МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра «Математического обеспечения и применения ЭВМ»

|  |  |  |
| --- | --- | --- |
| 1002262_584459121576624_356098510_n |  | logo МОПЭВМ |
|  |  |  |

**Индивидуальное задание**

по курсу

Методы оптимизации

Вариант№ 2

Выполнили:

студент группы КТбо3-9

Митина А.Г.

Ячменев М.И.

Проверила:

Липко Ю.Ю.

Таганрог, 2016

Содержание

[1. Постановка задачи 3](#_Toc462868330)

[2. Описание алгоритма Дейкстры 3](#_Toc462868331)

[3. Листинг 4](#_Toc462868332)

[3.1 Описание входных данных 4](#_Toc462868333)

[3.2 Описание выходных данных 4](#_Toc462868334)

[3.3 Дополнительная информация 4](#_Toc462868335)

[4. Проверка качества алгоритма 7](#_Toc462868336)

[4.1 тест 1 7](#_Toc462868337)

[4.2 Тест 2 7](#_Toc462868338)

[4.3 Тест 3 8](#_Toc462868339)

[5. Заключение 8](#_Toc462868340)

[6. Библиография 9](#_Toc462868341)

# Постановка задачи

Разработать алгоритм и программу построения кратчайшего маршрута движения доставщика пиццы на основе изученных алгоритмов теории графов.

Давайте разберемся, что такое путь. Путь – это последовательность ребер, соединяющих две вершины. [1] То есть, нам необходимо найти такую последовательность ребер, которая даст минимальное расстояние. Эту задачу решает алгоритм Дейксты.

# Описание алгоритма Дейкстры

Алгоритм Дейкстры является предпочтительным методом поиска кратчайших путей в графах со взвешенными ребрами(вершинами)[1]. Алгоритм находит кратчайшие пути от заданной вершины S ко всем другим вершинам графа, включая требуемую конечную вершину t.

Допустим, что кратчайший путь от вершины s к вершине t графа проходит через определенную промежуточную вершину x. Очевидно, что путь должен содержать кратчайший путь от вершины sдо вершины x, в качестве префикса, ибо в противном случае можно было бы сократить путь s-t, используя более короткий префиксный путь s-x. Таким образом, прежде чем найти кратчайший путь от начальной вершины s к конечной вершине t, нам нужно найти кротчайший путь от начальной вершины s к промежуточной вершине x.

Алгоритм Дейкстры работает поэтапно, находя на каждом этапе кратчайший путь от вершины s к некоторой новой вершине. Говоря конкретно, вершина X такова, что сумма dist(s,vi)+w(v1,x) минимальна для всех необработанных 1<=i<=n, где w(I,j)-длина ребра между вершинами I и j, а dist(I, j) длина кратчайшего пути между ними.

Здесь напрашивается стратегия, аналогичная ДП. Кратчайший путь от вершины s к самой себе является тривиальным, при условии отсутствия ребе с отрицательным весом, поэтому dist(s,s)=0. Если (s,y) является самым легким ребром, входящим в вершину s, то это означает, что dist(s,y)=w(s,y). Определив кратчайший путь к вершине x, мы проверяем все исходящие из нее реба, чтобы узнать, не существует ли лучшего пути из начальной вершины s к какой-нибудь неизвестной вершине через вершину x.   
рассмотрим псевдокод данного алгоритма.

1. ShortestPath-Dijkstra(G,s,t)
2. known = {s}
3. for i = 1 to n, dist[i] = infinity
4. for each adge(s,v), dist[v] = w(s,v)
5. last = s
6. while (last != t)
7. delect Vnext, неизвестная вершина, минимизирующая dist[v]
8. for each adge (Vnext, x), dist[x] = min[dist[x],dist[Vnext]+w(Vnext,s)]
9. last =  Vnext
10. known = known Объединение {Vnext}

В каждом цикле мы добавляем одну вершину к дереву вершин, для которых мы знаем кратчайший путь из вершины s. Мы сохраняем информацию о наилучшем пути на данное время для всех вершин вне дерев и вставляем их в дерево в порядке возрастания веса.

