**Содержание**

**Теоретическая часть** 3

**Пример** 5

**Листинг программы** 8

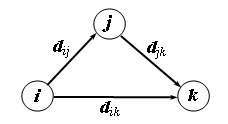
**Результаты работы программы** 13

**Теоретическая часть**

Алгоритм Флойда—Уоршелла—динамический алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного ориентированного графа. Разработан в 1962 году Робертом Флойдом и Стивеном Уоршеллом. При этом алгоритм впервые разработал и опубликовал Бернард в 1959 году.

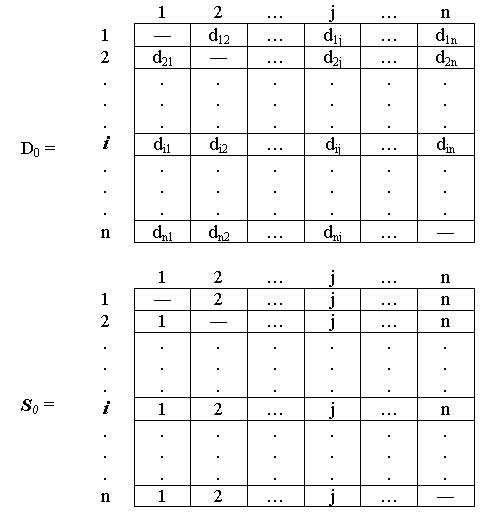
Этот алгоритм более общий по сравнению с алгоритмом Дейкстры, так как он находит кратчайшие пути между любыми двумя узлами сети. В этом алгоритме сеть представлена в виде квадратной матрицы с n строками и nстолбцами. Элемент (i, j) равен расстоянию dij от узла i к узлу j, которое имеет конечное значение, если существует дуга (i, j), и равен бесконечности в противном случае.

Покажем сначала основную идею метода Флойда. Пусть есть три узла i, j и k и заданы расстояния между ними. Если выполняется неравенство dij + djk < dik, то целесообразно заменить путь i -> k путем i -> j -> k. Такая замена (далее ее будем условно называть треугольным оператором) выполняется систематически в процессе выполнения алгоритма Флойда.



Алгоритм Флойда требует выполнения следующих действий:

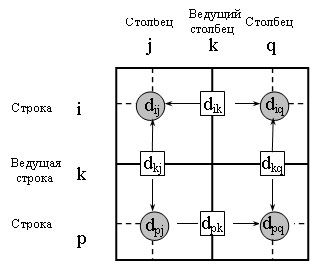
1. Определяем начальную матрицу расстояния D0 и матрицу последовательности узлов S0. Диагональные элементы обеих матриц помечаются знаком "-", показывающим, что эти элементы в вычислениях не участвуют. Полагаем k = 1:



1. . Задаем строку k и столбец k как ведущую строку и ведущий столбец. Рассматриваем возможность применения треугольного оператора ко всем элементам dij матрицы Dk-1. Если выполняется неравенство dik + dkj < dij, (i <> k, j <> k, i <> j), тогда выполняем следующие действия:

* создаем матрицу Dk путем замены в матрице Dk-1 элемента dij на сумму dik + dkj,
* создаем матрицу Sk путем замены в матрице Sk-1 элемента sij на k. Полагаем k = k + 1 и повторяем шаг k.

Поясним действия, выполняемые на k-м шаге алгоритма, представив матрицу Dk-1. Строка k и столбец k являются ведущими. Строка i - любая строка с номером от 1 до k - 1, а строка p - произвольная строка с номером от k + 1 до n. Аналогично столбец j представляет любой столбец с номером от 1 до k - 1, столбец q - произвольный столбец с номером от k + 1 до n. Треугольный оператор выполняется следующим образом. Если сумма элементов ведущих строки и столбца (показанных в квадратах) меньше элементов, находящихся в пересечении столбца и строки (показанных в кружках), соответствующих рассматриваемым ведущим элементам, то расстояние (элемент в кружке) заменяется на сумму расстояний, представленных ведущими элементами:



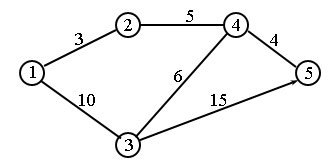
После реализации n шагов алгоритма определение по матрицам Dn и Sn кратчайшего пути между узлами i и jвыполняется по следующим правилам.

Расстояние между узлами i и j равно элементу dij в матрице Dn.

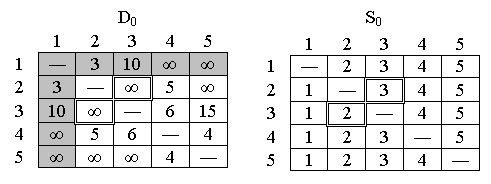
Промежуточные узлы пути от узла i к узлу j определяем по матрице Sn. Пусть sij = k, тогда имеем путь i -> k -> j. Если далее sik = k и skj = j, тогда считаем, что весь путь определен, так как найдены все промежуточные узлы. В противном случае повторяем описанную процедуру для путей от узла i к узлу k и от узла k к узлу j.

