14.04.2025

Потоки

Филиппов Михаил Витальевич

m.filippov@g.nsu.ru

89232283872

Императивное программирование, 2024-2025

Новосибирский государственный университет

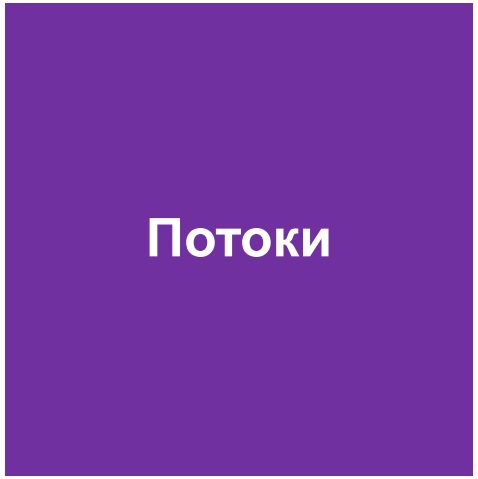
*НАСТОЯЩАЯ НАУКА

Давайте познакомимся

Филиппов Михаил Витальевич

- Окончил магистратуру ФФ НГУ
- Окончил аспирантуру ИТ СО РАН
- Являюсь м.н.с. ИТ СО РАН
- 7+ лет опыт в программировании C/C++

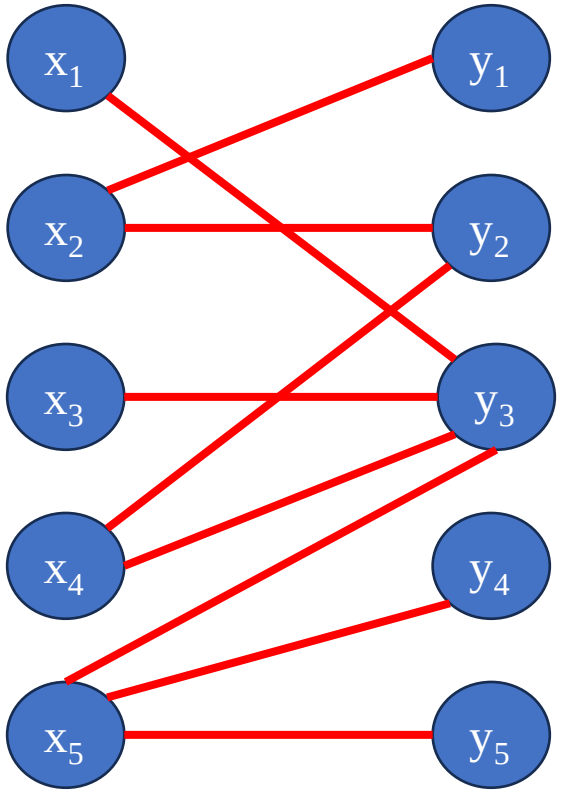
Адженда



90 минут

Введение

Так же, как дорожную карту можно смоделировать ориентированным графом, чтобы найти кратчайший путь из одной точки в другую, так и ориентированный граф можно интерпретировать как некоторую транспортную сеть и использовать для решения задач о потоках вещества в системе трубопроводов.

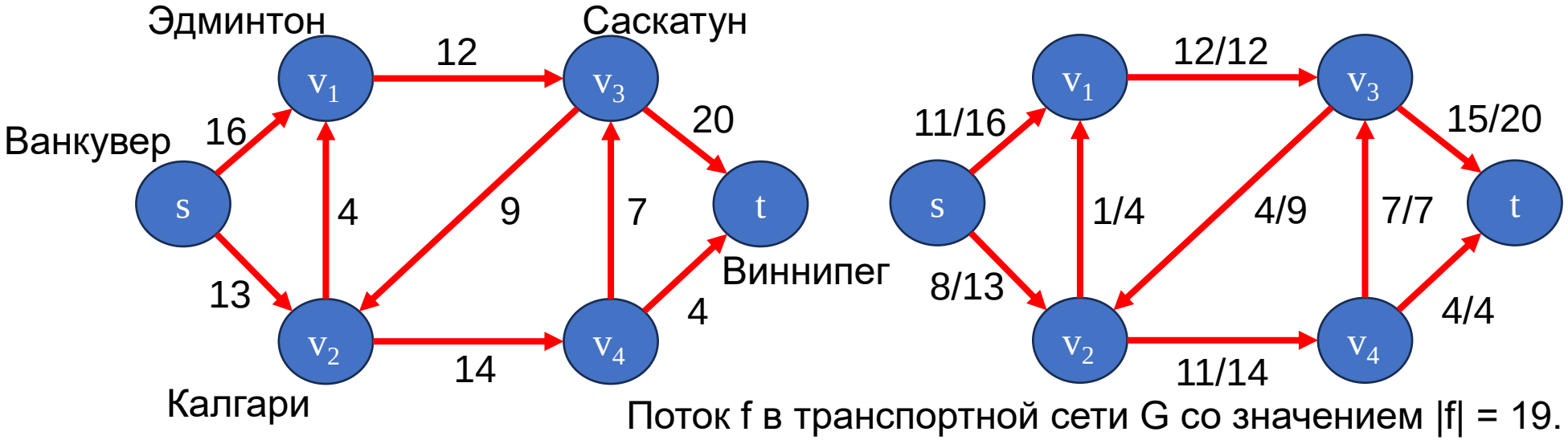


В задаче о максимальном потоке мы хотим найти максимальную скорость пересылки продукта от истока к стоку, при которой не будут нарушаться ограничения пропускной способности. Это задача, возникающая в транспортных сетях, и, существуют эффективные алгоритмы ее решения. Более того, основные методы, используемые в алгоритмах решения задач о максимальном потоке, можно применять для решения других задач, связанных с транспортными сетями.

Двудольным графом G = (V, E) называется ненаправленный граф, множество узлов которого может быть разбито на подмножества V = X ∪ Y, обладающее тем свойством, что один конец каждого ребра принадлежит X, а другой конец принадлежит Y. Двудольные графы часто изображаются так, как показано на рисунке: узлы X выстраиваются в столбец слева, узлы Y — в столбец справа, а ребра переходят между левым и правым столбцом.

Транспортные сети и потоки

Транспортная сеть (flow network) G = (V, E) представляет собой ориентированный граф, в котором каждое ребро (u,v) ∈ E имеет неотрицательную пропускную способность (capacity) c(u,v) > 0. Далее мы потребуем, чтобы в случае, если E содержит ребро (u,v), обратного ребра (v, u) не было (вскоре мы увидим, как обойти это ограничение). Если (u,v) ∉ E, то для удобства определим c(u,v) = 0, а также запретим петли. В транспортной сети выделяются две вершины: исток (source) s и сток (sink) t. Для удобства предполагается, что каждая вершина лежит на неком пути от истока к стоку, т.е. для любой вершины v ∈ V транспортная сеть содержит путь s ~ v ~ t. Таким образом, граф является связным и, поскольку каждая вершина, отличная от s, содержит как минимум одно входящее ребро, |E| > |V| - 1.



Поток f в транспортной сети G со значением |f| = 19.
Каждое ребро (u, v) имеет метку f(u,v)/c(u,v)

Транспортные сети и потоки

Пусть G = (V, E) - транспортная сеть с функцией пропускной способности c. Пусть s является истоком, а t - стоком. Потоком (flow) в G является действительная функция f : V x V → R, удовлетворяющая следующим двум условиям.

Ограничение пропускной способности. Для всех u,v ∈ V должно выполняться $0 \le f(u,v) \le c(u,v)$.

Сохранение потока. Для всех u ∈ V - {s, t} должно выполняться

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

Когда (u, v) ∉ E, потока из в v быть не может, так что f(u, v) = 0.

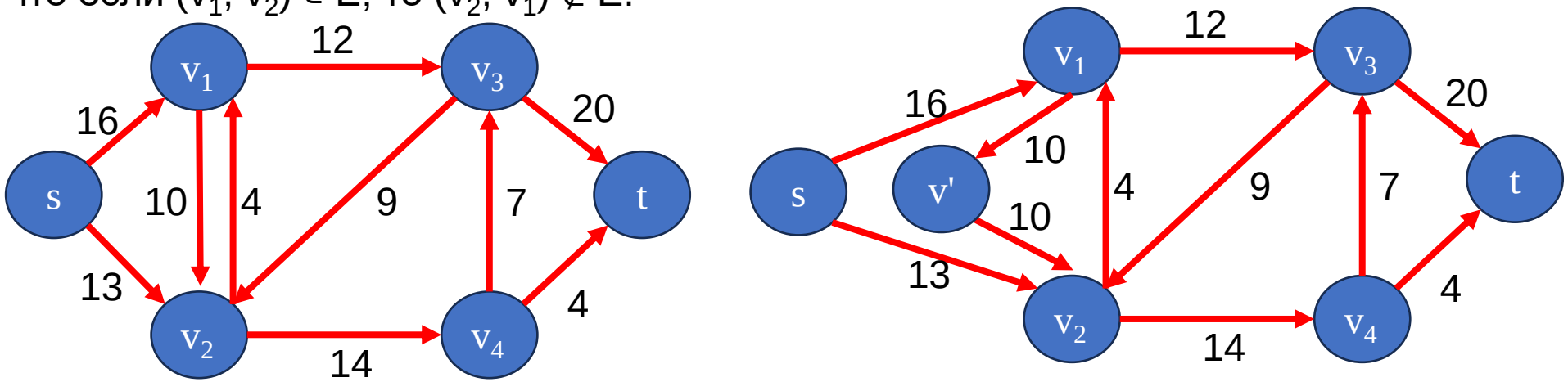
Неотрицательную величину f(u,v) мы называем потоком из вершины и в вершину v. Величина (value) |f| потока f определяется как

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

т.е. как суммарный поток, выходящий из истока, минус входящий в него. Обычно транспортная сеть не имеет ребер, входящих в исток, и поток в исток, задаваемый суммой $\sum_{v \in V} f(v,s)$, равен 0. В задаче о максимальном потоке (maximum flow problem) дана некоторая транспортная сеть G с истоком s и стоком t, и необходимо найти поток максимальной величины.

Моделирование задач с антипараллельными ребрами

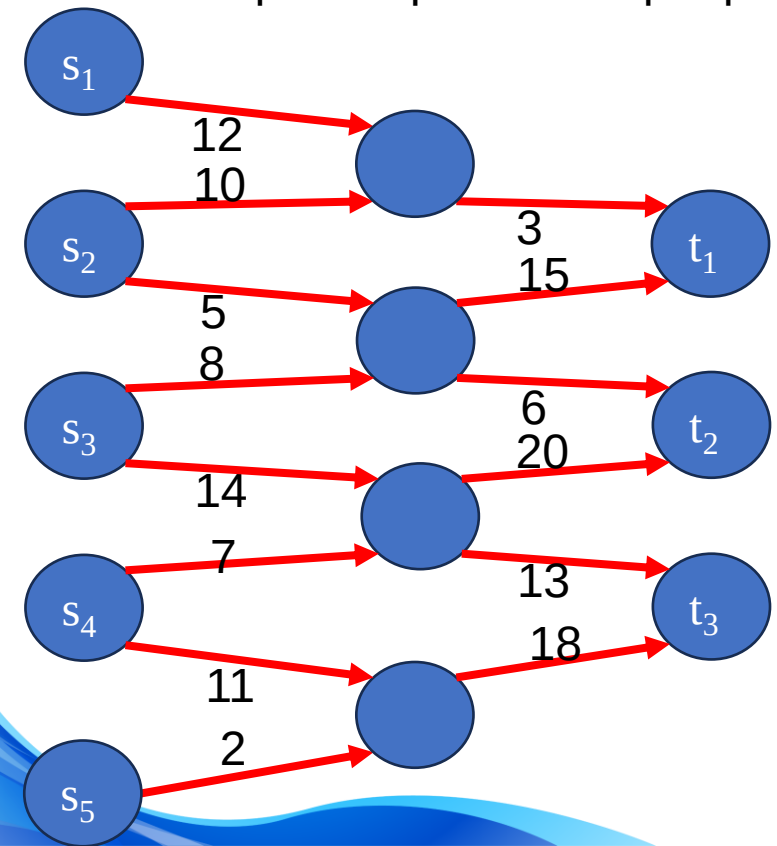
Предположим, что фирма-перевозчик предлагает Lucky Puck перевозку десяти ящиков в грузовике, идущем из Эдмонтона в Калгари. Представляется естественным добавить эту возможность в наш пример и получить транспортную сеть. Однако здесь возникает одна проблема: эта сеть нарушает исходное предположение о том, что если $(v_1, v_2) \in E$, то $(v_2, v_1) \notin E$.

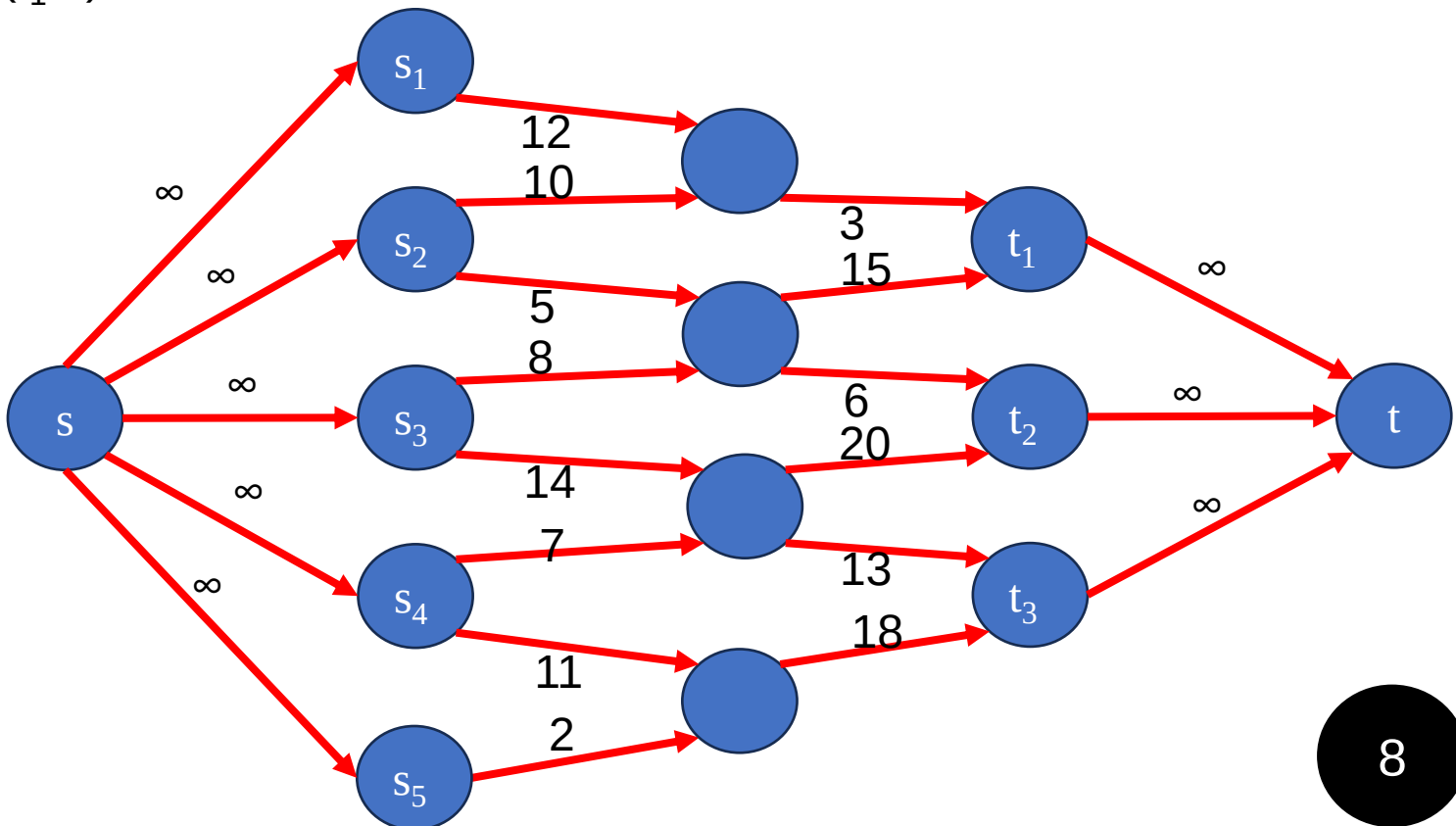


Преобразование сети с антипараллельными ребрами в эквивалентную сеть без таковых. (а) Транспортная сеть, содержащая как ребро (v_1, v_2), так и ребро (v_2, v_1). (б) Эквивалентная сеть без антипараллельных ребер. Добавлена новая вершина v', а ребро (v_1, v_2) заменено парой ребер (v_1, v') и (v', v_2) с одной и той же пропускной способностью (v_1, v_2).

Сети с несколькими истоками и стоками

В задаче о максимальном потоке может быть несколько истоков и стоков. Например, у компании Lucky Puck может быть m фабрик $\{s_1, s_2, \ldots, s_m\}$ и n складов $\{t_1, t_2, \ldots, t_n\}$. Задача определения максимального потока в сети с несколькими истоками и несколькими стоками сводится к обычной задаче о максимальном потоке. Для этого в сеть добавляются фиктивный исток (supersource) s и ориентированные ребра (s, s_i) с пропускной способностью $c(s, s_i) = \infty$ для каждого $i = 1,2,\ldots,m$. Точно так же создается фиктивный сток (supresink) t и добавляются ориентированные ребра (t_1, t) с $c(t_1, t) = \infty$ для каждого $i = 1,2,\ldots,n$.





Метод Форда-Фалкерсона

Метод Форда-Фалкерсона для решения задачи о максимальном потоке. Мы называем его методом, а не алгоритмом, поскольку он допускает несколько реализаций с различным временем выполнения. Метод Форда Фалкерсона базируется на трех важных идеях, которые выходят за рамки данного метода и применяются во многих потоковых алгоритмах и задачах. Это - остаточные сети, увеличивающие пути и разрезы. Метод Форда Фалкерсона итеративно увеличивает значение потока. Вначале поток обнуляется: $f(u, v) = 0$ для всех $u, v \in V$. На каждой итерации величина потока в G увеличивается посредством поиска "увеличивающего пути" в связанной "остаточной сети" G_f. Зная ребра увеличивающего пути в G_f, мы можем легко идентифицировать конкретные ребра в G, для которых можно изменить поток таким образом, что его величина увеличится. Хотя каждая итерация метода Форда-Фалкерсона увеличивает величину потока, мы увидим, что поток через конкретное ребро G может возрастать или уменьшаться; уменьшение потока через некоторые ребра может быть необходимым для того, чтобы позволить алгоритму переслать больший поток от истока к стоку. Мы многократно увеличиваем поток до тех пор, пока остаточная сеть не будет иметь ни одного увеличивающего пути.

