info = x;

        next = NULL;

    }

};

int A[MAX][MAX];

char vertex[MAX];

int n;

Node\* front, \*rear;

Node\* sp;

void initStack()

{

    sp = nullptr;

}

void push(int x)

{

    if (sp == NULL)

        sp = new Node(x);

    else

    {

        Node\* newNode = new Node(x);

        newNode->next = sp;

        sp = newNode;

    }

}

void pop()

{

    if (sp == nullptr)

        return;

    else

    {

        Node\* temp = sp;

        sp = sp->next;

        delete temp;

    }

}

int peekS()

{

    if (sp != NULL)

        return sp->info;

}

void initQueue()

{

    front = nullptr;

    rear = nullptr;

}

void enqueue(int x)

{

    if (front == NULL && rear == NULL)

    {

        front = new Node(x);

        rear = front;

    }

    else

    {

        rear->next = new Node(x);

        rear = rear->next;

    }

}

void dequeue()

{

    if (front == NULL && rear == NULL)

    {

        return;

    }

    else

    {

        if (front == rear)

        {

            delete front;

            front = nullptr;

            rear = nullptr;

        }

        else

        {

            Node\* temp = front;

            front = front->next;

            delete temp;

            temp = nullptr;

        }

    }

}

int peekQ()

{

    if (front != NULL && rear != NULL)

        return front->info;

}

void initGraph()

{

    n = 0;

}

void inputGraphFromTxt()

{

    string line;

    ifstream myfile("C:/Users/admin/Downloads/mtts4.txt");

    if (myfile.is\_open())

    {

        myfile >> n;

        for (int i = 0; i < n; i++)

            myfile >> vertex[i];

        for (int i = 0; i < n; i++)

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

                myfile >> A[i][j];

        myfile.close();

    }

}

void inputGraph()

{

    cout << "Nhap so dinh cua do thi: "; cin >> n;

    for (int i = 0; i < n; i++)

    {

        cout << "Nhap ten cho dinh thu tu [" << i << "]: "; cin >> vertex[i];

        cout << "Nhap du lieu cho dong thu [" << i << "]: " << endl;

        for (int j = 0; j < n; j++)

        {

            cin >> A[i][j];

        }

    }

}

void output(int a[], int n)

{

    for (int i = 0; i < n; i++)

        cout << vertex[a[i]] << " ";

}

void outputGraph()

{

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

            cout << A[i][j] << " ";

        cout << "\n";

    }

}

int C[100], bfs[100];

int nbfs = 0;

void initC()

{

    for (int i = 0; i < n; i++)

    {

        C[i] = 1;

    }

}

void BFS(int v)

{

    int w, p;

    enqueue(v);

    C[v] = 0;

    while (front != NULL) //check q rổng

    {

        p = peekQ();

        dequeue();

        bfs[nbfs] = p;

        nbfs++;

        for (w = 0; w < n; w++)

        {

            if (C[w] && A[p][w] != 1)

            {

                enqueue(w);

                C[w] = 0;

            }

        }

    }

}

int dfs[100];

int ndfs = 0;

void DFS(int s)

{

    push(s);

    dfs[ndfs] = s;

    ndfs++;

    C[s] = 0;

    int u = s, v = -1;

    while (sp != nullptr)

    {

        if (v == n)

        {

            u = peekS();

            pop();

        }

        for (v = 0; v < n; v++)

        {

            if (A[u][v] != 0 && C[v] == 1)

            {

                push(v);

                push(u);

                dfs[ndfs] = v;

                ndfs++;

                C[v] = 0;

                u = v;

                break;

            }

        }

    }

}

void searchByBFS(int x, int v)

{

    int w, p;

    enqueue(v);

    C[v] = 0;

    while (front != NULL)

    {

        p = peekQ();

        dequeue();

        if (x == p)

        {

            cout << "Tim thay x = " << x << endl;

            return;

        }

        for (w = 0; w < n; w++)

        {

            if (C[w] && A[p][w] == 1)

            {

                enqueue(w);

                C[w] = 0;

            }

        }

    }

}

void searchByDFS(int x, int s)

{

    push(s);

    dfs[ndfs] = s;

    ndfs++;

    C[s] = 0;

    int u = s, v = -1;

    while (sp != nullptr)

    {

        if (v == n)

        {

            u = peekS();

            pop();

        }

        if (x == u)

        {

            cout << "Tim thay x = " << x << endl;

            return;

