Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Сравнение некоторых алгоритмов поиска максимального потока

КУРСОВАЯ РАБОТА

студента 4 курса 431 группы
специальности 10.05.01 «Компьютерная безопасность»
факультета компьютерных наук и информационных технологий
Енца Михаила Владимировича

Научный руководитель
ассистент _____ Е.Н.Новокшонова
подпись, дата

Заведующий кафедрой
д.ф.-м.н., доцент _____ М. Б. Абросимов
подпись, дата

Саратов 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Теория графов – прикладная наука. Часто на практике возникают задачи, для которых довольно сложно найти решение, но если реальные объекты представить в виде графов и решить поставленную на практике задачу уже в понятиях теории графов, то найти решение часто становится заметно проще. Одной из таких задач является задача нахождения потока в транспортной сети. Эта задача имеет широкое применение во многих областях науки и жизни. Методы ее решения применяются на транспортных, коммуникационных, электрических сетях, различных физических и химических процессов, и даже для поиска Web-групп в сети Интернет. Исследования, посвященные данной задаче, проводятся во множестве крупнейших университетов мира. При использовании алгоритмов на практике очень важно, чтобы их скорость была высокой, поэтому для различных задач одной природы порой применяются различные алгоритмы, оптимизированные под определенные входные данные.

Целью данной работы является рассмотрение, реализация и сравнение некоторых алгоритмов поиска максимального потока.

В разделе «Определения и обозначения» даются основные определения и обозначения, используемые в работе.

В разделе «Потоки в сетях» даётся определение потока и его основные свойства.

В разделе «Алгоритмы поиска максимального потока» рассматриваются две известные концепции поиска максимального потока, а также три алгоритма, построенных на этих концепциях.

В разделе «Сравнение некоторых алгоритмов поиска максимального потока» показаны результаты сравнения времени работы двух реализованных алгоритмов поиска максимального потока на различных графах.

Для достижения поставленной цели проведен ряд экспериментов, также проведен анализ полученных данных для сравнения времени работы

реализованных алгоритмов – алгоритма Диница и алгоритма проталкивания предпотока (в его классической реализации). Для проведения ряда экспериментов был реализован генератор графов.

ОПРЕДЕЛЕНИЯ И ОБОЗНАЧЕНИЯ

Ориентированный граф (или *орграф*) – пара $\vec{G} = (V, \alpha)$, где V – конечное непустое множество (вершины графа), а $\alpha \subseteq V \times V$ – отношение на множестве V. Пара $(u, v) \in \alpha$ называется *дугой* орграфа с началом u концом v, где $u, v \in V$.

Две вершины u, v называются *смежными*, если какая-нибудь из двух дуг $(u, v) \in \alpha$ и $(v, u) \in \alpha$ или обе одновременно присутствуют в графе [2].

Последовательность вида $v_0(v_0, v_1)v_1(v_1, v_2)v_2 \dots v_{l-1}(v_{l-1}, v_l)v_l$, где $v_0, v_1, v_2, \dots, v_{l-1}, v_l \in V$, а $(v_0, v_1), (v_1 v_2), \dots, (v_{l-1} v_l) \in \alpha$, называется *маршрутом* (или *путём*) длины l из вершины v_0 (начало маршрута) в вершину v_l (конец маршрута).

Путь, каждая вершина которого принадлежит не более чем двум его дугам, является *простым*.

Говорят, что вершина v_l *достижима* из вершины v_0, если существует маршрут с началом в v_0 и концом в v_l [2].

Взвешенный ориентированный граф — ориентированный граф, каждой дуге которого поставлено в соответствие некое значение $w(u, v)$ (*вес дуги*).

Весом пути называется сумма весов дуг, входящих в этот путь. *Весом кратчайшего пути* из вершины u в v называется минимальный из весов путей из u в v, а если таких путей нет, то вес кратчайшего пути считается равным бесконечности.

Кратчайшим путем из u в v называется всякий путь из u в v, вес которого равен весу кратчайшего пути из u в v [4].

Петлёй называется дуга с совпадающим началом и концом [2].

Транспортная сеть $\vec{G} = (V, E)$ представляет собой ориентированный граф, в котором каждая дуга $(u, v) \in E$ имеет неотрицательный вес, называемый *пропускной способностью* $c(u, v) \geq 0$. Если $(u, v) \notin E$, предполагается, что $c(u, v) = 0$.

Говорят, что две вершины, v_1 и v_2, *односторонне связаны* в орграфе G, если существует путь либо из v_1 в v_2, либо из v_2 в v_1 [5].

Вершина орграфа, недостижимая ни из какой другой вершины, называется *истоком* [3].

Вершина орграфа, из которой недостижима ни одна другая вершина, называется *стоком* [3].

В работе будем рассматривать транспортные сети такого вида: есть один источник s, один сток t, из источника s достижимы все остальные вершины сети, а из стока t недостижима ни одна, причем такие вершины в сети единственны.

1 Потоки в сетях

Пусть $\vec{G} = (V, E)$ – транспортная сеть с функцией пропускной способности c. Потоком в $\vec{G}$ является действительная функция $f: V\ x\ V \rightarrow R$, удовлетворяющая следующим трем условиям.

- Ограничение пропускной способности: $f(u, v) \leq c(u, v)$ для всех $u, v \in V$.
- Антисимметричность: $f(u, v) = -f(u, v)$ для всех $(u, v) \in E$.
- Сохранение потока: для всех $u \in \{V \setminus \{s, t\}\}$

$$\sum_{v \in V} f(u, v) = 0$$

Количество $f(u, v)$, которое может быть положительным, нулевым или отрицательным, называется *потоком* из вершины u в вершину v. Величина потока f определяется как

$$|f| = \sum_{v \in V} f(s, v)$$

т.е. как суммарный поток, выходящий из источника.

Ограничение пропускной способности предполагает, что поток из вершины u в вершину v не превышает заданную пропускную способность дуги (u, v). Антисимметричность введена для удобства обозначения и заключается в том, что поток из вершины u в вершину v противоположен потоку в обратном направлении. Свойство сохранения потока утверждает, что суммарный поток, выходящий из вершины, не являющийся источником или стоком, равен нулю. Используя антисимметричность, можно записать свойство сохранения потока как

$$\sum_{u \in V} f(u, v) = 0$$

Для всех $v \in \{V \setminus \{s,t\}\}$, т.е. суммарный поток, входящий в вершину, отличную от источника и стока равен 0.

Если в E не присутствуют ни (u,v), ни (v,u), то между вершинами u и v нет потока, и $f(u,v) = f(v,u) = 0$.

Суммарный положительный поток определяется суммой всех положительных потоков, входящих в вершину v и задается выражением:

$$\sum_{\substack{u \in V \\ f(u,v)>0}} f(u,v)$$

Суммарный положительный поток, выходящий из некоторой вершины, определяется симметрично. *Суммарный чистый поток* в некоторой вершине равен разности суммарного положительного потока, выходящего из данной вершины, и суммарного положительного потока, входящего в нее. Одна из интерпретаций свойства сохранения потока состоит в том, что для отличной от источника и стока вершины, входящей в нее суммарный положительный поток должен быть равен выходящему суммарному положительному потоку.

Поток f^* называется *максимальным*, если для любого потока f справедливо неравенство $|f| \leq |f^*|$.

Задача о максимальном потоке формируется следующим образом: *в заданной сети $\vec{G}$ найти поток максимальной величины.*

2 Алгоритмы поиска максимального потока

Алгоритмов поиска максимального потока много, но большинство являются модификацией алгоритмов, которые представлены ниже.

2.1 Алгоритм Диница

Алгоритм Диница является частным случаем алгоритма Форда-Фалкерсона и является одним из самых применяемых на практике алгоритмов для поиска максимального потока в сети.

2.1.1 Алгоритм Форда-Фалкерсона

Алгоритм Форда-Фалкерсона решает задачу поиска максимального потока. Алгоритм является итеративным. Вначале величине потока присваивается значение 0. На каждой итерации величина потока увеличивается посредством поиска некоторого пути и последующего увеличения потока.

Остаточная сеть – это сеть, состоящая из дуг, допускающих увеличение потока. Пусть задана транспортная сеть $\vec{G} = (V, E)$ с источником s и стоком t. Пусть f – некоторый поток в $\vec{G}$. Рассмотрим пару вершин $u, v \in V$. Величина дополнительного потока, который можно направить из u в v, не превысив пропускную способность $c(u, v)$, и задается формулой:

$$c_f(u, v) = c(u, v) - f(u, v)$$

Когда поток $f(u, v)$ отрицателен, остаточная пропускная способность $c_f(u, v)$ больше, чем пропускная способность $c(u, v)$.

Для заданной транспортной сети $\vec{G} = (V, E)$ и потока f, остаточной сетью в $\vec{G}$, порожденной потоком f, является сеть $\vec{G}_f = (V, E_f)$, где $E_f = \{(u, v) \in VxV : c_f(u, v) > 0\}$.

Таким образом по каждой дуге остаточной сети, или остаточной дуге, можно направить поток, больший 0 [3].

Путь P из $v \in V$ в $w \in V$ называется *увеличивающим путем*, если для каждой дуги этого пути остаточная пропускная способность положительна [3].

Дугами E_f являются или дуги E, или обратные им. Если $f(u,v) < c(u,v)$ для некоторой дуги $(u,v) \in E$, то $c_f(u,v) = c(u,v) - f(u,v) > 0$ и $(u,v) \in E_f$. Если $f(u,v) > 0$ для некоторой дуги $(u,v) \in E$, то $f(v,u) < 0$. В таком случае $c_f(v,u) = c(v,u) - f(v,u) > 0$ и, следовательно, $(v,u) \in E_f$. Если в исходной сети нет ни дуги (u,v), ни (v,u), то $c(u,v) = c(v,u) = 0, f(u,v) = f(v,u) = 0$, и $c_f(u,v) = c_f(v,u) = 0$. Таким образом, можно сделать вывод, что дуга (u,v) может оказаться в остаточной сети только в том случае, если хотя бы одна из дуг (u,v) или (v,u) присутствует в исходной транспортной сети, поэтому $|E_f| \leq 2|E|$.

Остаточная сеть $\vec{G}_f$ является транспортной сетью со значениями пропускных способностей, заданными c_f.

Следующая лемма показывает, как поток в остаточной сети связан с потоком в исходной транспортной сети.

Лемма 1 (см. [2]). Пусть $\vec{G} = (V,E)$ – транспортная сеть с источником s и стоком t, а f – поток в $\vec{G}$. Пусть $\vec{G}_f$ – остаточная сеть в $\vec{G}$, построенная потоком f, а f' – поток в $\vec{G}_f$. Тогда сумма потоков $f + f'$, является потоком в $\vec{G}$, и величина этого потока равна $|f + f'| = |f| + |f'|$.

Доказательство. Необходимо проверить, выполняются ли ограничения антисимметричности, пропускной способности и сохранения потока.

Для подтверждения антисимметричности заметим, что для всех $u, v \in V$, справедливо $(f + f')(u,v) = f(u,v) + f'(u,v) = -f(v,u) - f'(v,u) = -\big(f(v,u) + f'(v,u)\big) = -(f + f')(v,u)$.

Покажем соблюдение ограничения пропускной способности. Заметим, что $f'(u,v) \le c_f(u,v)$ для всех $u,v \in V$. Поэтому $(f+f')(u,v) = f(u,v) + f'(u,v) \le f(u,v) + \big(c(u,v) - f(u,v)\big) = c(u,v)$.

Покажем соблюдение сохранения потока. Заметим, что для всех $u \in \{V \setminus \{s,t\}\}$ справедливо равенство

$$\sum_{v \in V} (f+f')(u,v) = \sum_{v \in V} \big(f(u,v) + f'(u,v)\big) = \sum_{v \in V} f(u,v) + \sum_{v \in V} f'(u,v) = 0 + 0 = 0$$

И наконец,

$$|f+f'| = \sum_{v \in V} (f+f')(s,v) = \sum_{v \in V} \big(f(s,v) + f'(s,v)\big) = \sum_{v \in V} f(s,v) + \sum_{v \in V} f'(s,v) = |f| + |f'|$$

■

Для заданной транспортной сети $\vec{G} = (V,E)$ и потока f увеличивающим путем p является простой путь из s в t в остаточной сети $\vec{G}_f$.

Согласно определению остаточной сети, каждая дуга (u,v) увеличивающего пути допускает некоторый дополнительный положительный поток из u в v без нарушения ограничения пропускной способности для данной дуги.

