Введение в дискретную математику и математическую логику

Лекция №11
Потоковые сети

Апанович Зинаида Владимировна

Потоковые сети

Проблема

Графы часто используются для моделирования транспортных сетей — сетей, по ребрам которых осуществляется какое-либо движение, а вершины действуют как «коммутаторы», пропускающие движение между различными ребрами.

Рассмотрим, например,

системму автомагистралей , в которой ребрами являются автомагистрали, а вершинами — развязки;

компьютерную сеть , в которой ребра представляют собой связи, по которым могут передаваться пакеты, а узлы — коммутаторы;

сеть трубопроводов , в которой ребра — это трубы, по которым течет жидкость, а вершины — это соединения, в которых трубы соединяются друг с другом.

Потоковые сети

Сетевые модели этого типа состоят из нескольких компонентов:

- пропускная способность ребер, указывающая, какой объем трафика они могут пропустить;
- вершины-источники в графе, которые генерируют трафик;
- вершины-приемники (или пункты назначения) в графе, которые могут «поглощать» трафик по мере его поступления;
- трафик сам по себе, который передается через ребра.

Потоковые сети

Мы будем называть трафик потоком — абстрактной сущностью, которая

генерируется в вершинах -истоках,

передается по ребрам ,

и поглощается в вершинах - стоках.

Формально мы будем говорить, что потоковая сеть— это ориентированный граф $G = (V, E)$ со следующими характеристиками.

- С каждым ребром e связана пропускная способность, которая является неотрицательным числом и обозначается c_e.
- Имеется одна вершина-исток $s \in V$.
- Имеется одна вершина-сток $t \in V$.
- Вершины, отличные от s и t, будут называться внутренними вершинами.

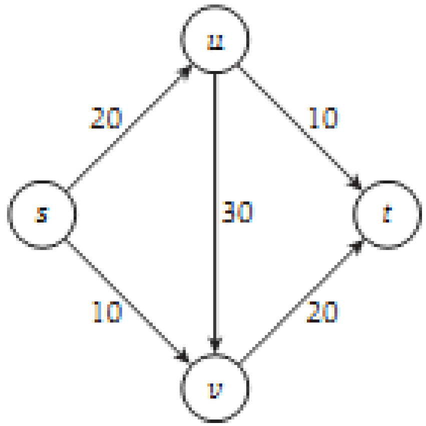


Рис.1 Потоковая сеть, с истоком s и стоком t. Числа рядом с ребрами показывают пропускные способности ребер.

Потоковые сети

Мы сделаем два предположения относительно потоковых сетей, с которыми мы имеем дело:

1) ни одно ребро не входит в исток s и ни одно ребро не выходит из стока t ;

2) каждой вершине инцидентно по крайней мере одно ребро;

3) все пропускные способности являются целыми числами.

Определение потока

Поток — это функция *f*, которая отображает каждое ребро *e* в неотрицательное вещественное число $f : E \to R^+$;

Значение $f(e)$ интуитивно представляет собой величину потока, переносимого ребром *e* .

Поток *f* должен удовлетворять *следующим* двум свойствам.

(i) (Условие пропускной способности) Для каждого $e \in E$,

имеем $0 \le f(e) \le c_e$.

(*ii*) (Условие сохранения) Для каждой вершины *v*, отличной от *s* и *t* , мы имеем

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e).$$

$\sum_{e \text{ into } v} f(e)$ представляет собой сумму значений потока $f(e)$ по всем ребрам, входящим в вершину *v*,

$\sum_{e \text{ out of } v} f(e)$ представляет собой сумму значений потока $f(e)$ по всем ребрам, исходящим из вершины *v* .

Определение потока

Таким образом, поток на ребре не может превышать пропускную способность ребра.

Для каждой вершины, кроме истока и стока, величина входящего потока должна быть равна величине исходящего потока .

Исток не имеет входящих рёбер (по нашему предположению), но ему разрешено иметь исходящий поток ; другими словами, он может генерировать поток .

Симметрично, в сток может поступать поток, даже если у него нет исходящих из него ребер.

Значение потока

Значение потока *f*, обозначаемое *v*(*f*), определяется как величина потока, генерируемого в истоке :

$$v(f) = \sum_{e \text{ out of } s} f(e).$$

Чтобы сделать запись более компактной, определим

$$f^{\text{out}}(v) = \sum_{e \text{ out of } v} f(e)$$

$$f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e).$$

Значение потока

Мы можем распространить формулы на множества вершин ;

если $S \subseteq V$, мы определяем

$$f^{\text{out}}(S) = \sum_{e \text{ out of } S} f(e)$$

$$f^{\text{in}}(S) = \sum_{e \text{ into } S} f(e).$$

В этой терминологии условие сохранения для вершин $v \neq s, t$

становится $f^{in}(v) = f^{out}(v)$;

и мы можем написать $v(f) = f^{out}(s)$.

Задача о максимальном потоке

Основная алгоритмическая проблема, которую мы рассмотрим, заключается в следующем:

Для заданной потоковой сети найти поток максимально возможной величины .

Когда мы думаем о разработке алгоритмов для этой задачи, полезно рассмотреть, как структура потоковой сети устанавливает верхние границы максимального значения *st*-потока .

Задача о максимальном потоке

Вот основное «препятствие» к существованию больших потоков:

Предположим, мы разделили вершины графа на 2 множества, A и B , так что $s \in A$ и $t \in B$.

Тогда интуитивно понятно, что любой поток, идущий из s в t, должен в какой-то момент перейти из множества A в множество B и тем самым использовать часть пропускной способности какого-то ребра из A в B.

Это говорит о том, что каждый такой «разрез» графа накладывает ограничение на максимально возможное значение потока.

Задача о максимальном потоке

Алгоритм поиска максимального потока, который мы здесь рассмотрим, будет переплетен с доказательством того, что значение максимального потока равно минимальной пропускной способности любого такого разбиения множества вершин, называемого минимальным разрезом.

В качестве бонуса наш алгоритм также вычислит минимальный разрез.

Мы увидим, что задача нахождения разрезов минимальной пропускной способности в потоковой сети оказывается столь же ценной с точки зрения приложений, как и задача нахождения максимального потока.

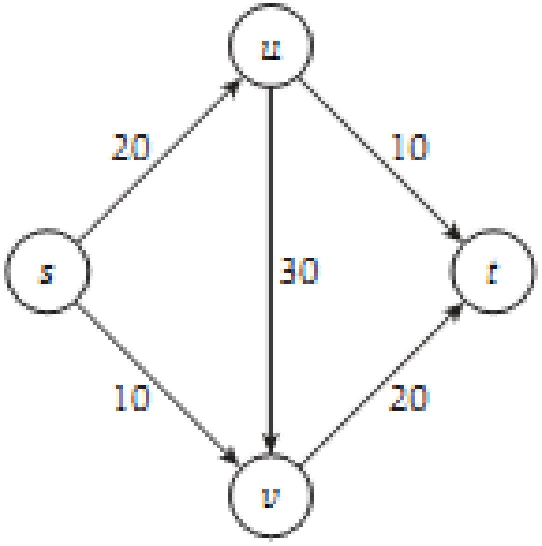
Разработка алгоритма

Предположим, мы хотим найти максимальный поток в сети.

Как нам следует это сделать?

Начнем с нулевого потока : $f(e) = 0$ для всех e .

Очевидно, что такой поток соответствует условиям пропускной способности и сохранения ; проблема в том, что его значение равно 0.



(a)

Разработка алгоритма

Теперь попытаемся увеличить значение *f*, «проталкивая» поток по пути от *s* до *t* , до пределов, налагаемых пропускной способностью ребер.

