**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

Пояснительная записка к курсовой работе

**по дисциплине «Алгоритмы и структуры данных»**

Тема : «Графы»

|  |  |  |
| --- | --- | --- |
| Студенты гр. 8307 |  | Никулин Л., Зуб В. |
| Преподаватель |  | Колинько П.Г. |

Санкт-Петербург

2019

1. **Текст индивидуального задания.**

Отыскивание кратчайшего пути между произвольной парой вершин в ориентированном графе с нагруженными ребрами (веса ребер неотрицательные).

1. **Задание на работу с деревьями.**

Пусть задан граф с весами рёбер и выделенной вершиной-источником *u*. Нам нужно найти кратчайшее расстояние *d(v)* от источника *u* до вершины *v*.

1. **Обоснование выбора способа представления графа в памяти ЭВМ**

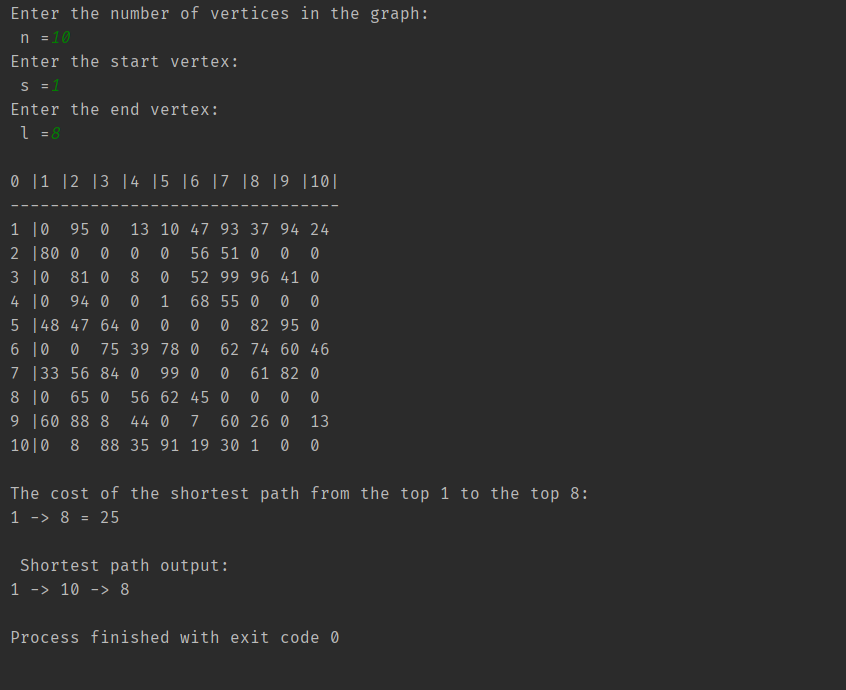
Был выбран способ представления графа в виде матрицы смежности, так как это самый оптимальный вариант для представления ориентированного графа в памяти ЭВМ.

1. **Описание алгоритма и оценка его временной сложности**

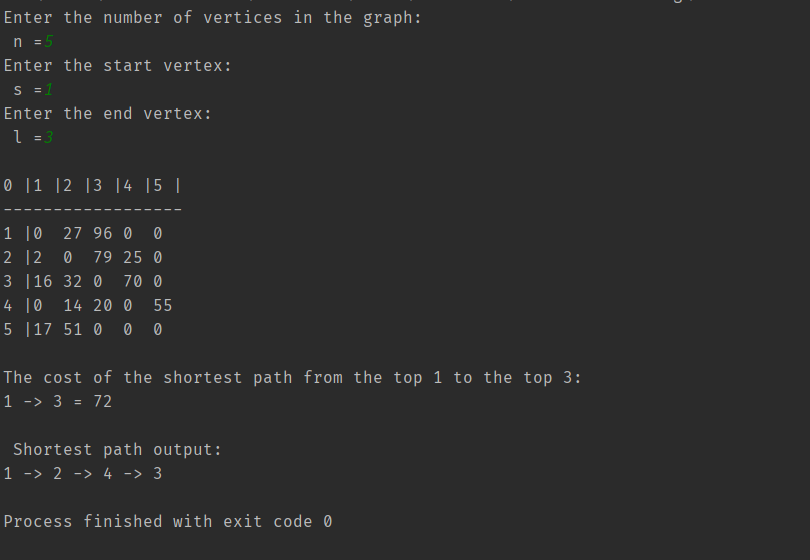
Для решения этой задачи проще всего использовать алгоритм Дейкстры: на каждом шаге цикла мы ищем вершину *v* с минимальным расстоянием и флагом равным нулю. Затем мы устанавливаем в ней флаг в 1 и проверяем все соседние с ней вершины *u*. Если в них (в *u*) расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его. Цикл завершается, когда флаги всех вершин становятся равны 1, либо когда у всех вершин c флагом 0 *d[i]=∞*. Последний случай возможен тогда и только тогда, когда граф *G* несвязный.

