Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
—
Институт компьютерных наук и технологий
Кафедра «Информационная безопасность компьютерных систем»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

по дисциплине «Дискретная Математика»

Выполнил
студент гр. 23508/4 Е.Г. Проценко

Проверила
ассистент Д.С. Лаврова

Санкт-Петербург
2016

1. Формулировка задания (Вариант 7)

Цели работы - изучение алгоритмов поиска кратчайших путей в сети между парой вершин и между всеми вершинами, алгоритмов поиска оптимального плана работ, а также алгоритмов поиска maxmin-путей в сети.

Задание 1. В сети S_1, заданной матрицей весов, определить кратчайшие пути из вершины v_1 в вершину u.

Задание 2. В сети S_2, заданной матрицей весов, определить:

a) Для алгоритма Флойда – матрицу кратчайших путей и матрицу длин этих путей. Найти кратчайший путь между вершинами v_2 и v_3 и указать его длину;
b) Для сетевого планирования. Найти оптимальный план работ T и критический путь. Для каждой работы определить наиболее ранний возможный срок ее начала и наиболее ранний возможный срок ее выполнения. При заданном плане T' определить для каждой работы наиболее поздний допустимый срок ее начала и ее выполнения, а также полный резерв времени;
c) Для задачи на «maxmin-путь»: maxmin-путь между вершинами v_2 и v_3 и его вес.

2. Ход работы

2.1. Алгоритм Дейкстры

$$\infty = 10000$$

```
Command: read_am
Command: Введите название файла из которого считать матрицу: input.txt
Done
Command: print_am
Матрица Смежности:
0 2 3 10000 10000 10000 10000 3 10000 3 1 10000 10000
10000 0 1 10000 5 10000 10000 10000 3 10000 10000 1 10000
10000 10000 0 10000 10000 10000 10000 3 10000 10000 10000 10000 12
2 1 1 0 -2 10000 4 10000 10000 10000 10000 10000 10000
10000 10000 10000 10000 0 3 10000 10000 10000 10000 10000 10000 10000
10000 10000 10000 10000 10000 0 10000 -4 2 10000 10000 10000 10000
10000 10000 10000 10000 10000 10000 0 3 1 10000 10000 10000 10000
10000 10000 10000 10000 10000 10000 10000 0 10000 10000 5 10000 10000
10000 10000 10000 10000 10000 10000 10000 10000 0 10000 4 10000 10000
10000 10000 10000 10000 10000 10000 -5 10000 10000 0 2 -2 2
10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 0 2 2
10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 0 -2
10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 0
Command: dijkstra
Command: Введите номер исходной вершины (v1) (1 <= var <= 13): 4
Command: Введите номер другой вершины (v2) (1 <= var <= 13): 13
Расстояние между вершинами v1 и v2: 0
```