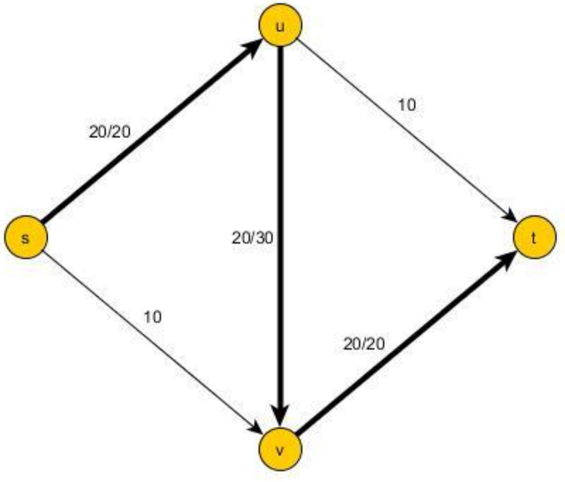
Например, мы могли бы выбрать путь, состоящий из ребер {(*s*, *u*), (*u*, *v*), (*v*, *t*)}, и увеличить поток на каждом из этих ребер до 20, а для двух других ребер оставить *f*(*e*) = 0.

Таким образом, мы по-прежнему соблюдаем условие пропускной способности, поскольку мы устанавливаем поток, насколько позволяют пропускные способности ребер.

— и условие сохранения — поскольку, когда мы увеличиваем поток на ребре, входящем во внутреннюю вершину, мы также увеличиваем его на ребре, выходящем из вершины.

Теперь значение нашего потока равно 20, и мы можем спросить:

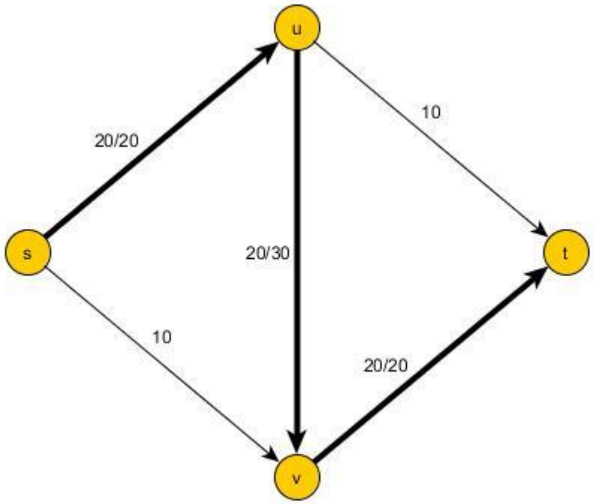
Является ли этот поток максимально возможным для данного графа?



Разработка алгоритма

Если задуматься, то мы увидим, что ответ — нет , поскольку возможно построить поток стоимостью 30.

Проблема в том, что теперь мы застряли — нет *пути* , по которому мы могли бы напрямую продвигать поток, не превышая некоторую пропускную способность, — и при этом у нас нет максимального потока.



Разработка алгоритма

Нам нужен более общий способ перемещения потока из *s* в *t* , чтобы в такой ситуации у нас был способ увеличить значение текущего потока .

По сути, мы хотели бы выполнить следующую операцию, обозначенную пунктирной линией.

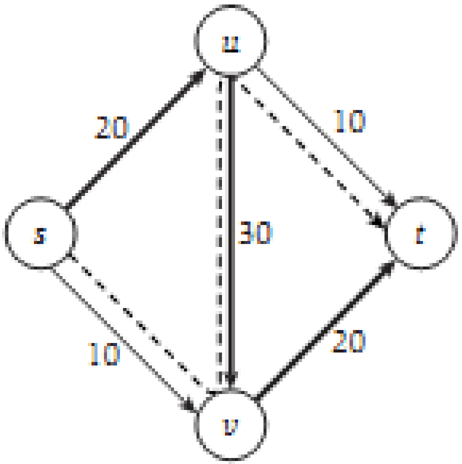
Мы проталкиваем 10 единиц потока вдоль ребра (*s*, *v*); это приводит к тому, что в *v* поступает слишком большой поток (20 + 10) .

Таким образом, мы «отменяем» 10 единиц потока на

Ребре (*u*, *v*); это восстанавливает условие сохранения в вершине *v* , но приводит к слишком малому потоку, покидающему *u* .

Тогда мы продвигаем 10 единиц потока вдоль ребра (*u*, *t*), восстанавливая условие сохранения в *u*.

Теперь у нас есть действительный поток, и его значение равно 30.

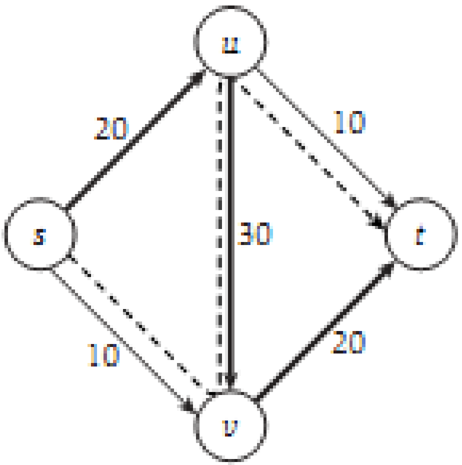


(c)

Разработка алгоритма

Это более общий способ продвижения потока:

Мы можем продвигаться вперед по ребрам с остаточной пропускной способностью, и мы можем продвигаться назад по ребрам, по которым уже идет поток, чтобы направить его в другом направлении .



(c)

Теперь мы определим остаточный граф.

Остаточный граф G_f

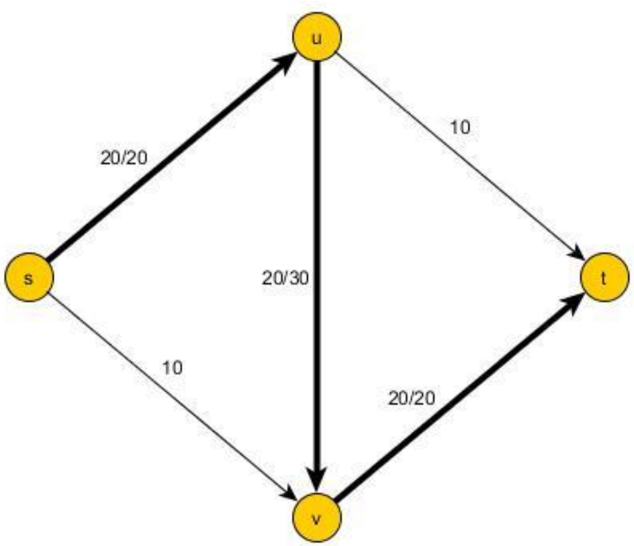
Дана потоковая сеть G и поток f в сети G, определим остаточный граф $G_f(G)$ относительно потока f следующим образом .

1) Множество вершин G_f *то же самое*, что у G.

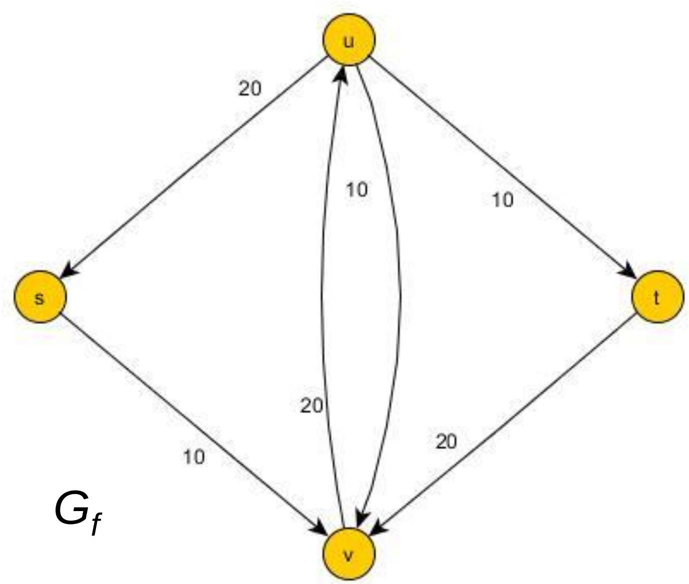
2) Для каждого ребра $e = (u, v)$ потоковой сети G, на котором $f(e) < c_e$, имеется $c_e - f(e)$ «остаточных» единиц пропускной способности, на которых мы могли бы попытаться продвинуть поток вперед.

