**Содержание**

**Теоретическая часть** 3

**Пример графа по алгоритму дейкстры** 4

**Листинг программы** 8

**Результаты работы программы** 10

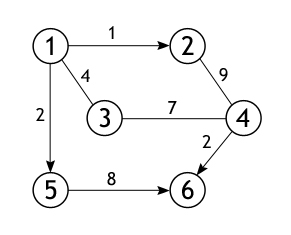
**Теоретическая часть**

Алгоритм голландского ученого Эдсгера Дейкстры находит все кратчайшие пути из одной изначально заданной вершины графа до всех остальных. С его помощью, при наличии всей необходимой информации, можно, например, узнать какую последовательность дорог лучше использовать, чтобы добраться из одного города до каждого из многих других, или в какие страны выгодней экспортировать нефть и тому подобное.

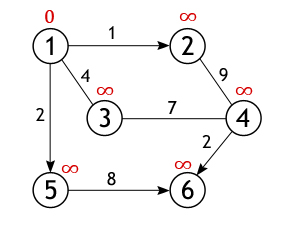
Минусом данного метода является невозможность обработки графов, в которых имеются ребра с отрицательным весом, т. е. если, например, некоторая система предусматривает убыточные маршруты, то для работы с ней следует воспользоваться отличным от алгоритма Дейкстры методом.

**Пример графа по алгоритму Дейкстры**

 На конкретном графе проследим работу алгоритма, найдем все кратчайшие пути между истоковой и всеми остальными вершинами. Размер (количество ребер) изображенного ниже графа равен **7 (E=7)**, а порядок (количество вершин) –**6 (V=6)**. Это взвешенный граф, каждому из его ребер поставлено в соответствие некоторое числовое значение, поэтому ценность маршрута необязательно определяется числом ребер, лежащих между парой вершин.



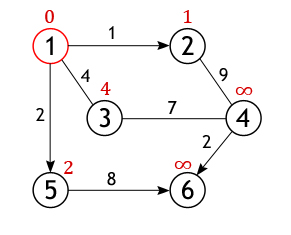
Из всех вершин входящих во множество **V** выберем одну, ту, от которой необходимо найти кратчайшие пути до остальных доступных вершин. Пусть таковой будет вершина **1**. Длина пути до всех вершин, кроме первой, изначально равна бесконечности, а до нее – **0**, т. к. граф не имеет петель.



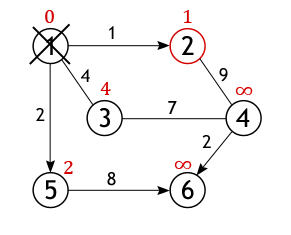
У вершины 1 ровно 3 соседа (вершины 2, 3, 5), и чтобы вычислить длину пути до них нужно сложить вес дуг, лежащих между вершинами: 1 и 2, 1 и 3, 1 и 5 со значением первой вершины (с нулем):

**2←1+0  
3←4+0  
5←2+0**

Как уже отмечалось, получившиеся значения присваиваются вершинам, лишь в том случае если они «лучше» (меньше) тех которые значатся на настоящий момент. А так как каждое из трех чисел меньше бесконечности, они становятся новыми величинами, определяющими длину пути из вершины 1 до вершин 2, 3 и 5.



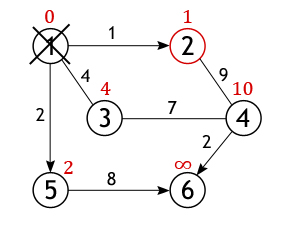
Далее, активная вершина помечается как посещенная, статус «активной» (красный круг) переходит к одной из ее соседок, а именно к вершине 2, поскольку она ближайшая к ранее активной вершине.



