**Введение**

Задача о максимальном потоке в сети изучается уже более 50-ти лет. Эта задача имеет огромную практическую значимость. Во многих практически важных случаях функционирование системы, моделируемой ориентированным графом, определяется передачей между её отдельными частями некоторых потоков. Например, в электрических, транспортных сетях, при моделировании различных химических и физических процессов, для поиска web-сайтов во всемирной сети интернет.

Существует множество решений этой задачи. В данной работе будут рассмотрены

**Алгоритм Диница**

Алгоритм Диница - полиномиальный алгоритм для вычисления *максимального потока в транспортной сети. Его придумал израильтянин,* когда-то советский программист, Ефим (Хаим) А.Диниц. Этот алгоритм является улучшенной версией алгоритма Эдмондса-Карпа. Его алгоритм справляется с работой за  O(V^2 E) и использует самые короткие пути увеличения. Заполучить эту оценку позволяет введение понятий вспомогательной сети и блокирующего потока.

Введём 3 определения, на которых основан алгоритм Диница. Каждое из этих определений не зависит от других.

**Остаточная сеть** **G^R** строится относительно сети G и некоторому потоку f в этой сети. В итоге построения получается сеть в которой каждому ребру  (u,v) \in G с пропускной способностью c_{uv} и потоком  f_{uv} соответствуют два ребра:

1.1 (u,v) с пропускной способностью c_{uv}^R = c_{uv} - f_{uv}

1.2 (v,u) с пропускной способностью c_{vu}^R = f_{uv}

Здесь можно заметить, что в остаточной сети появятся кратные рёбра, если до построения сеть G содержала как (u,v)  так и (v,u) .

Остаточное ребро несет в себе информацию о том, насколько ещё можно увеличить поток по этому ребру. Если по ребру  (u,v) с пропускной способностью c_{uv} проходит поток  f_{uv} , то оно способно вместить ещё  c_{uv}-f_{uv} единиц потока, а в обратную сторону может пропустить до  f_{uv} единиц потока, и это будет значить отмену потока в начальном направлении.

**Блокирующий поток** в исходной сети это таковой поток, что любой путь из истока s в сток t имеет насыщенное этим потоком ребро. Другими словами, в предоставленной сети никак не найдётся такого пути из истока в сток, вдоль которого позволительно увеличить поток.

Однако блокирующий поток не всегда является максимальным. Об этом и говориться в теореме Форда-Фалкерсона. Поток будет максимальным тогда и лишь тогда, когда в остаточной сети никак не будет пути из истока в сток.

**Слоистая сеть** для данной сети строится следующим образом. Первым делом определяются все длины кротчайших путей из истока ко всем остальным вершинам и обозначаются как \operatorname{dist}(v) . Далее в слоистую сеть добавляют все те рёбра (u,v) исходной сети, для которых справедливо dist(u)=dist(v)+1.

**Алгоритм**

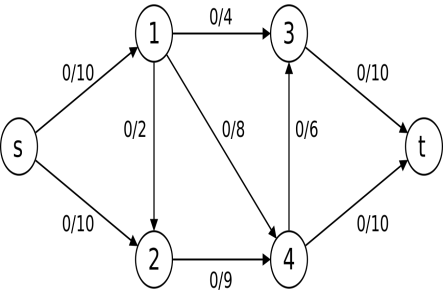
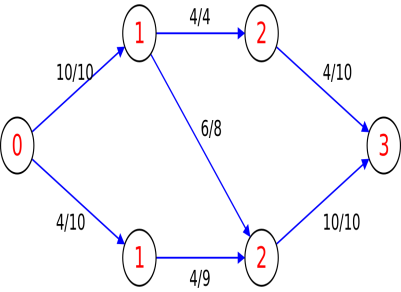
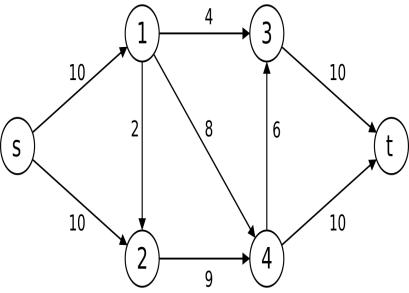
Алгоритм состоит из нескольких шагов. На вход алгоритму подаётся исходная сеть, в которой нужно найти максимальный поток. На выход алгоритм возвращает максимальный поток данной сети.

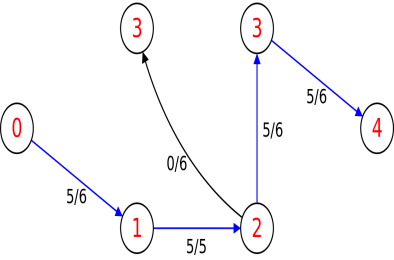
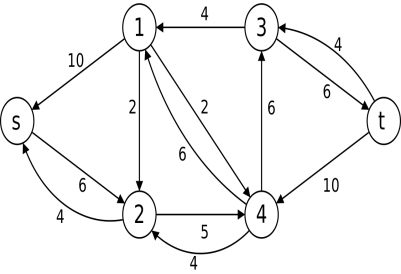
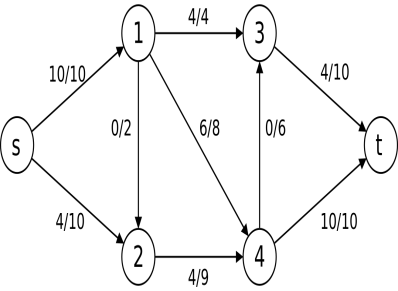
1. Строиться слоистая сеть, если в слоистой сети dist(t)=∞, алгоритм останавливается и выводит максимальный поток.
2. По отношения к слоистой сети строится блокирующий поток.
3. Основной поток дополняется блокирующим.