Ford-Fulkerson-Method(G,s,t)

1 Инициализация потока f нулевым значением
2 while существует увеличивающий путь p в остаточной сети G_f
3 увеличиваем поток f вдоль пути p
4 return f

Остаточные сети

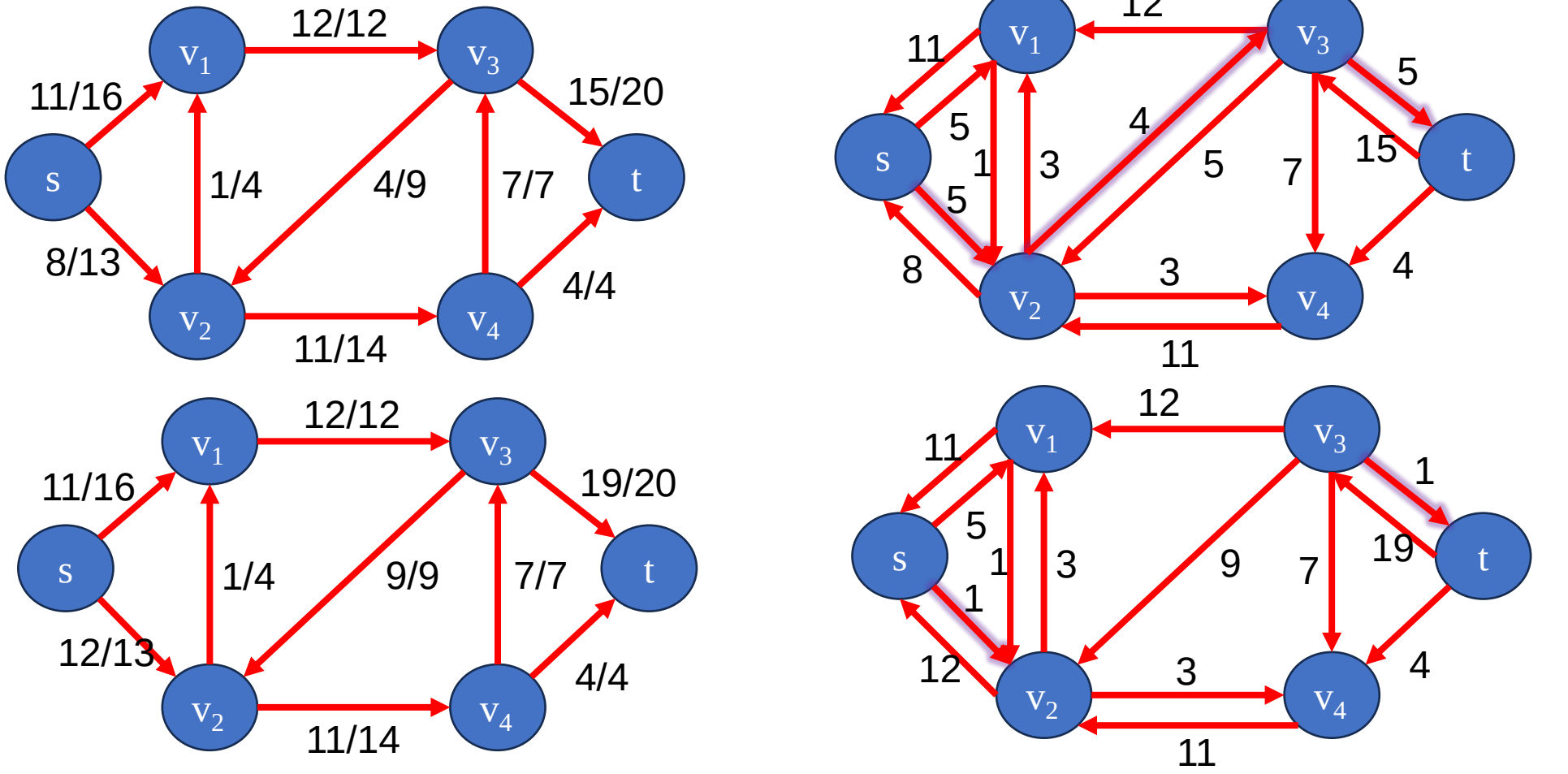
Интуитивно понятно, что если заданы некоторая транспортная сеть G и поток f, то остаточная сеть G_f – это сеть, состоящая из ребер с пропускными способностями, указывающими, как могут меняться потоки через ребра G. Ребро транспортной сети может пропустить дополнительный поток, равный пропускной способности ребра минус поток, проходящий через это ребро. Если это значение положительно, мы помещаем такое ребро в G_f с "остаточной пропускной способностью" $c_f(u, v) = c(u,v) - f(u, v)$. Дополнительный поток могут пропустить только те ребра в G, которые входят в G_f; ребра (u, v), поток через которые равен их пропускной способности, имеют $c_f(u, v) = 0$ и не входят в G_f.

Говоря более формально, предположим, что у нас есть транспортная сеть G = (V, E) с истоком s и стоком t. Пусть в G имеется поток f, и рассмотрим пару вершин u, v ∈ V. Определим ***остаточную пропускную способность*** $c_f(u, v)$ как

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

В силу нашего предположения том, что из (u,v) ∈ E вытекает (v,u) ∉ E, к каждой упорядоченной паре вершин применим только один случай из уравнения.

Остаточные сети



(а) Транспортная сеть G и поток f. (б) Остаточная сеть G с выделенным штриховкой увеличивающим путем p. (в) Поток в G, полученный в результате увеличения вдоль пути p на его остаточную пропускную способность 4. (г) Остаточная сеть, порожденная потоком в (в).

Остаточные сети

Для заданной транспортной сети G = (V, E) и потока f ***остаточная сеть*** (residual network) G, порожденная потоком f, представляет собой граф $G_f = (V, E_f)$, где

$$E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}.$$

Иначе говоря, как и отмечалось выше, по каждому ребру остаточной сети, или ***остаточному ребру***, можно направить поток, больший 0. Ребра в E_f являются либо ребрами из E, либо обратные им, и, таким образом,

$$\left|E_f\right| \leq 2\,|E]$$

Поток в остаточной сети предоставляет указания по добавлению потока к исходной транспортной сети. Если f представляет собой поток в G, а f' представляет собой поток в соответствующей остаточной сети G_f, определим ***увеличение*** (augmentation) $f \uparrow f'$ потока f на f' как функцию, отображающую V × V на R, определенную следующим образом:

$$(f \uparrow f')(u, v) = \begin{cases} \mathrm{f(u, v) + f'(u, v) - f'(v, u), если\ (u, v) \in E,} \\ 0 \text{ в противном случае.} \end{cases}$$

Интуитивно понятно, что это определение следует из определения остаточной сети. Мы увеличиваем поток в ребре (u, v) на f'(u, v), но уменьшаем его на f'(v, u), поскольку пропускание потока по обратным ребрам в остаточной сети означает уменьшение потока в исходной сети. Пропускание потока по обратному ребру в остаточной сети известно также как ***сокращение*** (cancellation).

Остаточные сети

Лемма 1.1 Пусть G = (V, E) является транспортной сетью с истоком s и стоком t и пусть f представляет собой поток в G. Пусть G_f - остаточная сеть G, порожденная f, и пусть f' - поток в G_f. Тогда функция $f \uparrow f'$, представляет собой поток в G с величиной $|f \uparrow f'| = |f| + |f'|$.

Доказательство. Сначала убедимся, что $f \uparrow f'$ подчиняется ограничению пропускной способности для каждого ребра в E и сохранению потока в каждой вершине в V - {s,t}.

В случае ограничения пропускной способности сначала заметим, что если $(u,v) \in E$, то $c_f(v,u) = f(u,v)$. Таким образом, мы имеем $f'(v,u) \le c_f(v,u) = f(u,v)$, а следовательно

$$(f \uparrow f')(u,v) = f(u,v) + f'(u,v) - f'(v,u) \ge f(u,v) + f'(u,v) - f(u,v) \text{ (поскольку } f'(v,u) \le f(u,v)) = f'(u,v) \ge 0.$$

Кроме того,

$$(f \uparrow f')(u,v) = f(u,v) + f'(u,v) - f'(v,u) \le f(u,v) + f'(u,v) \text{ (так как потоки неотрицательны)}$$
$$\le f(u,v) + c_f(u,v) \text{ (ограничение пропускной способности)} = f(u,v) + c(u,v) - f(u,v) \text{ (определение } c_f)$$
$$= c(u,v).$$

В случае сохранения потока, поскольку и f, и f' подчиняются свойству сохранения потока, для всех $u \in V - \{s,t\}$ имеем

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

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

где третье равенство следует из сохранения потока.

Наконец вычислим величину f ↑ f'. Вспомним, что антипараллельные ребра в G (но не в G_f) запрещены, а следовательно, для каждой вершины v ∈ V мы знаем, что может иметься либо ребро (s, v), либо ребро (v, s), но не оба одновременно. Определим V_1 = {v: (s, v) ∈ E} как множество вершин с ребрами из s, а V_2 = {v: (v, s) ∈ E} как множество вершин с ребрами в s. Мы имеем $V_1 \cup V_2 \subseteq V$ и, поскольку антипараллельные ребра запрещены, $V_1 \cap V_2 = \emptyset$. Теперь вычислим

$$|f \uparrow f'| = \sum_{v \in V} (f \uparrow f')(s, v) - \sum_{v \in V} (f \uparrow f')(v, s) = \sum_{v \in V_1} (f \uparrow f')(s, v) - \sum_{v \in V_2} (f \uparrow f')(v, s),$$

где вторая строка следует из того, что (f ↑ f')(w, x) равно 0, если (w, x) ∉ E. Теперь применим к уравнению определение f ↑ f', а затем переупорядочим и сгруппируем члены, чтобы получить

$$|f \uparrow f'| = \sum_{v \in V_1} (f(s, v) + f'(s, v) - f'(v, s)) - \sum_{v \in V_2} \big(f(v, s) + f'(v, s) - f'(s, v)\big)$$

$$= \sum_{v \in V_1} f(s, v) + \sum_{v \in V_1} f'(s, v) - \sum_{v \in V_1} f'(v, s) - \sum_{v \in V_2} f(v, s) - \sum_{v \in V_2} f'(v, s) + \sum_{v \in V_2} f'(s, v)$$

$$= \sum_{v \in V_1} f(s, v) - \sum_{v \in V_2} f(v, s) + \sum_{v \in V_1} f'(s, v) - \sum_{v \in V_2} f'(s, v) - \sum_{v \in V_1} f'(v, s) - \sum_{v \in V_2} f'(v, s)$$

$$= \sum_{v \in V_1} f(s, v) - \sum_{v \in V_2} f(v, s) + \sum_{v \in V_1 \cup V_2} f'(s, v) - \sum_{v \in V_1 \cup V_2} f'(v, s).$$

В уравнении можно распространить все четыре суммы на суммирование по всему множеству V, поскольку все дополнительные члены в этом случае имеют значения 0. Таким образом, мы имеем

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

Увеличивающие пути

Для заданных транспортной сети G = (V, E) и потока f ***увеличивающим путем*** (augmenting path) p является простой путь из s в t остаточной сети G_f. Согласно определению остаточной сети мы можем увеличить поток в ребре (u,v) увеличивающего пути до $c_f(u,v)$ без нарушения ограничения пропускной способности соответствующего ребра в исходной сети.

Максимальная величина, на которую можно увеличить поток в каждом ребре увеличивающего пути p, называется остаточной пропускной способностью (residual capacity) пути p и задается формулой

$$c_f(p) = min\{c_f(u,v) : (u,v) \text{ принадлежит } p\}.$$

Лемма 1.2

Пусть G = (V, E) является транспортной сетью, а f представляет собой поток в G, и пусть p является увеличивающим путем в G_f. Определим функцию f_p: V x V → R как

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

Тогда f_p является потоком в G_f с величиной $|f_p| = c_f(p) > 0$.

Следствие 1.3

Пусть G = (V, E) представляет собой транспортную сеть, а f является потоком в G, и пусть p представляет собой увеличивающий путь в G_f. Пусть также f_p определен и предположим, что мы увеличиваем f на f_p. Тогда функция $f \uparrow f_p$ является потоком в G с величиной $|f \uparrow f_p| = |f| + |f_p| > |f|$.

Разрезы транспортных сетей

В методе Форда-Фалкерсона проводится многократное увеличение потока вдоль увеличивающих путей до тех пор, пока не будет найден максимальный поток.
Разрезом (cut) (S,T) транспортной сети G = (V, E) называется разбиение множества вершин V на множества S и T = V - S, такие, что s ∈ S, а t ∈ T. Если f - поток, чистый поток (netflow) f(S,T) через разрез (S, T) определяется как

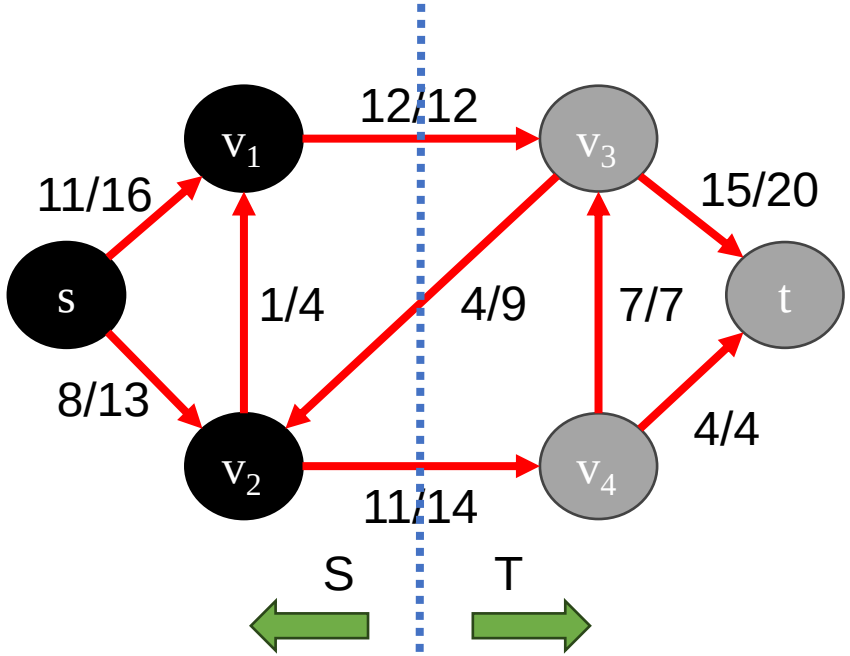
$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u).$$

Пропускной способностью (capacity) разреза (S, T) является

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

Минимальным разрезом (minimum cut) сети является разрез, пропускная способность которого среди всех разрезов сети минимальна.
Асимметрия между определениями потока и пропускной способности разреза является преднамеренной и существенной. В случае пропускной способности мы учитываем только пропускные способности ребер, идущих из S в T, игнорируя ребра, идущие в обратном направлении. Что касается потока, то мы рассматриваем поток из S в T минус поток, идущий в обратном направлении, из T в S.

Разрезы транспортных сетей



Разрез (S, T) в транспортной сети, где S = {s, v_1, v_2}, а T = {v_3, v_4, t}. Вершины в S показаны черными, а вершины в T - белыми. Чистый поток через разрез (S, T) равен f(S, T) = 19, а пропускная способность составляет c(S,T) = 26.

Чистый поток через данный разрез равен

$$f(v_1, v_3) + f(v_2, v_4) - f(v_3, v_2) = 12 + 11 - 4 = 19,$$

а пропускная способность данного разреза составляет

$$c(v_1, v_3) + c(v_2, v_4) = 12 + 14 = 26.$$

Разрезы транспортных сетей

Лемма 1.4

Пусть f представляет собой поток в транспортной сети G со стоком s и истоком t и пусть (S, T) - произвольный разрез G. Тогда чистый поток через разрез (S, T) равен f(S, T) = |f|.

Доказательство. Условие сохранения потока для любого узла u ∈ V - {s,t}можно переписать как

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

С учетом определения |f|, добавляя равную 0 левую часть уравнения, и суммируя по всем вершинам в S - {s}, получим

$$|f| \; = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) + \sum_{u \in S-\{s\}} \sum_{v \in V} f(u,v) - \sum_{v \in V} f(v,u)$$

Расширение правой суммы и перегруппировка членов дает

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) \; + \sum_{u \in S-\{s\}} \sum_{v \in V} f(u,v) - \sum_{u \in S-\{s\}} \sum_{v \in V} f(v,u)$$

$$= \sum_{v \in V} \left(f(s,v) + \sum_{u \in S-\{s\}} f(u,v) \right) - \sum_{v \in V} \left(f(v,s) + \sum_{u \in S-\{s\}} f(v,u) \right) = \sum_{v \in V} \sum_{u \in S} f(u,v) \; - \sum_{v \in V} \sum_{u \in S} f(v,u) \; .$$

Поскольку V = S ∪ T и S ∩ T = ∅, мы можем разбить каждое суммирование по V на суммы по S и T и получить

Разрезы транспортных сетей

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

$$= \sum_{v\in T}\sum_{u\in S} f(u,v) - \sum_{v\in T}\sum_{u\in S} f(v,u) + \left(\sum_{v\in S}\sum_{u\in S} f(u,v) - \sum_{v\in S}\sum_{u\in S} f(v,u)\right)$$

Две суммы в скобках на самом деле одинаковы, поскольку для всех вершин x, y ∈ S член f(x, y) в каждую сумму входит по одному разу. Следовательно, эти суммы сокращаются, и мы имеем

$$|f| = \sum_{u\in S}\sum_{v\in T} f(u,v) - \sum_{u\in S}\sum_{v\in T} f(v,u) = \mathrm{f(S,T)}\,.$$

Следствие леммы 1.4 показывает, как пропускные способности разрезов можно использовать для определения границы величины потока.

Следствие 1.5

Величина любого потока f в транспортной сети G ограничена сверху пропускной способностью произвольного разреза G.

Доказательство. Пусть (S,T) представляет собой произвольный разрез G и пусть f является произвольным потоком. Согласно лемме 1.4 и ограничению пропускной способности

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

Разрезы транспортных сетей

Теорема 1.6 (О максимальном потоке и минимальном разрезе)
Если f представляет собой поток в транспортной сети G = (V, E) с истоком s и стоком t, то следующие утверждения эквивалентны.

1. f является максимальным потоком в G.

2. Остаточная сеть G_f не содержит увеличивающих путей.

3. |f| = c(S, T) для некоторого разреза (S, T) транспортной сети G.

Доказательство. ***(1) = (2):*** предположим противное: пусть f является максимальным потоком в G, но G_f содержит увеличивающий путь p. Тогда, согласно следствию 1.3, поток, полученный путем увеличения потока f на поток f_p, представляет собой поток в G с величиной, строго большей, чем |f|, что противоречит предположению о том, что f является максимальным потоком.

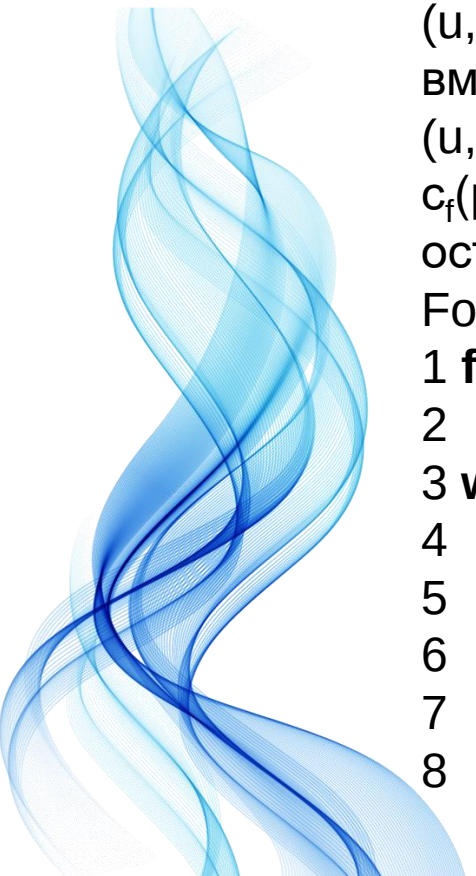
(2) => (3): предположим, что G_f не содержит увеличивающего пути, т.е. что в G_f нет пути из s в t. Определим S = {v ∈ V : в Gf имеется путь из s в v }и T = V - S. Разбиение (S,T) является разрезом: s ∈ S выполняется тривиально, а t ∉ S, поскольку в G_f нет пути из s в t. Теперь рассмотрим пару вершин u ∈ S и v ∈ T. Если (u,v) ∈ E, должно выполняться f(u,v) = c(u,v), поскольку в противном случае (u,v) ∈ E_f, что помещало бы вершину v в множество S.Если (v, u) ∈ E, должно выполняться f(v, u) = 0, поскольку в противном случае значение c_f(u, v) = f(v, u) должно было бы быть положительным, и мы должны были бы иметь (u, v) ∈ E_f, так что v должно было бы находиться в S. Конечно, если ни (u, v), ни (v, u) не находятся в E, то f(u, v) = f(v, u) = 0. Таким образом, имеем

$$f(S,T) = \sum_{u\in S}\sum_{v\in T} f(u,v) - \sum_{u\in S}\sum_{v\in T} f(v,u) = \sum_{u\in S}\sum_{v\in T} c(u,v) - \sum_{u\in S}\sum_{v\in T} 0 = c(S,T).$$

Следовательно, согласно лемме 1.4 |f| = f(S,T) = c(S,T).