        }

        for (v = 0; v < n; v++)

        {

            if (A[u][v] != 0 && C[v] == 1)

            {

                push(v);

                push(u);

                dfs[ndfs] = v;

                ndfs++;

                C[v] = 0;

                u = v;

                break;

            }

        }

    }

}

int dist[MAX];

int previous[MAX];

void initBell()

{

    for (int i = 0; i < MAX; i++)

    {

        dist[i] = INT\_MAX;

        previous[i] = 0;

    }

}

void BellmanFord(int src) {

    dist[src] = 0;

    for (int k = 0; k < n - 1; k++)

    {

        for (int i = 0; i < n; i++)

        {

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

            {

                if (dist[i] != INT\_MAX && A[i][j] && dist[j] > (dist[i] + A[i][j]))

                {

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

                    previous[j] = i;

                }

            }

        }

    }

    for (int k = 0; k < n - 1; k++)

    {

        for (int i = 0; i < n; i++)

        {

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

            {

                if (A[i][j] && dist[j] > (dist[i] + A[i][j]))

                {

                    dist[j] = INT\_MAX;

                    previous[j] = -1;

                    cout << "Graph contains negative weight cycle" << endl;

                    return;

                }

            }

        }

    }

    cout << "Vertex: \t\t";

    for (int i = 0; i < n; i++)

        cout << vertex[i] << "\t";

    cout << "\nDistance From source: \t";

    for (int i = 0; i < n; i++)

        cout << dist[i] << "\t";

    cout << "\n";

    return;

}

int minimumDist(int dist[], bool Tset[])

{

    int min = INT\_MAX, index;

    for (int i = 0; i < 6; i++)

    {

        if (Tset[i] == false && dist[i] <= min)

        {

            min = dist[i];

            index = i;

        }

    }

    return index;

}

void Dijkstra(int src)

{

    int dist[MAX];

    dist[src] = 0;

    for (int k = 0; k < n - 1; k++)

    {

        for (int i = 0; i < n; i++)

        {

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

            {

                if (dist[i] != INT\_MAX && A[i][j] && dist[j] >(dist[i] + A[i][j]))

                {

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

                    previous[j] = i;

                }

            }

        }

    }

    for (int k = 0; k < n - 1; k++)

    {

        for (int i = 0; i < n; i++)

        {

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

            {

                if (A[i][j] && dist[j] >(dist[i] + A[i][j]))

                {

                    dist[j] = INT\_MAX;

                    previous[j] = -1;

                    cout << "Graph contains negative weight cycle" << endl;

                    return;

                }

            }

        }

    }

    dist[src] = 0;

    bool Tset[MAX]; //visited array

    for (int i = 0; i < n; i++)

    {

        dist[i] = INT\_MAX;

        Tset[i] = false;

    }

    dist[src] = 0;

    //priority queue is replaced with minimumDist() and dist[]

    for (int i = 0; i < n; i++)

    {

        int m = minimumDist(dist, Tset);

        Tset[m] = true;

        for (int i = 0; i < n; i++)

        {

            if (!Tset[i] && A[m][i] && dist[m] != INT\_MAX && dist[m] + A[m][i] < dist[i])

                dist[i] = dist[m] + A[m][i];

        }

    }

    cout << "Vertex\t\tDistance from source" << endl;

    for (int i = 0; i < n; i++)

    {

        cout << vertex[i] << "\t\t\t" << dist[i] << endl;

    }

}

//khai bao tap E

int E1[MAX];

int E2[MAX];

int wE[MAX];

int nE = 0; //so phan tu tap E

//khai bao tap T

int T1[MAX];

int T2[MAX];

int wT[MAX];

int nT = 0; //so phan tu tap T

void TaoE()

{

    for (int i = 0; i < n; i++)

    {

        for (int j = i; j < n; j++)

        {

            if (A[i][j] != 0)

            {

                E1[nE] = i;

                E2[nE] = j;

                wE[nE] = A[i][j];

                A[i][j] = 0;

                A[j][i] = 0;

                nE++;

            }

        }

    }

}

int TonTai(int E, int T[], int nT)

{

    for (int i = 0; i < nT; i++)

    {

        if (E == T[i])

            return 1;

    }

    return 0;

}

void Swap(int &a, int &b)

{

    int temp = a;

    a = b;

    b = temp;

}

void SapXepE()

{

    for (int i = 0; i < nE - 1; i++)

    {

        for (int j = i + 1; j < nE; j++)