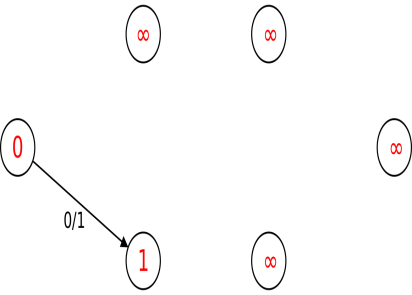
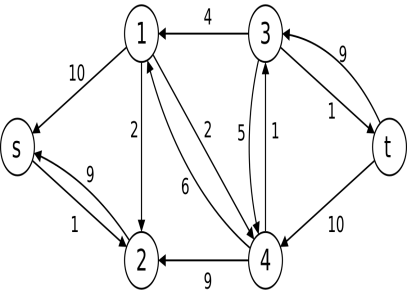
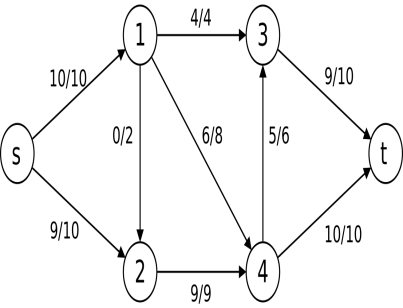
Пример работы алгоритма:

Рассмотрим пример работы на простой транспортной сети, которая имеет 6 вершин и 9 ребер.

Блокирующий поток помечен синим.





На последнем шаге алгоритм заканчивает работу, так как сток помечен меткой ∞. Получившийся максимальный поток равен 19.

**Алгоритм Эдмондса-Карпа**

Алгоритм Эдмондса-Карпа представляет собой улучшенный алгоритм Форда-Фалкерсона. Ключевым отличием является то, что на каждом шаге выбирается кратчайший дополняющий путь из истока в сток в остаточной сети (обычно используется алгоритм поиска в ширину). Алгоритм в 1972 году открыли независимо друг от друга Эдмондсон и Карп. Они доказали, что для алгоритма Форда-Фалкерсона справедливо утверждение, что если дополняющий путь является кратчайшим, то его длина не уменьшается.

Алгоритм Эдмондса-Карта описывается так:

1. Изначально считается, что потоки имеют нулевое значение, а остаточную суть полагают равной исходной.
2. При помощи поиска в ширину между стоком и источником находится кратчайший путь в остаточной сети. Если такой путь не найден, то алгоритм останавливается.
3. Через найденный кратчайший путь пускаем максимально возможный поток:
   1. Находим среди ребер пути ребро с наименьшим потоком Cmin
   2. Поток для каждого ребра в кратчайшем пути делаем больше на величину наименьшего потока Сmin, а в противоположных для каждого ребра – делаем меньше на найденную величину
   3. Изменяем остаточную сеть следующим образом: всех ребер в найденном пути, в том числе и противоположных, находим новые значения пропускной способности. Если у ребра получено ненулевая величина потока, то ребро остается в остаточной сети, в противном случае удаляем.
4. Возвращаемся к шагу 2 и повторяем алгоритм до тех пор, пока в остаточной сети будут пути между стоком и истоком.

Рассмотрим пример работы алгоритма. Дан граф c вершинами A,B,C,D,E,F,G. Будем считать истоком и стоком сети вершины A и G соответственно, f/s – потоком и пропускной способностью.

|  |  |
| --- | --- |
| 300px-Edmonds-Karp_flow_example_0.png | Каждый путь в графе находится при помощи алгоритма поиска в ширину. На первом шаге мы нашли путь A🡪D🡪E🡪G |
| 300px-Edmonds-Karp_flow_example_1.png | Теперь находим максимальное значение потока, которое мы можем пустить по данном пути. Min(Cf(A,D), Cf(D,E), Cf(E,G))=1 |
| 300px-Edmonds-Karp_flow_example_2.png | На следующем шаге мы вновь находим кратчайший путь в остаточной сети – путь A🡪D🡪F🡪G, и найдем величину потока для него:  Min(Cf(A,D), Cf(D,F), Cf(F,G))=2 |
| 300px-Edmonds-Karp_flow_example_3.png | Далее действия аналогичные:  Путь A🡪B🡪C🡪D🡪F🡪G  Min(Cf(A,B), Cf(B,C), Cf(C,D), Cf(D,F), Cf(F,G))=1 |
| 300px-Edmonds-Karp_flow_example_4.png | Путь A🡪B🡪C🡪E🡪D🡪F🡪G  Min(Cf(A,B), Cf(B,C), Cf(C,E), Cf(E,D), Cf(D,F), Cf(F,G))=1  Здесь через ребро DE поток идет в обратном направлении, поэтому мы уменьшаем его величину. |