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 № 4

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 максимального потока в сети.

$$S=\begin{pmatrix} 0 & 5 & 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 8 & 5 & 0 & 0 \\ 4 & 0 & 2 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 6 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 7 & 4 & 0 & 0 & 0 & 0 \end{pmatrix}$$

2. Ход работы

```
        lab 4:

max_fw  -       Поиск максимального потока в сети.

Command: read_am
Command: Введите название файла из которого считать матрицу: output.txt
Done
Command: print_am
Матрица Смежности:
0 5 0 0 0 6 0 0
0 0 0 0 0 0 1 0
0 0 0 0 8 5 0 0
4 0 2 0 0 9 0 0
0 0 0 0 0 2 6 0
0 2 0 0 0 0 2 0
0 0 0 0 0 0 0 0
6 0 7 4 0 0 0 0
Command: max_fw
Command: Введите номер вершины исхода (1 <= var <= 8): 8
Command: Введите номер вершины стока (1 <= var <= 8): 7
Максимальный поток в сети: 9
```

3. Контрольные вопросы

1. **Что такое *f*-дополняющая цепь?**

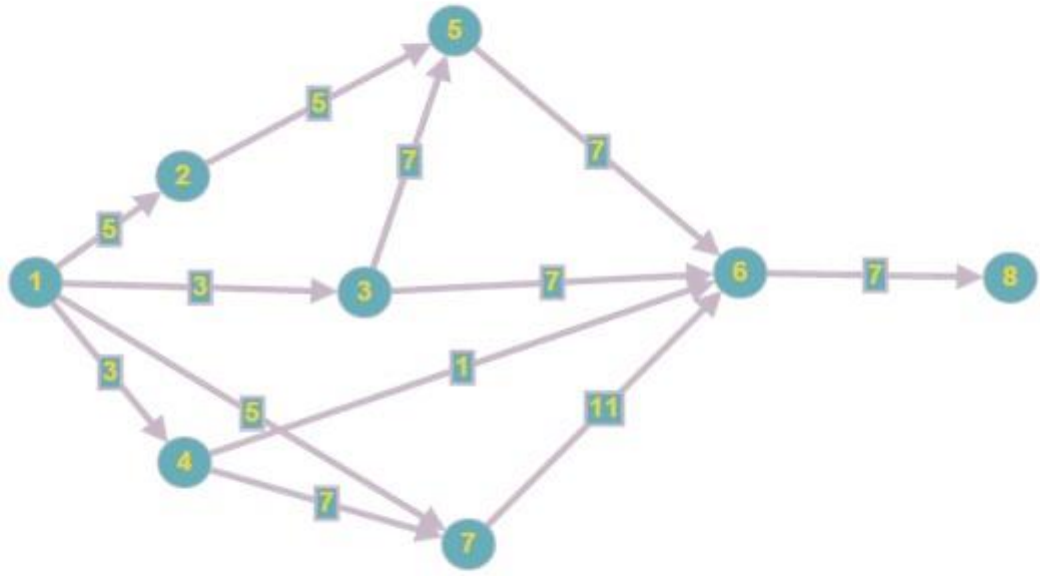
Пусть P(v,w) – цепь. Для любого её ребра e определим:

a) h(e) = c(e) – *f*(e), если e – прямая дуга;

b) h(e) = f(e), если e – обратная дуга.

Тогда h(P) = min {h(e), для всех e ϵ P}. И тогда P(s,t) является *f*-дополняющей цепью, если h(P) > 0.

2. **Приведите не тривиальный пример сети с n > 5 вершинами (под не тривиальной понимается сеть, имеющая, по меньшей мере, n+3 дуги), в которой максимальный поток проходит через цепь максимальной длины (под длиной понимается количество дуг).**



3. **Как найти максимальный поток в сети, в которой помимо пропускных способностей дуг заданы еще и пропускные способности вершин?**

Каждую вершину v с ограниченной пропускной способностью расщепляем на две вершины v_{in} и v_{out}. Все рёбра, до расщепления входящие в v, теперь входят в v_{in}. Все рёбра, до расщепления исходящие из v, теперь исходят из v_{out}. Добавляем ребро (v_{in},v_{out}) с заданной пропускной способностью. После чего применяем алгоритм для измененного графа.