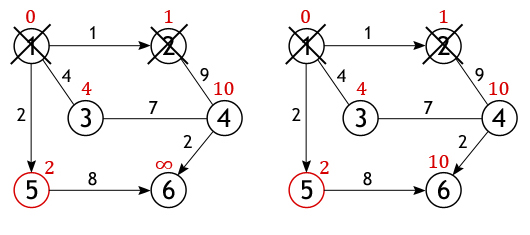
У вершины 2 всего один не рассмотренный сосед (вершина 1 помечена как посещенная), расстояние до которого из нее равно 9, но нам необходимо вычислить длину пути из истоковой вершины, для чего нужно сложить величину приписанную вершине 2 с весом дуги из нее в вершину 4

**4←1+9**

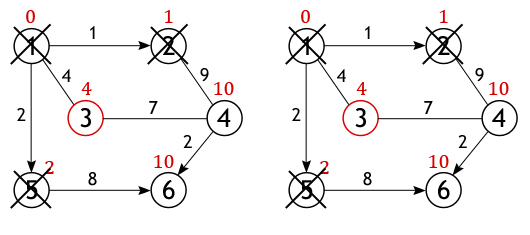
Условие «краткости» (10<∞) выполняется, следовательно, вершина 4 получает новое значение длины пути.



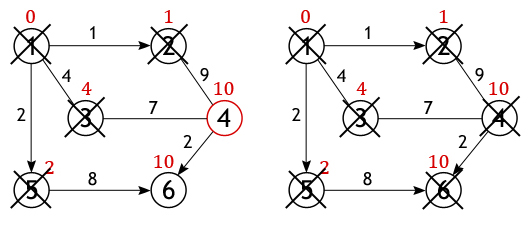
Вершина 2 перестает быть активной, также как и вершина 1 удаляется из списка не посещённых. Теперь тем же способом исследуются соседи вершины 5, и вычисляется расстояние до них.



Когда дело доходит до осмотра соседей вершины 3, то тут важно не ошибиться, т. к. вершина 4 уже была исследована и расстояние одного из возможных путей из истока до нее вычислено. Если двигаться в нее через вершину 3, то путь составит 4+7=11, а 11>10, поэтому новое значение игнорируется, старое остается.



Аналогичная ситуация с вершиной 6. Значение самого близкого пути до нее из вершины 1 равно 10, а оно получается только в том случае, если идти через вершину 5.



Когда все вершины графа, либо все те, что доступны из истока, будут помечены как посещенные, тогда работа алгоритма Дейкстры завершится, и все найденные пути будут кратчайшими. Так, например, будет выглядеть список самых оптимальных расстояний лежащих между вершиной 1 и всеми остальными вершинами, рассматриваемого графа:

**1→1=0**

**1→2=1**

**1→3=4**

**1→4=10**

**1→5=2**

**1→6=10**

**Листинг программы**

#include <iostream>

#include <algorithm>

/\* Алгоритм Дейкстры.\*/

#define INF 9000000 // Нет вершины

#define MatrixLen 9

/\* Исходная матрица расстояний \*/

int matrix[MatrixLen][MatrixLen] = {

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

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

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

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

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

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

{ 0, 0, 0, 14, 0, 2, 0, 1, 6 },

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

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

};

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

{

int min = INF, min\_index;

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

{

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

{

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

}

}

return min\_index;

}

void printSolution(int dist[], int n)

{

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

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

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

}

void dijkstra(int from)

{

int dist[MatrixLen];

bool shortestPath[MatrixLen];

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

{

dist[i] = INF, shortestPath[i] = false;

}

dist[from] = 0;

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

{

int u = minDistance(dist, shortestPath);

shortestPath[u] = true;

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

{

if (!shortestPath[v] && matrix[u][v] && dist[u] != INF

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

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

}

}

printSolution(dist, MatrixLen);

}

int main()

{

dijkstra(0);

system("pause");

return 1;

}

**Результат работы программы**

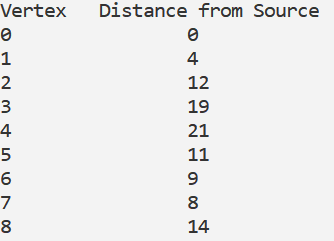
****

Рисунок 1 – результат работы программы