(3) => (1): согласно следствию 1.5 |f| ≤ c(S,T) для всех разрезов (S,T). Таким образом, из условия |f| = c(S,T) вытекает, что поток f является максимальным потоком.

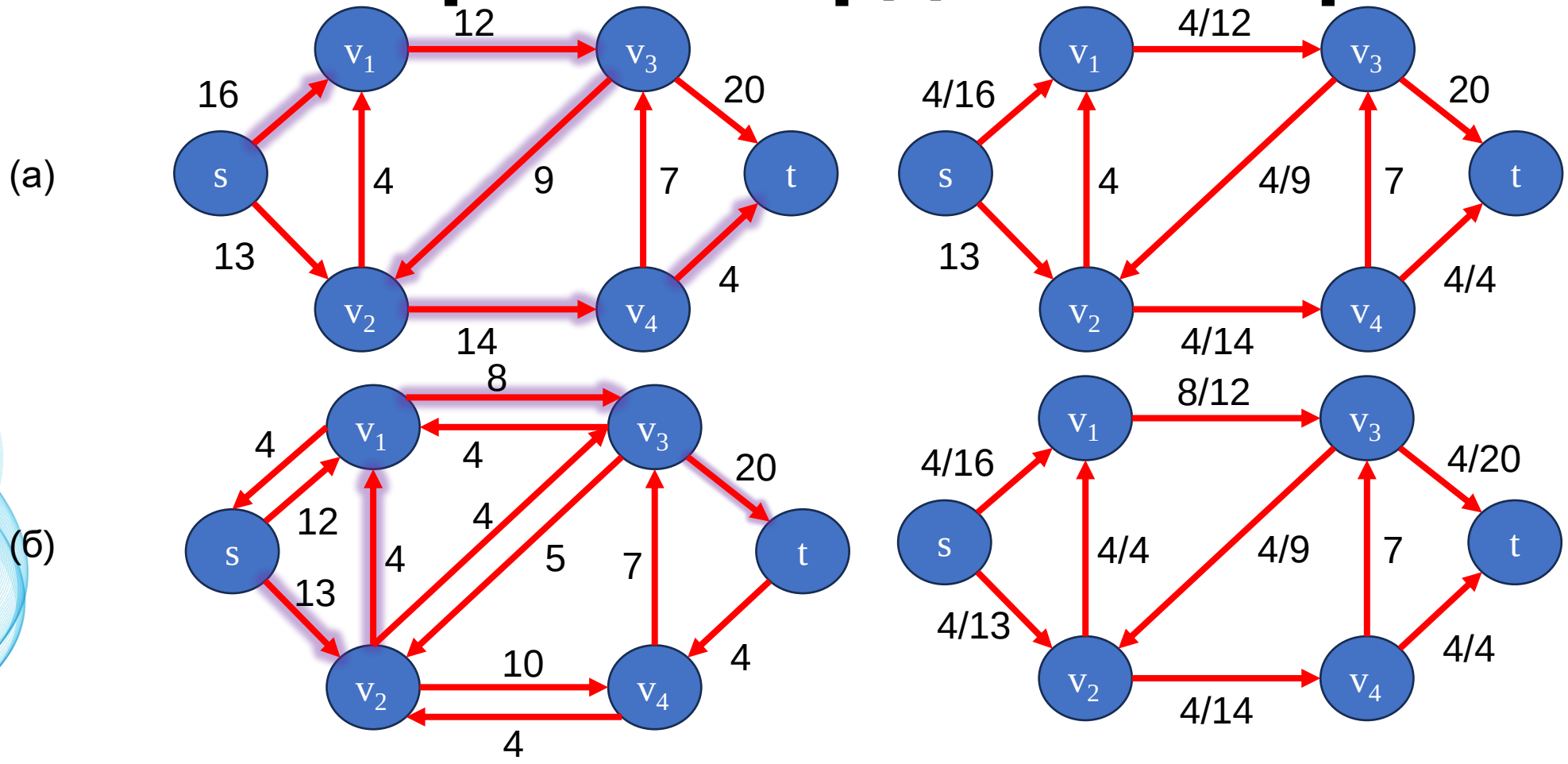
Базовый алгоритм Форда-Фалкерсона



При выполнении каждой итерации метода Форда-Фалкерсона мы находим некоторый увеличивающий путь p и используем p для того, чтобы изменять поток f. Как предлагается леммой 1.2 и следствием 1.3, мы заменяем f на $f \uparrow f_p$, получая новый поток, величина которого равна $|f| + |f_p|$. Приведенная далее реализация данного метода вычисляет максимальный поток в транспортной сети G = (V, E) путем обновления атрибута потока (u, v).f каждого ребра (u, v) ∈ E. Если (u, v) ∉ E, неявно предполагается, что (u, v).f = 0. Мы также считаем, что вместе с транспортной сетью задаются пропускные способности c(u, v) и что c(u,v) = 0, если (u,v) ∉ E. Остаточная пропускная способность $c_f(u,v)$ вычисляется по формуле f_p. Выражение $c_f(p)$ в коде процедуры является просто временной переменной, в которой хранится остаточная пропускная способность пути p.

```
Ford-Fulkerson(G, s, t)
1 for каждого ребра (u, v) ∈ G.E
2         (u, v).f = 0
3 while существует путь p из s в t в остаточной сети G_f
4         c_f(p) = min{c_f(u,v) : (u,v) содержится в p}
5         for каждого ребра (u, v) в p
6                 if (u, v) ∈ E
7                         (u,v).f = (u, v).f + c_f(p)
8                 else (v, u).f = (v, u).f - c_f(p)
```

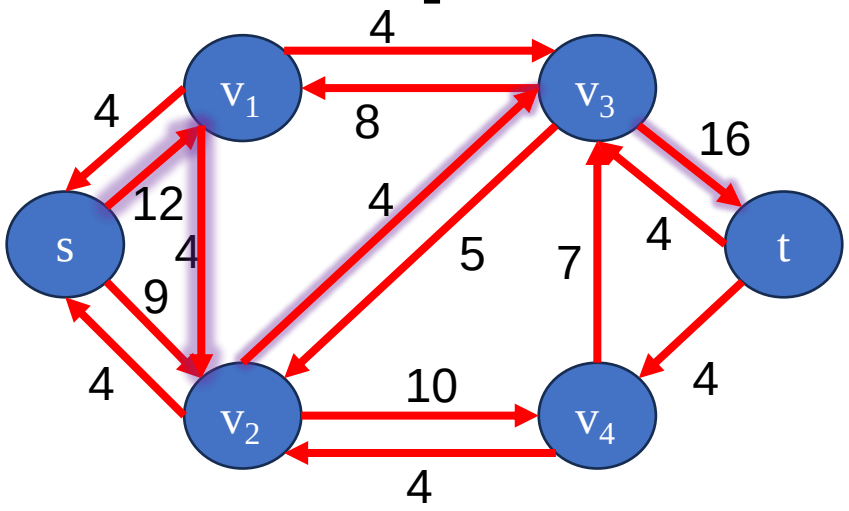
Базовый алгоритм Форда-Фалкерсона

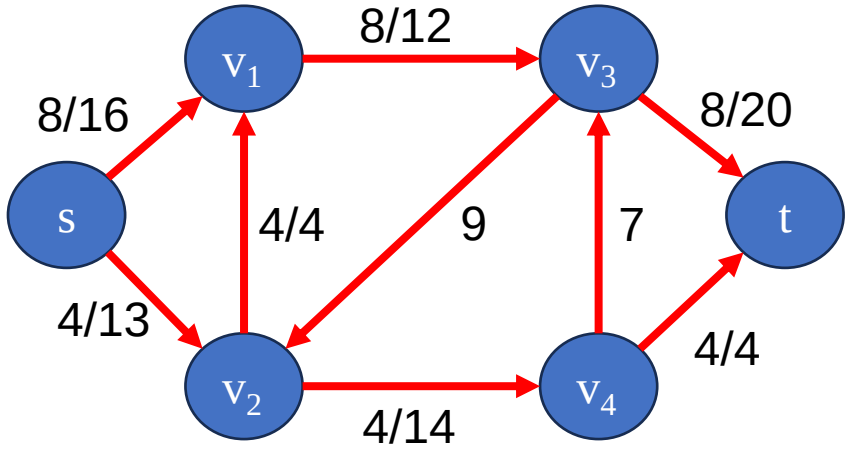


Работа базового алгоритма Форда Фалкерсона. (а) - (е) Последовательные итерации цикла while. Слева в каждой части показана остаточная сеть G_f из строки 3 с выделенным увеличивающим путем p. Справа показан новый поток f, который является результатом увеличения f на f_p. Остаточная сеть в (а) представляет собой входную сеть G.

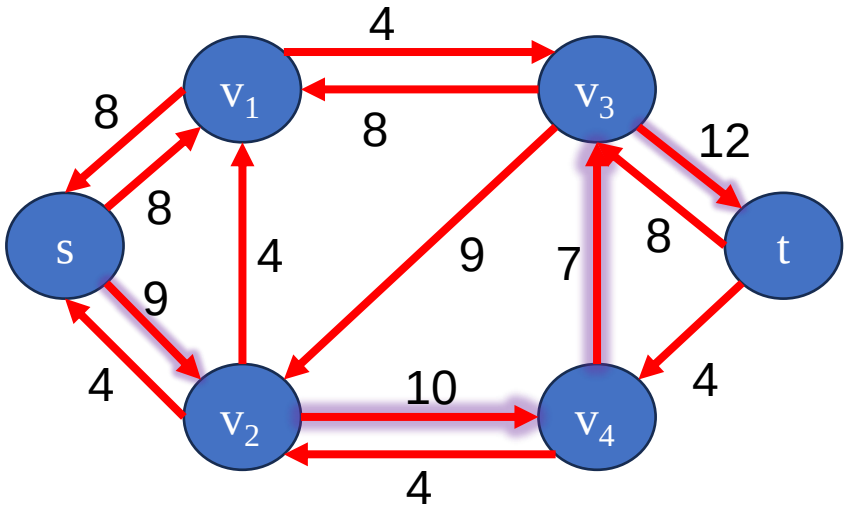
Базовый алгоритм Форда-Фалкерсона

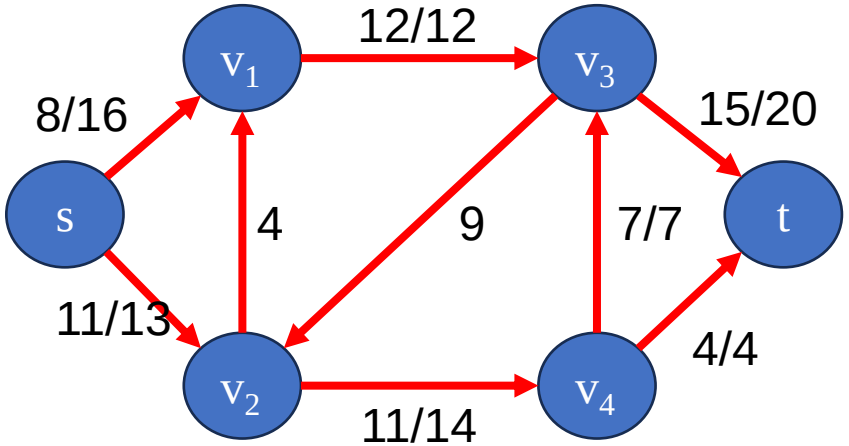
(в)





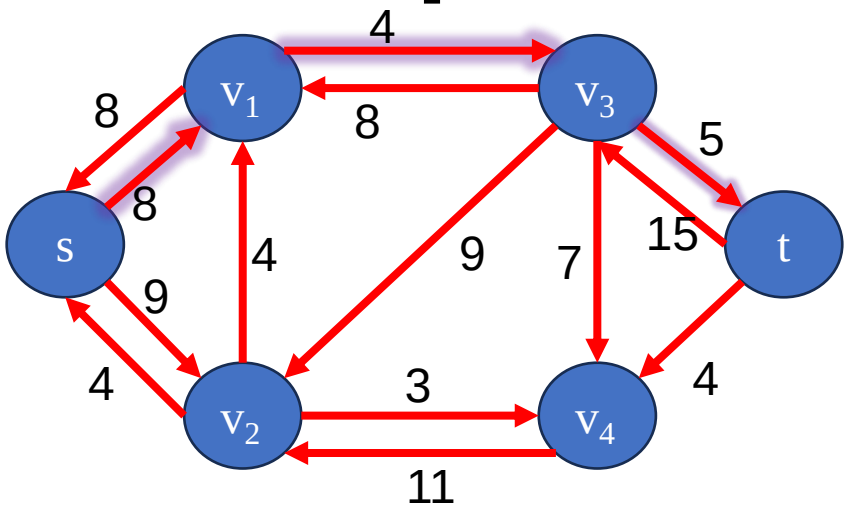
(г)

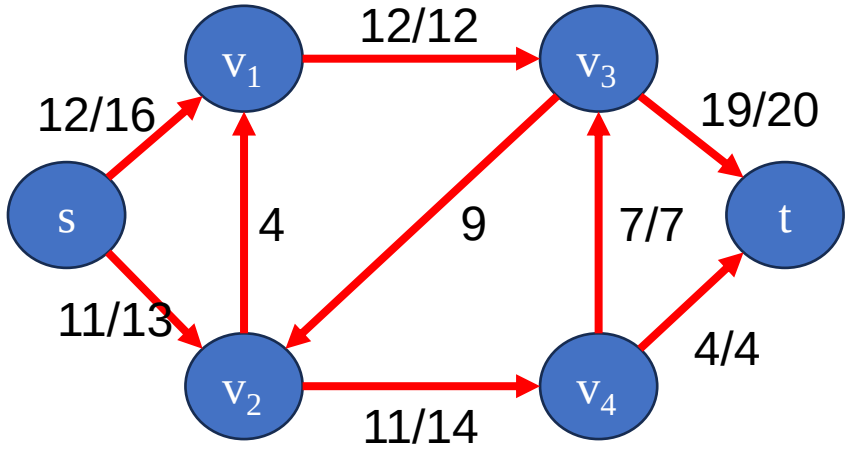




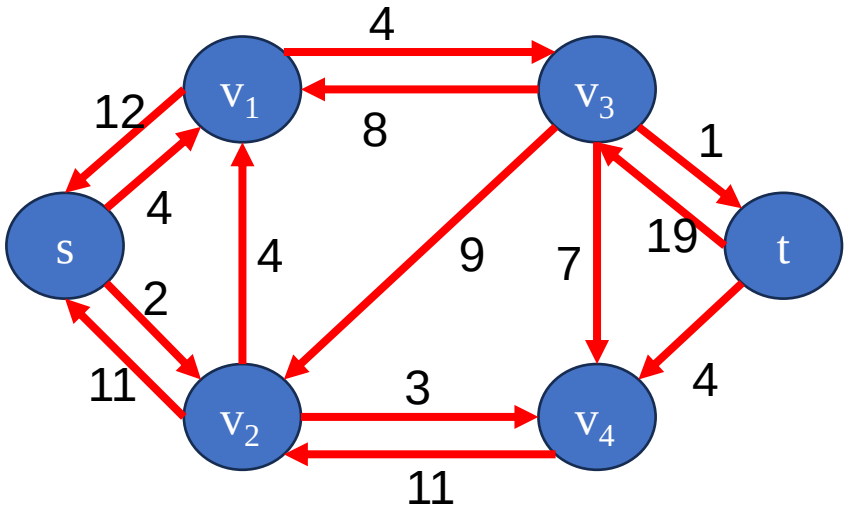
Базовый алгоритм Форда-Фалкерсона

(д)





(е)

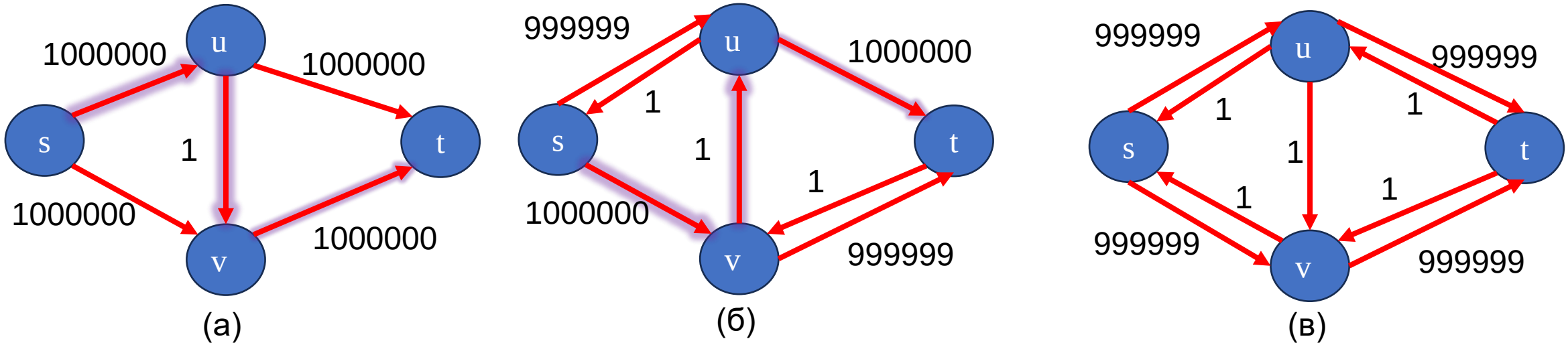


(е) Остаточная сеть при последней проверке цикла while. В ней нет увеличивающих путей, так что поток f, показанный в части (д), является максимальным. Величина найденного максимального потока равна 23.

Анализ метода Форда Фалкерсона

Время выполнения процедуры Ford-Fulkerson зависит от того, как именно выполняется поиск увеличивающего пути p в строке 3. При неудачном выборе метода поиска алгоритм может даже не завершиться: величина потока будет последовательно увеличиваться, но она не обязательно будет сходиться к максимальному значению потока. Если увеличивающий путь выбирается с использованием поиска в ширину, алгоритм выполняется за полиномиальное время. На практике задача поиска максимального потока часто возникает в целочисленной постановке. Если пропускные способности являются рациональными числами, можно использовать соответствующее масштабирование, которое сделает их целыми. Если обозначить максимальный поток в такой трансформированной сети как f*, то в случае непосредственной реализации процедуры Ford-Fulkerson цикл while в строках 3-8 выполняется не более |f*| раз, поскольку величина потока за каждую итерацию увеличивается по крайней мере на одну единицу. Цикл while будет выполняться эффективно, если реализовать транспортную сеть G = (V, E) с помощью правильно выбранной структуры данных и искать увеличивающий путь с помощью алгоритма с линейным временем работы. Предположим, что мы поддерживаем структуру данных, соответствующую ориентированному графу G' = (V, E'), где E' = {(u,v) : (u,v) ∈ E или (v, u) ∈ E}. Ребра сети G являются также ребрами графа G', поэтому в такой структуре данных можно довольно легко хранить пропускные способности и потоки. Для данного потока f в G ребра остаточной сети G_f состоят из всех ребер (u,v) графа G', таких, что $c_f(u,v) > 0$. Следовательно, время поиска пути в остаточной сети составляет O(V + E') = O(E), если используется либо поиск в глубину, либо поиск в ширину. Таким образом, каждая итерация цикла while занимает время O(E), что вместе с инициализацией в строках 1 и 2 делает общее время выполнения алгоритма Ford-Fulkerson равным O(E |f*|).

Анализ метода Форда Фалкерсона



(а) Транспортная сеть, для обработки которой алгоритму Ford-Fulkerson может по требоваться время Θ(E |f*|), где f* представляет собой максимальный поток, который в данном случае имеет величину |f*| = 2000000. Штриховкой выделен увеличивающий путь с остаточной пропускной способностью 1. (б) Полученная остаточная сеть с другим увеличивающим путем с остаточной пропускной способностью 1. (в) Полученная в результате остаточная сеть.

Алгоритм Эдмондса-Карпа

Можно улучшить временную границу алгоритма Ford-Fulkerson, если реализовать вычисление увеличивающего пути p в строке 3 как поиск в ширину, т.е. если в качестве увеличивающего пути выбрать кратчайший путь из s в t в остаточной сети, где каждое ребро имеет единичную длину (вес). Такая реализация метода Форда-Фалкерсона называется алгоритмом Эдмондса-Карпа (Edmonds-Karp algorithm). Докажем, что время выполнения алгоритма Эдмондса-Карпа составляет $O(VE^2)$.