        {

            if (wE[i] > wE[j])

            {

                swap(wE[i], wE[j]);

                swap(E1[i], E1[j]);

                swap(E2[i], E2[j]);

            }

        }

    }

}

int connectedComponent()

{

    int visited[MAX];

    int nV = 0;

    int i = 0, j = 0;

    int nC = 0;

    while (nV < n)

    {

        for (j = 0; j < n; j++)

        {

            if (A[i][j] != 0 && visited[j] != 1)

            {

                visited[j] = 1;

                nV++;

                i = j;

                break;

            }

        }

        if (j >= n && i < n - 1)

        {

            i++;

            visited[i] = 1;

            nV++;

            nC++;

        }

        else if (j >= n && i >= n - 1)

        {

            i = 1;

            visited[i] = 1;

            nV++;

            nC++;

        }

    }

    return nC;

}

void kruskal()

{

    for (int i = 0; i < nE; i++)

    {

        //kiem tra co gay vong lap neu add canh i

        if (TonTai(E1[i], T1, nT) == 1 && TonTai(E2[i], T2, nT) == 1)

            continue;

        if (TonTai(E1[i], T2, nT) == 1 && TonTai(E2[i], T1, nT) == 1)

            continue;

        //cho canh i vao T

        T1[nT] = E1[i];

        T2[nT] = E2[i];

        wT[nT] = wE[i];

        nT++;

        if (nT == n - 1)

            break;

    }

}

void output(bool VertexName)

{

    int tong = 0;

    for (int i = 0; i < nT; i++)

    {

        if (VertexName)

            cout << endl << "(" << vertex[T1[i]] << "," << vertex[T2[i]] << ") = " << wT[i];

        else

            cout << endl << "(" << T1[i] << "," << T2[i] << ") = " << wT[i];

        tong += wT[i];

    }

    cout << "\n Tong = " << tong;

}

void outputE(bool VertexName)

{

    int tong = 0;

    for (int i = 0; i < nE; i++)

    {

        if (VertexName)

            cout << endl << "(" << vertex[E1[i]] << "," << vertex[E2[i]] << ") = " << wT[i];

        else

            cout << endl << "(" << E1[i] << "," << E2[i] << ") = " << wT[i];

        tong += wT[i];

    }

    cout << "\n Tong = " << tong;

}

int main()

{

    int option, x, i;

    system("cls");

    cout << "----------BIEU DIEN DO THI-------------\n";

    cout << "1. Khoi tao ma tran ke rong\n";

    cout << "2. Nhap ma tran ke tu file text\n";

    cout << "3. Nhap tay ma tran ke\n";

    cout << "4. Xuat ma tran ke\n";

    cout << "5. BFS" << endl;

    cout << "6. DFS" << endl;

    cout << "7. Search by BFS: " << endl;

    cout << "8. Search by BFS: " << endl;

    cout << "9. Duong di ngan nhat BellFord: " << endl;

    cout << "10. Duong di ngan nhat Dijkstra: " << endl;

    cout << "11. Tim CAY BAO TRUM TOI THIEU bang Kruskal: " << endl;

    cout << "12. In ra tap E: " << endl;

    cout << "13. Connected Graph: " << endl;

    do

    {

        cout << "Nhap lua chon: "; cin >> option;

        switch (option)

        {

        case 1:

            initGraph();

            break;

        case 2:

            inputGraphFromTxt();

            break;

        case 3:

            inputGraph();

            break;

        case 4:

            outputGraph();

            break;

        case 5:

            initQueue();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            nbfs = 0;

            BFS(x);

            cout << "Thu tu dinh sau khi duyet BFS: ";

            output(bfs, n);

            cout << "\n";

            break;

        case 6:

            initStack();

            initC();

            cout << "Vui long nhap dinh xuat phat: ";

            cin >> x;

            ndfs = 0;

            DFS(x);

            cout << "Thu tu dinh sau khi duyet DFS: " << endl;

            output(dfs, n);

            cout << "\n";

            break;

        case 7:

            initQueue();

            initC();

            nbfs = 0;

            cout << "Vui long nhap dinh can tim: ";

            cin >> x;

            searchByBFS(x, 0);

            cout << "\n";

            break;

        case 8:

            initStack();

            initC();

            ndfs = 0;

            cout << "Vui long nhap dinh can tim: ";

            cin >> x;

            searchByDFS(x, 0);

            cout << "\n";

            break;

        case 9:

            cout << "Vui long nhap dinh: "; cin >> x;

            initBell();