2.2. Алгоритм Флойда

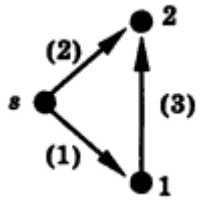
```
Command: read_am
Command: Введите название файла из которого считать матрицу: task.txt
Done
Command: print_am
Матрица Смежности:
0 1 2 10000 7 10000 10000 10000 1 10000 10000 10000 10000
10000 0 3 3 5 10000 10000 10000 10000 1 10000 10000 10000
10000 10000 0 10000 10000 10000 10000 10000 10000 10000 1 10000 10000
10000 10000 10000 0 1 10000 4 10000 10000 10000 10000 1 10000
10000 10000 10000 10000 0 3 10000 10000 10000 10000 10000 10000 10000
10000 10000 10000 1 2 0 10000 1 2 10000 10000 10000 10000
3 10000 10000 10000 10000 10000 0 2 1 10000 10000 10000 10000
10000 2 10000 10000 10000 10000 10000 0 10000 2 5 10000 10000
10000 10000 3 10000 10000 1 10000 1 0 10000 4 10000 10000
10000 10000 10000 2 10000 10000 10000 10000 10000 0 1 1 1
10000 10000 10000 10000 3 10000 10000 10000 10000 10000 0 1 1
10000 10000 10000 10000 10000 2 10000 10000 10000 10000 10000 0 1
10000 10000 10000 10000 10000 10000 10 10000 10000 10000 10000 10000 0
Command: floyd
Command: Введите номер исходной вершины (v1) (1 <= var <= 13): 1
Command: Введите номер другой вершины (v2) (1 <= var <= 13): 13
Матрица минимальных расстояний:
0 1 2 3 4 2 7 2 1 2 3 3 3
10 0 3 3 4 4 7 5 6 1 2 2 2
12 7 0 5 4 4 9 5 6 7 1 2 2
7 6 8 0 1 3 4 4 5 6 7 1 2
11 6 8 4 0 3 8 4 5 6 7 5 6
8 3 5 1 2 0 5 1 2 3 4 2 3
3 4 4 3 4 2 0 2 1 4 5 4 5
11 2 5 4 5 5 8 0 7 2 3 3 3
9 3 3 2 3 1 6 1 0 3 4 3 4
9 6 8 2 3 3 6 4 5 0 1 1 1
11 6 8 4 3 3 8 4 5 6 0 1 1
10 5 7 3 4 2 7 3 4 5 6 0 1
13 14 14 13 14 12 10 12 11 14 15 14 0
Расстояние между вершинами v1 и v2: 3
```

3. Контрольные вопросы

1. При каких условиях задача поиска кратчайшего пути между парой вершин имеет решение? Задача поиска кратчайшего пути между парой вершин имеет решение, когда в заданной сети нет контуров отрицательной длины.

2. Почему нельзя использовать модификацию алгоритма Дейкстры для поиска максимального пути? Приведите пример сети, где такой алгоритм найдет неверный путь. Модификацию алгоритма Дейкстры нельзя использовать для поиска максимального пути, поскольку на каждом шаге мы будем выбирать вершину с максимальной известной длинной пути, при этом можно допустить ошибку в определении максимального пути.

Для примера рассмотрим следующую сеть:



По алгоритму Дейкстры мы получим следующее:

1) (s,2) = 2, (s,1) = 1, тогда устанавливаем вершине 1 метку 1, а вершине 2 метку 2. Следующей просматриваемой вершиной будет 2, а s считается просмотренной.

2) 2 является стоком, поэтому метки вершин не поменяются, вершина 2 считается просмотренной.

3) Из вершины 1 существует единственная дуга в вершину 2, но она просмотрена, поэтому метки не поменяются. На этом шаге алгоритм закончится.

Таким образом, мы получили максимальный путь до 2, равный 2, что неверно.

3. Что произойдет, если на вход алгоритма Флойда подать сеть, содержащую контур отрицательной длины? На одном из шагов для некоторого i путь из вершины в саму себя станет отрицательным, что противоречит здравому смыслу.

4. Как можно определить истоки, стоки, компоненты сильной связности, изолированные вершины по результатам работы алгоритма Флойда? Это можно сделать, используя матрицу расстояний D и матрицу предков Previous, полученные в результате работы алгоритма Флойда. Вершина будет являться истоком, если до неё не будет пути ни из одной вершины т.е. если в i-ом столбце все элементы будут бесконечными по значению, то данная вершина – исток. Вершина будет являться стоком, если из этой вершины нет ни одного пути, т.е. если i-я строка будет состоять только из бесконечностей, то данная вершина – сток. Вершина будет являться изолированной, если нет ни одного пути в эту вершину и из этой вершины, таким образом, если i-е строка и столбец будут состоять только из бесконечностей, то данная вершина является изолированной.

Компоненты сильной связности можно найти следующим образом:

Если D[i,j] и D[j,i] !=∞, то все вершины входящие в путь из вершины i в j, а также и сами эти вершины составляют компоненту сильной связности. Вершины, входящие в данный путь можно восстановить при помощи матрицы предков Previous.