Анализ зависит от расстояний между вершинами остаточной сети G_f. В следующей лемме длина кратчайшего пути из вершины и в v в остаточной сети G_f, где каждое ребро имеет единичную длину, обозначена как $\delta_f(u, v)$.

Лемма 1.7

Если применить алгоритм Эдмондса-Карпа к транспортной сети $G = (V, E)$ с истоком s и стоком t, то для всех вершин $v \in V - \{s,t\}$ длина кратчайшего пути $\delta_f(s, v)$ в остаточной сети G_f монотонно увеличивается с каждым увеличением потока.

Доказательство. Предположим, что для некоторой вершины $v \in V - \{s,t\}$ существует такое увеличение потока, которое приводит к уменьшению длины кратчайшего пути из s в v, и покажем, что это предположение приведет нас к противоречию. Пусть f - поток, который был непосредственно перед первым увеличением, приведшим к уменьшению длины некоего 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, так что $\delta_{f'}(s,v) < \delta_f(s,v)$.

Алгоритм Эдмондса-Карпа

Пусть p = s ⇝ u -> v - кратчайший путь из s в v в G_f, такой, что (u, v) ∈ $E_{f'}$ 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 из истока s не уменьшилась, т.е.

$$\delta_{f'}(s,u) \geq \delta_{f}(s,u).$$

Мы утверждаем, что (u, v) ∉ E_f. Почему? Если мы имели (u, v) ∈ E_f, то должны были также выполняться соотношения

$$\delta_{f}(s,v) \leq \delta_{f}(s,u) + 1(\text{неравенство треугольника}) \leq \delta_{f'}(s,u) + 1 = \delta_{f'}(s,v),$$

что противоречит нашему предположению о том, что $\delta_{f'}(s, v) < \delta_f(s, v)$.

Как же может получиться (u, v) ∉ E_f и (u, v) ∈ $E_{f'}$? Увеличение должно привести к возрастанию потока из v в u. Алгоритм Эдмондса-Карпа всегда увеличивает поток вдоль кратчайших путей, поэтому последним ребром кратчайшего пути из s в u в G_f является ребро (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 нашему предположению о том, что $\delta_{f'}(s, v) < \delta_f(s, v)$. Мы заключаем, что наше предположение о существовании такой вершины v неверное.

Алгоритм Эдмондса-Карпа

Теорема 1.8

Если алгоритм Эдмондса-Карпа выполняется для некоторой транспортной сети G = (V, E) с истоком s и стоком t, то общее число увеличений потока, выполняемых данным алгоритмом, составляет O(VE).

Доказательство. Назовем ребро (u,v) остаточной сети G_f критическим (critical) для увеличивающего пути p, если остаточная пропускная способность p равна остаточной пропускной способности ребра (u, v), т.е. если $c_f(p) = c_f(u, v)$. После увеличения потока вдоль увеличивающего пути все критические ребра этого пути исчезают из остаточной сети. Кроме того, по крайней мере одно ребро любого увеличивающего пути должно быть критическим. Теперь покажем, что каждое из |E| ребер может становиться критическим не более |V|/2 раз. Пусть u и v являются вершинами из множества вершин V, соединенными некоторым ребром из множества E. Поскольку увеличивающие пути являются кратчайшими путями, то, когда ребро (u, v) становится критическим в первый раз, справедливо равенство

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

После того как поток увеличен, ребро (u, v) исчезает из остаточной сети. Оно 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 ребро (v,u). Если в этот момент в сети G поток представляет собой f', то справедливо следующее равенство

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1.$$

Алгоритм Эдмондса-Карпа

Поскольку $\delta_{f'}(s,v) \geq \delta_f(s,v)$, согласно лемме 1.7, мы имеем

$$\delta_{f'}(s,v) = \delta_{f'}(s,u) + 1 \geq \delta_f(s,u) + 1.$$

Следовательно, за время, прошедшее с момента, когда ребро (u, v) стало критическим, до момента, когда оно станет критическим в следующий раз, расстояние до u от истока увеличивается не менее чем на 2. Расстояние до u от истока 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 ребра (u,v) в увеличивающем пути влечет за собой u ≠ t). Следовательно, к тому моменту, когда вершина и станет недостижимой из истока (если такое произойдет), расстояние до нее будет не более |V| - 2. Таким образом, ребро (u, v) может стать критическим не более чем еще (|V| - 2)/2 = |V| /2 - 1 раз, т.е. всего не более |V| /2 раз. Поскольку в остаточном графе имеется O(E) пар вершин, которые могут быть соединены ребрами в остаточной сети, общее количество критических ребер в ходе выполнения алгоритма Эдмондса-Карпа равно O(VE). Каждый увеличивающий путь содержит по крайней мере одно критическое ребро, а значит, теорема доказана.

Поскольку, если использовать поиск в ширину, можно выполнять каждую итерацию процедуры Ford-Fulkerson за время O(E), общее время работы алгоритма Эдмондса-Карпа оказывается равным $O(VE^2)$.

Задача поиска максимального паросочетания в двудольном графе

Пусть дан неориентированный граф G = (V, E). Паросочетанием (matching) называется подмножество ребер M ⊆ E, такое, что для всех вершин v ∈ V в M содержится не более одного ребра, инцидентного v. Мы говорим, что вершина v ∈ V является связанной (matched) паросочетанием M, если в M есть ребро, инцидентное v; в противном случае вершина v называется открытой (unmatched). Максимальным паросочетанием называется паросочетание максимальной мощности, т.е. такое паросочетание M, что для любого паросочетания M' справедливо соотношение |M| ≥ |M'|. Мы ограничимся рассмотрением задачи поиска максимальных паросочетаний в двудольных графах, т.е. в графах, множество вершин которых можно разбить на два подмножества V = L ∪ R, где L и R не пересекаются и все ребра в E проходят между L и R. Далее мы предполагаем, что каждая вершина в V имеет по крайней мере одно инцидентное ребро.

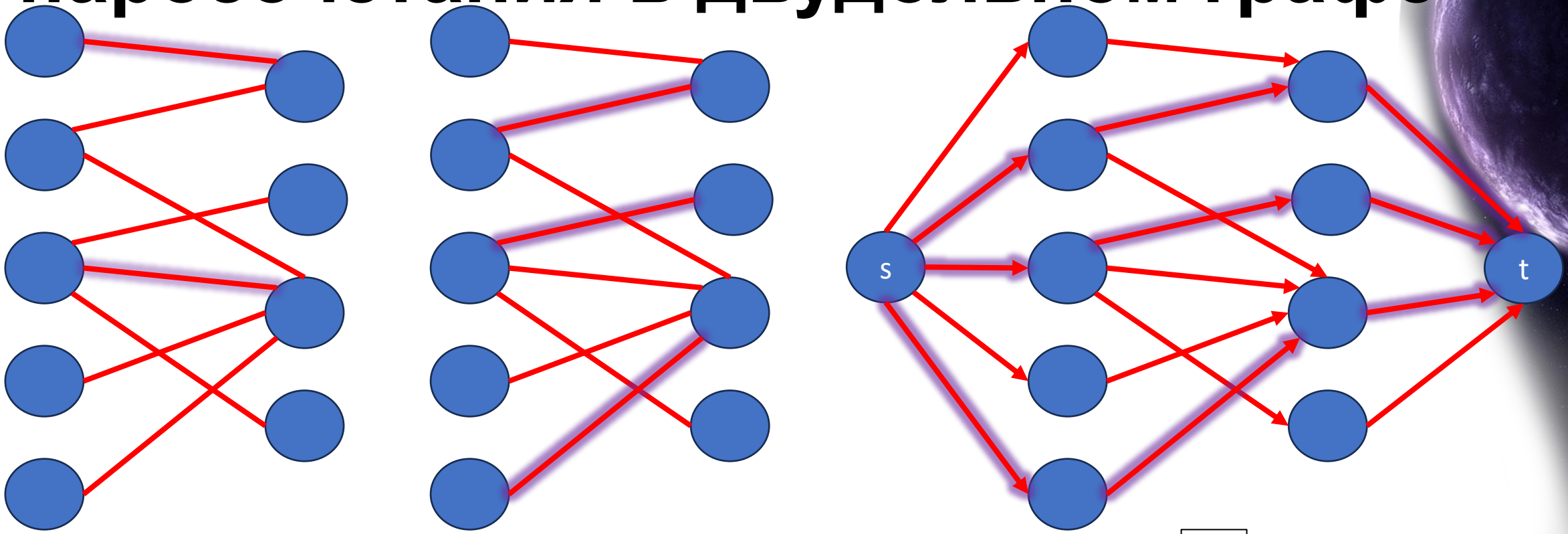
Задача поиска максимального паросочетания в двудольном графе имеет множество практических приложений. В качестве примера можно рассмотреть паросочетание множества машин L и множества задач R, которые должны выполняться одновременно. Наличие в E ребра (u,v) означает, что машина u ∈ L может выполнять задачу v ∈ R. Максимальное паросочетание обеспечивает максимальную загрузку машин.

Задача поиска максимального паросочетания в двудольном графе

Пусть дан неориентированный граф G = (V, E). Паросочетанием (matching) называется подмножество ребер M ⊆ E, такое, что для всех вершин v ∈ V в M содержится не более одного ребра, инцидентного v. Мы говорим, что вершина v ∈ V является связанной (matched) паросочетанием M, если в M есть ребро, инцидентное v; в противном случае вершина v называется открытой (unmatched). Максимальным паросочетанием называется паросочетание максимальной мощности, т.е. такое паросочетание M, что для любого паросочетания M' справедливо соотношение |M| ≥ |M'|. Мы ограничимся рассмотрением задачи поиска максимальных паросочетаний в двудольных графах, т.е. в графах, множество вершин которых можно разбить на два подмножества V = L ∪ R, где L и R не пересекаются и все ребра в E проходят между L и R. Далее мы предполагаем, что каждая вершина в V имеет по крайней мере одно инцидентное ребро.

Задача поиска максимального паросочетания в двудольном графе имеет множество практических приложений. В качестве примера можно рассмотреть паросочетание множества машин L и множества задач R, которые должны выполняться одновременно. Наличие в E ребра (u,v) означает, что машина u ∈ L может выполнять задачу v ∈ R. Максимальное паросочетание обеспечивает максимальную загрузку машин.

Задача поиска максимального паросочетания в двудольном графе



(а) (б) (в)

Двудольный граф G = (V, E) с разбиением вершин V = L ∪ R. (а) Паросочетание с мощностью 2. (б) Максимальное паросочетание с мощностью 3. (в) Соответствующая транспортная сеть G' с показанным максимальным потоком. Каждое ребро имеет единичную пропускную способность. Через заштрихованные ребра идет поток величиной 1, во всех остальных ребрах потока нет.

Поиск максимального паросочетания в двудольном графе

С помощью метода Форда-Фалкерсона можно найти максимальное паросочетание в неориентированном двудольном графе G = (V, E) за время, полиномиально зависящее от |V| и |E|. Фокус заключается в том, чтобы построить транспортную сеть, потоки в которой соответствуют паросочетаниям. Определим для заданного двудольного графа G соответствующую транспортную сеть G' = (V', E') следующим образом. Возьмем в качестве истока s и стока t новые вершины, не входящие в V, и положим V' = V ∪ {s,t}. Если разбиение вершин в графе G представляет собой V = L ∪ R, ориентированными ребрами G' являются ребра E, направленные из L в R, а также |V| новых ориентированных ребер

$$E' = \{(s,u)\colon u \in L\} \cup \{(u,v)\colon (u,v) \in E\} \cup \{(v,t) : v \in R\}.$$

Чтобы завершить построение, присвоим каждому ребру E' единичную пропускную способность. Поскольку каждая вершина из множества вершин V имеет по крайней мере одно инцидентное ребро, |E| > |V|/2. Таким образом, |E| ≤ |E'| = |E| + |V| ≤ 3|E|, так что |E'| = Θ(E).

Следующая лемма показывает, что паросочетание в G непосредственно соответствует потоку в соответствующей транспортной сети G'. Поток f в транспортной сети G = (V, E) называется целочисленным (integer-valued), если значения f(u, v) целые для всех (u, v) ∈ V × V.

Поиск максимального паросочетания в двудольном графе

Лемма 1.9

Пусть G = (V, E) является двудольным графом с разбиением вершин V = L ∪ R и пусть G' = (V', E') представляет собой соответствующую ему транспортную сеть. Если M является паросочетанием в G, то существует целочисленный поток f в G', величина которого - |f| = |M|. Справедливо и обратное утверждение: если f представляет собой целочисленный поток в G', то в G существует паросочетание M с мощностью |M| = |f|.

Доказательство. Покажем сначала, что паросочетанию M в графе G соответствует некоторый целочисленный поток f в сети G'. Определим f следующим образом. Если (u,v) ∈ M, то f(s, u) = f(u,v) = f(v,t) = 1. Для всех остальных ребер (u, v) ∈ E' определим f(u, v) = 0. Нетрудно убедиться, что f удовлетворяет ограничению пропускной способности и сохранению потока.

Интуитивно понятно, что каждое ребро (u, v) ∈ M соответствует единице потока в G', проходящего по пути s → и → v → t. Кроме того, пути, порожденные ребрами из M, представляют собой непересекающиеся множества вершин, не считая s и t.

Чистый поток через разрез (L ∪ {s}, R ∪ {t}) равен |M|; следовательно, согласно лемме 1.4 величина потока равна |f| = |M|.

Чтобы доказать обратное, предположим, что f - некоторый целочисленный поток в G', и пусть

$$M = \{(u,v) : u \in L, v \in R, \text{и } f(u,v) > 0\}.$$

Поиск максимального паросочетания в двудольном графе

Каждая вершина u ∈ L имеет только одно входящее ребро, а именно - (s, u), и его пропускная способность равна 1. Следовательно, в каждую вершину u ∈ L входит не более одной единицы положительного потока, и если она действительно входит, то из нее должна также выходить одна единица положительного потока согласно свойству сохранения потока. Более того, поскольку f – целочисленный поток, для каждой вершины u ∈ L одна единица потока может входить не более чем по одному ребру и выходить не более чем по одному ребру. Таким образом, одна единица положительного потока входит в и тогда и только тогда, когда существует в точности одна вершина v ∈ R, такая, что f(u, v) = 1, и из каждой вершины u ∈ L выходит не более одного ребра, несущего положительный поток. Симметричные рассуждения применимы для каждой вершины v ∈ R. Следовательно, M является паросочетанием.

Чтобы показать, что |M| = |f|, заметим, что f(s, u) = 1 для каждой связанной вершины u ∈ L, и для каждого ребра (u, v) ∈ E - M мы имеем f(u, v) = 0. Следовательно, f(L ∪{s}, R ∪{t}), чистый поток через разрез (L ∪ {s}, R ∪ {t}), равен |M|. Применив лемму 1.4, получаем, что |f| = f(L ∪ {s}, R ∪ {t}) = |M|.

На основании леммы можно сделать вывод о том, что максимальное паросочетание в двудольном графе G соответствует максимальному потоку в соответствующей ему транспортной сети G', следовательно, можно находить максимальное паросочетание в G с помощью алгоритма поиска максимального потока в G’.

Поиск максимального паросочетания в двудольном графе

Теорема 1.10 (Теорема о целочисленности)

Если функция пропускной способности c принимает только целые значения, то максимальный поток f, полученный с помощью метода Форда-Фалкерсона, обладает тем свойством, что значение потока |f| является целочисленным. Более того, для всех вершин u и v величина f(u, v) является целой.

Доказательство. Доказательство проводится индукцией по числу итераций.

Следствие 1.11

Мощность максимального паросочетания M в двудольном графе G равна величине максимального потока f в соответствующей транспортной сети G'.

Доказательство. Воспользуемся терминологией леммы 1.9. Предположим, что M представляет собой максимальное паросочетание в G, но соответствующий ему поток f в G' не максимален. Тогда в G' существует максимальный поток f', такой, что |f'| > |f|. Поскольку пропускные способности в G' являются целочисленными, теорема 26.10 позволяет считать поток f' целочисленным. Таким образом, f' соответствует некоторому паросочетанию M' в G мощностью |M'| = |f'| > |f| = |M|, что противоречит нашему предположению о том, что M является максимальным паросочетанием. Аналогично можно показать, что если f - максимальный поток в G', то соответствующее ему паросочетание является максимальным паросочетанием в G.

максимальное паросочетание в двудольном графе можно найти за время O(VE).

Алгоритмы проталкивания предпотока

В настоящее время многие асимптотически наиболее быстрые алгоритмы поиска максимального потока принадлежат данному классу, и на этом методе основаны реальные реализации алгоритмов поиска максимального потока. С помощью методов проталкивания предпотока можно решать и другие связанные с потоками задачи, например задачу поиска потока с минимальной стоимостью. Мы рассмотрим разработанный Голдбергом (Goldberg) "обобщенный" алгоритм поиска максимального потока, для которого существует простая реализация с временем выполнения $O(V^2E)$, что лучше времени работы алгоритма Эдмондса-Карпа $O(VE^2)$.

Предпоток (preflow) представляет собой функцию $f : V \times V \to R$, удовлетворяющую ограничениям пропускной способности и следующему ослабленному условию сохранения потока:

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

для всех вершин $u \in V - \{s\}$. Будем называть величину

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

избыточным потоком (excess flow), входящим в вершину и. Избыток в вершине представляет собой величину, на которую входящий поток превышает исходящий. Мы говорим, что вершина $u \in V - \{s,t\}$ ***переполненная*** (overflowing), если $e(u) > 0$.

Интуитивные соображения

Аналогия: Транспортная сеть G=(V,E) — система труб с пропускными способностями, вершины — перекрестки труб.

- ***Сравнение с Фордом-Фалкерсоном:*** Форд-Фалкерсон добавляет поток по увеличивающим путям от истока к стоку без ветвлений. Повторяет до исчерпания путей.
- ***Идея проталкивания предпотока:*** Вершины имеют резервуары для избыточного потока. Каждая вершина на платформе с изменяемой высотой. Поток проталкивается только вниз (от высокой вершины к низкой).
- ***Высоты вершин:*** Исток: фиксированная высота |V|. Сток: фиксированная высота 0. Остальные вершины: изначально 0, увеличиваются со временем.
- ***Процесс:*** Из истока отправляется поток, равный пропускной способности разреза $(s, V - \{s\})$. Поток накапливается в резервуарах транзитных вершин, затем проталкивается вниз.
- ***Операция подъема (relabeling):*** Если все выходящие трубы ведут к вершинам на том же уровне или выше, высота вершины увеличивается. Новая высота = (минимальная высота смежной вершины с незаполненной трубой) + 1.
- **Финал:**
  1. Поток достигает стока, ограниченный пропускной способностью разреза.
  2. Избыточный поток из резервуаров возвращается к истоку (высота вершин превышает |V|).
  3. После опустошения резервуаров предпоток становится максимальным потоком.

Операция проталкивания

Основная операция Push(u, v) может применяться тогда, когда u является переполненной вершиной, $c_f(u,v) > 0$ и $h(u) = h(v) + 1$. Представленный ниже псевдокод обновляет предпоток f и избыточный поток для u и v. Предполагается, что остаточные пропускные способности $c_f(u, v)$ при заданных f и c можно вычислить за фиксированное время. Излишний поток в вершине и поддерживается в виде атрибута u.e, а высота вершины u - в виде атрибута u.h. Выражение $\Delta_f(u,v)$ представляет собой временную переменную, в которой хранится количество потока, которое можно протолкнуть из u в v.

