1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский политехнический университет Петра Великого
3. —
4. Институт компьютерных наук и технологий
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ОТЧЕТ**

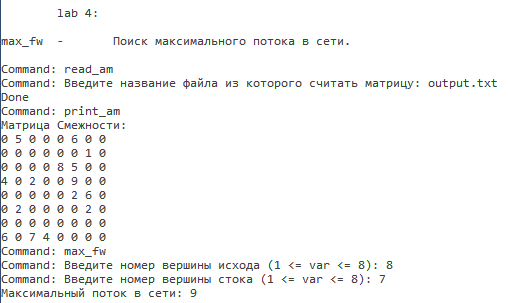
**по лабораторной работе № 4**

1. по дисциплине «Дискретная Математика»
2. Выполнил
3. студент гр. 23508/4 Е.Г. Проценко
4. Проверила
5. ассистент Д.С. Лаврова
6. Санкт-Петербург
7. 2016
8. **Формулировка задания (Вариант 7)**

Цели работы - изучение алгоритмов поиска максимального потока в сети.



1. **Ход работы**

****

1. **Контрольные вопросы**

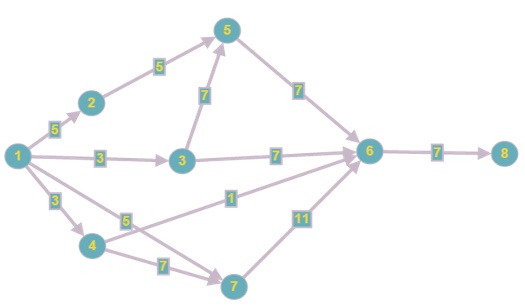
1. **Что такое *f*-дополняющая цепь?**

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

1. h(e) = c(e) – *f*(e), если е – прямая дуга;
2. h(e) = f(e), если е – 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 vin и vout. Все рёбра, до расщепления входящие в v, теперь входят в vin. Все рёбра, до расщепления исходящие из v, теперь исходят из vout. Добавляем ребро (vin,vout) 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.

1. **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;**

**}**