**Пример**

Найдем для данной сети кратчайшие пути между любыми двумя узлами. Расстояние между узлами этой сети проставлены на рисунке возле соответствующих ребер. Ребро (3, 5) ориентировано, поэтому не допускается движение от узла 5 к узлу 3. Все остальные ребра допускают движение в обе стороны:



Начальные матрицы D0 и S0 строятся непосредственно по заданной схеме сети. Матрица D0 симметрична, за исключением пары элементов d35 и d53, где d53 равно бесконечности, поскольку невозможен переход от узла 5 к узлу 3:

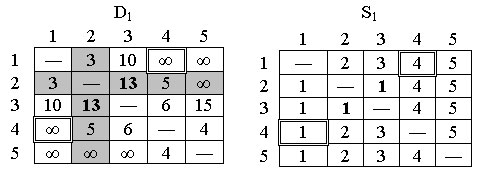


В матрице D0 выделены ведущие строка и столбец (k = 1). Двойной рамкой представлены элементы d23 и d32, единственные среди элементов матрицы D0, значения которых можно улучшить с помощью треугольного оператора. Таким образом, чтобы на основе матриц D0 и S0 получить матрицы D1 и S1, выполняем следующие действия.

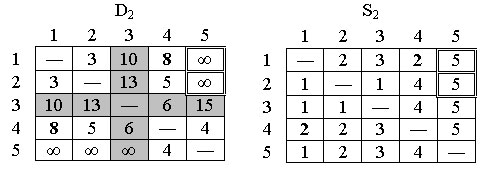
Заменяем d23 на d21 + d13 = 3 + 10 = 13 и устанавливаем s23 = 1.

Заменяем d32 на d31 + d12 = 10 + 3 = 13 и устанавливаем s32 = 1.

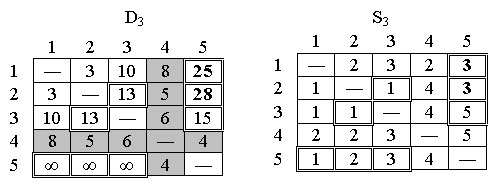
Матрицы D1 и S1 имеют следующий вид:



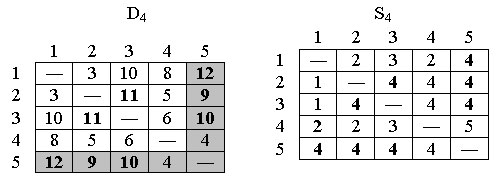
Полагаем k = 2; в матрице D1 выделены ведущие строка и столбец. Треугольный оператор применяется к элементам матрицы D1 и S1, выделенным двойной рамкой. В результате получаем матрицы D2 и S2:



Полагаем k = 3; в матрице D2 выделены ведущие строка и столбец. Треугольный оператор применяется к элементам матрицы D2 и S2, выделенным двойной рамкой. В результате получаем матрицы D3 и S3:



Полагаем k = 4, ведущие строка и столбец в матрице D3 выделены. Получаем новые матрицы D4 и S4:



Полагаем k = 5, ведущие строка и столбец в матрице D4 выделены. Никаких действий на этом шаге не выполняем; вычисления закончены.

Конечные матрицы D4 и S4 содержат всю информацию, необходимую для определения кратчайших путей между любыми двумя узлами сети. Например, кратчайшее расстояние между узлами 1 и 5 равно d15 = 12.

Для нахождения соответствующих маршрутов напомним, что сегмент маршрута (i, j) состоит из ребра (i, j) только в том случае, когда sij = j. В противном случае узлы i и j связаны, по крайней мере, через один промежуточный узел. Например, поскольку s15 = 4 и s45 = 5, сначала кратчайший маршрут между узлами 1 и 5 будет иметь вид 1->4->5. Но так как s14 не равно 4, узлы 1 и 4 в определенном пути не связаны одним ребром (но в исходной сети они могут быть связаны непосредственно). Далее следует определить промежуточный узел (узлы) между первым и четвертым узлами. Имеем s14 = 2 и s24 = 4, поэтому маршрут 1->4 заменяем 1->2->4. Поскольку s12 = 2 и s24 = 4, других промежуточных узлов нет. Комбинируя определенные сегменты маршрута, окончательно получаем следующий кратчайший путь от узла 1 до узла 5: 1->2->4->5. Длина этого пути равна 12 километрам.

**Листинг программы**

#include <iostream>

#define INF 9000000 // Нет вершины

#define MatrixLen 5

/\* Алгоритм Флойда — Уоршелла.\*/

/\* Исходная матрица расстояний \*/

int matrix[MatrixLen][MatrixLen] =

{

// 1 2 3 4 5

{0, 5, 2, INF, INF}, // 1

{5, 0, INF, 7, INF}, // 2

{2, INF, 0, 2, 8}, // 3

{INF, 7, 2, 0, 1}, // 4

{INF, INF, 8, 1, 0} // 5

};

int main()

{

/\* Поиск расстояния минимального пути от каждой до каждой вершины \*/

for (size\_t k(0); k < MatrixLen; ++k)

for (size\_t i(0); i < MatrixLen; ++i)

for (size\_t j(0); j < MatrixLen; ++j)

if (matrix[i][k] < INF && matrix[k][j] < INF)

if (matrix[i][k] + matrix[k][j] < matrix[i][j])

matrix[i][j] = matrix[i][k] + matrix[k][j];

int from = 0;

int to = 4;

/\* Теперь минимальное расстояние от любой до любой будет matrix[ОТКУДА][КУДА] \*/

std::cout << "Lenght from " << from << " to " << to << ": " << matrix[from][to] << std::endl;

system("pause");

return 1;

}

**Результат работы программы**

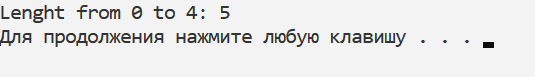


Рисунок 1 – результат работы программы