Максимальная величина, на которую можно увеличить поток вдоль каждой дуги увеличивающего пути p, называется *остаточной пропускной способностью* p и задается формулой

$$c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$$

Лемма 2 (см. [2]). Пусть $\vec{G} = (\mathrm{V}, \mathrm{E})$ – транспортная сеть, а f – некоторый поток в $\vec{G}$, и пусть p – некоторый увеличивающий путь в $\vec{G}_f$. Определим функцию $f_p: VxV \to R$ следующим образом:

$$f_p(u,v) = \begin{cases} c_f(p), \text{если } (u,v) \in p, \\ -c_f(p), \text{если } (v,u) \in p, \\ 0, \text{в противном случае.} \end{cases}$$

Тогда f_p является потоком в $\vec{G}$ и его величина составляет $|f_p| = c_f(p) > 0$.

Следствие 1 (см. [2]). Пусть $\vec{G} = (V, E)$ – транспортная сеть, а f – некоторый поток в $\vec{G}$, и пусть p – некоторый увеличивающий путь в $\vec{G}_f$. Пусть f_p определен в соответствии с уравнением представленном в лемме 2. Определим функцию $f': VxV \to R$ как $f' = f + f_p$. Тогда f' является потоком в $\vec{G}$ и имеет величину $|f'| = |f| + |f_p| > |f|$.

Доказательство. Непосредственно вытекает из лемм 1 и 2. ■

Разрезом (S, T) транспортной сети $\vec{G} = (V, E)$ называется разбиение множества V на S и $T = V - S$, такие что $s \in S$, а $t \in T$. Если f – поток, то *чистый поток* через разрез (S, T) по определению равен сумме потока, проходящего через дуги разреза и обозначается $f(S, T)$. Пропускной способностью разреза (S, T) является $c(S, T)$. *Минимальным разрезом* сети является разрез, пропускная способность котрого среди всех разрезов сети минимальна.

Лемма 3 (см. [2]). Пусть f – некоторый поток в транспортной сети $\vec{G}$ с источником s и стоком t, и пусть (S, T) – разрез $\vec{G}$. Тогда чистый поток через (S, T) равен $f(S, T) = |f|$.

Доказательство. Заметим, что согласно свойству сохранения потока $f(S - s, V) = 0$, так что

$$f(S,T) = f(S,V) - f(S,S) = f(S,V) = f(s,V) + f(S-s,V) = f(s,V) = |f|$$

■

Следствие 1 (см. [2]). Величина любого потока f в транспортной сети $\vec{G}$ не превышает пропускную способность произвольно разреза $\vec{G}$.

Доказательство. Пусть (S,T) – произвольный разрез $\vec{G}$, а f – некоторый поток. Согласно лемме 3 и ограничениям пропускной способности,

$$|f| = f(S,T) = \sum_{u\in S}\sum_{v\in T} f(u,v) \le \sum_{u\in S}\sum_{v\in T} c(u,v) = c(S,T)$$

■

Теорема 1 (см. [2]) (О максимальном потоке и минимальном разрезе). Если f – некоторый поток в транспортной сети $\vec{G} = (V,E)$ с источником s и стоком t, то следующие утверждения эквивалентны.

1. f – максимальный поток в $\vec{G}$.
2. Остаточная сеть $\vec{G}_f$ не содержит увеличивающих путей.
3. $|f| = c(S,T)$ для некоторого разреза (S,T) сети $\vec{G}$.

Доказательство. Докажем, что из первого утверждения следует второе.

Предположим противное: пусть f является максимальным потоком в $\vec{G}$, но $\vec{G}_f$ содержит увеличивающий путь p. Согласно следствию 1, суммарный поток $f + f_p$, заданный в лемме 2, является потоком в $\vec{G}$, величина которого строго больше, чем $|f|$, что противоречит предположению, что f – максимальны поток.

Докажем, что из второго утверждения следует третье.

Предположим, что $\vec{G}_f$ не содержит увеличивающего пути, т.е. $\vec{G}_f$ не содержит пути из s в t. Определим

$$S = \{v \in V: \text{в } \vec{G}_f \text{ существует путь из } s \text{ в } v\}$$

и $T = V \setminus S$. Разбиение (S,T) является разрезом: очевидно, что $s \in S$, а $t \notin S$, поскольку в $\vec{G}_f$ не существует пути из s в t. Для каждой пары вершин $u \in S, v \in$

T справедливо соотношение $f(u,v) = c(u,v)$, поскольку в противном случае $(u,v) \in E_f$ и v следует поместить во множество S. Следовательно, согласно лемме 4. $|f| = f(S,T) = c(S,T)$.

Докажем, что из третьего утверждения следует первое.

Согласно следствию 1 леммы 3, $|f| \leq c(S,T)$ для всех разрезов (S,T), поэтому из условия $|f| = c(S,T)$ следует, что f – максимальный поток. ■

Алгоритм Форда-Фалкерсона

Шаг 1. Для каждой дуги $(u,v) \in E[\vec{G}]$ положим $f(v,u) = 0,\ f(u,v) = 0$

Шаг 2. Пока существует путь p из s в t в остаточной сети $\vec{G}_f$: выполнять шаги 3-4

Шаг 3. Присваиваем $c_f(p)$ минимальное значение из $c_f(u,v)$, таких что $(u,v) \in p$

Шаг 4. Для каждой дуги $(u,v) \in p$: выполнять $f(u,v) = f(u,v) + c_f(p), f(v,u) = -f(u,v)$

На 1 шаге поток f инициализируется значением 0. На шагах 2-4 выполняется неоднократный поиск увеличивающего пути p в $\vec{G}_f$, и поток f вдоль пути p увеличивается на остаточную пропускную способность $c_f(p)$. Когда увеличивающих путей больше нет, поток f является максимальным.

Таким образом, при выполнении каждой итерации метода Форда-Фалкерсона находим некоторый увеличивающий путь p, и поток f вдоль каждой дуги данного пути увеличивается на величину остаточной пропускной способности $c_f(p)$. Простая реализация алгоритма вычисляет максимальный поток в графе $\vec{G} = (V,E)$ путем обновления потока $f(u,v)$ между каждой парой вершин u и v, соединенных дугой. Если вершины u и v не связаны дугой, неявно предполагается, что $f(u,v) = 0$. Предполагается, что значения пропускных

способностей задаются вместе с графом и $c(u,v) = 0$, если $(u,v) \notin E$. Остаточная пропускная способность $c_f(u,v)$ вычисляется по формуле

$$c_f(u,v) = c(u,v) - f(u,v).$$

Время выполнения алгоритма, указанного выше, зависит от того, как именно выполняется поиск увеличивающего пути p.

На практике задача поиска максимального потока чаще всего возникает в целочисленной постановке. Если пропускные способности – рациональные числа, можно использовать соответствующее масштабирование, которое сделает их целыми. Тогда реализация алгоритма Форда-Фалкерсона имеет время работы $O(|E| * |f^*|)$, где f^* – максимальный поток, найденный данным алгоритмом. Анализ производится следующим образом. Выполнение шага 1 занимает время $O(|E|)$. Цикл на шагах 2-4 выполняется не более $|f^*|$ раз, поскольку величина потока за каждую итерацию увеличивается по крайней мере на одну единицу [7].

2.1.2 Алгоритм Диница

Если в качестве увеличивающего пути p выбирается кратчайший путь из s в t в остаточной сети, где каждая дуга имеет единичную длину (вес), то такая реализация алгоритма Форда-Фалкерсона называется алгоритмом Диница. Время выполнения алгоритма Диница составляет $O(|V| * |E|^2)$.

Анализ времени выполнения алгоритма зависит от расстояний между вершинами остаточной сети $\vec{G}_f$. Обозначим длину кратчайшего пути из вершины u в v в остаточной сети $\vec{G}_f$, где каждая дуга имеет единичную длину, обозначается как $\delta_f(u,v)$.

Лемма 4 (см. [2]). Если для некоторой транспортной сети $\vec{G} = (V,E)$ с источником s и стоком t выполняется алгоритм Диница, то для всех вершин $v \in \{V \setminus \{s,t\}\}$ длина кратчайшего пути $\delta_f(s,v)$ в остаточной сети $\vec{G}_f$ монотонно возрастает с каждым увеличением потока.

Доказательство. Предположим, что для некоторой вершины $v \in \{V \setminus \{s,t\}\}$ существует такое увеличение потока, которое приводит к уменьшению длины кратчайшего пути из s в v, и покажем, что это предположение приводит к противоречию. Пусть f – поток, который был непосредственно перед первым увеличением, которое привело к уменьшению длины некого кратчайшего пути, а f' – поток сразу после этого увеличения. Пусть v – вершина с минимальной длиной кратчайшего пути $\delta'_f(s,v)$, которая уменьшилась в результате увеличения потока, т.е. $\delta'_f(s,v) < \delta_f(s,v)$. Пусть $p = s \dashrightarrow u \longrightarrow v$ – кратчайший путь от s к v в $\vec{G}'_f$, такой что $(u,v) \in E'_f$ и

$$\delta'_f(s,u) = \delta'_f(s,v) - 1$$

Исходя из того, как мы выбирали v, можно утверждать, что длина пути до вершины u не уменьшилась, т.е.

$$\delta'_f(s,u) \geq \delta_f(s,u).$$

В таком случае $(u,v) \notin E_f$. Если $(u,v) \in E_f$, тогда справедливо следующее:

$$\delta_f(s,v) \leq \delta_f(s,u) + 1 \leq \delta'_f(s,u) + 1 = \delta'_f(s,v)$$

Что противоречит предположению $\delta'_f(s,v) \leq \delta_f(s,v)$.

Рассмотрим, как может получиться, что $(u,v) \notin E_f$, но $(u,v) \in E'_f$. Увеличение должно привести к возрастанию потока из v в u. Алгоритм Диница всегда увеличивает поток вдоль кратчайших путей, поэтому последней дугой кратчайшего пути из s в u в $\vec{G}_f$ является дуга (v,u). Следовательно, $\delta_f(s,v) = \delta_f(s,u) - 1 \leq \delta'_f(s,u) - 1 = \delta'_f(s,v) - 2$, что противоречит предположению $\delta'_f(s,v) < \delta_f(s,v)$, а значит, наше предположение о существовании вершины v не верно. ■

Теорема 2 (см. [2]). Если для некоторой транспортной сети $\vec{G} = (V, E)$ с источником s и стоком t выполняется алгоритм Диница, то общее число увеличений потока, выполняемое данным алгоритмом, составляет $O(|V| * |E|)$.

Доказательство. Назовем дугу (u, v) остаточной сети $\vec{G}_f$ критической для увеличивающего пути p, если остаточная пропускная способность p равна остаточной пропускной способности дуги (u, v), т.е. если $c_f(p) = c_f(u, v)$. После увеличения потока вдоль некоторого увеличивающего пути, все критические дуги пути исчезают из остаточной сети. Кроме того, по крайней мере одна дуга любого увеличивающего пути должна быть критической. Теперь покажем, что каждая дуга из $|E|$ дуг может становиться критической не более $\frac{|V|}{2} - 1$ раз.

Пусть u и v – вершины из множества вершин V, соединенные некоторой дугой из множества E. Поскольку увеличивающие пути – это кратчайшие пути, то когда дуга (u, v) становится критической первый раз, справедливо равенство

$$\delta_f(s, v) = \delta_f(s, u) + 1.$$

После того, как поток увеличен, дуга (u, v) исчезает из остаточной сети. Она не может появиться в другом увеличивающем пути, пока не будет уменьшен поток из u в v, а это может произойти только в том случае, если на некотором увеличивающем пути встретится дуга (v, u). Если в этот момент поток в сети $\vec{G}$ составляет f', справедливо следующее равенство:

$$\delta'_f(s, u) = \delta'_f(s, v) + 1.$$

Поскольку, согласно лемме 4, $\delta_f(s, v) \leq \delta'_f(s, v)$, получаем: $\delta'_f(s, u) = \delta'_f(s, v) + 1 \geq \delta_f(s, v) + 1 = \delta_f(s, u) + 2$.

Следовательно, за время, прошедшее с момента, когда дуга (u, v) была критической, до момента, когда она становится критической в следующий раз, расстояние до u от источника увеличивается не менее чем на 2. Расстояние до u от источника в начальный момент было не меньше 0. Среди промежуточных

вершин на кратчайшем пути из s в u не могут находиться s, u или t (поскольку наличие дуги (u, v) в кратчайшем пути подразумевает, что $u \neq t$). Следовательно, к тому моменту, когда вершина u станет недостижимой из источника, расстояние до нее будет не более $|V| - 2$. Таким образом, дуга (u, v) может стать критической не более $\frac{|V|-2}{2} = \frac{|V|}{2} - 1$ раз. Поскольку в остаточном графе имеется не более $O(|E|)$ пар вершин, которые могут быть соединены дугами, общее количество критических дуг в ходе выполнения алгоритма Диница равно $O(|V| * |E|)$. Каждый увеличивающий путь содержит по крайней мере одну критическую дугу, следовательно, теорема доказана. ■

Если увеличивающий путь находится посредством поиска в ширину, каждую итерацию алгоритм Форда-Фалкерсона можно выполнить за время $O(|E|)$, следовательно, суммарное время выполнения алгоритма Диница составляет $O(|V| * |E|^2)$ [8].