4. Приложение

```
#define INF 10000

void Graph::dijkstra_rec(int * work_array, int * is_marked, int current, int to)
{
 is_marked[current] = 1;
 int changes = 0;
 for (int i = 0; i < vertices; i++)
 {
        if    (is_marked[i]    ==    0    &&    work_array[i]    >    work_array[current]    +
adjacency_matrix[current][i] && adjacency_matrix[current][i] != INF)
        {
               work_array[i] = work_array[current] + adjacency_matrix[current][i];
               changes++;
        }
 }

 if (changes == 0) return;
 int min = -1;
 for (int i = 0; i < vertices; i++)
 {
        if (is_marked[i] == 0)
        {
               if (min == -1) min = i;
               else if (work_array[i] < work_array[min]) min = i;
        }
 }

 if (min == -1) return;
 if (min == to) return;
 else dijkstra_rec(work_array, is_marked, min, to);
}

int Graph::dijkstra(int from, int to)
{
 //Рабочий массив
 int * work_array = (int *)malloc(vertices * sizeof(int));
 //Заполняем его
 for (int i = 0; i < vertices; i++) work_array[i] = INF;
 work_array[from] = 0;
 //
 int * mark_array = (int *)calloc(vertices, sizeof(int));

 dijkstra_rec(work_array, mark_array, from, to);

 return work_array[to];
}

#define min(a, b) (a < b) ? a : b
//под min_dist_matrix память заранее выделена
int ** Graph::floyd()
{
 int ** W = (int **)malloc(vertices * sizeof(int *));
 for (int i = 0; i < vertices; i++)
 {
        W[i] = (int *)malloc(vertices * sizeof(int));
 }
 for (int i = 0; i < vertices; i++)
 {
        for (int j = 0; j < vertices; j++)
        {
               W[i][j] = adjacency_matrix[i][j];
        }
 }


 /*for (int k = 0; k < vertices; k++)
```

```
 {
        for (int i = 0; i < vertices; i++)
        {
                for (int j = 0; j < vertices; j++)
                {
                        W[i][j] = (W[i][j] < W[i][k] + W[k][j]) ? W[i][j] : W[i][k] + W[k][j];
                }
        }
 }*/

 int ** TEMP = (int **)malloc(vertices * sizeof(int *));
 for (int i = 0; i < vertices; i++)
 {
        TEMP[i] = (int *)malloc(vertices * sizeof(int));
 }

 int changes = 1;
 while (changes)
 {
        for (int i = 0; i < vertices; i++) for (int j = 0; j < vertices; j++) TEMP[i][j] = (i
== j) ? 0 : INF;

        changes = 0;
        for (int i = 0; i < vertices; i++)
        {
                for (int j = 0; j < vertices; j++)
                {
                        if (W[i][j] != INF && i != j)
                        {
                                for (int k = 0; k < vertices; k++)
                                {
                                        if (W[j][k] != INF && j != k)
                                        {
                                                int min = min(W[i][k], TEMP[i][k]);
                                                //TEMP[i][k] = (min < W[i][j] + W[j][k]) ? min :
W[i][j] + W[j][k], changes++;
                                                if (min <= W[i][j] + W[j][k])
                                                {
                                                        TEMP[i][k] = min;
                                                }
                                                else
                                                {
                                                        TEMP[i][k] = W[i][j] + W[j][k];
                                                        changes++;
                                                }
                                        }
                                        else TEMP[i][k] = min(W[i][k], TEMP[i][k]);
                                }
                        }
                        else TEMP[i][j] = min(W[i][j], TEMP[i][j]);
                }

                /*for (int j = 0; j < vertices; j++) std::cout << TEMP[i][j] << " ";
                std::cout << std::endl;*/
        }
        for (int i = 0; i < vertices; i++)
        {
                for (int j = 0; j < vertices; j++)
                {
                        W[i][j] = (i == j) ? 0 : TEMP[i][j];
                }
        }
 }


 return W;
}
```