Поэтому мы включаем в G_f ребро $e = (u, v)$ с пропускной способностью $c_e - f(e)$.

Включенные таким образом ребра будем называть прямыми ребрами .



G



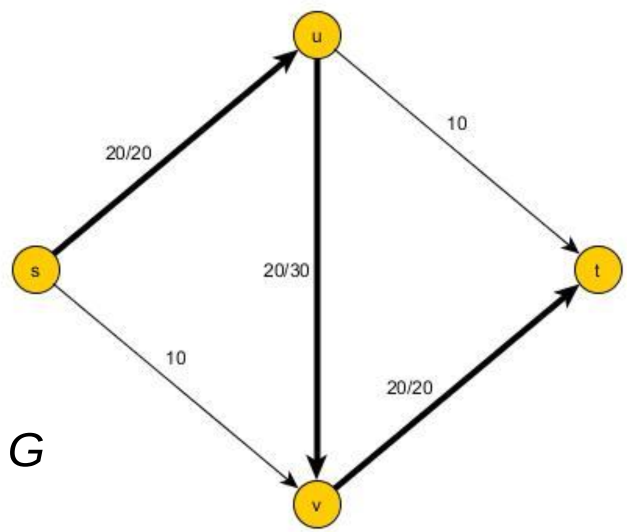
G_f

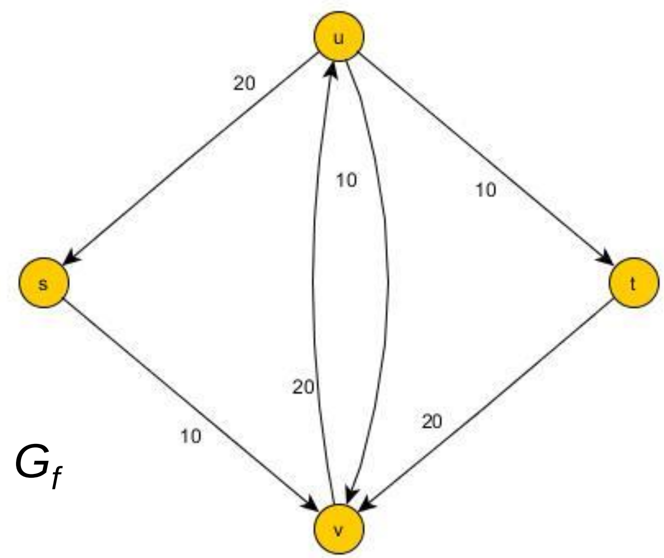
Остаточный граф G_f

3) Для каждого ребра $e = (u, v)$ графа G , на котором $f(e) > 0$, имеется $f(e)$ единиц потока, которые мы можем « отменить », если захотим, направив поток назад .

Поэтому мы включаем в G_f ребро $e = (v, u)$, с пропускной способностью $f(e)$.

Обратите внимание, что e имеет те же концы, что и e , но его направление противоположно; мы будем называть включенные таким образом ребра обратными ребрами .





Это завершает определение остаточного графа G_f .

Остаточный граф G_f

Заметим, что каждое ребро *e* в *G* может привести к появлению 1 или 2 ребер в G_f :

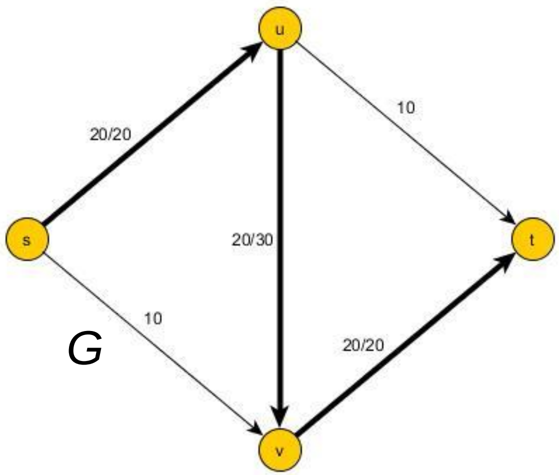
Если $0 < f(e) < c_e$ то и *прямое* и *обратное* ребро будут включены в G_f.

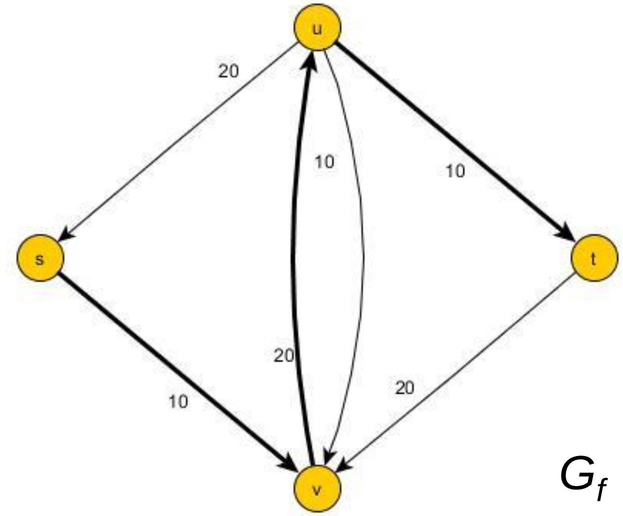
Таким образом, G_f имеет максимум вдвое больше ребер, чем *G:*

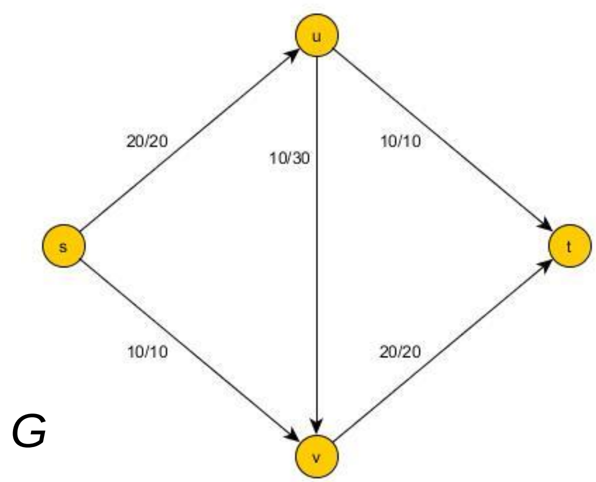
$|E_f| \leq 2\,|E|$.

Иногда мы будем называть пропускную способность ребра в остаточном графе *остаточной пропускной способностью,* чтобы отличить ее от пропускной способности соответствующего ребра в исходной потоковой сети *G.*

Остаточный граф G_f







Увеличивающий путь в остаточном графе

Теперь мы хотим уточнить способ, которым мы перемещаем поток из *s* в *t* в G_f.

Пусть *P* — простой *путь* в G_f, то есть *P* не посещает ни одну вершину более одного раза.

Определим (*bottleneck*(*P, f*), *критическая пропускная способность*) как минимальную остаточную пропускную способность любого ребра на *P* по отношению к потоку *f* .