Push(u, v)

1 // Применяется при: вершина u переполнена,
  // $c_f(u, v) > 0$ и u.h = v. h + 1.
2 // Действие: проталкивает $\Delta_f(u,v) = \min(u.e, c_f(u,v))$
  // единиц потока из и в v.
3 $\Delta_f(u, v) = \min(u.e, c_f(u,v))$
4 **if** $(u, v) \in E$
5 $\quad (u,v).f = (u,v).f + \Delta_f(u,v)$
6 **else**
$\quad (v, u).f = (v,u).f - \Delta_f(u,v)$
7 $\quad u.e = u.e - \Delta_f(u,v)$
8 $\quad v.e = v.e + \Delta_f(u,v)$

Операция проталкивания

Обратите внимание, что в коде процедуры PUSH ничто не зависит от высот вершин u и v; тем не менее мы запретили вызов процедуры, если не выполнено условие u.h = v.h + 1. Таким образом, избыточный поток проталкивается вниз только при разности высот, равной 1. Согласно лемме 1.12 между двумя вершинами, высоты которых отличаются более чем на 1, не существует остаточных ребер, а значит, поскольку атрибут h является функцией высоты, мы ничего не добьемся, разрешив проталкивать вниз поток при разности высот, превышающей 1.

Процедура PUSH(u, v) называется проталкиванием (push) из и в v. Если операция проталкивания применяется к некоторому ребру (u, v), выходящему из вершины и, будем говорить, что операция проталкивания применяется к и. Если в результате ребро (u, v) становится ***насыщенным*** (saturated) (после проталкивания $c_f(u, v) = 0$), то это ***насыщающее проталкивание*** (saturating push), в противном случае это ***ненасыщающее проталкивание*** (nonsaturating push). Если ребро становится насыщенным, оно исчезает из остаточной сети. Один из результатов ненасыщающего проталкивания характеризует следующая лемма.

Лемма 1.13

После ненасыщающего проталкивания из u в v вершина u более не является переполненной.

Доказательство. Поскольку проталкивание ненасыщающее, фактическое количество посланного потока $\Delta_f(u, v)$ должно быть равно величине u.e непосредственно перед проталкиванием. Поскольку избыток u.e уменьшается на эту величину, после проталкивания он становится равным 0.

Операция подъема

Основная операция Relabel(u) применяется, если вершина u переполнена и если u.h ≤ v.h для всех ребер (u,v) ∈ 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. (Напомним, что по определению ни исток s, ни сток t не могут быть переполнены; следовательно, ни s, ни t нельзя подвергать подъему.)

Relabel(u)

1 Применяется при: вершина и переполнена, и для всех v ∈ V, таких, что (u, v) ∈ E_f, имеем u.h ≤ v.h.

2 Действие: увеличивает высоту u.

3 u.h = 1 + min {v.h : (u,v) ∈ E_f}

Когда вызывается операция Relabel (u), мы говорим, что вершина и подвергается подъему (relabeled). Заметим, что когда выполняется подъем u, E_f должно содержать хотя бы одно ребро, выходящее из u, чтобы минимизация в коде операции осуществлялась по непустому множеству. Это свойство вытекает из предположения о том, что вершина и переполнена, что, в свою очередь, говорит нам, что

$$u.e = \sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v) > 0$$

Поскольку все потоки неотрицательны, должна быть по крайней мере одна вершина v, такая, что (v, u).f > 0. Но тогда c_f(u,v) > 0, откуда вытекает, что (u, v) ∈ E_f. Таким образом, операция Relabel(u) назначает и наибольшую высоту, допускаемую наложенными на функцию высоты ограничения

Обобщенный алгоритм

Обобщенный алгоритм проталкивания предпотока использует следующую подпрограмму для создания начального предпотока в транспортной сети.

Initialize-Preflow(G, s)

1 **for** каждой вершины v ∈ G. V
2 v.h = 0
3 v.e = 0
4 **for** каждого ребра (u, v) ∈ G. E
5 (u, v).f = 0
6 s.h = |G. V|
7 **for** каждой вершины v ∈ s.Adj
8 (s, v).f = c(s, v)
9 v.e = c(s,v)
10 s.e = s.e - c(s, v)

Initialize-Preflow создает начальный предпоток f, определяемый как

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

Иначе говоря, каждое ребро, выходящее из истока s, заполняется до его пропускной способности, а все остальные ребра не несут потока. Для каждой вершины v, смежной с истоком, изначально мы имеем v.e = c(s,v) и инициализируем s.e суммой этих значений с обратным знаком.

Обобщенный алгоритм

Обобщенный алгоритм начинает работу с начальной функцией высоты h, задаваемой следующим образом:

$$u.h = \begin{cases} |V|, \text{если } u = s, \\ 0 \text{ в противном случае}. \end{cases}$$

Уравнение определяет функцию высоты, поскольку единственными ребрами (u,v), для которых u.h > v.h + 1, являются те, для которых u = s, и эти ребра заполнены, а это означает, что их нет в остаточной сети.

Инициализация, за которой следует ряд операций проталкивания и подъема, выполняемых без определенного порядка, образует алгоритм Generic-Push-Relabel.

Generic-Push-Relabel(G)

1 **Initialize-Preflow(G, s)**

2 **while** существует применимая операция проталкивания или подъема

3 выбрать и выполнить операцию проталкивания или подъема

Лемма 1.14 (Для переполненной вершины можно выполнить либо проталкивание, либо подъем)

Пусть G = (V, E) представляет собой транспортную сеть с истоком s и стоком t, а f является предпотоком, и пусть h является произвольной функцией веса для f.

Если u представляет собой произвольную переполненную вершину, то к ней применимо либо проталкивание, либо подъем.

Корректность метода проталкивания предпотока

Лемма 1.15 (Высота вершины никогда не уменьшается)
При выполнении процедуры Generic-Push-Relabel над транспортной сетью G = (V, E) для любой вершины и ∈ V ее высота и. h никогда не уменьшается. Более того, всякий раз, когда к вершине и применяется операция подъема, ее высота u.h увеличивается как минимум на 1.

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

Корректность метода проталкивания предпотока

Лемма 1.16

Пусть G = (V, E) представляет собой транспортную сеть с истоком s и стоком t. Во время выполнения процедуры Generic-Push-Relabel над сетью G атрибут h сохраняет свойства функции высоты.

Доказательство. Доказательство проводится индукцией по числу выполненных основных операций. Изначально, как уже отмечалось, h является функцией высоты.

Мы утверждаем, что если h является функцией высоты, то операция Relabel(u) оставляет h функцией высоты. Если рассмотреть остаточное ребро $(u, v) \in E_f$, которое выходит из u, то операция Relabel(u) гарантирует, что после нее будет выполняться соотношение $u.h \leq v.h + 1$. Теперь рассмотрим остаточное ребро (w, u), входящее в u. Согласно лемме 1.15 из $w.h \leq u.h + 1$ перед выполнением операции Relabel(u) вытекает $w.h < u.h + 1$ после нее. Таким образом, операция Relabel(u) оставляет h функцией высоты.

Теперь рассмотрим операцию PUSH(u,v). Эта операция может добавить ребро (v,u) к E_f, и может удалить ребро (u,v) из E_f. В первом случае имеем $v.h = u.h - 1 < u.h + 1$, так что h остается функцией высоты. Во втором случае удаление ребра (u,v) из остаточной сети приводит к удалению соответствующего ограничения, так что h по-прежнему остается функцией высоты.

Корректность метода проталкивания предпотока

Лемма 1.17

Пусть G = (V, E) представляет собой транспортную сеть с истоком s и стоком t, f является предпотоком в G, а h - функцией высоты, определенной на множестве V. Тогда в остаточной сети G_f не существует пути из истока s в сток t.

Доказательство. Предположим, что в G_f имеется некоторый путь $p = \langle v_0, v_1, \ldots, v_k \rangle$ 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 $i = 0, 1, \ldots, k - 1$ ребра $(v_i, v_{i+1}) \in E_f$. Поскольку h является функцией высоты, для $i = 0,1,\ldots,k - 1$ справедливы соотношения $h(v_i) \leq h(v_{i+1}) + 1$. Объединив эти неравенства вдоль пути p, получаем, что $h(s) \leq h(t) + k$. Но поскольку $h(t) = 0$, получаем $h(s) < k < |V|$, что противоречит требованию $h(s) = |V|$ к функции высоты.

Теперь мы готовы показать, что после завершения обобщенного алгоритма проталкивания предпотока вычисленный алгоритмом предпоток является максимальным потоком.

Корректность метода проталкивания предпотока

Теорема 1.18 (Корректность обобщенного алгоритма проталкивания предпотока)
Если алгоритм Generic-Push-Relabel, выполняемый над транспортной сетью G = (V, E) с истоком s и стоком t, завершается, то вычисленный им предпоток f является максимальным потоком в G.

Доказательство. Воспользуемся следующим инвариантом цикла. Всякий раз, когда в строке 2 процедуры Generic-Push-Relabel выполняется проверка условия цикла while, f является предпотоком.

Инициализация. Initialize-Preflow делает f предпотоком.

Сохранение. Внутри цикла while в строках 2 и 3 выполняются только операции проталкивания и подъема. Операции подъема влияют только на атрибуты высоты, но не на величины потока, следовательно, от них не зависит, будет ли f предпотоком. Анализируя работу процедуры Push, мы доказали, что если f является предпотоком перед выполнением операции проталкивания, он остается предпотоком и после ее выполнения.

Завершение. По завершении процедуры каждая вершина из множества V - {s,t} должна иметь нулевой избыток, поскольку из леммы 1.14 и инварианта, что f всегда остается предпотоком, вытекает, что переполненных вершин нет. Следовательно, f является потоком. Лемма 1.16 показывает, что при завершении h является функцией высоты; таким образом, согласно лемме 1.17 в остаточной сети G_f не существует пути из 8 в t. Согласно теореме о максимальном потоке и минимальном разрезе (теорема 1.6) f является максимальным потоком.

Анализ метода проталкивания предпотока

Лемма 1.19

Пусть G = (V, E) представляет собой транспортную сеть с истоком s и стоком t, а f является предпотоком в G. Тогда в остаточной сети G_f для любой переполненной вершины x существует простой путь из x в s.

Доказательство. Для переполненной вершины x введем U = {v : существует простой путь из x в v G_f}. Теперь предположим, что s ∉ U, и покажем, что это приводит к противоречию. Обозначим $\overline{U}$ = V - U.

Возьмем определение избытка из уравнения, выполним суммирование по всем вершинам в U и заметим, что V = U ∪ $\overline{U}$. Это даст

$$\sum_{u\in U} e(u) = \sum_{u\in U}\left(\sum_{v\in V} f(v,u) - \sum_{v\in V} f(u,v)\right) = \sum_{u\in U}\left(\left(\sum_{v\in U} f(v,u) - \sum_{v\in \overline{U}} f(v,u)\right) - \left(\sum_{v\in U} f(u,v) - \sum_{v\in \overline{U}} f(u,v)\right)\right)$$

$$= \sum_{u\in U}\sum_{v\in U} f(v,u) + \sum_{u\in U}\sum_{v\in \overline{U}} f(v,u) - \sum_{u\in U}\sum_{v\in U} f(u,v) - \sum_{u\in U}\sum_{v\in \overline{U}} f(u,v) = \sum_{u\in U}\sum_{v\in \overline{U}} f(v,u) - \sum_{u\in U}\sum_{v\in \overline{U}} f(u,v)$$

Мы знаем, что величина $\sum_{u\in U} e(u)$ должна быть положительной, поскольку e(x) > 0, x ∈ U, что все вершины, отличные от s, имеют неотрицательный избыток и что согласно нашему предположению s ∉ U. Таким образом, имеем

$$\sum\sum f(v,u) - \sum\sum f(u,v) > 0$$

Анализ метода проталкивания предпотока

Все потоки ребер неотрицательны, так что, чтобы выполнялось уравнение, необходимо иметь $\sum_{u \in U} \sum_{v \in \bar{U}} f(v,u) > 0$. Следовательно, должна существовать как минимум одна пара вершин u' ∈ U и v' ∈ U, обладающих тем свойством, что f(v', u') > 0. Но если f(v', u') > 0, должно существовать остаточное ребро (u', v'), а это означает, что имеется простой путь из в v' (путь x ⇝ u' → v'), что противоречит определению U.

Лемма 1.20

Пусть G = (V, E) представляет собой транспортную сеть с истоком s и стоком t. В любой момент в процессе выполнения процедуры Generic-Push-Relabel над сетью G для всех вершин и ∈ V выполняется соотношение u.h ≤ 2|V| - 1.

Доказательство. Высоты истока s и стока t никогда не изменяются, поскольку эти вершины по определению не переполняются. Таким образом, всегда s.h = |V| и t.h = 0, причем оба значения не превышают 2|V| - 1.

Рассмотрим теперь произвольную вершину u ∈ V - {s,t}. Изначально u.h = 0 ≤ 2|V| - 1. Покажем, что после каждого подъема неравенство u.h ≤ 2|V| - 1 остается справедливым. При подъеме вершины и она является переполненной, и согласно лемме 1.19 имеется простой путь p из u в s в G_f. Пусть p = (v_0, v_1, …, v_k), где v_0 = u, v_k = s и k ≤ |V| - 1, поскольку p - простой путь.

Для i = 0,1,…, k - 1 имеем (v_i, v_{i+1}) ∈ E_f, а следовательно, v_i.h ≤ v_{i+1}.h + 1 согласно лемме 1.16. Расписав неравенства для всех составляющих пути p, получаем u.h = v_0.h ≤ v_k.h + k ≤ s.h + (|V| - 1) = 2 |V| - 1.

Анализ метода проталкивания предпотока

Следствие 1.21 (Верхний предел числа подъемов)

Пусть G = (V, E) представляет собой транспортную сеть с истоком s и стоком t. Тогда в процессе выполнения процедуры Generic-Push-Relabel над G число подъемов не превышает 2|V| - 1 на вершину, а их общее количество – не более (2 |V| - 1)(|V| - 2) < $2|V|^2$.

Доказательство. В множестве V - {s,t} могут быть подняты только |V| - 2 вершин. Пусть u ∈ V - {s,t}. Операция Relabel(u) увеличивает высоту u.h. Значение u.h изначально равно 0 и согласно лемме 1.20 возрастает не более чем до 2|V| - 1. Таким образом, каждая вершина u ∈ V - {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 1.22 (Граница количества насыщающих проталкиваний)

В процессе выполнения алгоритма Generic-Push-Relabel над любой транспортной сетью G = (V, E) число насыщающих проталкиваний меньше, чем 2|V||E|.

Доказательство. Для любой пары вершин u, v ∈ V рассмотрим насыщающие проталкивания от u к v и от v к u (в обе стороны) и назовем их насыщающими проталкиваниями между u и v. Если есть хотя бы одно такое проталкивание, то хотя бы одно из ребер (u, v) и (v, u) является ребром из E. Теперь предположим, что произошло насыщающее проталкивание из и в v. В этот момент v. h = u.h-1. Чтобы позднее могло произойти еще одно проталкивание из и в v, алгоритм сначала должен протолкнуть поток из v в u, что невозможно до тех пор, пока не будет выполнено условие v.h = u.h + 1. Поскольку u.h никогда не уменьшается, для того ч… выполнялось условие v.h = u.h + 1, значение v.h должно увеличиться по меньшей мере на 2.

Анализ метода проталкивания предпотока

Аналогично u.h должно увеличиться между последовательными насыщающими проталкиваниями из v в и как минимум на 2. Высота изначально принимает значение 0 и согласно лемме 1.20 никогда не превышает 2|V|-1, откуда следует, что количество раз, когда высота вершины может увеличиться на 2, меньше |V|. Поскольку между двумя насыщающими проталкиваниями между u и v хотя бы одна из высот u.h и v. h должна увеличиться на 2, всего имеется меньше 2|V| насыщающих проталкиваний между u и v.

Умножив это число на число ребер, получим, что общее число насыщающих проталкиваний меньше, чем 2 |V||E|.

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

Лемма 1.23 (Граница количества ненасыщающих проталкиваний)

В процессе выполнения алгоритма Generic-Push-RelabeL над любой транспортной сетью G = (V, E) число ненасыщающих проталкиваний меньше $4|V|^2$ (|V| + |E|).

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

Анализ метода проталкивания предпотока

Рассмотрим два пути увеличения Ф. Во-первых, подъем вершины и увеличивает Ф менее чем на 2|V|, поскольку множество, для которого вычисляется сумма, остается прежним, а подъем не может увеличить высоту вершины и больше, чем ее максимально возможная высота, которая составляет не более 2 |V|- 1 согласно лемме 1.20. Во-вторых, насыщающее проталкивание из вершины и в вершину v увеличивает Ф 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. Почему? Перед ненасыщающим проталкиванием вершина u была переполненной, а v могла быть переполненной или непереполненной. Согласно лемме 1.13 после этого проталкивания и больше не является переполненной. Кроме того, если только v не является истоком, она может быть как переполненной, так и не быть таковой после проталкивания. Следовательно, потенциальная функция Ф уменьшилась ровно на u.h, а увеличилась на 0 или на v.h. Поскольку u. h - v. h = 1, в итоге потенциальная функция уменьшается как минимум на 1.

Итак, в ходе выполнения алгоритма увеличение Ф происходит благодаря подъемам и насыщающим проталкиваниям; согласно следствию 26.21 и лемме 1.22 это увеличение ограничено и составляет менее $(2|V|)(2|V|^2) + (2|V|)(2|V||E|) = 4|V|^2 (|V| + |E|)$. Поскольку Ф > 0, суммарная величина уменьшения u, следовательно, общее число ненасыщающих проталкиваний меньше, чем $4|V|^2 (|V| + |E|)$.

Определив границу числа подъемов, насыщающего проталкивания и ненасыщающего проталкивания, мы заложили основу дальнейшего анализа процедуры Generic-Push-Relabel, а следовательно, любых других алгоритмов, основанных на методе проталкивания предпотока.

Анализ метода проталкивания предпотока

Теорема 1.24

При выполнении процедуры Generic-Push-Relabel над любой транспортной сетью G = (V, E) число основных операций составляет $O(V^2E)$.

Доказательство. Непосредственно вытекает из следствия 1.21 и лемм 1.22 и 1.23.

Таким образом, алгоритм завершается после O(V2E) операций. Итак, осталось предложить эффективные методы реализации каждой операции и выбора подходящей выполняемой операции.

Следствие 1.25

Существует реализация обобщенного алгоритма проталкивания предпотока, которая для любой транспортной сети G = (V, E) выполняется за время $O(V^2E)$.

Алгоритм "поднять-в-начало"

Метод проталкивания предпотока позволяет применять основные операции в произвольном порядке. Однако путем тщательного выбора порядка их выполнения и при эффективном управлении структурой сетевых данных можно решить задачу поиска максимального потока быстрее, чем за предельное время $O(V^2E)$. Далее мы рассмотрим алгоритм "поднять-в-начало" (relabel-to-front), основанный на методе проталкивания предпотока, время выполнения которого составляет $O(V^3)$, что асимптотически не хуже, чем $O(V^2E)$, а для плотных сетей даже лучше.

Алгоритм "поднять-в-начало" поддерживает список вершин сети. Алгоритм многократно сканирует список с самого начала, выбирает некоторую переполненную вершину u, а затем "разгружает" ее, т.е. выполняет операции проталкивания и подъема до тех пор, пока избыток в u не перестанет быть положительным. Если выполнялось поднятие вершины, то она переносится в начало списка (отсюда и название алгоритма: "поднять-в-начало"), и алгоритм начинает сканирование списка заново.

Для исследования корректности и временных характеристик данного алгоритма используется понятие "допустимых" ребер: это ребра остаточной сети, через которые можно протолкнуть поток. Доказав некоторые свойства сети, состоящей из допустимых ребер, мы рассмотрим операцию разгрузки, а затем представим и проанализируем сам алгоритм "поднять-в-начало"

Допустимые ребра и сети

Если G = (V, E) представляет собой некоторую транспортную сеть с истоком s и стоком t, f - предпоток в G, а h - функция высоты, то мы говорим, что ребро (u, v) является ***допустимым ребром*** (admissible edge), если $c_f(u, v) > 0$ и $h(u) = h(v) + 1$. В противном случае ребро (u,v) называется ***недопустимым*** (inadmissible). ***Допустимой сетью*** (admissible network) является сеть $G_{f,h} = (V, E_{f,h})$, где $E_{f,h}$ - множество допустимых ребер.

Допустимая сеть состоит из тех ребер, через которые можно протолкнуть поток. Следующая лемма показывает, что такая сеть является ориентированным ациклическим графом.

