#include <iostream>

#include<fstream>

#include<cstdio>

using namespace std;

#define MAX 32767

#define MVNUM 100

typedef char VerTexType;

typedef int ArcType;

class EdgeNode {

public:

VerTexType Head;

VerTexType Tail;

ArcType lowcost;

EdgeNode(VerTexType H, VerTexType T, ArcType N) {

Head = H;

Tail = T;

lowcost = N;

}

EdgeNode(EdgeNode &E) {

Head = E.Head;

Tail = E.Tail;

lowcost = E.lowcost;

}

friend class AMGraph;

friend void Sort(AMGraph& G);

friend void MiniSpanTree\_Kruskal(AMGraph& G);

};

int Vexset[MVNUM];

class AMGraph{

private:

VerTexType\* vexs=nullptr;

ArcType\* arcs=nullptr;

int vexnum=0, arcnum=0;

EdgeNode\*\* Edge=nullptr;

public:

AMGraph() {

vexnum = 0; arcnum = 0;

}

int Locate(char x) {

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

if (vexs[i] == x)return i;

}

return -1;

}

AMGraph(const char\* path) {//读取文件，输入（点数、边数，各点，边的辅助数组）创造图

ifstream ifs;

ifs.open(path,ios::in);

if (ifs.is\_open()) {

ifs>> vexnum;

ifs>> arcnum;

vexs = new VerTexType[MVNUM];

arcs = new ArcType[MVNUM \* MVNUM];

for (int i = 0; i < MVNUM \* MVNUM; ++i) {

arcs[i] = MAX;

}

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

char x;

ifs >> x;

vexs[i] = x;

}

Edge = new EdgeNode \* [arcnum];

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

char a, b;

int c;

ifs >> a >> b >> c;

Edge[j] = new EdgeNode(a, b, c);

int a1 = Locate(a), b1 = Locate(b);

arcs[a1 \* MVNUM + b1] = c;

arcs[b1 \* MVNUM + a1] = c;

}

ifs.close();

}

/\*FILE\* Ifile;

int err= fopen\_s(&Ifile, path, "r");

if (err == 0) {

vexnum = fgetc(Ifile);

arcnum = fgetc(Ifile);

vexs = new VerTexType[vexnum];

arcs = new ArcType[vexnum\*vexnum];

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

vexs[i] = i;

}

Edge = new EdgeNode \* [arcnum];

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

Edge[j] = new EdgeNode(fgetc(Ifile), fgetc(Ifile), fgetc(Ifile));

}

fclose(Ifile);

}\*/

}

~AMGraph() {

if (Edge) {

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

delete Edge[i];

}

delete[]Edge;

}

if (vexs) {

delete[]vexs;

}

if (arcs)delete[]arcs;

}

void getin() {

int n;

char m, p;

cin >> n;

vexnum = n;

cin >> n;

arcnum=n;

vexs = new VerTexType[vexnum];

arcs = new ArcType[vexnum \* vexnum];

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

cin >> m;

vexs[i] = m;

}

Edge = new EdgeNode \* [arcnum];

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

cin >> m >> p >> n;

Edge[j] = new EdgeNode(m, p, n);

int a = Locate(m), b = Locate(p);

arcs[a \* vexnum + b] = n;

arcs[b \* vexnum + a] = n;

}

}

friend void Sort(AMGraph& G);

friend void MiniSpanTree\_Kruskal(AMGraph& G);

friend void InsertVex(AMGraph& G, VerTexType v);

friend void DeleteVex(AMGraph& G, VerTexType v);

friend void InsertArc(AMGraph& G, VerTexType v, VerTexType w);

friend void DeleteArc(AMGraph& G, VerTexType v, VerTexType w);

};

void Sort(AMGraph &G) {

for (int i = 0; i < G.arcnum; ++i) {

for (int j = 0; j < G.arcnum - 1; ++j) {

if (G.Edge[j]->lowcost > G.Edge[j + 1]->lowcost) {

EdgeNode\* temp=G.Edge[j];

G.Edge[j] = G.Edge[j + 1];

G.Edge[j + 1] = temp;

}

}

}

}

void MiniSpanTree\_Kruskal(AMGraph &G) {

Sort(G);

for (int i = 0; i < G.vexnum; ++i) {

Vexset[i] = i;

}

for (int i = 0; i < G.arcnum; ++i) {

VerTexType v1 = G.Locate(G.Edge[i]->Head);

VerTexType v2 = G.Locate(G.Edge[i]->Tail);

int vs1 = Vexset[v1];

int vs2 = Vexset[v2];

if (vs1 != vs2) {

cout << G.Edge[i]->Head << '-' << G.Edge[i]->Tail <<endl;

for (int j = 0; j < G.vexnum; j++) {

if (Vexset[j] == vs2)Vexset[j] = vs1;

}

}

}

}

void InsertVex(AMGraph& G, VerTexType v) {

if (G.vexnum + 1 > MVNUM) { cout << "wrong"; return; }

else {

G.arcs[G.vexnum] = v;

G.vexnum++;

}

}

void DeleteVex(AMGraph& G, VerTexType v) {

int n=G.Locate(v);

G.vexnum--;

for (int i = n; i < G.vexnum; i++) {

G.vexs[i] = G.vexs[i + 1];

}

for (int i = 0; i <= G.vexnum; i++) {

for (int j = n; j < G.vexnum; i++) {

if (G.arcs[i \* MVNUM + j] != MAX)G.arcnum--;

G.arcs[i\*MVNUM+j] = G.arcs[i\*MVNUM + j + 1];

}

}

for (int i = n; i < G.vexnum; i++) {

for (int j = 0; j < G.vexnum; i++) {

if (G.arcs[i \* MVNUM + j] != MAX)G.arcnum--;

G.arcs[i \* MVNUM + j] = G.arcs[(i + 1) \* MVNUM + j];

}

}

}

void InsertArc(AMGraph& G, VerTexType v, VerTexType w) {

int n = G.Locate(v);

int m = G.Locate(w);

G.arcnum++;

G.arcnum++;

G.arcs[n \* MVNUM + m] = 1;

G.arcs[m \* MVNUM + n] = 1;

}

void DeleteArc(AMGraph& G, VerTexType v, VerTexType w) {

int n = G.Locate(v);

int m = G.Locate(w);

G.arcnum--;

G.arcnum--;

G.arcs[n \* MVNUM + m] = MAX;

G.arcs[m \* MVNUM + n] = MAX;

}