            BellmanFord(x);

            break;

        case 10:

            cout << "Vui long nhap dinh: "; cin >> x;

            Dijkstra(x);

            break;

        case 11:

            nT = 0;

            TaoE();

            SapXepE();

            kruskal();

            cout << "Cay bao trum toi thieu voi Kruskal: " << endl;

            output(true);

            cout << endl;

            break;

        case 12:

            outputE(true);

            break;

        case 13:

            i = connectedComponent();

            cout << "i = " << i << endl;

            break;

        default:

            break;

        }

    } while (option <= 13 && option >= 1);

    cin.get();

}

**PRIM**

#include <iostream>

#include <string>

#include <fstream>

#include <Windows.h>

#include <iomanip>

#define MAXCHOICE 9

#define MAX 20

#define W 4

#define WD 50

using namespace std;

int a[MAX][MAX], n = 0;

char vertex[MAX];

int dist[MAX];

struct Node

{

    int info;

    Node\* link;

};

Node\* first[MAX];

void initFirst()

{

    for (int i = 0; i < MAX; i++)

        first[i] = NULL;

}

void insertLast(Node\*& first, int newInfo)

{

    Node\* p = new Node;

    p->info = newInfo;

    p->link = NULL;

    if (first == NULL)

        first = p;

    else

    {

        Node\* q = first;

        while (q->link != NULL)

            q = q->link;

        q->link = p;

    }

}

void colorText(int colorCode)

{

    HANDLE h = GetStdHandle(STD\_OUTPUT\_HANDLE);

    SetConsoleTextAttribute(h, colorCode);

}

void initGraph() { n = 0; }

void initMatrix(int a[MAX][MAX], int& n)

{

    cout << "Enter n = ";

    cin >> n;

    while (n <= 0 || n > MAX)

    {

        colorText(12);

        cout << "Invalid! Again: ";

        colorText(7);

        cin >> n;

    }

    for (int i = 0; i < n; i++)

    {

        cout << "Name of vertex: ";

        cin >> vertex[i];

        for (int j = 0; j < n; j++)

            a[i][j] = rand() % 50;

    }

}

void inputMatrix(int a[MAX][MAX], int& n)

{

    cout << "Enter n = ";

    cin >> n;

    while (n <= 0 || n > MAX)

    {

        colorText(12);

        cout << "Invalid! Again: ";

        colorText(7);

        cin >> n;

    }

    for (int i = 0; i < n; i++)

    {

        cout << "Name of vertex: ";

        cin >> vertex[i];

        for (int j = 0; j < n; j++)

        {

            cout << "a[" << i + 1 << "][" << j + 1 << "] = ";

            cin >> a[i][j];

        }

    }

}

void inputMatrix\_txt(int a[MAX][MAX], int& n, string filetxt)

{

    ifstream inFile(filetxt);

    if (inFile.is\_open())

    {

        inFile >> n;

        if (n <= 0 || n > MAX)

        {

            colorText(12);

            cout << "Invalid!";

            colorText(7);

            return;

        }

        for (int i = 0; i < n; i++)

        {

            inFile.ignore();

            inFile >> vertex[i];

        }

        for (int i = 0; i < n; i++)

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

            {

                inFile.ignore();

                inFile >> a[i][j];

            }

    }

    else

    {

        colorText(12);

        cout << "The file can't be opened!\n";

        colorText(7);

    }

    inFile.close();

}

void outMatrix(int a[MAX][MAX], int n)

{

    colorText(6);

    cout << " " << setfill('=') << setw((W + 3) \* n + 1) << "=" << endl;

    for (int i = 0; i < n; i++)

        cout << setfill(' ') << " | " << setw(W) << left << vertex[i];

    cout << " | \n";

    cout << " " << setfill('=') << setw((W + 3) \* n + 1) << "=" << endl;

    colorText(7);

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            colorText(6);

            cout << " | ";

            colorText(7);

            cout << setfill(' ') << setw(W) << left << a[i][j];

        }

        colorText(6);

        cout << " | \n";

        cout << " " << setfill('-') << setw((W + 3) \* n + 1) << "-" << endl;

    }

    colorText(7);

}

void initAdList\_txt(string filetxt)

{

    ifstream inFile(filetxt);

    if (inFile.is\_open())

    {

        inFile >> n;

        if (n <= 0)