Лемма 1.26 (Допустимая сеть является ациклической)

Если G = (V, E) является транспортной сетью, f представляет собой предпоток в G, а h - функция высоты на G, то допустимая сеть $G_{f,h} = (V, E_{f,h})$ ациклична.

Доказательство. Проведем доказательство методом от противного. Предположим, что $G_{f,h}$ содержит некоторый цикл $p = (v_0, v_1, \ldots, v_k)$, где $v_0 = v_k$ и $k > 0$. Поскольку каждое ребро в p является допустимым, справедливо равенство $h(v_{i-1}) = h(v_i) + 1$ для $i = 1,2, \ldots,k$. Просуммировав эти равенства вдоль циклического пути, получаем

$$\sum_{i=1}^{k} h(v_{i-1}) = \sum_{i=1}^{k} (h(v_i) + 1) = \sum_{i=1}^{k} h(v_i) + k .$$

Поскольку каждая вершина цикла p встречается при суммировании по одному разу, приходим к выводу, что 0 = k, что противоречит первоначальному предположению.

Допустимые ребра и сети

Лемма 1.27

Пусть G = (V, E) представляет собой транспортную сеть, f - предпоток в G, и предположим, что атрибут h является функцией высоты. Если вершина u переполнена, а (u,v) является допустимым ребром, то применима процедура

Push(u,v). Эта операция не создает новых допустимых ребер, но может привести к тому, что ребро (u, v) станет недопустимым.

Доказательство. По определению допустимого ребра из u в v можно протолкнуть поток. Поскольку вершина и переполнена, применяется операция

Push(u,v). В результате проталкивания потока из u в v может быть создано только одно новое остаточное ребро (v, u). Поскольку v.h = u.h - 1, ребро (v, u) не может стать допустимым. Если примененная операция является насыщающим проталкиванием, то после ее выполнения $c_f(u,v) = 0$ и ребро (u,v) становится недопустимым.

Допустимые ребра и сети

Лемма 1.28

Пусть G = (V, E) представляет собой транспортную сеть, f является предпотоком в G, и предположим, что атрибут h является функцией высоты. Если вершина и переполнена и не имеется допустимых ребер, выходящих из и, то применяется операция Relabel(u). После подъема появляется по крайней мере одно допустимое ребро, выходящее из и, но нет допустимых ребер, входящих в u.

Доказательство. Если вершина и переполнена, то согласно лемме 1.14 к ней применяется или операция проталкивания, или операция подъема. Если не существует допустимых ребер, выходящих из и, то протолкнуть поток из и невозможно, следовательно, применяется операция Relabel(u). После данного подъема $u.h = 1 + \min \{v.h : (u, v) \in E_f\}$. Таким образом, если v - вершина, в которой реализуется минимум указанного множества, ребро (u, v) становится допустимым. Следовательно, после подъема имеется по крайней мере одно допустимое ребро, выходящее из u.

Чтобы показать, что после подъема не существует входящих в и допустимых ребер, предположим, что существует некоторая вершина v, такая, что ребро (v, u) допустимо. Тогда после подъема $v.h = u.h + 1$, так что непосредственно перед подъемом $v.h > u.h + 1$. Но согласно лемме 1.12 не существует остаточных ребер между вершинами, высоты которых отличаются более чем на 1. Кроме того, подъем вершины не меняет остаточную сеть. Таким образом, ребро (v, u) не принадлежит остаточной сети, а следовательно, оно не может находиться в допустимой сети.

Списки соседей

Ребра в алгоритме "поднять-в-начало" объединены в так называемые "списки соседей". В заданной транспортной сети G = (V, E) списком соседей (neighbor list) u.N некоторой вершины u ∈ V является односвязный список вершин, смежных с u в G. Таким образом, вершина v оказывается в списке u.N, если (u, v) ∈ E или (v, u) ∈ E. Список соседей u. N содержит только такие вершины v, для которых может существовать остаточное ребро (u,v). На первую вершину в списке u. N указывает указатель u.N.head. Указатель v.next-neighbor указывает на вершину, следующую в списке соседей за v; этот указатель имеет значение NIL, если v является последней вершиной в списке соседей.

Алгоритм "поднять-в-начало" циклически обрабатывает каждый список соседей в произвольном порядке, который фиксируется в процессе выполнения алгоритма. Для каждой вершины и атрибут u. current указывает на текущую вершину списка u.N. Изначально u. current устанавливается равным u.N.head.

Разгрузка переполненной вершины

Переполненная вершина и разгружается (discharged) путем проталкивания всего ее избыточного потока через допустимые ребра в смежные вершины, при этом, если необходимо, выполняется подъем вершины и, чтобы ребра, выходящие из вершины и, стали допустимыми. Псевдокод разгрузки выглядит следующим образом.

Discharge(u)

```
1 while u.e > 0
2         v = u. current
3         if v == NIL
4                 Relabel(u)
5                 u.current = u. N. head
6         else if c_f(u, v) > 0 и u.h == v.h + 1
7                 Push(u, v)
8         else u.current = v.next-neighbor
```

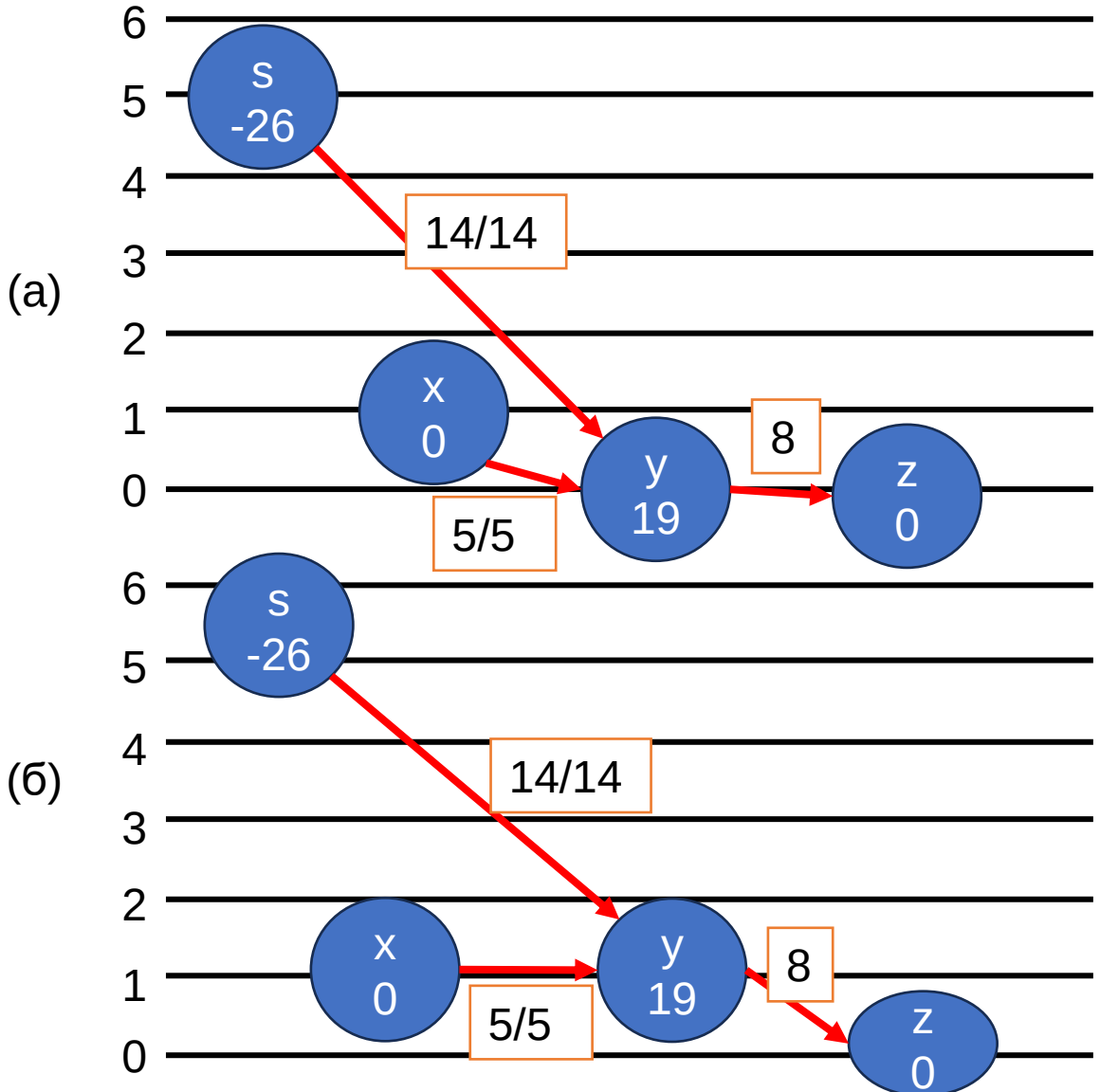
Лемма 1.29

Если процедура Discharge вызывает в строке 7 процедуру Push(u,v), то к (u,v) применима операция проталкивания. Если процедура Discharge вызывает в строке 4 процедуру Relable(u), к вершине и применим подъем.

Разгрузка переполненной вершины

Доказательство. Проверки в строках 1 и 6 гарантируют, что операция проталкивания вызывается только тогда, когда она применима; таким образом, первое утверждение леммы доказано. Чтобы доказать второе утверждение, исходя из проверки в строке 1 и леммы 1.28, необходимо только показать, что все ребра, выходящие из u, являются недопустимыми. Если вызов Discharge(u) начинается с указателем u.current на голову списка соседей, а по завершении он указывает за конец списка, то все выходящие из u ребра недопустимы, и применяется операция подъема. Возможно, однако, что во время вызова Discharge(u) указатель u.current проходит только по части списка перед возвратом из процедуры. После этого мгут призойти вызовы процедуры Discharge с другими вершинами, но указатель u.current будет перемещаться по списку в процессе следующего вызова Discharge(u). Рассмотрим теперь, что произойдет при полном проходе по списку, который начинается с заголовка u. N и заканчивается значением u.current = NIL. Когда u.current достигает конца списка, процедура поднимает u и начинает новый проход. Чтобы в процессе прохода указатель u current переместился за вершину v ∈ u.N, ребро (u,v) должно быть признано недопустимым проверкой в строке 6. Таким образом, к моменту завершения прохода каждое ребро, покидающее и, определено как недопустимое в некоторый момент этого прохода. Ключевым является тот факт, что к концу прохода все ребра, покидающие u, остаются недопустимыми. Почему? Согласно лемме 1.27 операции проталкивания не могут приводить к созданию допустимых ребер, независимо от того, из какой вершины выполняется проталкивание. Таким образом, любое допустимое ребро должно быть создано операцией подъема. Но вершина u не подвергается подъему в процессе прохода, а любая другая вершина v, подвергшаяся подъему в процессе данного прохода (в результате вызова DISCHARGE(v)), не имеет после подъема допустимых входящих ребер согласно лемме 1.28. Итак, в конце прохода все ребра, выходящие из и, остаются недопустимыми, и лемма доказана.

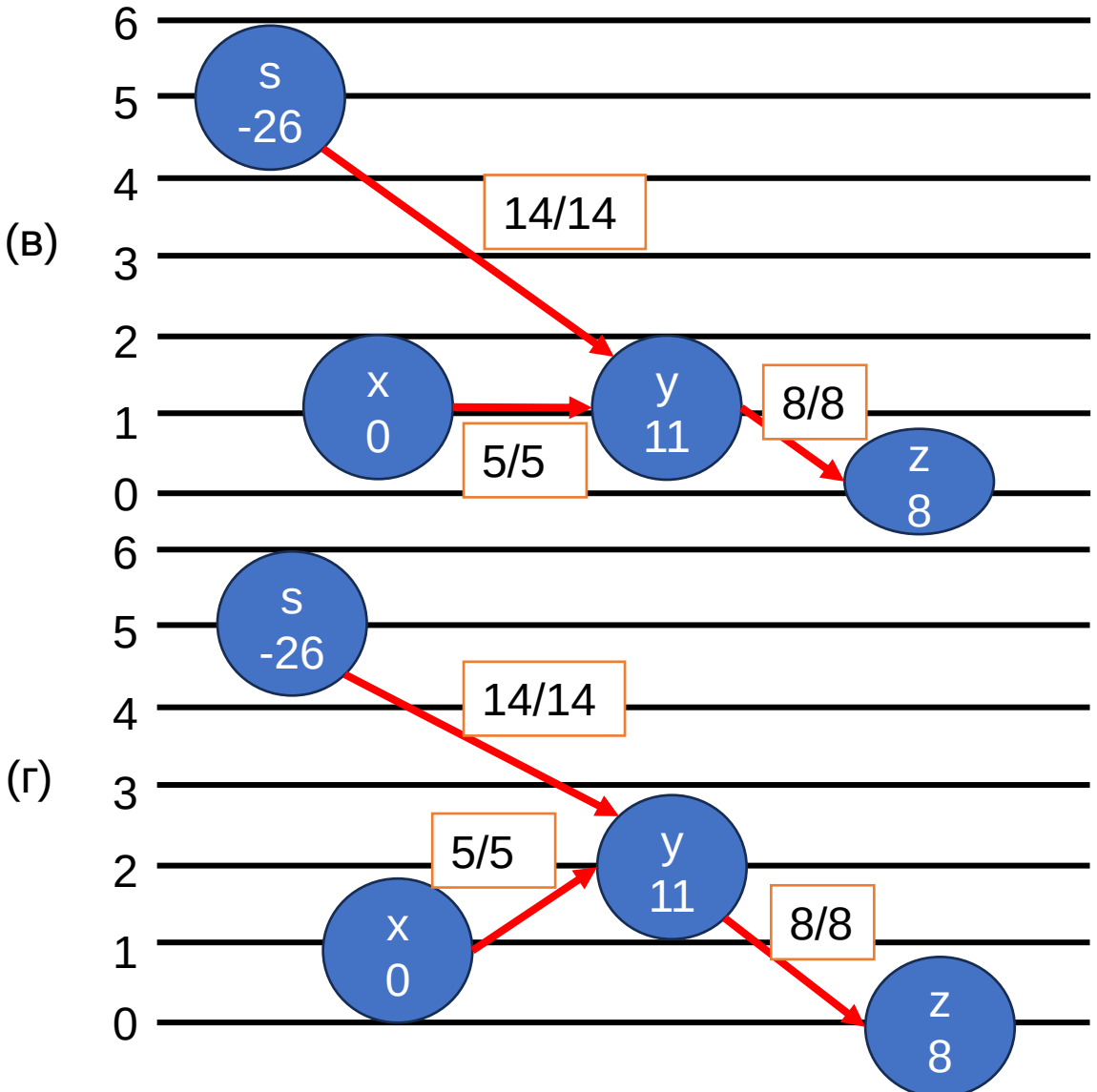
Разгрузка переполненной вершины

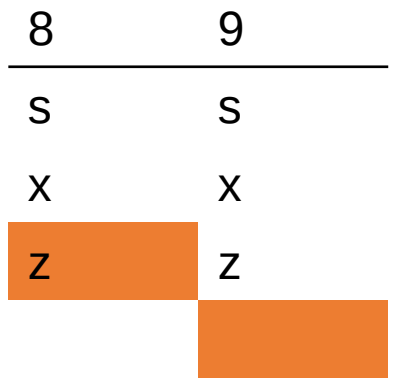


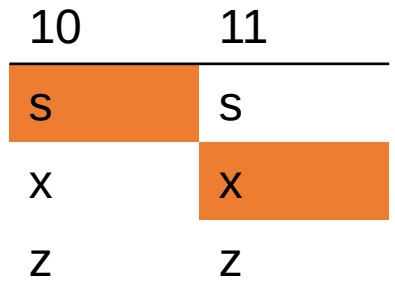
1	2	3	4
s	s	s	s
x	x	x	x
z	z	z	z

5	6	7
s	s	s
x	x	x
z	z	z

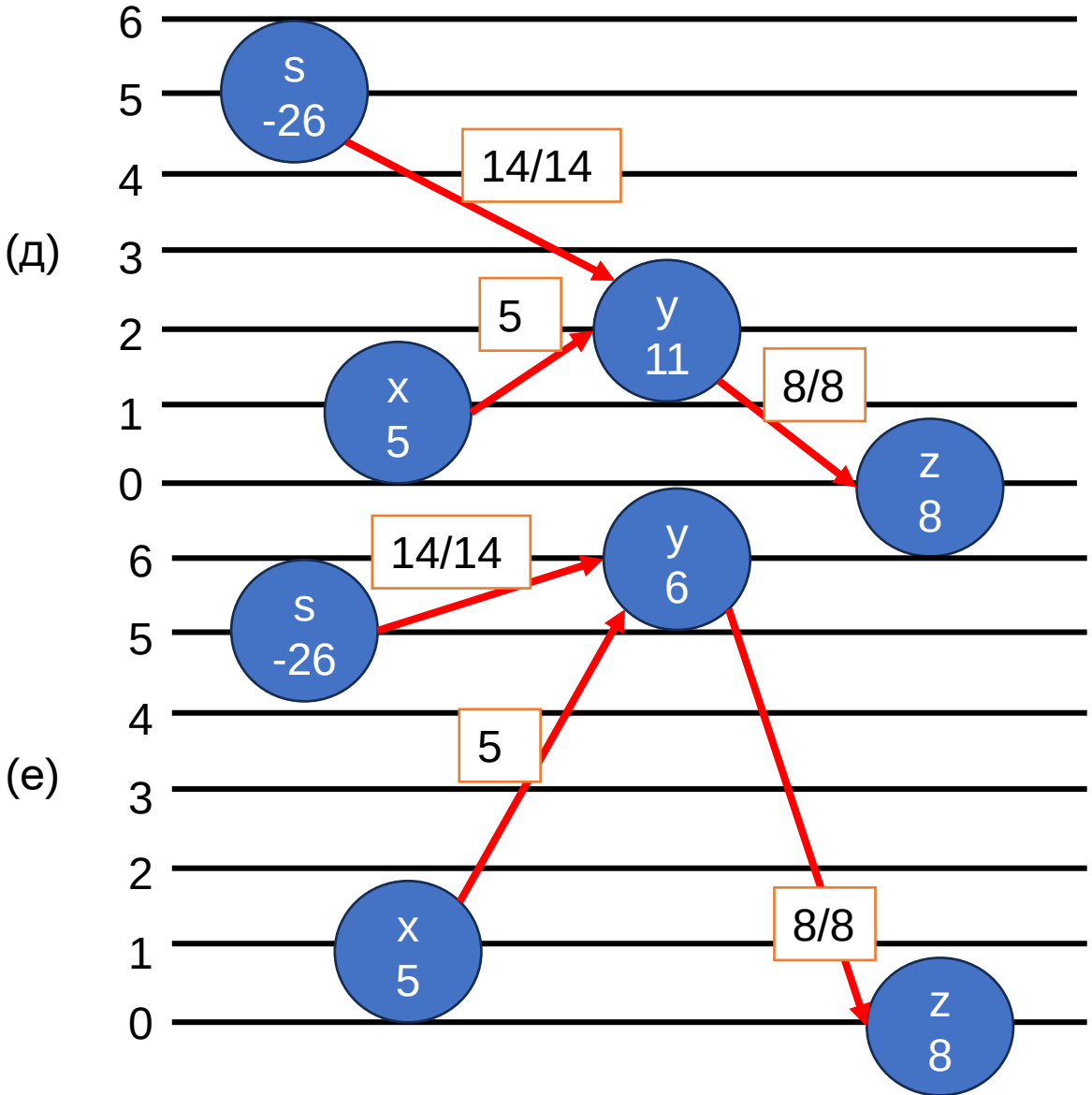
Разгрузка переполненной вершины

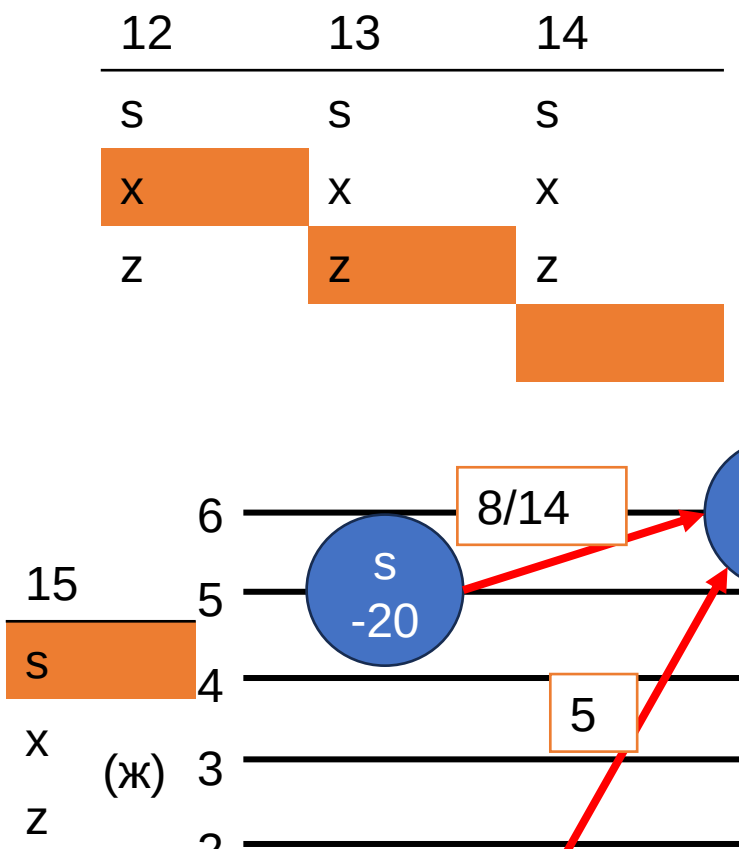






Разгрузка переполненной вершины





Алгоритм "поднять-в-начало"