2.2 Алгоритм проталкивания предпотока

Многие наиболее асимптотически быстрые алгоритмы поиска максимального потока принадлежат данному классу, и на этом методе основаны реальные реализации алгоритма поиска максимального потока. С помощью методов проталкивания предпотока можно решать и другие связанные с потоками задачи, например, задачу поиска потока с минимальными затратами [6]. Время работы простой реализации $O(|V|^2 * |E|)$, также существует усовершенствованная версия алгоритма, работающая за время $O(|V|^3)$.

Алгоритм проталкивания предпотока обрабатывают вершины по одной, рассматривая только соседей данной вершины в остаточной сети. Алгоритмы проталкивания предпотока не обеспечивают в ходе своего выполнения свойство сохранения потока. При этом они поддерживают предпоток, который представляет собой функцию $f: VxV \to R$, обладающую свойством антисимметричности, удовлетворяющую ограничениям пропускной

способности и следующему ослабленному условию сохранения потока: $f(V,u) \geq 0$ для всех вершин $u \in \{V \setminus \{s\}\}$. Это количество называется *избыточным потоком*, входящим в вершину u, и обозначается

$$e(u) = f(V,u)$$

Вершина $u \in \{V \setminus \{s,t\}\}$, называется *переполненной*, если $e(u) > 0$.

2.2.1 Основные операции

В алгоритме проталкивая предпотока выполняются две основные операции: проталкивание избытка потока от вершины к одной из соседних с ней вершин и подъем вершины.

Пусть $\vec{G} = (V,E)$ транспортная сеть с источником s и стоком t, а f – некоторый предпоток в $\vec{G}$. Функция $h: V \to N$ является функцией высоты, если $h(s) = |V|, h(t) = 0$ и

$$h(u) \leq h(v) + 1$$

для любой остаточной дуги $(u,v) \in E_f$. Сразу же можно сформулировать следующую лемму.

Лемма 5 (см. [2]). Пусть $\vec{G} = (V,E)$ – транспортная сеть, а f – некоторый предпоток в $\vec{G}$, и пусть h – функция высоты, заданная на множестве V. Для любых двух вершин $u, v \in V$ справедливо следующее утверждение: если $h(u) > h(v) + 1$, то (u,v) не является дугой остаточного графа.

Основная операция проталкивания может применяться тогда, когда u является переполненной вершиной, $c_f(u,v) > 0$ и $h(u) = h(v) + 1$. Предполагается, что остаточные пропускные способности при заданных f и c можно вычислить за фиксированное время. Излишний поток, хранящийся в вершине u, поддерживается в виде атрибута $e[u]$, а высота вершины u – в виде атрибута $h[u]$. Выражение $d_f(u,v)$ – это временная переменная, в которой хранится количество потока, которое можно протолкнуть из u в v.

Операция проталкивания

Условия применения: u переполнена, $c_f(u,v) > 0$, и $h[u] = h[v] + 1$.

Шаг 1. Количество потока равно минимуму из $e[u]$ и $c_f(u,v)$: $d_f(u,v) = \min(e[u], c_f(u,v))$

Шаг 2. Величина потока из u в v увеличивается на возможное количество потока, которое можно протолкнуть из u в v: $f(u,v) = f(u,v) + d_f(u,v)$

Шаг 3. Величина потока из v в u становится равна величине потока из u в v, но с отрицательным знаком: $f(v,u) = -f(u,v)$

Шаг 4. Избыток вершины u уменьшается на величину возможного количества потока, которое можно протолкнуть из u в v: $e[u] = e[u] - d_f(u,v)$

Шаг 5. Избыток вершины v увеличивается на величину возможного количества потока, которое можно протолкнуть из u в v: $e[v] = e[v] + d_f(u,v)$

Операция проталкивания работает следующим образом. Предполагается, что вершина u имеет положительный избыток $e[u]$ и остаточная пропускная способность дуги (u,v) положительна. Тогда можно увеличить поток из u в v на величину равную минимуму из $e[u]$ и $c_f(u,v)$, при этом избыток $e[u]$ не становится отрицательным и не будет превышена пропускная способность $c(u,v)$. Если функция f является предпотоком перед применением операции проталкивания, она останется предпотоком и после ее применения.

Операция проталкивания называется *проталкиванием* из u к v. Если операция проталкивания применяется к некоторой дуге (u,v), выходящей из вершины u, будем говорить, что операция проталкивания применяется к u. Если в результате дуга (u,v) становится насыщенной (после проталкивания $c_f(u,v) = 0$), то это *насыщающее проталкивание*, в противном случае это *ненасыщающее проталкивание*. Если дуга насыщена, она не входит в остаточную сеть. Один из результатов ненасыщающего проталкивания характеризует следующая лемма.

Лемма 6 (см. [2]). После ненасыщающего проталкивания из u в v вершина u более не является переполненной.

Доказательство. Поскольку проталкивание ненасыщающее, количество переданного потока должно быть равно величине $e[u]$ непосредственно перед проталкиванием. Поскольку избыток $e[u]$ уменьшается на эту величину, после проталкивания он становится равным 0. ■

Основная операция подъема применяется, если вершина u переполнена и $h[u] \leq h[v]$ для всех дуг $(u, v) \in E_f$. Иными словами, переполненную вершину u можно подвергнуть подъему, если все вершины v, для которых имеется остаточная пропускная способность от u к v, расположены не ниже u, так что протолкнуть поток из u нельзя. Ни источник s, ни сток t нельзя подвергать подъему.

Операция подъема

Условие применения: u переполнена и для всех $v \in V$, таких что $(u, v) \in E_f, h[u] \leq h[v]$.

Шаг 1. $h[u]$ становится равна минимальной высоте из всех дуг $(u, v) \in E_f$ вершины u плюс 1

Когда вызывается операция подъема, говорим, что вершина u подвергается подъему. Заметим, что когда производится подъем u, остаточная сеть E_f должна содержать хотя бы одну дугу, выходящую из u, чтобы минимизация на шаге 1 производилась по непустому множеству. Это свойство вытекает из предположения, что вершина u переполнена. Поскольку $e[u] > 0$, имеем $e[u] = f(V, u) > 0$ и, следовательно, должна существовать по крайней мере одна вершина v, такая что $f(v, u) > 0$. Но тогда

$$c_f(u, v) = c(u, v) - f(u, v) = c(u, v) + f(v, u) > 0,$$

Откуда вытекает, что $(u, v) \in E_f$. Таким образом, операция подъема дает u наибольшую высоту, допускаемую наложенными на функцию высоты ограничениями.

2.2.2 Универсальный алгоритм.

Универсальный алгоритм проталкивания предпотока использует следующую процедуру для создания начального предпотока в транспортной сети:

Алгоритм создания начального потока в транспортной сети

Шаг 1. Для каждой вершины $u \in V[G]$: выполнять шаги 2-3

Шаг 2. Высоте вершины u присваивается значение 0: $h[u] = 0$

Шаг 3. Излишнему поток вершины u присваивается значение 0: $e[u] = 0$

Шаг 4. Для каждой дуги $(u, v) \in E|G|$: выполнять шаги 5-6

Шаг 5. Потоку из вершины u в v присваивается значение 0: $f[u, v] = 0$

Шаг 6. Потоку из вершины v в u присваивается значение 0: $f[v, u] = 0$

Шаг 7. Высоте истока присваивается мощность множества вершин в сети $\vec{G}$: $h[s] = |V[G]|$

Шаг 8. Для каждой вершины u смежной с s: выполнять шаги 9-12

Шаг 9. Величине потока из s в u присвоить значение пропускной способности из s в u: $f[s, u] = c(s, u)$

Шаг 10. Величине потока из u в s присвоить значение пропускной способности из s в u с отрицательным знаком: $f[u, s] = -c(s, u)$

Шаг 11. Величине излишнего потока u присвоить значение пропускной способности из s в u: $e[u] = c(s, u)$

Шаг 12. Из величины излишнего потока s вычесть значение пропускной способности из s в u: $e[s] = e[s] - c(s, u)$

Алгоритм, указанный выше, создает начальный предпоток f, определяемый формулой

$$f[u,v] = \begin{cases} c(u,v), \text{если } u = s \\ -c(v,u), \text{если } v = s \\ 0, \text{в противном случае.} \end{cases}$$

Это, действительно, функция высоты, поскольку единственной дугой (u,v), для которых $h[u] > h[v] + 1$, являются дуги, для которых $u = s$, и эти дуги заполнены, а это значит, что их нет в остаточной сети.

Инициализация, за которой следует ряд операций проталкивания и подъема, выполняемых без определенного порядка, образует алгоритм:

Алгоритм проталкивания предпотока

Шаг 1. Создать начальный предпоток алгоритмом инициализации

Шаг 2. Пока можно выполнить одну из операций проталкивания или подъема: выполнять шаг 3

Шаг 3. Выбрать операцию проталкивания или подъема и выполнить её

Следующая лемма утверждает, что до тех пор, пока существует хотя бы одна переполненная вершина, применима хотя бы одна из этих операций.

Лемма 7 (см. [2]). Пусть $\vec{G} = (V,E)$ – транспортная сеть с источником s и стоком t, f – предпоток, а h – некоторая функция высоты для f. Если u – некоторая переполненная вершина, то к ней можно применить или операцию проталкивания, или операцию подъема.

Доказательство. Для любой остаточной дуги (u,v) выполняется соотношение $h(u) \leq h(v) + 1$, поскольку h – функция высоты. Если к u не применима операция проталкивания, то для всех остаточных дуг (u,v) должно выполняться условие $h(u) < h(v) + 1$, откуда следует, что $h(u) \leq h(v)$. В этом случае к u можно применить операцию подъема. ■

Чтобы показать, что универсальный алгоритм проталкивания предпотока позволяет решить задачу максимального потока, сначала докажем, что после его завершения предпоток f является максимальным потоком. Затем докажем, что алгоритм завершается. Начнем с рассмотрения некоторых свойств функции высоты h.

Лемма 8 (см. [2]). При выполнении алгоритма проталкивания предпотока над транспортной сетью $\vec{G} = (V, E)$, для любой вершины $u \in V$ ее высота $h[u]$ никогда не уменьшается. Более того, всякий раз, когда к вершине u применяется операция подъема, ее высота $h[u]$ увеличивается как минимум на 1.

Доказательство. Поскольку высота вершины меняется только при выполнении операции подъема, достаточно доказать второе утверждение леммы. Если вершина u должна подвергнуться подъему, то для всех вершин v, таких что $(u, v) \in E_f$, выполняется условие $h[u] \leq h[v]$. Таким образом, $h[u] < 1 + \min\{h[v]: (u, v) \in E_f\}$, и операция должна увеличить значение $h[u]$. ■

Лемма 9 (см. [2]). Пусть $\vec{G} = (V, E)$ – транспортная суть с источником s и стоком t. Во время выполнения алгоритма проталкивания предпотока над сетью $\vec{G}$ атрибут h сохраняет свойство функции высоты.

Доказательство. Доказательство проводится индукцией по числу выполненных основных операций. h является функцией высоты.

Утверждается, что если h – функция высоты, то после выполнения операции подъема она останется функцией высоты. Если посмотреть на остаточную дугу $(u, v) \in E_f$, выходящее из u, то операция подъема гарантирует, что после ее выполнения $h[u] \leq h[v] + 1$. Рассмотрим теперь некоторую остаточную дугу (w, u), входящее в u. Согласно лемме 8, $h[w] \leq h[u] + 1$ перед выполнением операции подъема; следовательно, после ее выполнения $h[w] < h[u] + 1$. Таким образом, операция подъема оставляет h функцией высоты.

Теперь рассмотрим операцию проталкивания. Данная операция может добавить дугу (v,u) к E_f или удалить дугу (u,v) из E_f. В первом случае имеем $h[v] = h[u] - 1 < h[u] + 1$, так что h остается функцией высоты. Во втором случае удаление дуги (u,v) из остаточной сети приводит к удалению соответствующего ограничения, так что h по-прежнему остается функцией высоты. ■

Следующая лемма характеризует важное свойство функции высоты.