Теперь определим операцию увеличения потока *augment*(*f, P*), *которая* дает новый поток *f* в *G.*

augment(*f*, *P*)

Let b = *bottleneck*(*P*, *f*)

For each edge $(u, v) \in P$

If $e = (u, v)$ is a forward edge **then**

increase $f(e)$ in G by b

Else ((u, v) is a backward edge, and let $e = (v, u)$)

decrease $f(e)$ in G by b

Endif

Endfor

Return(*f*)

Увеличивающий путь в остаточном графе

Мы определили остаточный граф исключительно для того, чтобы иметь возможность выполнить операцию увеличения потока.

Чтобы отразить важность увеличения потока, часто любой

st-путь в остаточном графе G_f называют увеличивающим путем.

Результатом augment(f, P) является новый поток f' в G, полученный путем увеличения и уменьшения значений потока на ребрах пути P.

Давайте сначала проверим, что f' действительно является потоком.

Свойства увеличивающего пути

Лемма 1 f' — поток в G.

Доказательство. Мы должны проверить *условия пропускной способности и сохранения для f'* .

Поскольку f' отличается от f только на ребрах пути P , нам нужно проверить условия пропускной способности только на этих ребрах.

Таким образом, пусть (u, v) будет ребром P.

Неформально, условие пропускной способности продолжает выполняться, поскольку если $e = (u, v)$ является прямым ребром , мы специально избегаем увеличения потока на ребре e больше чем c_e ;

и если (u, v) — обратное ребро, возникающее из ребра $e = (v, u) \in E$, мы специально избегали уменьшения потока на e ниже чем 0.

Свойства увеличивающего пути

Более конкретно, отметим, что *bottleneck(P, f)* $\leq$ остаточная пропускная способность $f\,(u,v)$.

Если $e = (\ u,\ v\ )$ — прямое ребро , то его остаточная пропускная способность равна $c_e - f(e)$;

таким образом, мы имеем

$f'\,(e) = f\,(e) + bottleneck\,(P\,,f) \geq\ f(e) \geq 0$

$f'\,(e) = f\,(e) +\ bottleneck\,(P\,,f\,) \leq f\,(e) + (c_e - f\,(e)) = c_e\,,$

поэтому условие пропускной способности выполняется.

Если $(\ u,\ v\ )$ — обратное ребро, возникающее из ребра $e = (\ v\ ,\ u\ ) \in E$, то его остаточная пропускная способность равна $f(e)$, так что у нас есть

$f'(e\ ) = f(e\ ) -\ bottleneck\ (\ P\ ,f\ ) \leq f\,(e) \leq c_e$

$f'(e\ ) = f(e\ ) -\ bottleneck\ (\ P\ ,f\ ) \geq f\,(e) - f\,(e) = 0,$

и снова условие пропускной способности выполняется.

Свойства увеличивающего пути

Далее нам необходимо проверить условие сохранения в каждой внутренней вершине, лежащей на пути *P.*

Пусть *v* будет такой вершиной; мы можем проверить, что изменение величины потока, входящего в *v,* такое же, как изменение величины потока, выходящего из *v* ;

поскольку *f* удовлетворяет условию сохранения в *v*, то и *f*' должен удовлетворять ему.

Технически, необходимо проверить 4 случая, в зависимости от того,

является ли ребро пути *P*, входящее *в v,* прямым или обратным ребром,

является ли ребро пути *P*, выходящее *из v,* прямым или обратным ребром.

Однако каждый из этих случаев легко прорабатывается, и мы оставляем их в качестве упражнения.

Алгоритм Форда-Фалкерсона (метод)

Давайте теперь рассмотрим следующий алгоритм вычисления st-потока в *G.*

Max-Flow

Initially $f(e) = 0$ for all e in G

While there is an s-t path in the residual graph G_f

Let P be a simple s-t path in G_f

$f' = augment(f, P)$

Update f to be f'

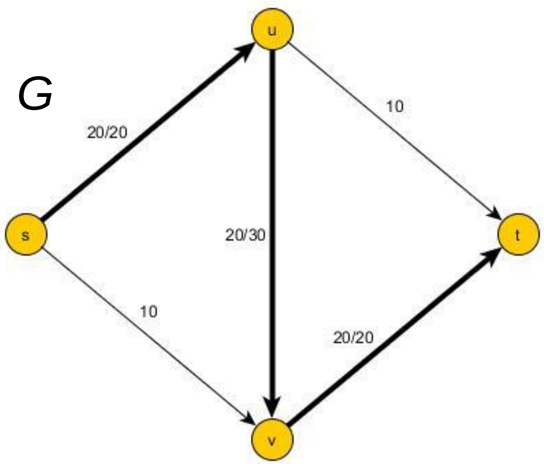
Update the residual graph G_f to be G_f'

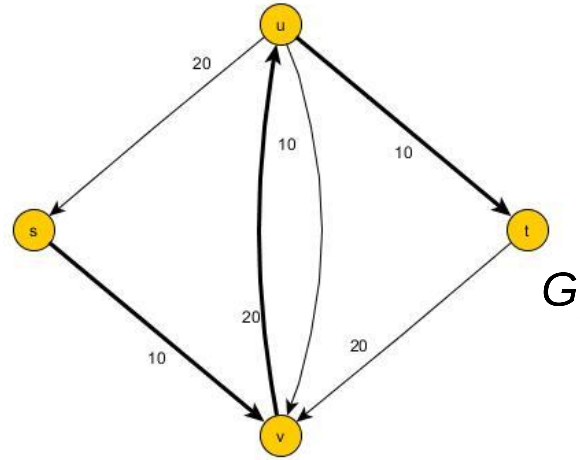
Endwhile

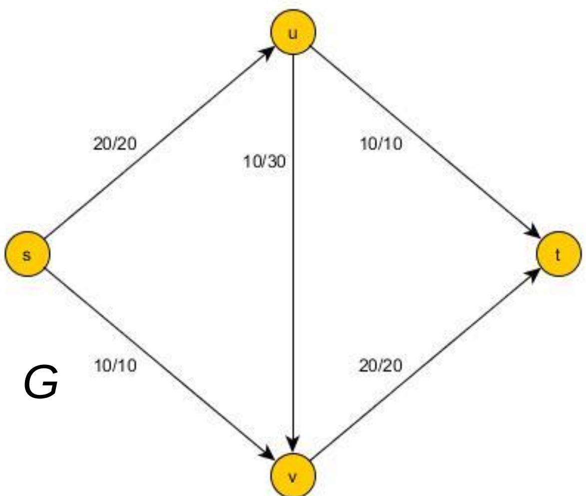
Return f

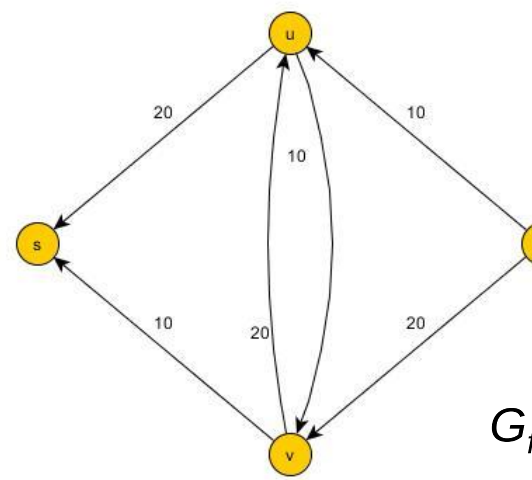
Он называется алгоритмом Форда-Фалкерсона, в честь двух исследователей, которые разработали его в 1956 году.