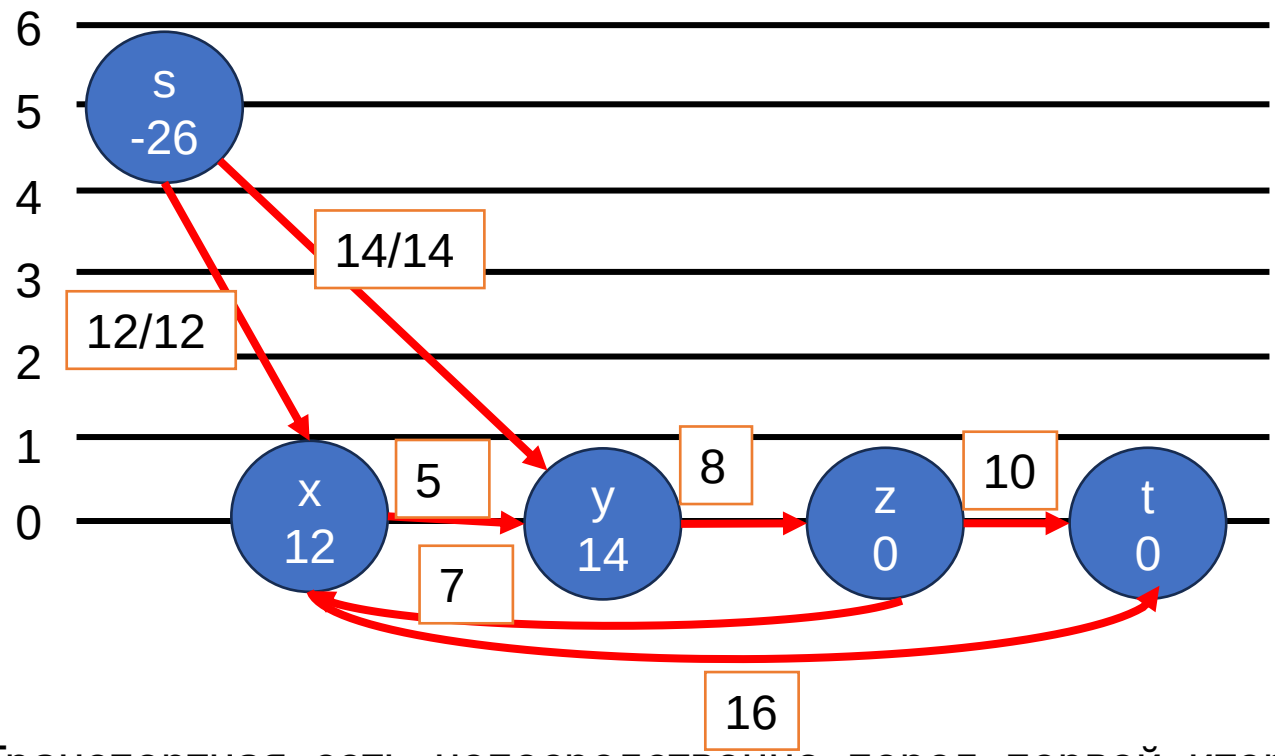
В алгоритме "поднять-в-начало" поддерживается связанный список L, состоящий из всех вершин множества V - {s,t}. Ключевым свойством данного списка является то, что вершины в нем топологически отсортированы в соответствии с допустимой сетью, как будет показано при рассмотрении инварианта цикла ниже.

В приведенном ниже псевдокоде алгоритма "поднять-в-начало" предполагается, что для каждой вершины и уже создан список соседей и.N. Кроме того, предполагается, что u.next указывает на вершину, следующую за и в списке L, и что, как обычно, если и - последняя вершина данного списка, то u.next = NIL.

```
Relabel-To-Front(G, s, t)
1 Initialize-Preflow(G, s)
2 L = G. V - {s,t}, в произвольном порядке
3 for каждой вершины u ∈ G. V - {s, t}
4         u.current = u. N.head
5 u = L.head
6 while u ≠ NIL
7         old-height = u.h
8         Discharge(u)
9         if u.h > old-height
10                переместить и в начало списка L
11        u = u.next
```

Алгоритм "поднять-в-начало"

(a)

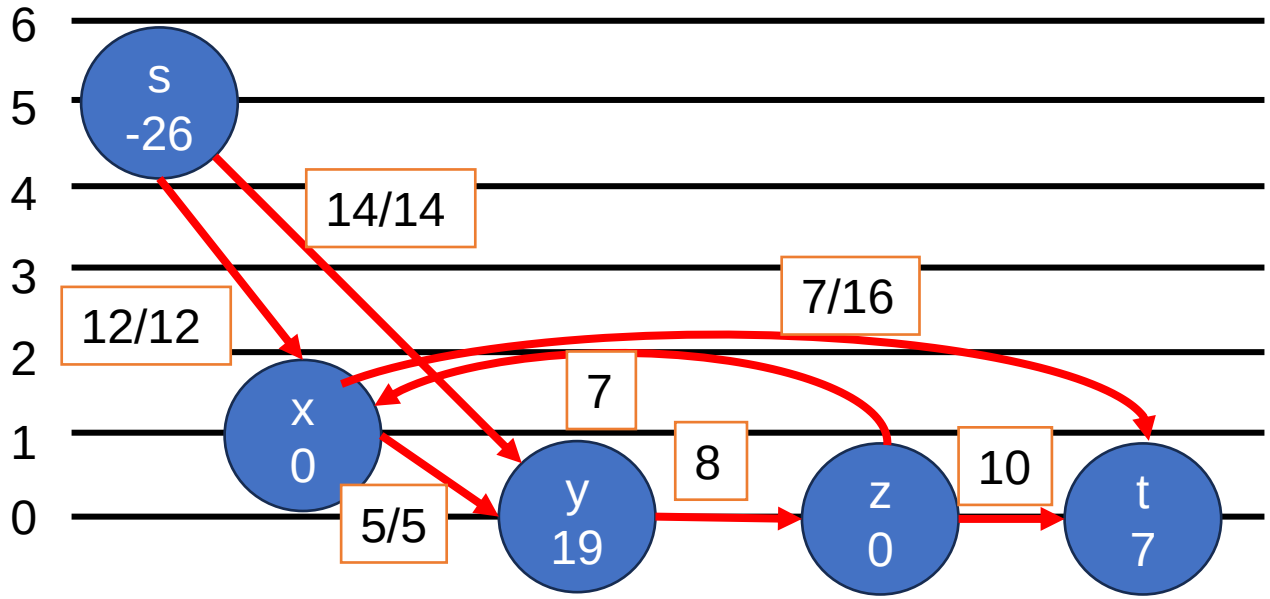


L:	x	y	z
N:	s	s	x
	y	x	y
	z	z	t
	t		

(a) Транспортная сеть непосредственно перед первой итерацией цикла while. Изначально источник s покидают 26 единиц потока. В правой части показан исходный список L = (x, y, z), где изначально u = x. Под каждой вершиной в списке L приведен ее список соседей, в котором выделен текущий сосед. Он поднимается до высоты 1, 5 единиц избыточного потока проталкиваются в y, а 7 оставшихся единиц избытка проталкиваются в сток t. Поскольку x поднята, она перемещается в начало списка L, что в данном случае не приводит к изменению структуры L.

Алгоритм "поднять-в-начало"

(б)

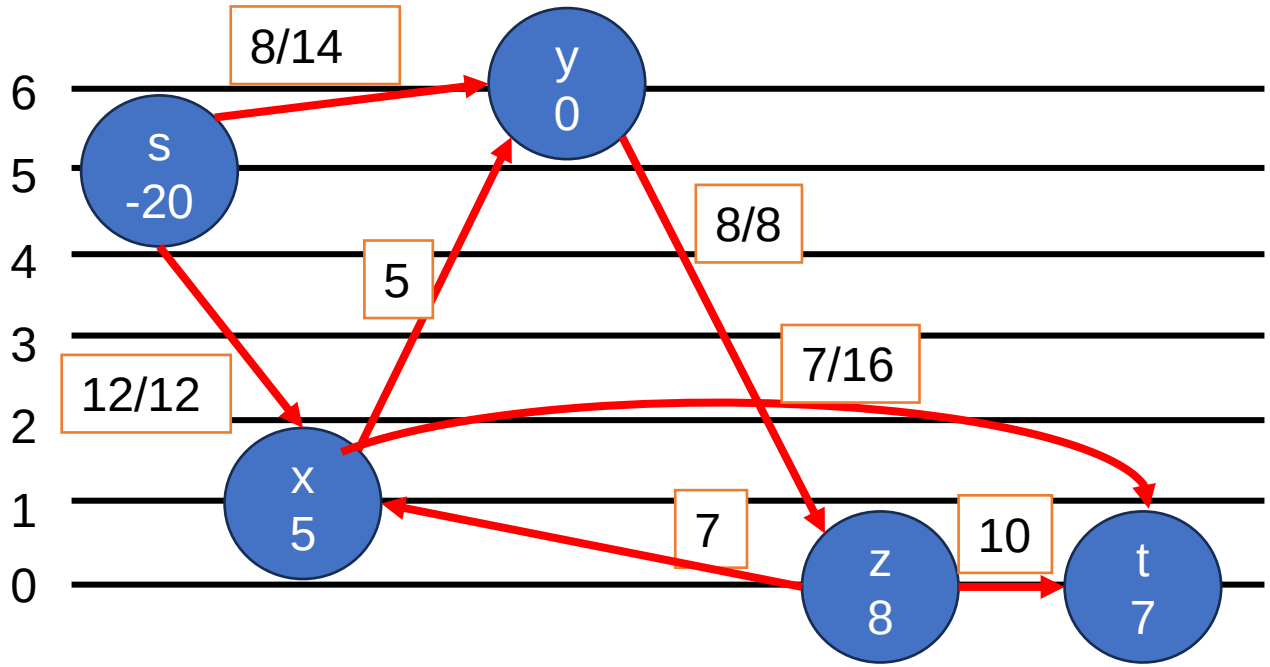


L:	x	y	z
N:	s	s	x
	y	x	y
	z	z	t
	t		

(б) После x следующей в L вершиной, подвергающейся разгрузке, является вершина y. Поскольку вершина y поднята, она перемещается в начало списка L.

Алгоритм "поднять-в-начало"

(в)

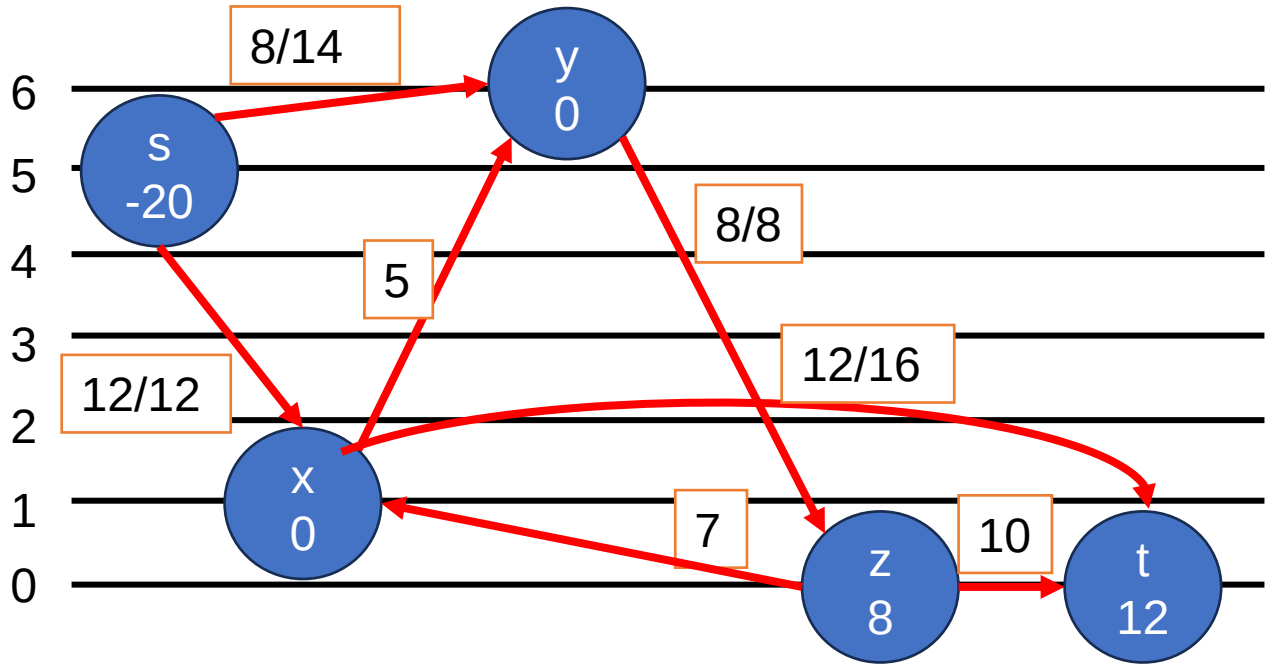


L:	y	x	z
N:	s	s	x
	x	y	y
	z	z	t
		t	

(в) Теперь вершина x следует в списке L за y, так что она вновь разгружается с проталкиванием всех 5 единиц избытка в t. Поскольку вершина x в этой операции разгрузки не поднимается, она остается на своем месте в списке L.

Алгоритм "поднять-в-начало"

(г)

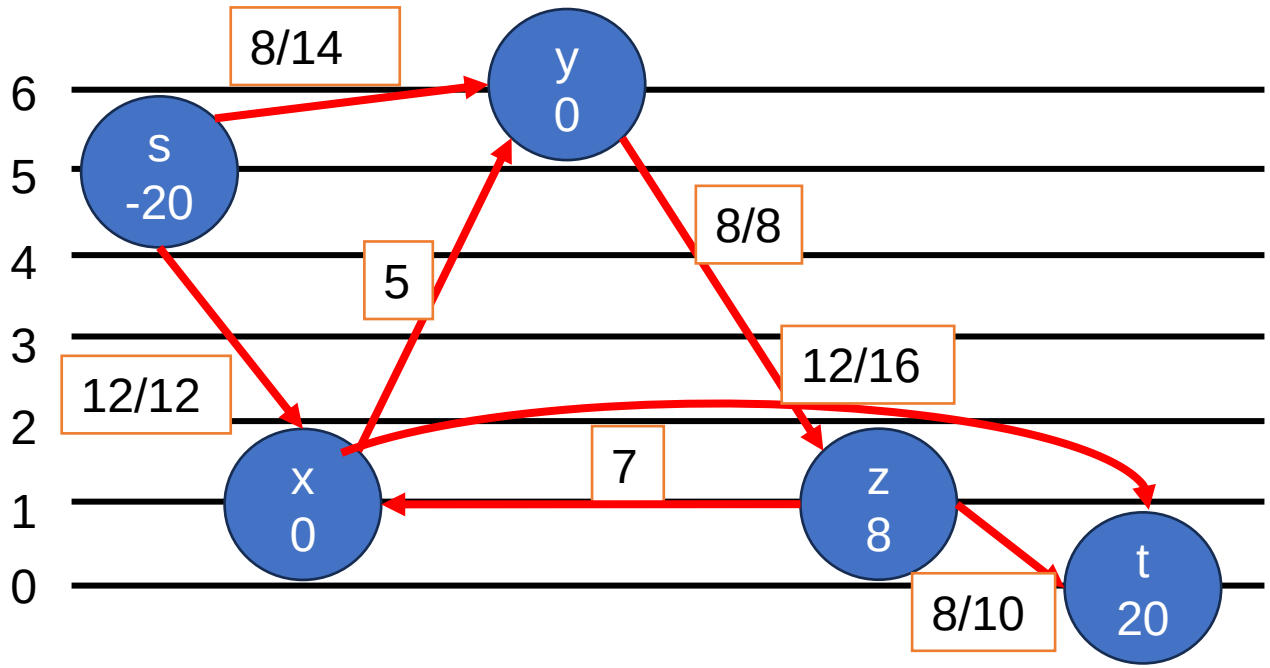


L:	y	x	z
N:	s	s	x
	x	y	y
	z	z	t
		t	

(г) Поскольку в списке L за вершиной x следует вершина z, она подвергается разгрузке. Эта вершина поднимается до высоты 1, а все 8 единиц избытка потока проталкиваются в t. Поскольку z поднята, она перемещается в начало списка L.

Алгоритм "поднять-в-начало"

(д)



L:	z	y	x
N:	x	s	s
	y	x	y
	t	z	z
			t

(д) Теперь за вершиной z в списке L следует вершина y, так что должна быть выполнена ее разгрузка. Но поскольку в вершине y избыток отсутствует, процедура DISCHARGE тут же выполняет возврат, и y остается в L на своем месте. Затем выполняется разгрузка вершины x. Поскольку она также не имеет избытка, процедура DISCHARGE вновь выполняет немедленный возврат, и также остается на собственном месте в L. Процедура RELABEL-TO-FRONT достигает конца списка L и завершается. Переполненных вершин нет, и предпоток представляет собой максимальный поток.

Алгоритм "поднять-в-начало"

Чтобы показать, что процедура Relabel-To-Front вычисляет максимальный поток, покажем, что она является реализацией обобщенного алгоритма проталкивания предпотока. Во-первых, заметим, что она выполняет операции проталкивания и подъема только тогда, когда они применимы, что гарантируется леммой 1.29. Остается показать, что, когда процедура Relabel-To-Front завершается, не применима ни одна основная операция. Дальнейшее доказательство корректности построено на следующем инварианте цикла:

При каждом выполнении проверки в строке 6 процедуры Relabel-To-Front список L является топологическим упорядочением вершин допустимой сети $G_{f.h} = (V, E_{f.h})$, и ни одна вершина, стоящая в списке перед и, не имеет избыточного потока.

Инициализация. Непосредственно после запуска процедуры Initialize-Preflow s.h = |V| и v.h = 0 для всех $v \in V - \{s\}$. Поскольку $|V| \geq 2$ (так как V содержит как минимум исток s и сток t), ни одно ребро не является допустимым. Следовательно, $E_{f,h} = \emptyset$, и любое упорядочение множества $V - \{s,t\}$ является топологическим упорядочением $G_{f,h}$.

Поскольку изначально вершина и является заголовком списка L, перед ней нет вершин, следовательно, перед ней нет вершин с избытком потока.

Сохранение. Чтобы убедиться в том, что данное топологическое упорядочение сохраняется при проведении итераций цикла while, прежде всего заметим, что к изменению допустимой сети приводят только операции проталкивания и подъема. Согласно лемме 1.27 операции проталкивания не приводят к тому, что ребра становятся допустимыми. Поэтому допустимые ребра могут создаваться только подъемами.