Лемма 10 (см. [2]). Пусть $\vec{G} = (V,E)$ – транспортная сеть с источником s и стоком t, f – предпоток в $\vec{G}$, а h – функция высоты, определенная на множестве V. Тогда не существует пути из источника s к стоку t в остаточной сети $\vec{G}_f$.

Доказательство. Предположим, что в $\vec{G}_f$ существует некоторый путь $p = \{v_0, v_1, \dots, v_k\}$ из s в t, где $v_0 = s$, а $v_k = t$, и покажем, что это приводит к противоречию. Без потери общности можно считать, что p – простой путь, так что $k < |V|$. Для $i = \overline{0, k-1}$, дуги $(v_i, v_{i+1}) \in E_f$. Поскольку h – функция высоты, для $i = \overline{0, k-1}$ справедливы соотношения $h(v_i) \leq h(v_{i+1})$. Объединяя эти неравенства вдоль пути p, получим, что $h(s) \leq h(t) + k$. Но поскольку $h(t) = 0$, получаем $h(s) \leq k < |V|$, что противоречит требованию $h(s) = |V|$ к функции высоты. ■

Покажем, что после завершения универсального алгоритма проталкивания предпотока вычисленный алгоритмом предпоток является максимальным потоком.

Теорема 3 (см. [2]). (О корректности универсального алгоритма проталкивания предпотока). Если алгоритм проталкивания предпотока, выполняемый над сетью $\vec{G} = (V < E)$ с источником s и стоком t, завершается, то вычисленный им предпоток f является максимальным потоком в $\vec{G}$.

Доказательство. Используем следующий инвариант цикла: всякий раз, когда производится проверка условия цикла на 2 шаге алгоритма проталкивания предпотока, f является предпотоком.

Инициализация. Алгоритм создания начального потока делает f предпотоком.

Сохранение. Внутри цикла на шаге 2 выполняется только операции проталкивания и подъема. Операции подъема влияют только на атрибуты высоты, но не на величину потока, следовательно, от них не зависит, будет ли f предпотоком. Анализируя работу операции проталкивания, мы доказали, что, если f является предпотоком перед выполнением операции проталкивания, он останется предпотоком и после ее выполнения.

Завершение. По завершению процедуры каждая вершина из множества $\{V \setminus \{s,t\}\}$ должна иметь избыток, равный 0, поскольку из лемм 7 и 9 и инварианта, что f всегда останется предпотоком, вытекает, что переполненных вершин нет. Следовательно, f является потоком. Поскольку h – функция высоты, согласно лемме 10 не существует пути из s в t в остаточной сети $\vec{G}_f$. Согласно теореме 1 о максимальном потоке и минимальном разрезе, f является максимальным потоком. ■

Лемма 11 (см. [2]). Пусть $\vec{G} = (V, E)$ – транспортная сеть с источником s и стоком t, а f – предпоток в $\vec{G}$. Тогда для любой переполненной вершины u существует простой путь из u в s в остаточной сети $\vec{G}_f$.

Доказательство. Пусть u – некоторая переполненная вершина, и пусть $U = \{v$: в $\vec{G}_f$ существует простой путь из u в $v\}$. Предположим, что $\mathrm{s} \notin \mathrm{U}$, и покажем, что это приведет к противоречию. Обозначим $\overline{U} = V - U$.

Утверждение, что для каждой пары вершин $w \in \overline{U}$ и $v \in U$ выполняется соотношение $f(w,v) \leq 0$. Если $f(w,v) > 0$, то $f(v,w) < 0$, откуда в свою очередь вытекает, что $c_f(v,w) = c(v,w) - f(v,w) > 0$. Следовательно,

существует дуга $(v, w) \in E_f$ и существует простой путь вида $u \dashrightarrow v \rightarrow w$ в остаточной сети G_f, что противоречит тому, как мы выбирали w.

Таким образом, должно выполняться неравенство $f(\overline{U}, U) \leq 0$, поскольку каждое слагаемое в этом неявном суммировании неположительное, и, следовательно, $e(U) = f(V, U) = f(\overline{U}, U) + f(U, \overline{U}) = f(\overline{U}, U) \leq 0$.

Излишки неотрицательны для всех вершин из множества $\{V \setminus \{s\}\}$, поскольку предположили, что $U \subseteq \{V \setminus \{s\}\}$, то для всех вершин $v \in U$ должно выполняться $e(v) = 0$. В частности, $e(u) = 0$, что противоречит предположению о том, что u переполнена. ■

Следующая лемма устанавливает границы высот вершин, а вытекающее из нее следствие устанавливает предел общего числа выполненных операций подъема.

Лемма 12 (см. [2]). Пусть $G = (V, E)$ – транспортная сеть с источником s и стоком t. В любой момент в процессе выполнения алгоритма проталкивания предпотока в сети $\vec{G}$ для всех вершин $u \in V$ выполняется соотношение $h[u] \leq 2|V| - 1$.

Доказательство. Высота источника s и стока t никогда не изменяется, поскольку эти вершины по определению не переполняются. Таким образом, всегда $h[s] = |V|$ и $h[t] = 0$, и обе не превышают $2|V| - 1$.

Рассмотрим теперь произвольную вершину $u \in \{V \setminus \{s, t\}\}$. Изначально $h[u] = 0 \leq 2|V| - 1$. Покажем, что после каждого подъема неравенство $h[u] \leq 2|V| - 1$ остается справедливым. При подъеме вершины u она является переполненной и, согласно лемме 11, имеется простой путь p из u в s в $\vec{G}_f$. Пусть $p = \{v_0, \dots, v_k\}$, где $v_0 = u$, а $v_k = s$, и $k \leq |V| - 1$, поскольку p – простой путь. Для $i = \overline{0, k-1}$ имеем $(v_i, v_{i+1}) \in E_f$, следовательно, $h[v_i] \leq h[v_{i+1}] + 1$ согласно лемме 9. Расписав неравенства для всех составляющих пути p, получаем $h[u] = h[v_0] \leq h[v_k] + k \leq h[s] + (|V| - 1) = 2|V| - 1$. ■

Следствие 1 (см. [2]) (верхний предел числа подъемов). Пусть $\vec{G} = (V, E)$ – транспортная сеть с источником s и стоком t. Тогда в процессе выполнения алгоритма проталкивания предпотока в $\vec{G}$ число подъемов не превышает $2|V| - 1$ для одной вершины, а их общее количество не более $(2|V| - 1)(|V| - 2) < 2|V|^2$.

Доказательство. Во множестве $\{V \setminus \{s, t\}\}$ только $|V| - 2$ вершин могут быть подняты. Пусть $u \in \{V \setminus \{s, t\}\}$. Операция подъема увеличивает высоту $h[u]$. Значение $h[u]$ первоначально равно 0 и, согласно лемме 12, возрастает не более чем на $2|V| - 1$. Таким образом, каждая вершина $u \in \{V \setminus \{s, t\}\}$ подвергается подъему не более $2|V| - 1$ раз, а общее число выполненных подъемов не превышает $(2|V| - 1)(|V| - 2) < 2|V|^2$. ■

Лемма 12 также помогает определить границу количества насыщающих проталкиваний.

Лемма 13 (см. [2]) (Граница количества насыщающих проталкиваний). В процессе выполнения алгоритма проталкивания предпотока для любой транспортной сети $\vec{G} = (V, E)$ число насыщающих проталкиваний меньше, чем $2|V||E|$.

Доказательство. Для любой пары вершин $u, v \in V$ рассмотрим насыщающие проталкивания от u к v и от v к u, и назовем их насыщающими проталкиванием между u и v. Если есть хотя бы одно такое проталкивание, то хотя бы одна из дуг (u, v) и (v, u) является дугой в E. Теперь предположим, что произошло насыщающее проталкивание из u в v. В этот момент $h[v] = h[u] - 1$. Чтобы позднее могло произойти еще одно проталкивание из u в v, алгоритм сначала должен протолкнуть поток из v в u, что невозможно до тех пока не будет выполнено условие $h[v] = h[u] + 1$. Поскольку $h[u]$ никгда не уменьшается, для того чтобы выполнялось условие $h[v] = h[u] + 1$, значение $h[v]$ должно увеличиться по меньшей мере на 2. Аналогично, $h[u]$ должно увеличиться между последовательными насыщающими проталкиваниями из v в u как минимум на

2. Высота изначально принимает значение 0 и, согласно лемме 12, никогда не превышает $2|V| - 1$, откуда следует, что количество раз, когда высота вершины может увеличить на 2, меньше $2|V|$ насыщающих проталкиваний между u и v. Умножив это число на число дуг, получим, что общее число насыщающих проталкиваний меньше, чем $2|V| * |E|$. ■

Следующая лемма устанавливает границу числа ненасыщающих проталкиваний в обобщенном алгоритме проталкивания предпотока.

Лемма 14 (см. [2]) (Граница количества ненасыщающих проталкиваний). В процессе выполнения алгоритма проталкивания предпотока для любой транспортной сети $\vec{G} = (V, E)$ число ненасыщающих проталкиваний меньше $4|V|^2(|V| + |E|)$.

Доказательство. Определим потенциальную функцию $\Phi = \sum_{v:e(v)>0} h[v]$. Изначально $\Phi = 0$ и значение Φ может изменяться после каждого подъема, насыщающего и ненасыщающего проталкивания. Найдем предел величины, на которую насыщающие проталкивания и подъемы могут увеличивать Φ. Затем покажем, что каждое ненасыщающее проталкивание должно увеличивать Φ. Затем покажем, что каждое ненасыщающее проталкивание должно уменьшать Φ как минимум на 1, и используем эти оценки для определения верхней границы числа ненасыщающих проталкиваний.

Рассмотрим два пути увеличения Φ. Во-первых, подъем вершины u увеличивает Φ менее чем на $2|V|$, поскольку множество, для которого вычисляется сумма, остается прежним, а подъем не может увеличить высоту вершины u больше, чем ее максимально возможная высота, которая составляет не более $2|V| - 1$ согласно лемме 12. Во-вторых, насыщающее проталкивание из вершины u в вершину v увеличивает Φ менее чем на $2|V|$, поскольку никаких изменений высот при этом не происходит, и только вершина v, высота которой не более $2|V| - 1$, может стать переполненной.

Теперь покажем, что ненасыщающее проталкивание из u в v уменьшает Ф не менее чем на 1. Перед ненасыщающим проталкиванием вершина u была переполненной, а v могла быть переполненной или непереполненной. Согласно лемме 6, после этого проталкивания u больше не является переполненной. Кроме того, после данного проталкивания v должна быть переполненной, если только она не является источником. Следовательно, потенциальная функция Ф уменьшилась ровно на $h[u]$, а увеличилась на 0 или на $h[v]$. Поскольку $h[u] - h[v] = 1$, в итоге потенциальная функция уменьшается как минимум на 1.

Таким образом, в ходе выполнения алгоритма увеличение Ф просиходит благодаря подъемам и насыщающим проталкиваниям; согласно следствию 1 и лемме 13, это увеличение ограничено, и составляет менее $(2|V|)(2|V|^2) + (2|V|)(2|V| * |E|) = 4|V|^2(|V| + |E|)$. Поскольку $\Phi \geq 0$, суммарная величина уменьшения и, следовательно, общее число ненасыщающих проталкиваний меньше, чем $4|V|^2(|V| + |E|)$. ■

Определив границу числа подъемов, насыщающего проталкивания и ненасыщающего проталкивания, заложили основу дальнейшего анализа алгоритма проталкивания предпотока, а также любых других алгоритмов, основанных на методе проталкивания предпотока.

Теорема 4 (см. [2]). При выполнении алгоритма проталкивания предпотока для любой транспортной сети $\vec{G} = (V, E)$ число основных операций составляет $O(|V|^2 * |E|)$.

Доказательство. Непосредственно вытекает из уже доказанных следствия 1 и лемм 13 и 14. ■

3 Сравнение некоторых алгоритмов поиска максимального потока

Для сравнения некоторых алгоритмов поиска максимального потока были реализованы указанные выше алгоритмы: алгоритм Диница и алгоритм проталкивания предпотока, реализован генератор графов для проведения серии экспериментов на различных графах.

Программа была написана на языке Python 3.6 и имеет графический интерфейс, для построения которого использованы модули PyQt5, matplotlib (см. [10]) и networkx (см. [9]). Полный листинг программы представлен в приложении А и в приложении Б.

Генератор графов

Генератор имеет 5 возможных параметров, которые соответственно задаются в поле «параметры» интерфейса программы. Полный листинг генератора графов представлен в приложении Б.