Пример работы алгоритма









Свойства алгоритма Форда-Фалкерсона

Алгоритм Форда-Фалкерсона на самом деле довольно прост.
Что не совсем ясно, так это:

- завершается ли его центральный цикл **While** , и
- является ли возвращаемый поток максимальным потоком.

Ответы на оба эти вопроса оказываются довольно тонкими.

Анализ алгоритма: завершение и время выполнения

Сначала рассмотрим некоторые свойства, которые поддерживает алгоритм, посредством индукции по числу итераций цикла **While** , полагаясь на наше предположение, что все пропускные способности являются целыми числами.

Лемма 2 На каждом промежуточном этапе алгоритма Форда-Фалкерсона значения потока { $f(e)$} и остаточные пропускные способности ребер в G_f являются целыми числами .

Доказательство. Утверждение очевидно истинно до начала любых итераций цикла **While** .

Теперь предположим, что это верно после j итераций.

Тогда, поскольку все остаточные пропускные способности в G_f являются целыми числами, значение bottleneck(P, f) для увеличивающего пути, найденного на итерации j + 1, будет целым числом.

Таким образом, поток f будет иметь целочисленные значения, а значит, и пропускные способности ребер нового остаточного графа.

Свойства алгоритма Форда-Фалкерсона

Мы можем использовать это свойство, чтобы доказать, что алгоритм Форда-Фалкерсона заканчивается.

Мы будем искать меру прогресса , которая будет означать прекращение работы алгоритма.

Сначала мы покажем, что значение потока строго увеличивается, когда мы применяем операцию увеличения потока.

Свойства алгоритма Форда-Фалкерсона

Лемма 3 Пусть f — поток в потоковой сети G , а P — простой st- путь в остаточном графе G_f .

Тогда $v(f') = v(f) + bottleneck(P, f);$

и поскольку $bottleneck(P, f) > 0$, то $v(f') > v(f)$.

Доказательство. Первое ребро e из P должно быть ребром, исходящим из s в остаточном графе G_f;

и поскольку путь P простой, он не посещает s снова.

Поскольку G не имеет ребер, входящих в s, ребро e должно быть прямым ребром .

Мы увеличиваем поток на этом ребре на $bottleneck(P, f)$ и не изменяем поток на любом другом ребре, инцидентном s .

Следовательно, значение f' превышает значение f на $bottleneck(P, f)$.

Граница максимального возможного значения потока

Для доказательства остановки работы алгоритма нам нужно еще одно наблюдение:

Нам необходимо иметь возможность ограничить максимальное возможное значение потока.

Вот одна верхняя граница:

Если бы все ребра, исходящие из *s,* могли быть полностью насыщены потоком, значение потока будет

$$\sum_{e \text{ out of } s} c_e.$$

Обозначим эту сумму *C.*

Таким образом, мы имеем $v(f) \leq C$ для всех *st*-потоков f .

(*C* может быть сильно завышенной оценкой максимального значения потока в *G*, но это нам удобно в качестве конечной, просто выраженной границы.)

Используя лемму 3, мы теперь можем доказать остановку.

Свойства алгоритма Форда-Фалкерсона

Лемма 4 Предположим, что все пропускные способности в потоковой сети G являются целыми числами.

Тогда алгоритм Форда-Фалкерсона заканчивается за $\leq C$ итераций **While** .

Доказательство. Выше мы отметили, что никакой поток в G *не* может иметь значение $> C$ из-за условия пропускной способности на ребрах, выходящих из s .

Теперь, согласно лемме 3, значение потока, поддерживаемого алгоритмом Форда-Фалкерсона, увеличивается с каждой итерацией ;

поэтому по лемме 2 он увеличивается на ≥ 1 в каждой итерации.

Поскольку он начинается со значения 0 и не может превышать C, цикл **While** в алгоритме Форда-Фалкерсона может выполняться $\leq C$ итераций.

Далее рассмотрим время работы алгоритма Форда-Фалкерсона.

Свойства алгоритма Форда-Фалкерсона

Пусть n обозначает количество узлов в G, а m обозначает количество ребер в G.

Мы предположили, что все вершины в G имеют по крайней мере 1 инцидентное ребро, следовательно, $m \geq n/2$, и поэтому мы можем использовать $O(m + n) = O(m)$ для упрощения границ.

Лемма 5 Предположим, как и выше, что все пропускные способности в потоковой сети G *являются* целыми числами.

Тогда алгоритм Форда-Фалкерсона может быть реализован так, чтобы он выполнялся за время $O(mC)$.

Доказательство. Из леммы 4 мы знаем, что алгоритм завершается максимум через C итераций цикла **While** .

Поэтому мы рассматриваем объем работы, выполняемой за 1 итерацию, когда текущий поток равен f .

Остаточный граф G_f имеет $\leq 2m$ ребер, поскольку каждое ребро графа G порождает ≤ 2 ребер в остаточном графе.

Мы будем поддерживать G_f с использованием представления списка смежности;

у нас будет 2 списка связностей для каждой вершины v, один из которых содержит ребра, входящие в v, а другой — ребра, исходящие из v .

Свойства алгоритма Форда-Фалкерсона

а) Чтобы найти st-*путь* в G_f, можно использовать поиск в ширину или поиск в глубину, которые выполняются за время $O(m + n)$;

В силу предположения, что $m \geq n/2$, $O(m + n)$ то же самое, что и $O(m)$.

б) Процедура *augment* (f , P) занимает время $O(n)$, так как путь P имеет не более $n - 1$ ребра.

с) Для нового потока f, можно построить новый остаточный граф G_f за время $O(m)$:

Для каждого ребра e графа G строятся прямые и обратные ребра в G_f .

Таким образом, алгоритм Форда-Фалкерсона может быть реализован за время $O(mC)$.

Максимальные потоки и минимальные разрезы в сети

Теперь продолжим анализ алгоритма Форда-Фалкерсона.

Наша следующая цель — показать, что поток, возвращаемый алгоритмом Форда-Фалкерсона, имеет максимально возможное значение среди всех потоков в *G*.

Анализ алгоритма: потоки и разрезы

Мы уже видели одну верхнюю границу: значение $v(f)$ любого st -потока f не превышает

$$C = \sum_{e \text{ out of } s} c_e.$$

Иногда эта граница полезна, но иногда она очень слаба. Теперь мы используем понятие разреза, чтобы рассмотреть гораздо более общие средства установления верхних границ значения максимального потока.

Анализ алгоритма: потоки и разрезы

Рассмотрим разбиение вершин графа *G* на 2 множества, *A* и *B* , так что $s \in A$, *a* $t \in B$.

Любое такое разбиение устанавливает верхнюю границу максимально возможного значения потока, поскольку весь поток должен где-то перейти из вершины в *A* в вершину в *B.*

Формально мы говорим, что *st* -разрез является разбиением (*A, B*) множества вершин *V*, так что $s \in A$ и $t \in B$.

Пропускная способность разреза (*A, B*), которую мы обозначим *c*(*A, B*), представляет собой просто сумму пропускных способностей всех ребер, исходящих из *A* :

$$c(A, B) = \sum_{e \text{ out of } A} c_e.$$

Оказывается, разрезы дают очень естественные верхние границы значений потоков.

Анализ алгоритма: потоки и разрезы

Лемма 6 Пусть f — любой st -поток, а (A, B) — любой st-разрез. Тогда $v(f) = f^{out}(A) - f^{in}(A)$.

Это утверждение на самом деле гораздо сильнее, чем простая верхняя граница.

В нем говорится, что, наблюдая за величиной потока f, проходящего через разрез, мы можем точно измерить значение потока:

Это общая сумма, которая покидает A , за вычетом суммы, которая «возвращается обратно» в A.

Доказательство. По определению $v(f) = f^{\text{out}}(s)$.