# Листинг

## Описание входных данных

Входные данные задаются в текстовом файле in.txt. В первой строке вводится размер исходного графа. В новой строке номера вершин начала и окончания пути. В последующих строках вводится по 3 числа: номер вершины от куда идет путь, и номер вершины куда этот путь ведет, и расстояние.

## Описание выходных данных

Результат выводится в out.txt, и представляет из себя число – минимальный размер пути и последовательность вершин, образующих минимальный путь.

## Дополнительная информация

Программа написана на языке С++ в среде Visual Studio 2015

1. #include <functional>
2. #include <algorithm>
3. #include <iterator>
4. #include <iostream>
5. #include <sstream>
6. #include <iomanip>
7. #include <numeric>
8. #include <fstream>
9. #include <utility>
10. #include <cstring>
11. #include <cstdlib>
12. #include <cstddef>
13. #include <vector>
14. #include <bitset>
15. #include <cstdio>
16. #include <cctype>
17. #include <deque>
18. #include <stack>
19. #include <queue>
20. #include <cmath>
21. #include <ctime>
22. #include <list>
23. #include <map>
24. #include <set>
26. using namespace std;
28. #define mp make\_pair
29. #define pb push\_back
30. #define mref mem\_fun\_ref
31. #define bis back\_inserter
32. #define all(x) (x).begin(), (x).end()
33. #define forn(i, n) for(int64 i = 0; i < int(n); ++i)
35. typedef long long int64;
36. typedef pair<int64, int64> pint;
37. typedef vector<int64> vint;
38. const int64 INF = 1e9;
39. const int N = 1e5 + 4;
41. struct Edge
42. {
43. Edge(){}
44. Edge(int \_v, int \_len) : v(\_v), len(\_len)  {}
45. int v; *// вершина в которую приходим*
46. int len;
47. };
48. struct Node
49. {
50. Node(): dist(INF), prev(-1){}
51. Node(int \_dist) :dist(\_dist) {};
52. vector<Edge> edges;
53. int dist; *// длина пути*
54. int prev;
55. };
57. vector <Node> graph;
59. void Dij(int s)
60. {
61. priority\_queue<pair<int, int>, vector< pair<int, int> >, greater< pair<int, int> > > que; *// очередь*
62. graph[s].dist = 0;
63. que.push(mp(0, s));
64. while (!que.empty())
65. { *// пока очередь не пуста*
66. int u = que.top().second;
67. int dist = que.top().first;*// достаём расстояние (FIRST!!!)*
68. que.pop();
69. if (graph[u].dist != dist) continue; *// если не то, переходим к след.*
70. for (int i = 0; i < graph[u].edges.size(); ++i)
71. { *// цикл по списку*
72. int v = graph[u].edges[i].v;
73. int newDist = graph[u].dist + graph[u].edges[i].len;
74. if (graph[v].dist > newDist)
75. { *// если лучше*
76. graph[v].dist = newDist; *// то улучшаем*
77. graph[v].prev = u; *// запоминаем предшеств.*
78. que.push(mp(graph[v].dist, v)); *// в очередь !!!*
79. }
80. }
81. }
82. }


86. int main()
87. {
88. freopen("in.txt", "r", stdin);
89. freopen("out.txt", "w", stdout);
90. int n;
91. cin >> n;
92. int firstNode, lastNode;
93. cin >> firstNode >> lastNode;
94. graph.resize(n);
95. int start, end, dist;
96. while(cin >> start >> end >> dist)
97. {
98. start--, end--;
99. graph[start].edges.push\_back(Edge(end, dist));
101. }
103. Dij(--firstNode);
104. lastNode--;
106. if (graph[lastNode].dist == INF)
107. {
108. [cout](http://www.opengroup.org/onlinepubs/009695399/functions/cout.html) << "не удалось построить путь" << endl;
109. return 0;
110. }
111. [cout](http://www.opengroup.org/onlinepubs/009695399/functions/cout.html) << graph[lastNode].dist << endl;
112. vint res;
113. res.push\_back(lastNode + 1);
114. while (graph[lastNode].prev != -1)
115. {
116. res.push\_back(graph[lastNode].prev + 1);
117. lastNode = graph[lastNode].prev;
118. }
120. for(int i = res.size() - 1; i > -1; --i)
121. {
122. [cout](http://www.opengroup.org/onlinepubs/009695399/functions/cout.html) << res[i] << ' ';
123. }
124. [cout](http://www.opengroup.org/onlinepubs/009695399/functions/cout.html) << endl;
125. return 0;
126. }