Параметр «-n» отвечает за количество вершин в сгенерированном графе, если параметр не указывается, то он выбирается случайно в интервале $[2, 20]$. Параметр «-m» отвечает за количество дуг, если параметр не указывается, то он выбирается случайно в интервале $[n, n * (n - 1)]$, где n – количество вершин в графе. Параметр «-cup» отвечает за установку фиксированной пропускной способности, если параметр не указывается, то пропускная способность для каждой дуги выбирается случайно в интервале $[1, max\_cup]$. Параметр «-max_cup» отвечает за максимально возможную пропускную способность, если параметр не указывается, то max_cup имеет значение по умолчанию равное 20. Параметр «-file» отвечает за считывание графа из файла, если параметр указан, то все остальные параметры игнорируются и граф считывается из указанного файла. На рисунке 1 представлен пример установки параметров.

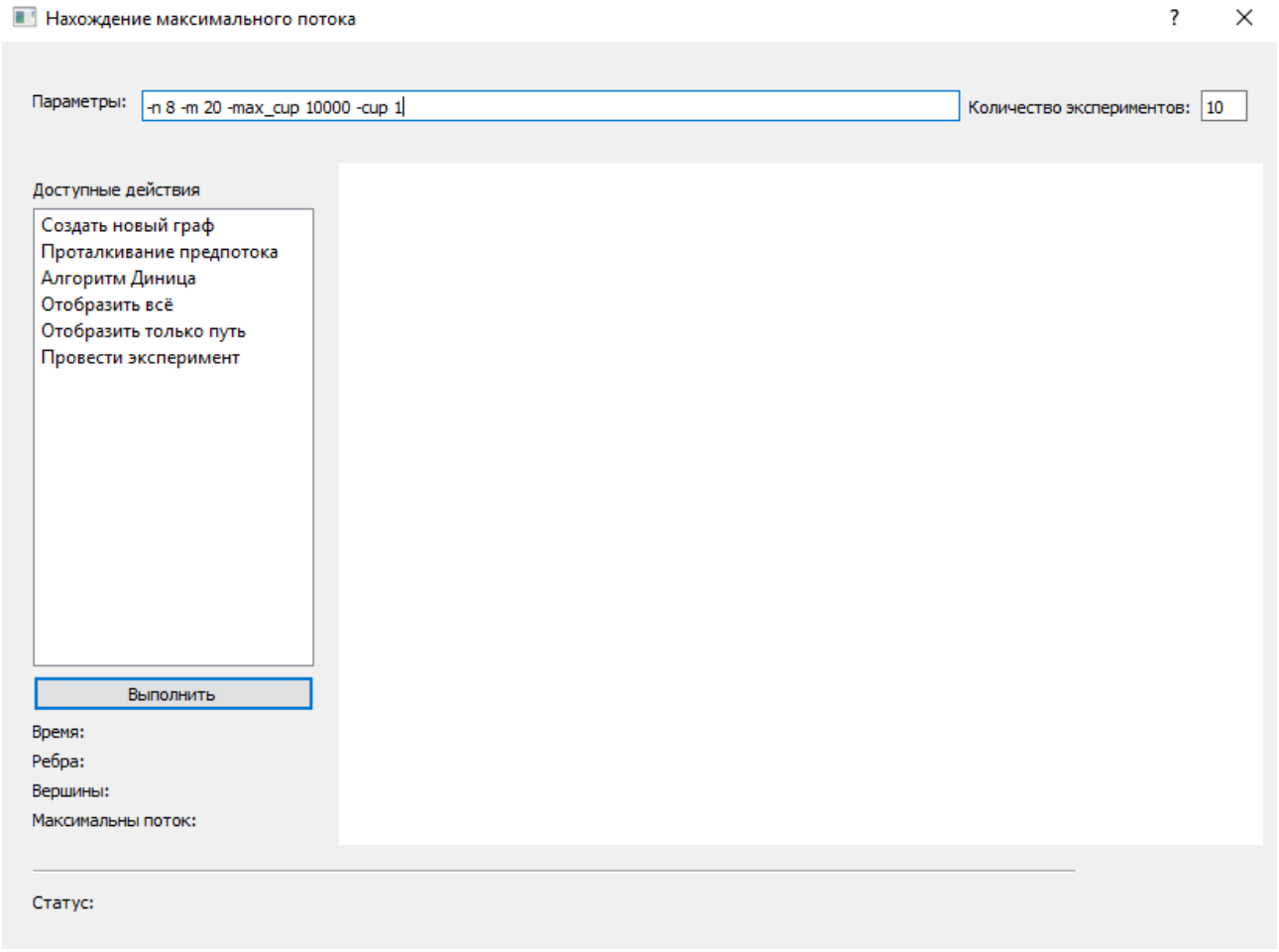

Рисунок 1 – Пример установки параметров генератора.

Генератор графов создает граф случайным образом, имеющий n вершин и m дуг, также полученный граф не имеет петель. Гарантируется, что путь из 0 вершины в $n-1$ вершину будет существовать в созданном графе, если задано корректное количество дуг, а именно должно выполняться неравенство $m \geq n-1$.

Нахождение максимального потока

Для того, чтобы найти максимальный поток, из истока 0 в сток $n-1$, в сгенерированном или считанном из файла графе, необходимо

Шаг 1. По необходимости задать параметры для генератора графов

Шаг 2. Выбрать поле «Создать новый граф» в списке слева

Шаг 3. Нажать на кнопку «Выполнить»

Шаг 4. Выбрать один из двух алгоритмов: «Алгоритм Диница» или «Проталкивание предпотока» в списке слева

Шаг 5. Нажать на кнопку «Выполнить»

Алгоритм исполнится на графе, полученном на шагах 1-3. Время выполнения и найденный максимальный поток отобразятся под списком слева.

На рисунке 2 показан результат работы программы используя алгоритм проталкивания предпотока на графе, созданном случайно.

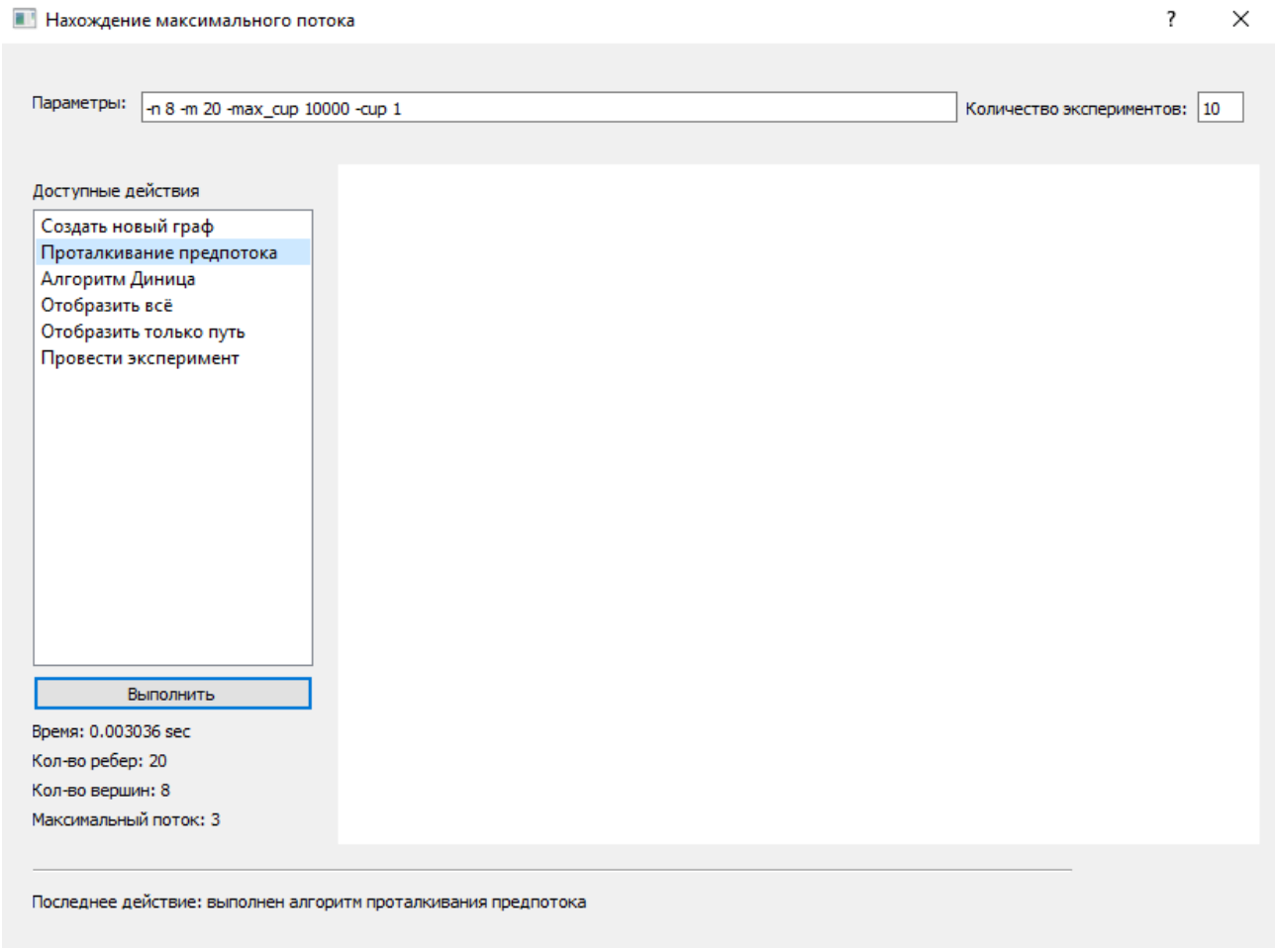

Рисунок 2 – Результат работы программы, после выполнения алгоритма проталкивания предпотока.

Для того, чтобы отобразить граф, необходимо:

Шаг 1. Выбрать один из режимов отображения: «Отобразить всё» или «Отобразить только путь»

Шаг 2. Нажать на кнопку «Выполнить»

На отобразившимся графе дуги имеют определенные цвета. Насыщенные дуги имеют светло-серый оттенок, дуги по которым проходит поток ненасыщенный и не нулевой, имеют серый оттенок. Дуги с нулевым потоком имеют темный оттенок.

На рисунке 3 показан результат при выборе режима «Отобразить всё» на шаге 1. Отображается весь граф, считанный из файла, с максимальным потоком, найденным на нем алгоритмом проталкивания предпотока, где на каждой дуге отображен проходящий поток и пропускная способность дуги.

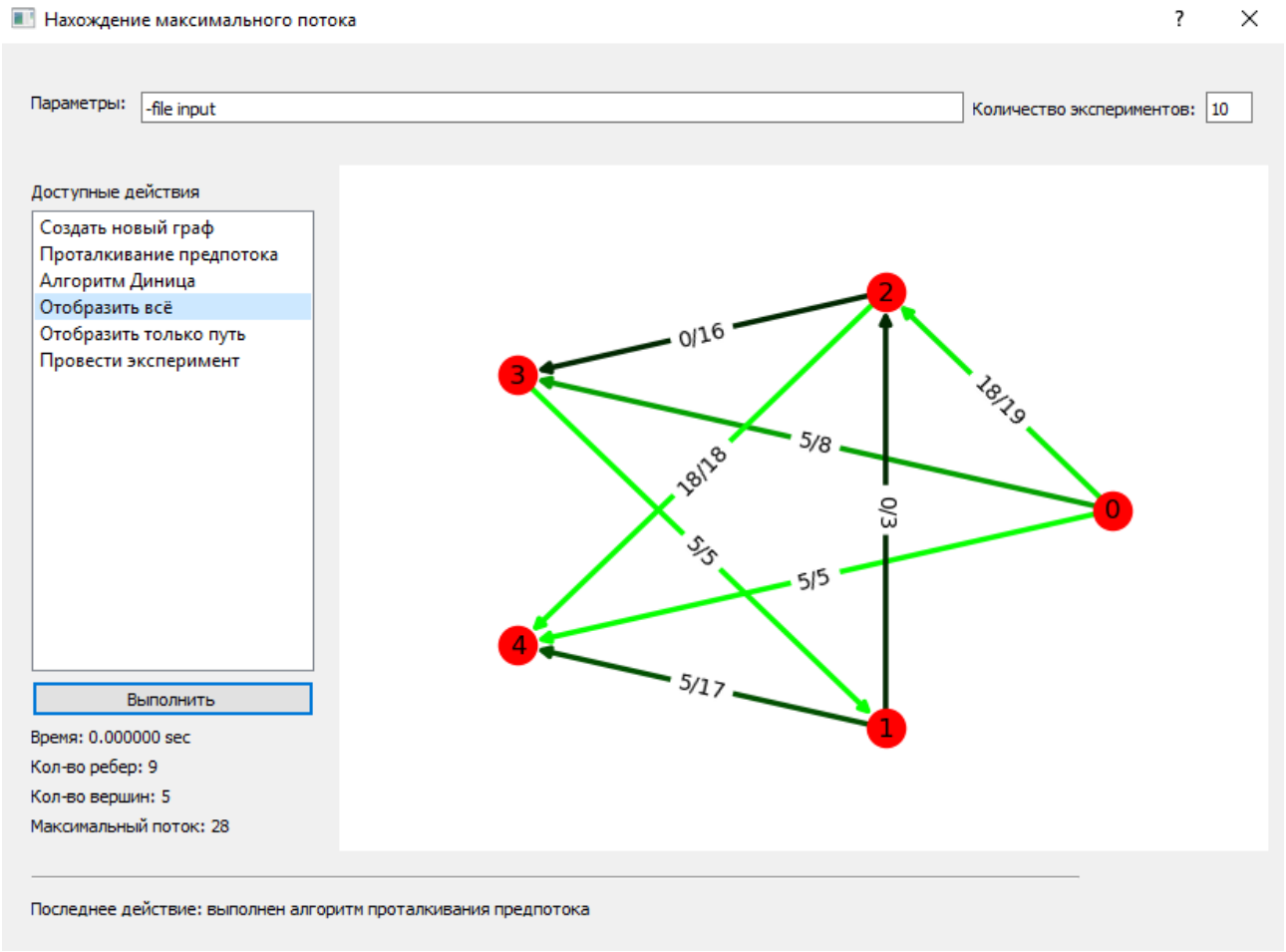

Рисунок 3 – Результат работы программы после выполнения действия «Отобразить всё».