По предположению $f^{in}(s) = 0$, поскольку исток s не имеет входящих ребер, поэтому мы можем записать $v(f) = f^{out}(s) - f^{in}(s)$.

Поскольку каждый узел v в A, кроме s, является внутренним, мы знаем, что

$f^{out}(v) - f^{in}(v) = 0$ для всех таких узлов.

Анализ алгоритма: потоки и разрезы

Таким образом

$$\nu(f) = \sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v))$$

поскольку единственный член в этой сумме, который отличен от нуля, — это тот, в котором $v = s$.

Давайте попробуем переписать сумму справа следующим образом.

•Если ребро *e* имеет оба конца в *A*, то *f* (*e*) появляется в сумме один раз со знаком «+» и один раз со знаком «−», и, следовательно, эти два термина сокращаются .
•Если *e* имеет в *A* только хвост, то *f* (*e*) появляется в сумме только один раз со знаком «+».
•Если *e* имеет в *A* только голову, то *f* (*e*) также появляется в сумме только один раз, со знаком «−».
•Если *e* не имеет ни одного конца в *A*, то *f* (*e*) вообще не появляется в сумме.

Анализ алгоритма: потоки и разрезы

- В связи с этим, мы имеем

$$\sum_{v \in A} f^{\text{out}}(v) - f^{\text{in}}(v) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$$

$$= f^{\text{out}}(A) - f^{\text{in}}(A).$$

Объединяя эти два уравнения, получаем утверждение Леммы 6.

Если $A = \{ s \}$, то $f^{\text{out}}(A) = f^{\text{out}}(s)$ и $f^{\text{in}}(A) = 0$, поскольку по предположению в исток не входит ни одно ребро.
Таким образом, утверждение для этого множества $A = \{ s \}$ является в точности определением значения потока $v(f)$.

Анализ алгоритма: потоки и разрезы

Обратите внимание, что если (A, B) — разрез, то ребра, входящие в B, — это в точности ребра, исходящие из A.

Аналогично, ребра исходящие из B — это в точности ребра, входящие в A.

Таким образом, мы имеем $f^{out}(A) = f^{in}(B)$ и

$f^{in}(A) = f^{out}(B)$,

просто сравнив определения этих двух выражений.

Итак, мы можем перефразировать Лемму 6 следующим образом.

Лемма 7. Пусть f — любой st-поток , а (A, B) — любой st-разрез .

Тогда $v(f) = f^{in}(B) - f^{out}(B)$.

Если мы положим $A = V - \{ t \}$ и $B = \{ t \}$ в лемме 7,

имеем $v(f) = f^{in}(B) - f^{out}(B) = f^{in}(t) - f^{out}(t)$.

По нашему предположению сток t не имеет исходящих ребер, поэтому

$f^{out}(t) = 0$.

Это говорит о том, что мы могли бы изначально определить значение потока с тем же успехом в терминах стока t :

Это $f^{in}(t)$, величина потока, поступающего в сток.

Анализ алгоритма: потоки и разрезы

Очень полезным следствием леммы 6 является следующая верхняя оценка.

Лемма 8. Пусть f — любой st-поток, а (A, B) - любой st- разрез.

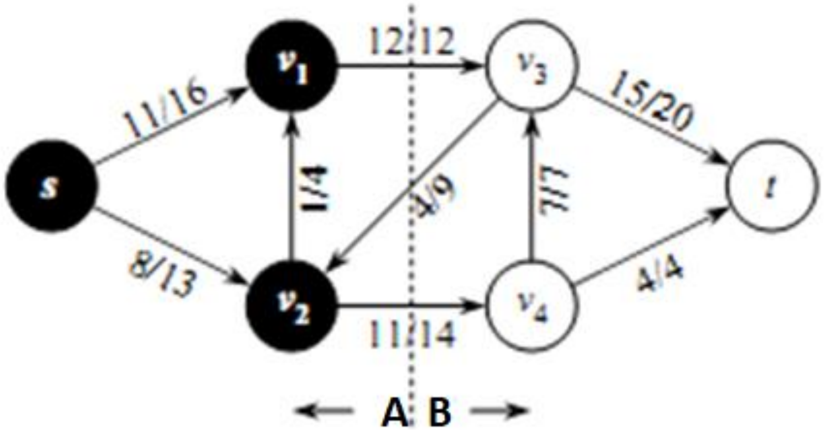
Тогда $\nu(f) \le c(A, B)$.

Доказательство .

$$\begin{aligned}
\nu(f) &= f^{\text{out}}(A) - f^{\text{in}}(A) \\
&\le f^{\text{out}}(A) \\
&= \sum_{e \text{ out of } A} f(e) \\
&\le \sum_{e \text{ out of } A} c_e \\
&= c(A, B).
\end{aligned}$$

Здесь первая строка — это просто Лемма 6; переходим от первого равенства ко второму неравенству, так как $f^{\text{in}}(A) \ge 0$, и переходим от третьего равенства к четвертому неравенству, применяя условие пропускной способности для каждого члена суммы.

Пример. Разрез и пропускная способность разреза



Разрез (A , B) в потоковой сети, где $A = \{ s, v_1 , v_2 \}$ и $B = \{ v_3, v_4 , t \}$.
Вершины в A черные, а вершины в B белые.
Поток через разрез (A , B) равен $f(A , B) = f (v_1 , v_3) + f (v_2 , v_4) - f (v_3 , v_2) = 12+ 11- 4 = 19$,
а пропускная способность разреза равна $c(A , B) = c(v_1 , v_3) + c(v_2 , v_4) = 12 + 14 = 26$.

Анализ алгоритма: потоки и разрезы

В некотором смысле лемма 8 выглядит слабее леммы 6, поскольку она представляет собой всего лишь неравенство, а не равенство.

Однако она будет чрезвычайно полезна, поскольку ее правая часть не зависит от какого-либо конкретного потока f .

Лемма 8 утверждает, что значение каждого потока ограничено сверху пропускной способностью каждого разреза.

Другими словами, если мы демонстрируем любой *st*- разрез в *G* с некоторым значением c^* , мы немедленно узнаем по лемме 8, что не может быть *st* -потока в *G* со значением $> c^*$.

Обратно, если мы демонстрируем любой *st*- поток в *G* с некоторым значением v^*, мы немедленно узнаем по лемме 8, что не может быть *st*- разреза в *G* со значением $< v^*$.

Анализ алгоритма: максимальный поток равен минимальному разрезу

Пусть f обозначает поток, возвращаемый алгоритмом Форда-Фалкерсона.

Мы хотим показать, что f имеет максимально возможное значение из всех потоков в G , и мы делаем это методом, описанным выше:

Мы находим st-разрез (A^*, B^*) для которого $v(f) = c(A^*, B^*)$.

Это немедленно устанавливает, что f имеет максимальное значение среди всех потоков, и что (A^*, B^*) имеет минимальную пропускную способность среди всех st-разрезов.

Алгоритм завершается, когда для данного потока f нет *ни одного* st-пути в остаточном графе G_f .

Это оказывается единственным свойством, необходимым для доказательства его максимальности.

Анализ алгоритма: максимальный поток равен минимальному разрезу

Лемма 9. Если f — *st-поток,* такой что в остаточном графе G_f нет *st- пути, то* в G существует *st*- разрез (A^*, B^*), для которого $v(f) = c(A^*, B^*)$.