4. **Как найти максимальный поток в сети с несколькими источниками или несколькими стоками?**

Если источников больше одного, добавляем новую вершину S, которую делаем единственным источником. Добавляем рёбра с бесконечной пропускной способностью от S к каждому из старых источников. Аналогично, если стоков больше одного, добавляем новую вершину T, которую делаем единственным стоком. Добавляем рёбра с бесконечной пропускной способностью от каждого из старых стоков к T. После чего применяем алгоритм для измененного графа.

4. Приложение

```
int Graph::FindPath(int head, int end, int ** MR)
{
 int ** MW = adjacency_matrix;
 int N = vertices;
 const int inf = 10000; // некоторое, условное число обозначающее бесконечность
 int* fw = (int*)calloc(N, sizeof(int));// [] ->>> [N]
 int* link = (int*)calloc(N, sizeof(int));// ->>> [N]
 int* course = (int*)calloc(N, sizeof(int));; //кладем все в очередь ->>> [N]
 int cb, ct; //Для очереди, заводим вспомогательные переменные cb,ct, где cb - указатель начала
очереди и ct - число эл-тов в очереди
 cb = 0; ct = 1; course[0] = head;
 link[end] = -1; // ставим особую метку для стока
 int i;
 int course_begin; //Вершина
 memset(fw, 0, sizeof(int)*N); // в начале из всех вершин, кроме истока, течет 0
 fw[head] = inf; // а из истока может вытечь сколько угодно
 while (link[end] == -1 && cb < ct)
 {
	// смотрим какие вершины могут быть достигнуты из начала очереди
	course_begin = course[cb];
	for (i = 0; i < N; i++)
		// проверяем можем ли мы пустить поток по ребру (course_begin,i):
	if ((MW[course_begin][i] - MR[course_begin][i])>0 && fw[i] == 0)
	{
		// если можем, то добавляем i в конец очереди
		course[ct] = i; ct++;
		link[i] = course_begin; // указываем, что в i добрались из course_begin
		// и находим значение потока текущее через вершину i
		if (MW[course_begin][i] - MR[course_begin][i] < fw[course_begin])
			fw[i] = MW[course_begin][i];
		else
			fw[i] = fw[course_begin];
	}
	cb++;// прерходим к следующей в очереди вершине
 }
 // закончили поиск пути
 if (link[end] == -1) return 0; // мы или не находим путь и выходим
 // или находим:
 // тогда fw[end] будет равен потоку который дотек по данному пути из истока в сток
 // тогда изменяем значения массива g для  данного пути на величину fw[end]
 course_begin = end;
 while (course_begin != head) // путь из стока в исток мы восстанавливаем с помощбю массива
link
 {
	MR[link[course_begin]][course_begin] += fw[end];
	course_begin = link[course_begin];
 }
 return fw[end]; // Возвращаем значение потока которое мы еще смогли пустить по графу
}

int Graph::max_fw(int head, int end)
{
 int N = vertices;
 int ** MR = (int **)malloc(N * sizeof(int *));
 for (int i = 0; i < N; i++) MR[i] = (int *)calloc(N, sizeof(int)); // по графу ничего не течет
 // инициализируем переменные:
 int max_fw = 0; // начальное значение потока
 int add_fw;//добавить в поток
 do
 {	// каждую итерацию ищем какй-либо простой путь из истока в сток
	// и какой еще поток мажет быть пущен по этому пути
	add_fw = FindPath(head, end, MR);
	max_fw += add_fw;
 } while (add_fw >0);// повторяем цикл пока поток увеличивается
 return max_fw;
}
```