        {

            colorText(12);

            cout << "Invalid!\n";

            colorText(7);

            return;

        }

        for (int i = 0; i < n; i++)

        {

            inFile.ignore();

            inFile >> vertex[i];

        }

        initFirst();

        for (int i = 0; i < n; i++)

        {

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

            {

                inFile.ignore();

                int x;

                inFile >> x;

                if (x == 0)

                    continue;

                else

                    insertLast(first[i], j);

            }

        }

    }

    else

    {

        colorText(12);

        cout << "The file can't be opened!\n";

        colorText(7);

    }

    inFile.close();

}

void initAdList\_matrix()

{

    initFirst();

    for (int i = 0; i < n; i++)

        for (int j = 0; j < n; j++)

            if (a[i][j] == 0)

                continue;

            else

                insertLast(first[i], j);

}

void outAdList()

{

    for (int i = 0; i < n; i++)

    {

        colorText(6);

        cout << vertex[i] << ": ";

        Node\* p = new Node;

        p = first[i];

        while (p != NULL)

        {

            colorText(7);

            cout << p->info;

            colorText(8);

            cout << " -> ";

            p = p->link;

        }

        if (p == NULL)

        {

            colorText(7);

            cout << "NULL\n";

        }

        cout << endl;

    }

}

int E1[MAX];

int E2[MAX];

int wE[MAX];

int nE = 0;

int T1[MAX];

int T2[MAX];

int wT[MAX];

int nT = 0;

int tonTai(int d, int D[], int nD)

{

    for (int i = 0; i < nD; i++)

        if (D[i] == d)

            return 1;

    return 0;

}

void XoaViTriE(int i)

{

    for (int j = i; j < nE; j++)

    {

        E1[j] = E1[j + 1];

        E2[j] = E2[j + 1];

        wE[j] = wE[j + 1];

    }

    nE--;

}

void XoaCanhE(int u, int v)

{

    for(int i = 0; i < nE; i++)

        if (E1[i] == u && E2[i] == v)

        {

            XoaViTriE(i);

            break;

        }

}

void Prim(int s)

{

    //d1, d2 de chua dinh s va e cua canh co weight chua trong min

    int u = s, min, i, d1, d2;

    while (nT < n - 1)

    {

        //add canh ke cua u vao E

        for (int v = 0; v < n; v++)

            if (a[u][v] != 0)

                if (tonTai(v, T2, nT) == 0) //neu canh ke chua co trong T

                {

                    E1[nE] = u;

                    E2[nE] = v;

                    wE[nE] = a[u][v];

                    nE++;

                }

        //lay canh dau tien thoa dieu kien lam min

        for (i = 0; i < nE; i++)

            if (tonTai(E2[i], T2, nT) == 0) //neu co 1 dinh dich chua co trong T2 (1 canh chua co trong T)

            {

                min = wE[i];

                d1 = E1[i];

                d2 = E2[i];

                break;

            }

        //tim canh min that

        for (; i < nE; i++)

            if (tonTai(E2[i], T2, nT) == 0)

                if (min > wE[i])

                {

                    min = wE[i];

                    d1 = E1[i];

                    d2 = E2[i];

                }

        //add canh min vao T

        T1[nT] = d1;

        T2[nT] = d2;

        wT[nT] = a[d1][d2];

        //xoa canh min khoi ma tran ke

        a[d1][d2] = 0;

        a[d2][d1] = 0;

        nT++;

        XoaCanhE(d1, d2);

        //set dinh dang visit la dinh dich cua canh min

        u = d2;

    }

}

void outGraph(bool vertexName)

{

    int tong = 0;

    for (int i = 0; i < nT; i++)

    {

        if (vertexName)

            cout << "(" << vertex[T1[i]] << ", " << vertex[T2[i]] << ") = " << wT[i] << endl;

        else

            cout << "(" << T1[i] << ", " << T2[i] << ") = " << wT[i] << endl;

        tong += wT[i];

    }

    cout << "\tTong = " << tong << endl;

}

int main()

{

    int choice, x;

    bool cont = true;

    string filetxt;

    do {

        system("cls");

        colorText(11);

        cout << "\t+" << setfill('=') << setw(WD / 2 + 3) << "=" << " SPANNING TREE " << setw(WD / 2 + 3) << "=" << "+\n"

            << "\t|" << setfill(' ') << setw(3) << " " << setw(WD + 10) << left << "1. Intialize a random matrix" << setw(3) << "|\n"