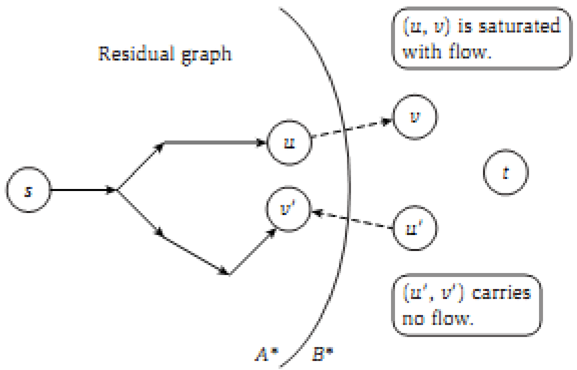
Следовательно, f имеет максимальное значение среди всех потоков в G, а (A^*, B^*) имеет минимальную пропускную способность среди всех *st*-разрезов в *G.*

Доказательство. В утверждении постулируется существование разреза, удовлетворяющего определенному желаемому свойству; таким образом, теперь мы должны идентифицировать такой разрез.

Для этого пусть A^* обозначает множество всех узлов v в *G,* для которых существует *st*-путь в G_f .

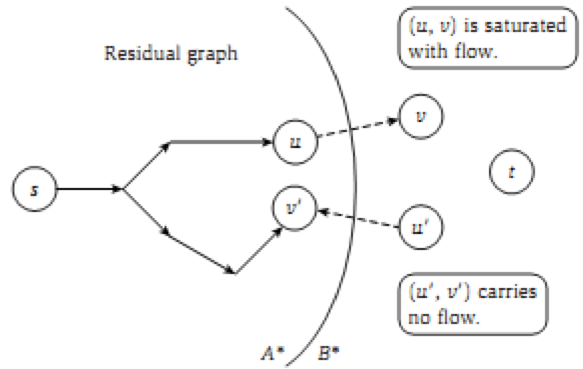
Пусть B^* обозначает множество всех остальных узлов: $B^* = V - A^*$.

Анализ алгоритма: максимальный поток равен минимальному разрезу



Сначала установим, что (*A**, *B**) действительно является
st-разрезом.
Во первых, это разбиение множества вершин *V.*
Источник *s* принадлежит *A*,* поскольку всегда существует путь из *s* в *s* .
Более того, $t \notin A^*$ в предположении, что в остаточном графе нет *st*-пути *;*
следовательно*,* $t \in B^*$.

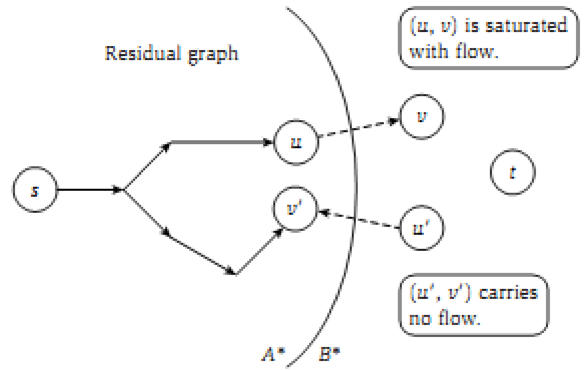
Анализ алгоритма: максимальный поток равен минимальному разрезу



Далее предположим, что $e = (u, v)$ — ребро в G, для которого $u \in A^*$ и $v \in B^*$, как показано выше.
утверждается, что $f(e) = c_e$.
В противном случае e будет прямым ребром в остаточном графе G_f, и поскольку $u \in A^*$, в G_f есть su-путь;

Добавив e к этому пути, мы получим sv-путь в G_f , что противоречит нашему предположению, что $v \in B^*$.

Анализ алгоритма: максимальный поток равен минимальному разрезу



Теперь предположим, что $e' = (u', v')$ — ребро в G, для которого $u' \in B^*$ и $v' \in A^*$.
Мы утверждаем, что $f(e') = 0$.
В противном случае e' приведет к появлению обратного ребра $e = (v', u')$ в остаточном графе G_f,
и так как $v' \in A^*$, в G_f есть sv –путь.
Добавив e к этому пути, мы получим su'-путь в G_f, что противоречит нашему предположению, что $u' \in B^*$.
Таким образом, все ребра, исходящие из A^*, полностью насыщены потоком, в то время как все ребра, входящие в B^* полностью не используются.

Анализ алгоритма: максимальный поток равен минимальному разрезу

Теперь мы можем использовать лемму 6, чтобы прийти к желаемому выводу:

$$\begin{aligned}
\nu(f) &= f^{\text{out}}(A^*) - f^{\text{in}}(A^*) \\
&= \sum_{e \text{ out of } A^*} f(e) - \sum_{e \text{ into } A^*} f(e) \\
&= \sum_{e \text{ out of } A^*} c_e - 0 \\
&= c(A^*, B^*). \quad \blacksquare
\end{aligned}$$

Анализ алгоритма: максимальный поток равен минимальному разрезу

Теперь мы можем понять, почему два типа остаточных ребер — прямое и обратное — имеют решающее значение при анализе двух членов в выражении из леммы 6.

Зная, что алгоритм Форда Фалкерсона останавливается, когда в остаточном графе нет *ни одного s-t-пути, из леммы 6 немедленно следует его оптимальность.*

Лемма 10. Поток *f*, возвращаемый алгоритмом Форда-Фалкерсона, является максимальным потоком.

Анализ алгоритма: максимальный поток равен минимальному разрезу

Мы также видим, что наш алгоритм можно легко расширить для вычисления минимального *st*- разреза (A^*, B^*) следующим образом.

Лемма 11. При наличии потока f максимального значения мы можем вычислить *st*- разрез минимальной пропускной способности за время $O(m)$.

Доказательство. Мы просто следуем конструкции в доказательстве леммы 9.

Построим остаточный граф G_f, и выполним поиск в ширину или в глубину, чтобы определить множество A^* всех вершин, достижимых из s .

Затем мы определяем $B^* = V - A^*$, и возвращаем разрез (A^*, B^*).

Анализ алгоритма: максимальный поток равен минимальному разрезу

Заметим, что в графе G может быть много разрезов минимальной пропускной способности ;

Процедура доказательства леммы 11 заключается просто в нахождении одного из этих разрезов, начиная с максимального потока f .

В качестве бонуса в результате анализа алгоритма мы получили следующий поразительный факт.

Лемма 12 В каждой потоковой сети существует поток f и разрез (A, B), такие что

$v(f) = c(A, B)$.

Дело в том, что f в Лемме 12 должен быть максимальным потоком st ; поскольку, если бы существовал поток f' большего значения, значение f' превысило бы пропускную способность (A , B), и это противоречило бы лемме 8.

Аналогично, отсюда следует, что разрез (A, B) в лемме 12 является минимальным разрезом — никакой другой разрез не может иметь меньшую пропускную способность — поскольку, если бы существовал разрез (A', B') меньшей пропускной способности, он был бы меньше значения f , и это снова противоречило бы лемме 8.

Анализ алгоритма: максимальный поток равен минимальному разрезу

В связи с этими следствиями лемму 12 часто называют

Теоремой о максимальном потоке и минимальном разрезе Она формулируется следующим образом.

В каждой потоковой сети, максимальное значение *s*-*t* потока равно минимальной пропускной способности *s*-*t* разреза .

Выбор хороших увеличивающих путей

Давайте теперь обсудим, как выбирать увеличивающие пути, чтобы избежать потенциально плохого поведения алгоритма.