Временная сложность при использовании этого алгоритма для нашего представления графа будет .

1. **Результаты работы программы**

****

На скриншоте изображен результат работы программы, ошибки не обнаружены.



На скриншоте изображен результат работы программы, ошибки не обнаружены.

1. **Вывод**

1. При проведенных тестах не были замечены ошибки в решении поставленной задачи.

2. Была оценена временная сложность алгоритма.

3. При выполнении курсовой работы был исследован и реализован алгоритм Дейкстры, также научились отлаживать рекурсивные функции; закрепили и улучшили навыки работы с классами в языке C++.

1. **Список использованных источников**
2. Колинько П.Г. Алгоритмы и структуры данных. Часть 1: Методические указания к самостоятельной работе на ПЭВМ и курсовому проектированию. Вып. 1909. — СПб.: СПбГЭТУ «ЛЭТИ», 2019. — 69 с.
3. https://ru.wikipedia.org/wiki/Алгоритм\_Дейкстры - Алгоритм Дейкстры
4. **Приложение**

**Код:**

#include <iostream>

#include <time.h>

#include <stdlib.h>

#include <limits.h>

#include <locale.h>

class Graph{

private:

int\*\* a = nullptr;

int n;

public:

Graph(int rows) : n(rows) {

a = new int\* [n];

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

a[i] = new int [n];

}

~Graph(){

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

delete [] a[i];

delete [] a;

}

void Input(int);

void Output() const;

void Search(int, int);

};

void Graph::Input(int f) {

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

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

if(rand() % 3 == 2)

a[i][j] = 0;

else

a[i][j] = rand() % 99 + 1;

}

a[i][i] = 0;

}

}

void Graph::Output() const {

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

{

std::cout << i;

if(i < 10) std::cout << " |";

else std::cout << "|";

}

std::cout << std::endl;

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

std::cout << "---";

std::cout << std::endl;

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

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

if(j == 0) {

std::cout << i + 1;

if(i < 9) std::cout << " |";

else std::cout << "|";

}

std::cout << a[i][j];

if(a[i][j] < 10)

std::cout << " ";

else

std::cout << " ";

}

std::cout << std::endl;

}

}

void Graph::Search(int st, int second)

{

using namespace std;

int V = n;

int distance[V], count, index, i, u, m = st + 1;

bool visited[V];

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

distance[i] = INT\_MAX;

visited[i] = false;

}

distance[st] = 0;

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

int min = INT\_MAX;

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

if (!visited[i] && distance[i] <= min) {

min = distance[i];

index = i;

}

u = index;

visited[u] = true;

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

if (!visited[i] && a[u][i] && distance[u] != INT\_MAX &&

distance[u] + a[u][i] < distance[i])

distance[i] = distance[u] + a[u][i];

}

std::cout<<"The cost of the shortest path from the top "<< m << " to the top " << second << ":\n";

if (distance[second - 1] != INT\_MAX)

std::cout << m << " -> " << second <<" = "<< distance[second - 1] << endl;

else std::cout << m << " -> " << second << " = " << "route unavailable" << endl;

//------------------------------------

int ver[V];

int b[V][V];

int end = second - 1;

ver[0] = end + 1;

int k = 1;

int weight = distance[end];

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

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

b[i][j] = a[j][i];

while (end != st ) {

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

if (b[end][i] != 0) {

int temp = weight - b[end][i];

if (temp == distance[i]) {

weight = temp;

end = i;

ver[k] = i + 1;

k++;

}

}

}

printf("\n Shortest path output: \n");

for (int i = k - 1; i >= 0; i--) {

cout << ver[i];

if(i - 1 >= 0)

cout << " -> ";

}

cout << endl;

}

int main() {

srand(time(nullptr));

using namespace std;

int n, s, l;

cout << "Enter the number of vertices in the graph: \n n =";

cin >> n;

cout << "Enter the start vertex: \n s =";

cin >> s;

cout << "Enter the end vertex: \n l =";

cin >> l;

cout << endl;

Graph gr(n);

gr.Input(1);

gr.Output();

cout << endl;

gr.Search(s-1,l);

return 0;

}