На рисунке 4 показан результат при выборе режима «Отобразить только путь» на шаге 1. Отображаются лишь те дуги, по которым проходит поток $f \neq 0$, из истока 0 в сток 4.

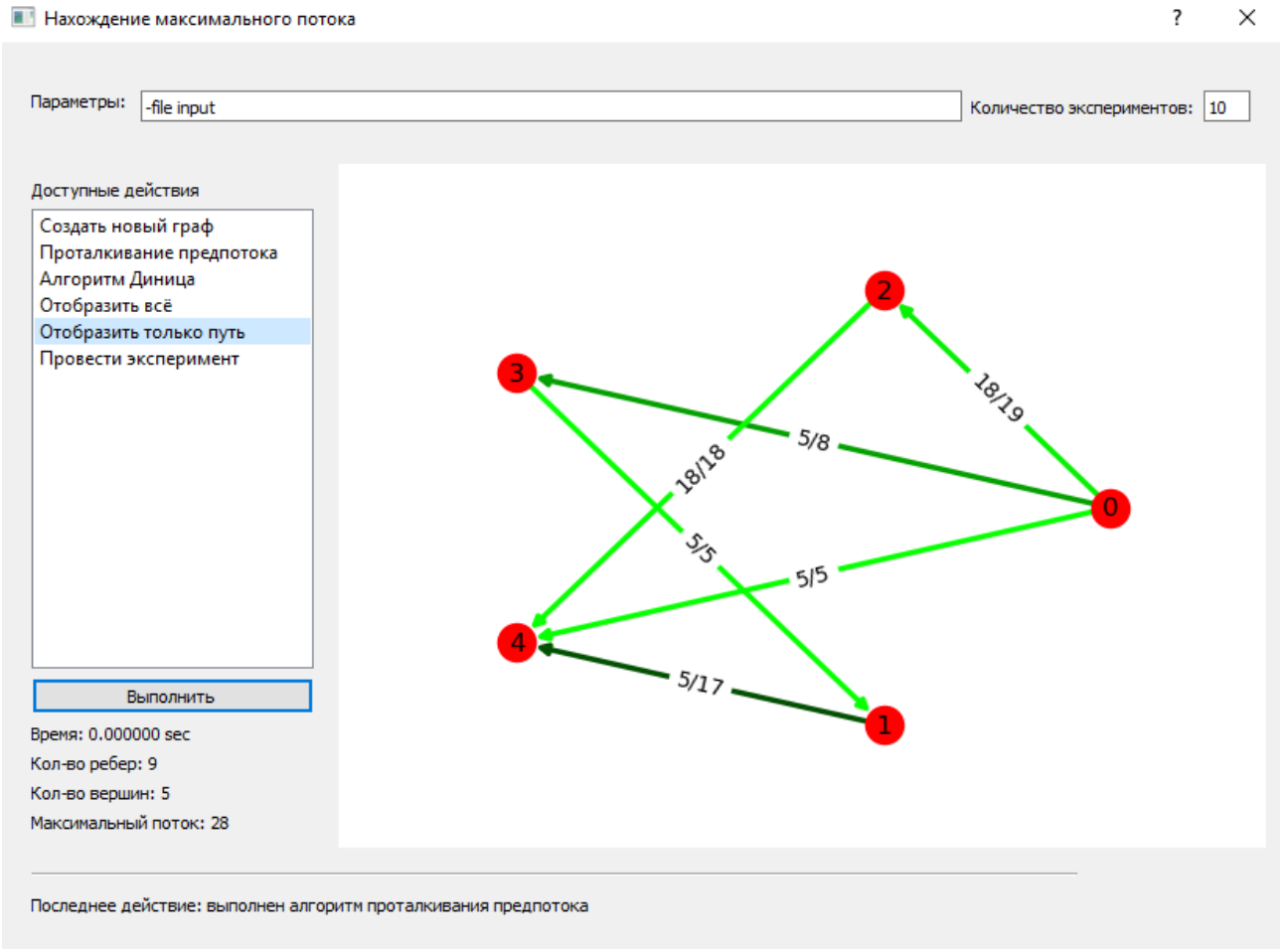

Рисунок 4 – Результат работы программы после выполнения действия «Отобразить только путь».

Сравнение скорости работы алгоритмов

Для того, чтобы узнать среднюю скорость работы алгоритмов на серии экспериментов, необходимо

Шаг 1. Задать число экспериментов (сколько раз сгенерировать случайный граф) в поле «Количество экспериментов»

Шаг 2. Выбрать поле «Провести эксперимент» в списке слева

Шаг 3. Нажать на кнопку «Выполнить»

Результат эксперимента будет отображен в поле снизу.

Результаты проведения серии экспериментов

Была проведена серия экспериментов с различными параметрами генератора. Для каждого замера времени работы алгоритмов создается 1000 графов с помощью генератора графов. Результаты проведения экспериментов можно увидеть в таблице 1.

Таблица 1.

Количество вершин (n)	Количество дуг (m)	Фикс. пропускная способность дуг (cup)	Максимально возможная пропускная способность дуг (max_cup)	Среднее время выполнения алгоритма Диница	Среднее время выполнения алгоритма проталкивания предпотока
8	15	1	-	0.0002862	0.0004316
8	15	20	-	0.0002961	0.0003547
8	15	1000000	-	0.000282	0.000374
8	15	-	20	0.0003511	0.0004765
8	15	-	1000000	0.0003611	0.0004744
8	30	1	-	0.0005344	0.0006132
8	30	20	-	0.0005329	0.0006351
8	30	1000000	-	0.0005529	0.0006748
8	30	-	20	0.0007623	0.0007777
8	30	-	1000000	0.0007918	0.0007916
20	30	1	-	0.0008189	0.002854
20	30	20	-	0.0008321	0.002794
20	30	1000000	-	0.0008338	0.002877
20	30	-	20	0.0011499	0.0043733
20	30	-	1000000	0.0011424	0.0044208
20	250	1	-	0.0037824	0.0106184
20	250	20	-	0.003924	0.0118431

Продолжение таблицы 1.

20	250	1000000	-	0.0040553	0.0129596
20	250	-	20	0.0044356	0.014597
20	250	-	1000000	0.0046785	0.0155492
50	250	1	-	0.0178203	0.1002994
50	250	20	-	0.0181634	0.0970283
50	250	1000000	-	0.0259241	0.1511811
50	250	-	20	0.0440775	0.2203837
50	250	-	1000000	0.039249	0.1870216
50	1500	1	-	0.0252064	0.2037404
50	1500	20	-	0.0251774	0.1977448
50	1500	1000000	-	0.0234274	0.1827549
50	1500	-	20	0.0296276	0.2639327
50	1500	-	1000000	0.0324092	0.332643
100	1500	1	-	0.0805785	0.9427662
100	1500	20	-	0.0815019	0.705341
100	1500	1000000	-	0.0812312	0.9122211
100	1500	-	20	0.0959796	1.0226776
100	1500	-	1000000	0.0974356	1.4153233
100	7000	1	-	0.0935304	1.4735126
100	7000	20	-	0.0905643	1.7993784
100	7000	1000000	-	0.0927838	1.887954
100	7000	-	20	0.1067766	1.2774868
100	7000	-	1000000	0.1085376	2.4338243

Анализируя таблицу 1, можно заметить следующее: алгоритм Диница стабильно быстро работает на сетях с пропускными способностями одинакового веса. Так как время работы алгоритма в таком случае составляет $O(|V| * |E|)$ за

счет того, что алгоритм совершает всего одну фазу поиска блокирующего потока, что эквивалентно поиску в ширину. Если же пропускная способность имеет большой разброс, как в случае, когда параметр генератора «-max_cup» равен 1000000, то время работы алгоритма увеличивается, так как количество блокирующих потоков стремится к $n-1$, также асимптотика алгоритма стремится к $O(|V|^2 * |E|)$. В среднем, алгоритм Диница работает медленнее на плотном графе, исходя из таблицы 1, в среднем, в 2.0569 раза.

Алгоритм проталкивания предпотока показывает похожие результаты, но в отличии от алгоритма Диница имеет меньшую разницу по времени, в зависимости от параметров. В среднем, алгоритм проталкивания предпотока работает медленнее на плотном графе, исходя из таблицы 1, в среднем, в 2.17104 раза.

В целом алгоритм Диница работает быстрее алгоритма проталкивания предпотока, хоть они и имеют одинаковые асимптотики, но алгоритм проталкивания предпотока, в отличии от алгоритма Диница, имеет большую скрытую константу, которая сильно влияет на производительность алгоритма в среднем.

ЗАКЛЮЧЕНИЕ

В ходе работы было рассмотрено несколько алгоритмов для решения задачи поиска максимального потока в транспортных сетях. В частности, были рассмотрены алгоритм Диница и алгоритм проталкивания предпотока (в его классической реализации). Эти алгоритмы имеют совершенно разные подходы для решения задачи поиска максимального потока, но асимптотически одинаковые оценки. Для оценки их времени работы на практике был реализован генератор графов и указанные алгоритмы поиска максимального потока. Была проведена серия экспериментов, в ходе которых были получены экспериментальные данные, подтверждающие теоретические. На случайных графах алгоритм Диница показал значительно лучшую скорость работы, чем алгоритм проталкивания предпотока в классической реализации (имеющий большую скрытую константу в асимптотической оценке). На практике алгоритм Диница применяется для решения многих прикладных задач за счет своего быстродействия. Алгоритм проталкивания предпотока, уступает в скорости работы алгоритму Диница, поэтому имеет меньшую популярность.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Богомолов, А.М. Алгебраические основы теории дискретных систем / А.М. Богомолов, В.Н. Салий – Москва: «Физико-математическая литература» РАН, 1997. – 367с.

2. Кормен, Томас Х., Алгоритмы: построение и анализ, 2-е издание / Кормен, Томас Х., Лейзернон, Чарльз И., Ривест, Рональд Л., Штайн, Клиффорд – Издательский дом “Вильямс”, 2005. – 1296с.

3. Зыков, А.А. Основы теории графов / А.А. Зыков – Москва: «Наука», 1987. –382с.

4. Абросимов, М.Б. Практические задания по графам: учебное пособие / М.Б. Абросимов, А.А. Долгов –Саратов: «Научная книга», 2016. –82с.

5. Новиков, Ф.А. Дискретная математика: учебник для вузов. 2-е издание / Ф.А. Новиков – СПб.: Питер, 2013. – 432 с.

6. Лапенок, А. Поток минимальной стоимости [Электронный ресурс]. URL: https://neerc.ifmo.ru/wiki/index.php?title=Поток_минимальной_стоимости (Дата обращения 11.04.2018). Яз. Рус.

7. Свами, М. Графы, сети и алгоритмы / М. Свами, К. Тхуласирамин – М.: Мир, 1984. – 455с.

8. Харари, Ф. Теория графов / Ф. Харари – М.: Едиториал УРСС, 2003. – 296с.