Ранее мы видели, что любой способ выбора увеличивающегося пути увеличивает значение потока, и это привело к ограничению *C* на количество увеличений, где

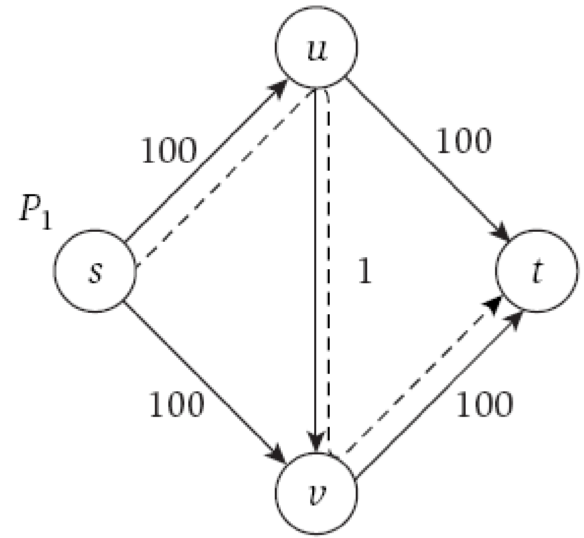
$$C = \sum_{e \text{ out of } s} c_e.$$

Если *C* не очень велико, это может быть разумной границей;
Однако она очень слаба, когда *C* велико.

Выбор хороших увеличивающих путей

Чтобы понять, насколько плохим может быть это ограничение, рассмотрим предыдущую потоковую сеть, но на этот раз предположим, что пропускны способности следующие.

Ребра (*s, v*), (*s, u*), (*v, t*) и (*u , t*) имеют пропускную способность 100, а ребро (*u, v*) имеет пропускную способность 1.



(a)

Легко видеть, что максимальный поток имеет значение 200 и *f*(*e*) = 100 для ребер (*s, v*), (*s, u*), (*v, t*) и (*u, t*) и *f*(*e*)= 0 на ребре (*u, v*).

Этот поток может быть получен путем последовательности двух увеличений, используя путь *s* , *u* , *t* и путь *s* , *v* , *t*

Выбор хороших увеличивающих путей

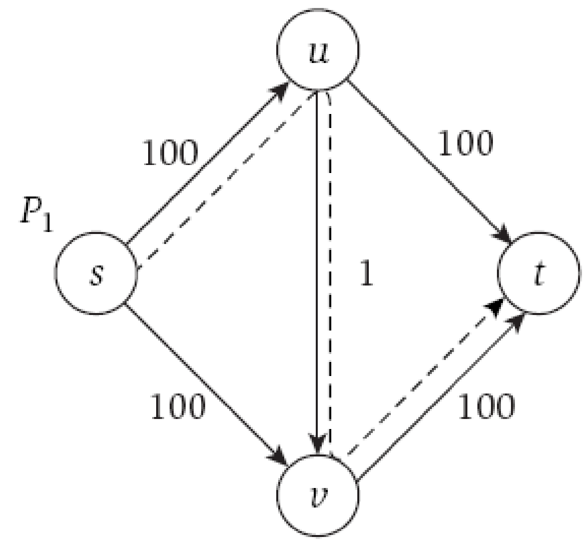
Но подумайте, насколько плохо может работать Алгоритм Форда-Фалкерсона с плохим выбором пути для увеличения потока.

Предположим, что мы начинаем с увеличения потока вдоль пути P_1 состоящего из вершин s , u , v , t в этом порядке (как показано справа) .

Этот путь имеет критическую пропускную способность *bottleneck*(P_1, f) = 1.

После увеличения, мы имеем

$f(e) = 1$ на ребре $e = (u, v)$, поэтому обратное ребро находится в остаточном графе.



(a)

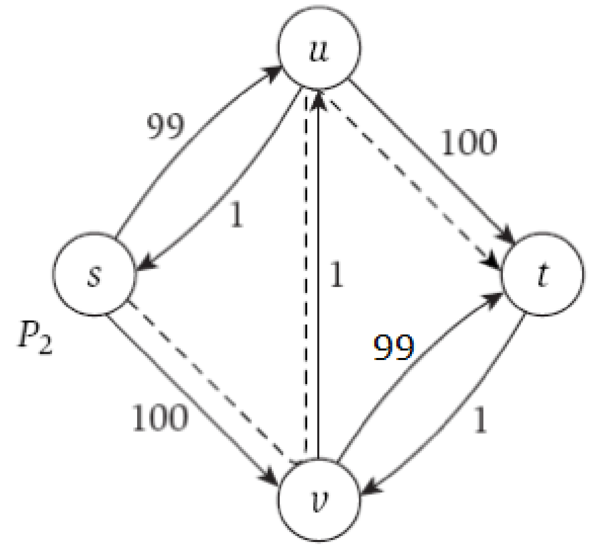
Выбор хороших увеличивающих путей

Для следующего пути увеличения потока мы выбираем путь P_2 , состоящий из вершин *s*, *v*, *u*, *t* в этом порядке.

После этого второго увеличения мы имеем $f(e) = 0$ для ребра

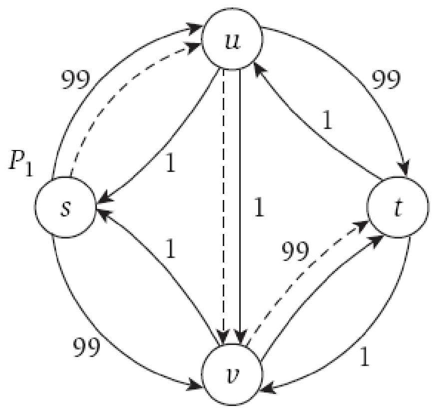
$e = (u, v)$, поэтому ребро снова находится в остаточном графе.

Предположим, что мы попеременно выбираем пути P_1 и P_2 для увеличения потока.

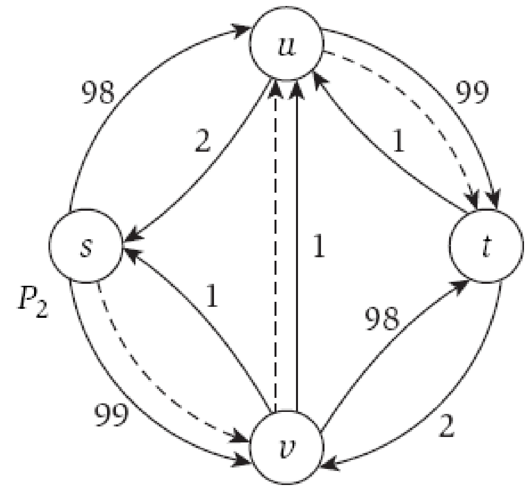


(b)

Выбор хороших увеличивающих путей



(c)



(d)

В этом случае, каждое увеличение потока будет равно критической пропускной способности =1, и потребуется сделать 200 увеличений потока, чтобы получить желанный поток значения 200.

Этот в точности граница, которая была доказана в Лемм4 4, так как *C* = 200 в этом примере.

Алгоритм Эдмондса-Карпа

Мы можем улучшить границу FORD-FULKERSON, найдя увеличивающийся путь P с помощью поиска в ширину .

То есть мы выбираем увеличивающийся путь как кратчайший путь из s *в* t в остаточной сети, где каждое ребро имеет единичный вес.

Реализованный таким образом метод Форда-Фалкерсона, называется алгоритмом Эдмондса-Карпа .

Утверждается, что алгоритм Эдмондса-Карпа выполняется за время $O(VE^2)$.

Алгоритм Эдмондса-Карпа

Теорема 13

Если алгоритм Эдмондса-Карпа запущен на потоковой сети $G = (V, E)$ с истоком s и стоком t , то общее количество увеличений потока, выполняемых алгоритмом, равно $O(VE)$.

Поскольку каждую итерацию алгоритма FORD-FULKERSON можно реализовать за время $O(E)$, когда мы находим увеличивающийся путь с помощью поиска в ширину, общее время выполнения алгоритма Эдмондса-Карпа составляет $O(VE^2)$.

- Ваши вопросы?

- Контакты лектора: apanovich_09@mail.ru