            << "\t|" << setfill(' ') << setw(3) << " " << setw(WD + 10) << left << "2. Input a matrix" << setw(3) << "|\n"

            << "\t|" << setfill(' ') << setw(3) << " " << setw(WD + 10) << left << "3. Input a matrix from file .txt" << setw(3) << "|\n"

            << "\t|" << setfill(' ') << setw(3) << " " << setw(WD + 10) << left << "4. Ouput the matrix" << setw(3) << "|\n"

            << "\t|" << setfill(' ') << setw(3) << " " << setw(WD + 10) << left << "5. Intialize the adjacency list from file .txt" << setw(3) << "|\n"

            << "\t|" << setfill(' ') << setw(3) << " " << setw(WD + 10) << left << "6. Intialize the adjacency list from the matrix" << setw(3) << "|\n"

            << "\t|" << setfill(' ') << setw(3) << " " << setw(WD + 10) << left << "7. Output the adjacency list" << setw(3) << "|\n"

            << "\t|" << setfill(' ') << setw(3) << " " << setw(WD + 10) << left << "8. Prim" << setw(3) << "|\n"

            << "\t|" << setfill(' ') << setw(3) << " " << MAXCHOICE << setw(WD + 10 - 2) << left << ". Exit" << setw(3) << "|\n"

            << "\t+" << setfill('=') << setw(WD + 13) << "=" << "+\n"

            << "Your choice: ";

        colorText(7);

        cin >> choice;

        while (choice <= 0 || choice > MAXCHOICE)

        {

            colorText(12);

            cout << "Invalid! Again: ";

            colorText(7);

            cin >> choice;

        }

        switch (choice)

        {

        case 1:

            initMatrix(a, n);

            colorText(2);

            cout << "Done!\n";

            break;

        case 2:

            inputMatrix(a, n);

            colorText(2);

            cout << "Done!\n";

            break;

        case 3:

            cout << "File: ";

            cin >> filetxt;

            inputMatrix\_txt(a, n, filetxt);

            colorText(2);

            cout << "Done!\n";

            break;

        case 4:

            if (n > 0)

            {

                colorText(6);

                cout << "The matrix:\n";

                outMatrix(a, n);

            }

            else

            {

                colorText(12);

                cout << "The matrix is unavailable!\n";

            }

            break;

        case 5:

            cout << "File: ";

            cin >> filetxt;

            initAdList\_txt(filetxt);

            colorText(2);

            cout << "Done!\n";

            break;

        case 6:

            if (n > 0)

            {

                initAdList\_matrix();

                colorText(2);

                cout << "Done!\n";

            }

            else

            {

                colorText(12);

                cout << "The matrix is unavailable!\n";

            }

            break;

        case 7:

            outAdList();

            break;

        case 8:

            cout << "Vertex: ";

            cin >> x;

            Prim(x);

            cout << "Cay khung toi tieu:\n";

            outGraph(true);

            break;

        default:

            cont = false;

            colorText(12);

            cout << "Thanks for visiting us! Goodbye and see you again!\n";

            break;

        }

        colorText(7);

        system("pause");

    } while (cont == true);

}

**COUNTING:**

#pragma once

//Time Complex: O(n + k)

void countingSort(int a[], int sorted[], int n)

{

    if (n == 0 || n == 1)

        return;

    int tmpn = 0;

    //step 1:

    //decide the size k of tmp by going through each entry of a, if the entry > current tmpn (currently 0), assign entry to tmpn

    for (int i = 0; i < n; i++)

    {

        if (a[i] + 2 > tmpn)

            tmpn = a[i] + 2;

    }

    int\* tmp = new int[tmpn];

    for (int i = 0; i < tmpn; i++)

    {

        tmp[i] = 0;

    }

    //for each entry in a, increment its corresponding entry in tmp where a's entry is tmp's index

    for (int i = 0; i < n; i++)

    {

        tmp[a[i]]++;

    }

    //step 2:

    //do addition: tmp[i] = tmp[i] + tmp[i - 1], where 1 <= i <= tmpn - 1

    for (int i = 1; i < tmpn; i++)

    {

        tmp[i] += tmp[i - 1];

    }

    //step 3:

    //add sorted values to sorted array based on current tmp, but before adding, decrement the value in tmp

    for (int i = n - 1; i >= 0; i--)

    {

        tmp[a[i]]--;

        sorted[tmp[a[i]]] = a[i];

    }

    delete[] tmp;

}