9. NetworkX is a Python package for the creation networks [Электронный ресурс]. – URL: https://networkx.github.io (Дата обращения 6.02.2018). Яз. Англ.

10. Matplotlib. Python 2D plotting library [Электронный ресурс]. URL: https://matplotlib.org (Дата обращения 07.02.2018). Яз. Англ.

ПРИЛОЖЕНИЕ А

Листинг программы, реализующей некоторые алгоритмы поиска максимального потока

Программа написана на языке Python (версия интерпретатора 3.6). Используемые модули: PyQt5, matplotlib и networkx.

Модуль viz.py

```
# coding=utf-8
import sys

import matplotlib.pyplot as plt
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QGridLayout
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas

import generator
from algorithms.dinica import Dinica
from algorithms.pushFlow import PushFlow
from graph import Graph
from ui import design
from ui.graphViz import GraphViz
from utils import clear_log, get_matrix_stats, parse_gen_args


class ExampleApp(QtWidgets.QDialog, design.Ui_Dialog):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        self.HANDLER_DICT = {"Алгоритм Диница": self.handler_dinica,
                             "Проталкивание предпотока": self.handler_push_flow,
                             "Создать новый граф": self.handler_generate_next,
                             "Отобразить всё": self.handler_draw_graph,
                             "Отобразить только путь": self.handler_draw_onlyway_graph,
                             "Провести эксперимент": self.handler_experiment}
        self.flow_algorithm = None
        self.graph = None
        self.matrix = []

        grid = QGridLayout()
        self.setLayout(grid)
        self.figure = plt.figure()
        self.canvas = FigureCanvas(self.figure)
        self.gridLayout.addWidget(self.canvas, 0, 1, 9, 9)
```

```
        print(self.listWidget.selectedItems())

self.executeBtn.clicked.connect(self.handler_execute_from_listbox)

self.listWidget.itemDoubleClicked.connect(self.handler_double_clicked_lis
t)

    def handler_draw_graph(self):
        self.__draw(self.flow_algorithm.getGraph())

    def handler_draw_onlyway_graph(self):
        self.__draw(self.flow_algorithm.getGraph(), True)

    def handler_generate_next(self):
        self.matrix = self.get_matrix()
        stat = get_matrix_stats(self.matrix)
        self.label_vertexes.setText('Кол-во вершин: ' + str(stat['v']))
        self.label_edges.setText('Кол-во ребер: ' + str(stat['e']))
        self.set_status('создан новый граф')

    def handler_experiment(self):
        count = int(self.editExperimentCount.text())

        params = self.__get_gen_atributes()
        params = parse_gen_args(params.split())
        gen = generator.GraphGenerator(params)

        ans_dinica = 0
        ans_preflow = 0

        for idx in range(count):
            M = next(gen)
            s, t = 0, len(M) - 1

            graph = Graph.initGraphFromMatrix(M)
            preflow_algorithm = PushFlow(graph, len(M), s, t)

            graph = Graph.initGraphFromMatrix(M)
            dinica_algorithm = Dinica(graph, len(M), s, t)

            assert            dinica_algorithm.getMaxFlow()            ==
preflow_algorithm.getMaxFlow()
            ans_dinica += dinica_algorithm.time
            ans_preflow += preflow_algorithm.time
        self.set_status(
            '{} тестов (Средний результат), Диница: {}, проталкивание
предпотока: {}'.format(count, ans_dinica / count,

ans_preflow / count))

    def handler_execute_from_listbox(self):
        commands = []
        for item in self.listWidget.selectedItems():
            commands.append(item.text())
        for command in commands:
            self.HANDLER_DICT[command]()
```

```
    def handler_double_clicked_list(self, item):
        self.HANDLER_DICT[item.text()]()

    def __get_gen_atributes(self):
        try:
            params = self.lineEdit.text()
        except:
            params = []
        return params

    def get_matrix(self):
        params = self.__get_gen_atributes()
        params = parse_gen_args(params.split())
        M = generator.GraphGenerator(params)
        M = next(M)
        return M

    def handler_push_flow(self):
        M = self.matrix
        self.graph = Graph.initGraphFromMatrix(M)
        s, t = 0, len(M) - 1
        self.flow_algorithm = PushFlow(self.graph, len(M), s, t)
        self.label_time.setText("Время:                                {:.6f}
sec".format(self.flow_algorithm.time))
        self.label_flow.setText("Максимальный                          поток:
{}".format(self.flow_algorithm.getMaxFlow()))
        self.set_status('выполнен алгоритм проталкивания предпотока')

    def handler_dinica(self):
        M = self.matrix
        self.graph = Graph.initGraphFromMatrix(M)
        s, t = 0, len(M) - 1
        self.flow_algorithm = Dinica(self.graph, len(M), s, t)
        self.label_time.setText("Время:                                {:.6f}
sec".format(self.flow_algorithm.time))
        self.label_flow.setText("Максимальный                          поток:
{}".format(self.flow_algorithm.getMaxFlow()))
        self.set_status('выполнен алгоритм Диница')

    def set_status(self, text):
        self.label_status.setText('Последнее действие: {}'.format(text))

    def __draw(self, G, only_way=False):
        self.figure.clf()
        gv = GraphViz(G, only_way)
        plt.axis('off')
        gv.draw()
        self.canvas.draw_idle()


def run():
    clear_log()
    app = QtWidgets.QApplication(sys.argv)
    window = ExampleApp()
    window.show()
    app.exec_()
```

```
if __name__ == '__main__':
    run()
```

Модуль design.py

```
# -*- coding: utf-8 -*-

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(824, 585)
        self.gridLayout_2 = QtWidgets.QGridLayout(Dialog)
        self.gridLayout_2.setObjectName("gridLayout_2")
        self.frame_left = QtWidgets.QFrame(Dialog)
        self.frame_left.setMinimumSize(QtCore.QSize(201, 0))
        self.frame_left.setMaximumSize(QtCore.QSize(201, 16777215))
        self.frame_left.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.frame_left.setFrameShadow(QtWidgets.QFrame.Raised)
        self.frame_left.setObjectName("frame_left")
        self.verticalLayout_3 = QtWidgets.QVBoxLayout(self.frame_left)
        self.verticalLayout_3.setObjectName("verticalLayout_3")
        self.verticalLayout = QtWidgets.QVBoxLayout()
        self.verticalLayout.setObjectName("verticalLayout")
        self.label = QtWidgets.QLabel(self.frame_left)
        self.label.setObjectName("label")
        self.verticalLayout.addWidget(self.label)
        self.listWidget = QtWidgets.QListWidget(self.frame_left)
        self.listWidget.setObjectName("listWidget")
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        self.verticalLayout.addWidget(self.listWidget)
        self.executeBtn = QtWidgets.QPushButton(self.frame_left)
        self.executeBtn.setObjectName("executeBtn")
        self.verticalLayout.addWidget(self.executeBtn)
        self.verticalLayout_3.addLayout(self.verticalLayout)
        self.label_time = QtWidgets.QLabel(self.frame_left)
        self.label_time.setObjectName("label_time")
        self.verticalLayout_3.addWidget(self.label_time)
        self.label_edges = QtWidgets.QLabel(self.frame_left)
        self.label_edges.setObjectName("label_edges")
        self.verticalLayout_3.addWidget(self.label_edges)
        self.label_vertexes = QtWidgets.QLabel(self.frame_left)
        self.label_vertexes.setObjectName("label_vertexes")
        self.verticalLayout_3.addWidget(self.label_vertexes)
```

```
        self.label_flow = QtWidgets.QLabel(self.frame_left)
        self.label_flow.setObjectName("label_flow")
        self.verticalLayout_3.addWidget(self.label_flow)
        self.gridLayout_2.addWidget(self.frame_left, 1, 0, 1, 1)
        self.gridLayout = QtWidgets.QGridLayout()
        self.gridLayout.setObjectName("gridLayout")
        self.gridLayout_2.addLayout(self.gridLayout, 1, 1, 1, 1)
        self.frame_bottom = QtWidgets.QFrame(Dialog)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(11)

sizePolicy.setHeightForWidth(self.frame_bottom.sizePolicy().hasHeightForW
idth())
        self.frame_bottom.setSizePolicy(sizePolicy)
        self.frame_bottom.setMinimumSize(QtCore.QSize(691, 51))
        self.frame_bottom.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.frame_bottom.setFrameShadow(QtWidgets.QFrame.Raised)
        self.frame_bottom.setObjectName("frame_bottom")
        self.gridLayout_6 = QtWidgets.QGridLayout(self.frame_bottom)
        self.gridLayout_6.setObjectName("gridLayout_6")
        self.line_2 = QtWidgets.QFrame(self.frame_bottom)
        self.line_2.setFrameShape(QtWidgets.QFrame.HLine)
        self.line_2.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line_2.setObjectName("line_2")
        self.gridLayout_6.addWidget(self.line_2, 0, 0, 1, 1)
        self.label_status = QtWidgets.QLabel(self.frame_bottom)
        self.label_status.setObjectName("label_status")
        self.gridLayout_6.addWidget(self.label_status, 1, 0, 1, 1)
        self.gridLayout_2.addWidget(self.frame_bottom, 2, 0, 1, 2)
        self.frame_top = QtWidgets.QFrame(Dialog)
        self.frame_top.setMinimumSize(QtCore.QSize(691, 61))
        self.frame_top.setStyleSheet("")
        self.frame_top.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.frame_top.setFrameShadow(QtWidgets.QFrame.Raised)
        self.frame_top.setObjectName("frame_top")
        self.gridLayout_4 = QtWidgets.QGridLayout(self.frame_top)
        self.gridLayout_4.setObjectName("gridLayout_4")
        self.verticalLayout_5 = QtWidgets.QVBoxLayout()
        self.verticalLayout_5.setObjectName("verticalLayout_5")
        self.editExperimentCount = QtWidgets.QLineEdit(self.frame_top)
        self.editExperimentCount.setMaximumSize(QtCore.QSize(30,
16777215))
        self.editExperimentCount.setObjectName("editExperimentCount")
        self.verticalLayout_5.addWidget(self.editExperimentCount)
        self.gridLayout_4.addLayout(self.verticalLayout_5, 0, 3, 2, 1)
        self.lineEdit = QtWidgets.QLineEdit(self.frame_top)
        self.lineEdit.setMinimumSize(QtCore.QSize(250, 0))
        self.lineEdit.setObjectName("lineEdit")
        self.gridLayout_4.addWidget(self.lineEdit, 0, 1, 2, 1)
        self.verticalLayout_4 = QtWidgets.QVBoxLayout()
        self.verticalLayout_4.setObjectName("verticalLayout_4")
        self.label_6 = QtWidgets.QLabel(self.frame_top)
        self.label_6.setObjectName("label_6")
        self.verticalLayout_4.addWidget(self.label_6)
        self.gridLayout_4.addLayout(self.verticalLayout_4, 0, 2, 2, 1)
```

```
        self.label_load_from_file = QtWidgets.QLabel(self.frame_top)
        self.label_load_from_file.setObjectName("label_load_from_file")
        self.gridLayout_4.addWidget(self.label_load_from_file, 0, 0, 1, 1)
        self.line = QtWidgets.QFrame(self.frame_top)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.line.sizePolicy().hasHeightForWidth())
        self.line.setSizePolicy(sizePolicy)
        self.line.setFrameShape(QtWidgets.QFrame.HLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line.setObjectName("line")
        self.gridLayout_4.addWidget(self.line, 1, 0, 1, 1)
        self.gridLayout_2.addWidget(self.frame_top, 0, 0, 1, 2)
        self.frame_top.raise_()
        self.frame_bottom.raise_()
        self.frame_left.raise_()
        self.label_load_from_file.setBuddy(self.listWidget)

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)
        Dialog.setTabOrder(self.lineEdit, self.listWidget)
        Dialog.setTabOrder(self.listWidget, self.executeBtn)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog",            "Нахождение
максимального потока"))
        self.label.setText(_translate("Dialog", "Доступные действия"))
        __sortingEnabled = self.listWidget.isSortingEnabled()
        self.listWidget.setSortingEnabled(False)
        item = self.listWidget.item(0)
        item.setText(_translate("Dialog", "Создать новый граф"))
        item = self.listWidget.item(1)
        item.setText(_translate("Dialog", "Проталкивание предпотока"))
        item = self.listWidget.item(2)
        item.setText(_translate("Dialog", "Алгоритм Диница"))
        item = self.listWidget.item(3)
        item.setText(_translate("Dialog", "Отобразить всё"))
        item = self.listWidget.item(4)
        item.setText(_translate("Dialog", "Отобразить только путь"))
        item = self.listWidget.item(5)
        item.setText(_translate("Dialog", "Провести эксперимент"))
        self.listWidget.setSortingEnabled(__sortingEnabled)
        self.executeBtn.setText(_translate("Dialog", "Выполнить"))
        self.label_time.setText(_translate("Dialog", "Время:"))
        self.label_edges.setText(_translate("Dialog", "Ребра:"))
        self.label_vertexes.setText(_translate("Dialog", "Вершины:"))
        self.label_flow.setText(_translate("Dialog",         "Максимальны
поток:"))
        self.label_status.setText(_translate("Dialog", "Статус:"))
        self.editExperimentCount.setText(_translate("Dialog", "10"))
        self.lineEdit.setText(_translate("Dialog", "-file input"))
        self.label_6.setText(_translate("Dialog",             "Количество
экспериментов:"))
```

```
        self.label_load_from_file.setText(_translate("Dialog",
"Параметры: "))
```