# Проверка качества алгоритма

## тест 1

входные данные

5

1 5

1 2 5

1 5 40

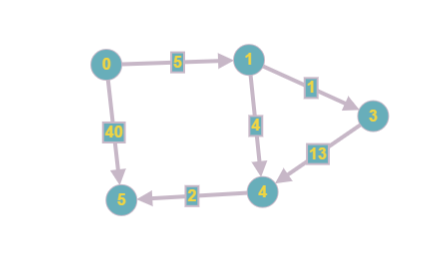
2 3 1

2 4 4

3 4 13

4 5 2

Граф выглядит подобным образом, нетрудно заметить, что минимальный путь будет проходить через вершины 1-2-4-5 и значение пути равно 11



Выходные данные

11

1 2 4

## Тест 2

Входные данные

5

3 1

1 2 5

2 1 5

1 5 40

5 1 40

2 3 1

3 2 1

2 4 4

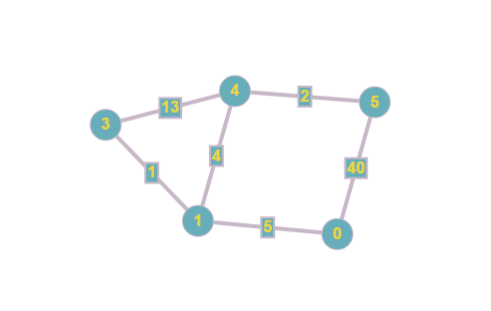
4 2 4

3 4 13

4 3 13

4 5 2

5 4 2



Выходные данные

6

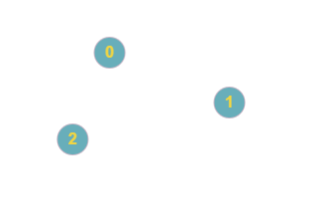
3 2 1

## Тест 3

Входные данные

3

1 3



Выходные данные

“не удалось построить путь”

# Заключение

Алгоритм Дейкстры работает правильно только в тех графах, в которых нет отрицательных ребер. Дело в том, что при построении пути может встретиться ребро с отрицательным весом настолько большим по модулю, что оно полностью изменит оптимальный путь вершины s к какой-нибудь другой вершине, которая уже включена в дерево.

Асимптотическая сложность алгоритма Дейкстры в наивной реализации О( понятно, что эта асимптотика является оптимальной для плотных графов, т.е. когда  . Чем более разрежен граф (т.е. чем меньше m по сравнению с максимальным количество рёбер ), тем менее оптимальной становится эта оценка, и по вине первого слагаемого. Таким образом, надо улучшать время выполнения операций первого типа, не сильно ухудшая при этом время выполнения операций второго типа.

Для этого надо использовать различные вспомогательные структуры данных. Наиболее привлекательными являются Фибоначчиевы кучи, которые позволяют производить операцию первого вида за O(logn), а второго — за O(1). Поэтому при использовании Фибоначчиевых куч время работы алгоритма Дейкстры составит O(n\*logn+m), что является практически теоретическим минимумом для алгоритма поиска кратчайшего пути. Однако, Фибоначчиевы кучи довольно сложны в реализации (и, надо отметить, имеют немалую константу, скрытую в асимптотике).

В качестве компромисса можно использовать структуры данных, позволяющие выполнять оба типа операций (фактически, это извлечение минимума и обновление элемента) за O(n\*logn). Тогда время работы алгоритма Дейкстры составит:

O(n\*logn+m\*logn)=O(m\*logn)

В качестве такой структуры данных программистам на C++ удобно взять стандартный priority\_queue, основанный на бинарной куче.

# Библиография

1. Стивен С. Скиена “Алгоритмы Руководство по разработке” 2-е издание, Санкт-Петербург 2016г
2. Википедия https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B