Алгоритм "поднять-в-начало"

Однако после того, как вершина и подверглась подъему, согласно лемме 1.28 больше не существует допустимых ребер, входящих в u, но могут быть допустимые ребра, выходящие из нее. Таким образом, перемещая и в начало списка L, алгоритм гарантирует, что все допустимые ребра, выходящие из и, удовлетворяют условию топологического упорядочения.

Чтобы убедиться в том, что ни одна вершина, предшествующая и в списке L, не имеет избытка потока, обозначим вершину, которая будет текущей вершиной и на следующей итерации, как u'. Среди вершин, которые будут предшествовать u' на следующей итерации, находится текущая вершина u (согласно строке 11), и либо больше таких вершин нет (если u подвергалась подъему), либо там находятся те же вершины, что и ранее (если u не поднималась). Поскольку u подверглась разгрузке, она не содержит избытка потока. Следовательно, если u подвергалась подъему в процессе разгрузки, то ни одна вершина, предшествующая u', не содержит избытка потока. Если же u в процессе разгрузки не поднималась, ни одна вершина, стоящая в списке L перед ней, не получила избыток потока при этой разгрузке, поскольку список L остается топологически упорядоченным все время в процессе разгрузки (как уже отмечалось, допустимые ребра создаются только подъемами, а не операциями проталкивания), поэтому каждая операция проталкивания заставляет избыток потока двигаться только к вершинам, расположенным в списке дальше (или же s или t). И вновь ни одна вершина, предшествующая u', не имеет избытка потока.

Завершение. Когда цикл завершается, u оказывается за последним элементом списка L, поэтому инвариант цикла гарантирует, что избыток всех вершин равен 0. Следовательно, ни одна основная операция неприменима.

Анализ

Покажем теперь, что процедура Relabel-To-Front выполняется за время $O(V^3)$ для любой транспортной сети $G = (V,E)$. Поскольку данный алгоритм является реализацией обобщенного алгоритма проталкивания предпотока, воспользуемся следствием 1.21, которое устанавливает границу $O(V)$ для числа подъемов, применяемых к одной вершине, и $O(V^2)$ для общего числа подъемов.

Кроме того, граница $O(VE)$ для суммарного времени, затраченного на выполнение подъемов, а лемма 1.22 устанавливает границу $O(VE)$ для суммарного числа операций насыщающих проталкиваний.

Теорема 1.30

Время выполнения процедуры Relable-To-Front для любой транспортной сети $G = (V, E)$ составляет $O(V^3)$.

Доказательство. Будем считать "фазой" алгоритма "поднять-в-начало" время между двумя последовательными операциями подъема. Поскольку всего выполняется $O(V^2)$ подъемов, в алгоритме насчитывается $O(V^2)$ фаз. Каждая фаза сдержит не более $|V|$ вызовов процедуры Discharge, что можно показать следующим образом. Если процедура Discharge не выполняет подъем, то следующий ее вызов происходит ниже по списку L, а длина L меньше $|V|$. Если же процедура Discharge выполняет подъем, следующий ее вызов происходит уже в другой фазе алгоритма. Поскольку каждая фаза содержит не более $|V|$ обращений к процедуре Discharge, а всего в алгоритме насчитывается $O(V^2)$ фаз, число вызовов данной процедуры в строке 8 процедуры Relable-To-Front составляет $O(V^3)$.

Анализ

Таким образом, цикл while процедуры Relable-To-Front выполняет работу (не учитывая работу, выполняемую внутри процедуры Discharge), не превышающую $O(V^3)$.

Теперь необходимо проанализировать работу процедуры Discharge в ходе выполнения данного алгоритма. Каждая итерация while в процедуре Discharge заключается в выполнении одного из трех действий. Проанализируем объем работы при выполнении каждого из них.

Начнем с подъемов (строки 4 и 5). Время выполнения всех $O(V^2)$ подъемов ограничивается пределом $O(VE)$.

Теперь предположим, что действие заключается в обновлении указателя u.current в строке 8. Это действие выполняется $O(degree(u))$ раз всякий раз, когда вершина и подвергается подъему, что в целом для вершины составляет $O(V \bullet degree(u))$ раз. Следовательно, для всех вершин общий объем работы по перемещению указателей в списках соседей составляет $O(VE)$ согласно лемме о рукопожатиях.

Третий тип действий, выполняемых процедурой Discharge, - операция проталкивания (строка 7). Мы уже знаем, что суммарное число насыщающих операций проталкивания составляет $O(VE)$. Заметим, что если выполняется ненасыщающее проталкивание, процедура Discharge немедленно выполняет возврат, поскольку такое проталкивание уменьшает избыток до 0. Поэтому при каждом обращении к процедуре Discharge может выполняться не более одного ненасыщающего проталкивания. Как мы знаем, процедура Discharge вызывается $O(V^3)$ раз, следовательно, общее время, затраченное на ненасыщающие проталкивания, составляет $O(V^3)$. Таким образом, время выполнения процедуры Relable-To-Front составляет $O(V^3 + VE)$, что эквивалентно $O(V^3)$.

Пример: сегментация изображений

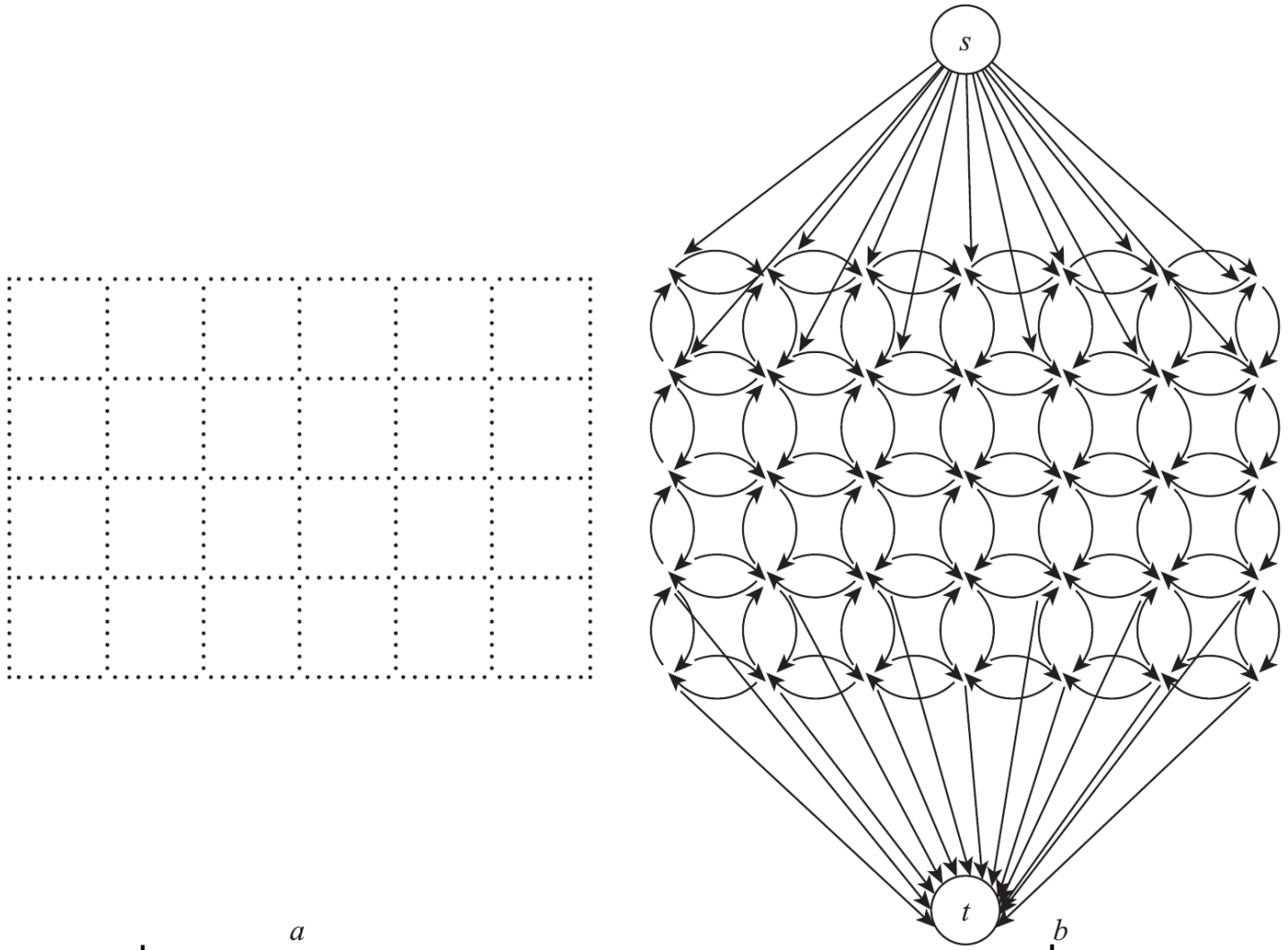
Центральное место в компьютерной обработке изображений занимает сегментация изображения. Например, на изображении могут быть представлены три человека на сложном фоне. Естественная, но непростая задача заключается в идентификации каждого из трех людей как отдельного объекта.

Задача Одна из основных задач этого направления — отделение переднего плана от фона: каждый пиксел изображения помечается как принадлежащий либо к переднему плану, либо к фону. Как выясняется, чрезвычайно естественная модель такого разбиения приводит к задаче, эффективно решаемой посредством вычисления минимального разреза.

Обозначим V множество пикселов анализируемого изображения. Некоторые пары пикселов объявляются соседями; пусть E — множество всех пар соседних пикселов. Таким образом будет получен ненаправленный граф G = (V, E). Здесь мы намеренно не даем четкого определения того, что подразумевается под «пикселом» или «соседским» отношением. Для любого графа G будет получено эффективное решение задачи, поэтому мы можем определять эти концепции так, как считаем нужным. Конечно, пикселы естественно представлять как точки, образующие сетку; при этом соседями считаются пикселы, занимающие смежные позиции в этой сетке.

Для каждого пиксела i имеется степень правдоподобия a_i того, что он принадлежит переднему плану, и степень правдоподобия b_i того, что он принадлежит фону. Условно будем считать, что величины правдоподобия представляют собой произвольные неотрицательные числа, содержащиеся в постановке задачи, и они указывают, насколько желательно, чтобы пиксел i считался относящимся к переднему плану или фону. В остальном не так важно, какие физические свойства изображения оценивают эти характеристики и как они вычисляются.

Пример: сегментация изображений



a — матрица пикселов; b — схема соответствующего потокового графа; на схеме изображены не все ребра от источника к стоку

Пример: сегментация изображений

Каждой отдельный пиксел i было бы разумно отнести к переднему плану, если $a_i > b_i$, или к фону в противном случае. Однако решение относительно i зависит от решений, принимаемых относительно соседей i. Например, если многие соседи i помечены как относящиеся к фону, то у нас появляется больше оснований отнести к фону i; это приводит к «сглаживанию» границ между фоном и передним планом, и снижению длины границы. Для каждой пары соседних пикселов (i, j) вводится штраф за разделение $p_{ij} \geq 0$, применяемый в том случае, если один пиксел пары относится к переднему плану, а другой к фону.

А теперь мы можем точно определить задачу сегментации в контексте параметров правдоподобия и штрафа за разделение. Требуется найти разбиение множества пикселов на подмножества A и B (передний план и фон соответственно), максимизирующее

$$q(A,B) = \sum_{i \epsilon A} a_i + \sum_{j \epsilon B} b_j - \sum_{\substack{(i,j) \epsilon E \\ |A \cap (i,j)|=1}} p_{ij}\,.$$

Таким образом, высокие значения правдоподобия награждаются, а соседние пары (i, j), в которых один пиксел принадлежит A, а другой принадлежит B, штрафуются. Задача заключается в вычислении оптимальной разметки — разбиения (A, B), максимизирующей q(A, B).

Пример: сегментация изображений

Разработка и анализ алгоритма

Нетрудно заметить сходство между задачей о минимальном разрезе и задачей нахождения оптимальной разметки. Тем не менее между этими задачами также существует ряд важных различий. Во-первых, целевая функция максимизируется, а не минимизируется. Во-вторых, в задаче разметки нет источника и стока; кроме того, нам приходится иметь дело со значениями a_i и b_i узлов. В-третьих, граф G является ненаправленным, тогда как в задаче о минимальном разрезе используется направленный граф. Все эти проблемы кратко рассмотрены ниже.

Первая проблема — тот факт, что задача сегментации направлена на максимизацию — решается следующим наблюдением. Пусть $Q = \sum_i (a_i + b_i)$. Сумма $\sum_{i \epsilon A} a_i + \sum_{j \epsilon B} b_j$ не отличается от суммы $Q - \sum_{i \epsilon A} b_i - \sum_{j \epsilon B} a_j$, что позволяет записать

$$q(A,B) = Q - \sum_{i \epsilon A} b_i - \sum_{j \epsilon B} a_j - \sum_{\substack{(i,j) \epsilon E \\ |A \cap (i,j)|=1}} p_{ij} .$$

Таким образом, мы видим, что задача максимизации q(A, B) эквивалентна задаче минимизации величины

$$q'(A,B) = + \sum_{i \epsilon A} b_i + \sum_{j \epsilon B} a_j + \sum_{\substack{(i,j) \epsilon E \\ |A \cap (i,j)|=1}} p_{ij} .$$

Пример: сегментация изображений

Что касается отсутствия источника и стока, мы поступим так же, как в других предшествующих построениях: создадим новый «супериточник» s, представляющий передний план, и новый «суперсток», представляющий фон. Заодно это позволит разобраться со значениями a_i и b_i, находящимися в узлах (тогда как минимальные разрезы ограничиваются числами, связанными с ребрами). А именно, каждый из узлов s и t соединяется с каждым пикселом, а a_i и b_i используются для определения соответствующих пропускных способностей ребер между пикселом i и источником и стоком соответственно.

Наконец, чтобы разобраться с направленностью ребер, мы смоделируем каждую из соседних пар (i, j) двумя направленными ребрами (i, j) и (j, i), как было сделано в ненаправленной задаче о непересекающихся путях. Этот прием очень хорошо работает в данном случае, так как в любом разрезе s–t не более одного из двух противоположно направленных ребер может переходить со стороны s на сторону t разреза (потому что другое ребро в этом случае должно переходить со стороны t на сторону s).

Конкретнее, мы определяем следующую потоковую сеть G' = (V’, E). Множество узлов V' состоит из множества пикселов V с двумя дополнительными узлами s и t. Для каждой соседней пары пикселов i и j добавляются направленные ребра (i, j) и (j, i), каждое из которых обладает пропускной способностью p_{ij}. Для каждого пиксела i добавляются ребро (s, i) с пропускной способностью a_i, и ребро (i, t) с пропускной способностью b_i.

Пример: сегментация изображений

Теперь разрез s–t (A, B) соответствует разбиению пикселов на множества A и B. Рассмотрим, как пропускная способность разреза c(A, B) связана с величиной q'(A, B), которую мы пытаемся минимизировать Ребра, пересекающие разрез (A, B), делятся на три естественные категории.

- Ребра (s, j), для которых j ∈ B; такое ребро вносит вклад aj в пропускную способность разреза.
- Ребра (i, t), для которых i ∈ A; такое ребро вносит вклад bi в пропускную способность разреза.
- Ребра (i, j), для которых i ∈ A и j ∈ B; такое ребро вносит вклад p_{ij} в пропускную способность разреза.

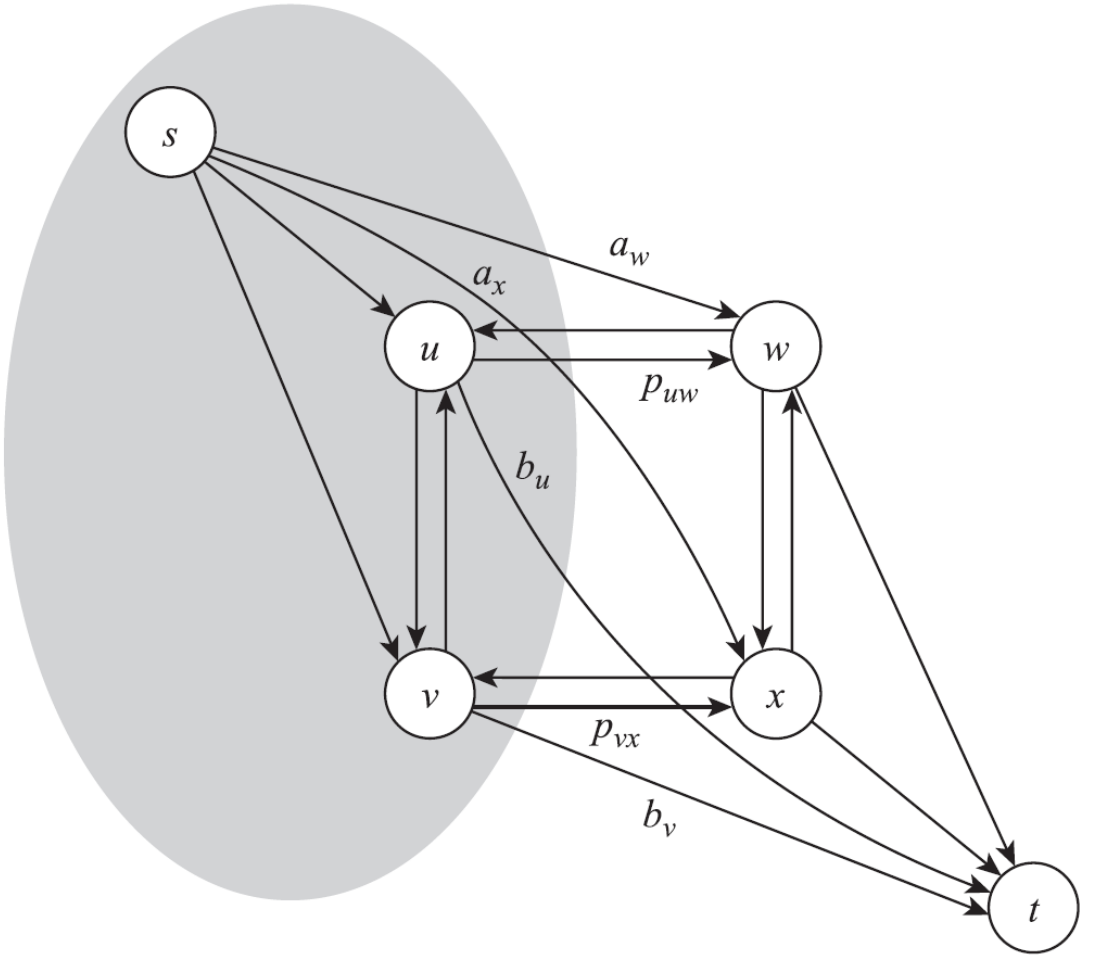
Суммируя вклады этих трех типов ребер, получаем

$$c(A,B) = \sum_{i \epsilon A} a_i b + \sum_{j \epsilon B} a_j - \sum_{\substack{(i,j) \epsilon E \\ |A \cap (i,j)|=1}} p_{ij} = q'(A,B).$$

Итак, все идеально складывается. Потоковая сеть устроена так, что пропускная способность разреза (A, B) в точности соответствует величине q'(A, B): три типа ребер, пересекающих разрез (A, B), которые мы определили (ребра от источника, ребра к стоку и ребра, не связанные ни с источником, ни со стоком), соответствуют трем типам слагаемых в выражении q'(A, B).

Пример: сегментация изображений

Разрез s–t на графе, построенном для четырех пикселов. Обратите внимание на то, как в разрезе отражены три типа слагаемых в выражении q'(A, B)



Следовательно, если мы хотим минимизировать q'(A, B) (что, как было показано ранее, эквивалентно максимизации q(A, B)), достаточно найти разрез с минимальной пропускной способностью — а мы уже знаем, как эффективно решить эту задачу.

Таким образом, решение задачи о минимальном разрезе дает оптимальный алгоритм для решения задачи об отделении фона от переднего плана.

Утверждение 2.33. Решение задачи сегментации может быть получено при помощи алгоритма нахождения минимального разреза в графе G', построенного ранее. Для минимального разреза (A', B') разбиение (A, B), полученное удалением s* и t*, максимизирует метрику сегментации q(A, B).