Модуль __init__.py

```
# coding=utf-8


class MaxFlowAlgo:
    def __init__(self, G, n, start, finish):
        self.n = n
        self.maxFlow = 0
        self.start = start
        self.finish = finish
        self.graph = G

    def findMaxFlow(self):
        pass

    def getGraph(self):
        return self.graph

    def printFlow(self):
        print('Поток:')
        n = self.n
        for i in range(n):
            for j in range(n):
                print(self.graph[i][j].flow, end=' ')
            print()

    def printCup(self):
        print('Пропускная способность:')
        n = self.n
        for i in range(n):
            for j in range(n):
                print(self.graph[i][j].cup, end=' ')
            print()

    def getMaxFlow(self):
        return self.maxFlow
```

Модуль dinica.py

```
# coding=utf-8
import time

import collections
from defines import *
from algorithms import MaxFlowAlgo


class Dinica(MaxFlowAlgo):
    def __init__(self, G, n, start, finish):
        super().__init__(G, n, start, finish)
        self.p = []
```

```
        self.d = []
        self._time_start = time.time()
        self.findMaxFlow()
        self.time = time.time() - self._time_start

    def __bfs(self):
        self.d = [INF for i in range(self.n)]
        self.d[self.start] = 0
        q = collections.deque()
        q.append(self.start)
        while q:
            u = q.popleft()
            for v in range(len(self.graph[u])):
                if  self.graph[u][v].flow  <  self.graph[u][v].cup  and
self.d[v] == INF:
                    self.d[v] = self.d[u] + 1
                    q.append(v)
        return self.d[self.finish] != INF

    def __dfs(self, u, flow):
        if u == self.finish or flow == 0:
            return flow
        while self.p[u] < self.n:
            v = self.p[u]
            if self.d[v] == self.d[u] + 1:
                delta = self.__dfs(v, min(flow, self.graph[u][v].cup -
self.graph[u][v].flow))
                if delta:
                    self.graph[u][v].flow += delta
                    self.graph[v][u].flow -= delta
                    return delta
            self.p[u] += 1
        return 0

    def findMaxFlow(self):
        while self.__bfs():
            self.p = [0 for i in range(self.n)]
            flow = self.__dfs(self.start, INF)
            while flow != 0:
                self.maxFlow += flow
                flow = self.__dfs(self.start, INF)
        return self.maxFlow
```

Модуль pushFlow.py

```
import time

from algorithms import MaxFlowAlgo


class PushFlow(MaxFlowAlgo):

    def __init__(self, G, n, start, finish):
        super().__init__(G, n, start, finish)
        self.high = [0 for _ in range(n)]
        self.e = [0 for _ in range(n)]
        for i in range(0, self.n):
```

```
            self.graph[0][i].flow = self.graph[0][i].cup
            self.graph[i][0].flow = -self.graph[0][i].cup
            self.e[i] = self.graph[0][i].cup
            self.e[0] -= self.graph[0][i].cup
        self.high[0] = self.n
        self._time_start = time.time()
        self.findMaxFlow()
        self.time = time.time() - self._time_start

    def __push(self, u, v):
        d = min(self.e[u], self.graph[u][v].cup - self.graph[u][v].flow)
        self.graph[u][v].flow += d
        self.graph[v][u].flow = -self.graph[u][v].flow
        self.e[u] -= d
        self.e[v] += d

    def findMaxFlow(self):

        n = self.n
        while True:
            v = n
            for i in range(n - 1):
                if self.e[i] > 0:
                    v = i
                    break

            if v == n:
                break

            while self.e[v]:
                for i in range(0, n):
                    if self.graph[v][i].cup > self.graph[v][i].flow and
self.high[v] > self.high[i]:
                        self.__push(v, i)

                if self.e[v] > 0:
                    self.high[v] += 1

        self.maxFlow = self.e[n - 1]
        return self.maxFlow
```

Модуль graphViz.py

```
import networkx as nx


class GraphViz:
    def __init__(self, G, only_way=False):
        self.G = G
        self.DG = nx.DiGraph()
        self.edge_colors = []
        self.node_pos = {}
        self.ONLY_WAY = only_way
        self.n = len(G)
        self.__init_graph()
        self.edge_labels = nx.get_edge_attributes(self.DG, 'route')
        self.edge_colors2 = nx.get_edge_attributes(self.DG, 'color')
```

```
        self.pos = nx.shell_layout(self.DG)

    def __init_graph(self):
        def get_indexes(n):
            for i in range(n):
                for j in range(n):
                    yield i, j
        if self.ONLY_WAY:
            [self.__initEdge(i, j, self.G[i][j].flow, self.G[i][j].cup)
for i, j in
             get_indexes(self.n) if self.G[i][j].cup > 0 and
self.G[i][j].flow > 0]
        else:
            [self.__initEdge(i, j, self.G[i][j].flow, self.G[i][j].cup)
for i, j in
             get_indexes(self.n) if self.G[i][j].cup > 0 and
self.G[i][j].flow >= 0]

    def __getEdgeColor(self, flow, cup):
        def get_str_hex(value):
            if value >= 255:
                return 'FF'
            result = str(hex(value))[2:4].rjust(2, '0')
            return result
        if flow < 0: return '#000000'
        delimiter = 6
        RGB = [1, 20, 1]
        cf = int(flow / (cup / delimiter) + 1) * 2
        return '#{}{}{}'.format(get_str_hex(RGB[0] * (cf // 2)),
get_str_hex(RGB[1] * cf), get_str_hex(RGB[2]))

    def __drawEdges(self):
        nx.draw_networkx_edge_labels(self.DG, self.pos,
edge_labels=self.edge_labels, clip_on=False)

    def __drawNodes(self):
        nx.draw_networkx_nodes(self.DG, self.node_pos)

    def draw(self):
        self.__drawEdges()
        nx.draw_shell(self.DG, with_labels=True,
edge_color=self.edge_colors2.values(), width=2.5)

    def __initEdge(self, x, y, flow, cup):
        edge_label = "{0}/{1}".format(flow, cup)
        self.DG.add_edge(x, y, route=edge_label,
color=self.__getEdgeColor(flow, cup))
        self.edge_colors.append(self.__getEdgeColor(flow, cup))
```

Модуль graph.py

```
import utils


class Graph:
    def __init__(self):
        pass
```

```
    def updateVisual(self):
        pass

    @staticmethod
    def readMatrixFromTerminal():
        n = int(input())
        M = [[0 for y in range(n)] for x in range(n)]
        for i in range(n):
            M[i] = list(map(int, input().split()))
        return M

    @staticmethod
    def readMatrixFromFile(file_name):
        with open(file_name) as f:
            n = int(f.readline().strip())
            M = [[0 for y in range(n)] for x in range(n)]
            for i in range(n):
                line = f.readline()
                M[i] = list(map(int, line.split()))
        return M

    @staticmethod
    def readAdjacencyListToMatrix():
        n, m = map(int, input().split())
        M = [[0 for y in range(n)] for x in range(n)]
        for i in range(m):
            x, y, c = map(int, input().split())
            M[x - 1][y - 1] = c
        return M

    @staticmethod
    def initGraphFromMatrix(M):
        n = len(M)
        G = [[utils.edge(cup=M[x][y]) for y in range(n)] for x in range(n)]
        return G

    @staticmethod
    def default(n):
        G = [[utils.edge() for y in range(n)] for x in range(n)]
        G[0][1].cup = 3
        G[1][3].cup = 1
        G[0][2].cup = 1
        G[2][3].cup = 2
        G[1][2].cup = 1
        return G
```

Модуль utils.py

```
import argparse
import time


class Profiler(object):
    def __enter__(self):
        self._startTime = time.time()
```

```
    def __exit__(self, type, value, traceback):
        print("Elapsed    time:    {:.3f}    sec".format(time.time()    -
self._startTime))


class edge:
    def __init__(self, flow=0, cup=0):
        self.flow = flow
        self.cup = cup


def printMatrix(M):
    n = len(M)
    for i in range(n):
        for j in range(n):
            print(M[i][j], end=' ')
        print()


def matrix_to_str(M):
    result = ''
    for i in range(len(M)):
        for j in range(len(M)):
            result += str(M[i][j]) + ' '
        result += '\n'
    return result


def get_matrix_stats(M):
    vertexes = len(M)
    edges = 0
    for i in range(len(M)):
        for j in range(len(M)):
            if M[i][j] != 0:
                edges += 1
    return {'e': edges, 'v': vertexes}


def log_report(*args):
    with open('log.log', 'a') as file:
        file.write(' '.join(map(str, args)))


def clear_log():
    with open('log.log', 'w') as file:
        file.write('')
    print("Log is cleared")


def generator_read_file(name):
    file = open(name, 'r')
    n = int(file.readline())
    while file.readline():
        M = []
        for i in range(n):
            line = file.readline()
            M.append(list(map(int, line.split())))
```

```
        yield M


def parse_gen_args(args):
    parser = argparse.ArgumentParser()
    parser.add_argument('-n', type=int)
    parser.add_argument('-m', type=int)
    parser.add_argument('-cup', type=int)
    parser.add_argument('-max_cup', type=int)
    parser.add_argument('-file', type=str)
    return parser.parse_args(args)
```

Модуль main.py

```
import ui.viz as ui

def main():
    ui.run()

main()
```

ПРИЛОЖЕНИЕ Б

Листинг программы, реализующей генератор графов

Программа написана на языке Python (версия интерпретатора 3.6). Используемые модули: PyQt5, matplotlib и networkx.

Модуль generator.py

```
import random

from defines import *
from graph import Graph
from utils import log_report, matrix_to_str


def generate_allow_edges(n):
    result = []
    for idx in range(n-1):
        for idy in range(1, n):
            if idx != idy:
                result.append((idx, idy))
    return result


class GraphGenerator:
    def __init__(self, params):
        self.file = params.file if not (params.file is None) else None
        self.n = random.randint(2, 20) if params.n is None else params.n
        self.m = random.randint(self.n - 1, (self.n * (self.n - 1)) // 2)
if params.m is None else params.m
        if self.m > (self.n * (self.n - 1)) // 2:
            self.m = (self.n * (self.n - 1)) // 2
        self.const_cup_mod,   self.const_cup_value   =   (False,   0)   if
params.cup is None else (True, params.cup)
        self.max_cup = 20 if params.max_cup is None else params.max_cup
        self.duo_mod = False
        self.allowable_edges = []

    def gen_matrix(self):
        def get_indexes():
            if len(self.allowable_edges) == 0:
                return None, None
            x, y = random.choice(self.allowable_edges)
            self.allowable_edges.remove((x, y))
            try:
                self.allowable_edges.remove((y, x))
            except: pass
            return x, y

        result = [[0 for x in range(self.n)] for y in range(self.n)]
        for x in range(self.m):
            idx, idy = get_indexes()
```

```
            if idx is None:
                break
            result[idx][idy] = self.generate_cup()
        return result

    def check_way(self, M):
        if M is None:
            return False
        result = False

        def dfs(u):
            used[u] = True
            if u == len(M) - 1:
                return
            for to in range(self.n):
                if not used[to] and M[u][to]:
                    dfs(to)

        used = [False for i in range(self.n)]
        dfs(0)
        if used[len(M) - 1]:
            result = True

        return result

    def gen(self):
        result = None
        if not (self.file is None):
            return Graph.readMatrixFromFile(self.file)

        while not self.check_way(result):
            self.allowable_edges = generate_allow_edges(self.n)
            result = self.gen_matrix()
        log_report('Generate graph:\n', matrix_to_str(result))
        return result

    def generate_cup(self):
        if self.const_cup_mod:
            return self.const_cup_value
        return random.randint(1, self.max_cup)

    def __next__(self):
        return self.gen()
```