Исследование операций

Поток в сети и его приложения

В.В. Лепин
Институт математики
НАН Беларуси, Минск

План

- Задача о максимальном потоке (MaxFlow): алгоритм Форда-Фалкерсона, теорема о макмимальном потоке и минимальном разрезе;
- Двойственное объяснение алгоритма Форда-Фалкерсона и теоремы о макмимальном потоке и минимальном разрезе;
- Эффективные алгоритмы для задачи о максимальном потоке: метод масштабирования, алгоритм Эдмонса-Карпа (Edmonds-Karp), алгоритм Диница (Dinic)
- Обобщения задачи о максимальном потоке : нижняя граница на пропускную способность, много источникков & много стоков, косвенный граф.

Краткая история Задачи о максимальном потоке I

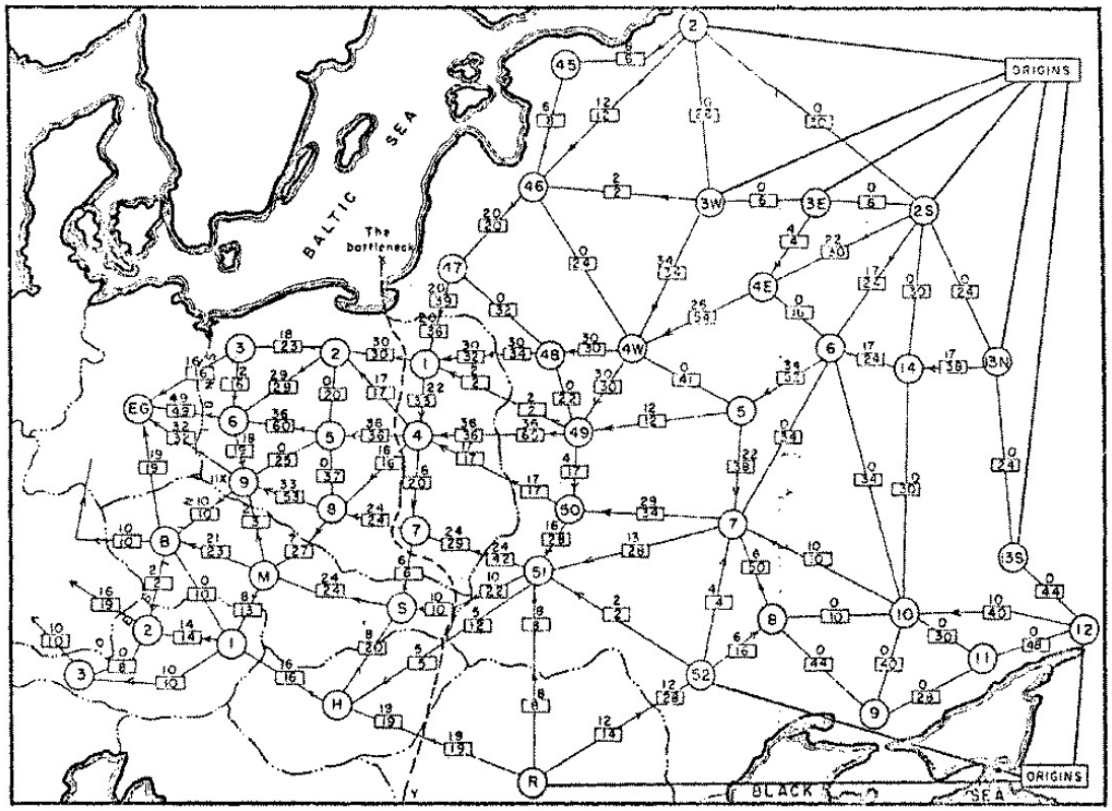


Рис.: Железно-дорожная сеть СССР, 1955

Краткая история Задачи о максимальном потоке II

- *“Принципиальная схема железнодорожной сети Советского Союза и стран Восточной Европы. Максимальный поток 163 000 тонн из России в Восточную Европу и разрез с пропускной способности 163 000 тонн, обозначен как "узкое место"...”*
- Недавно рассекреченный отчет ВВС США указывает, что первоначальная мотивация Задачи о максимальном потоке и алгоритма Форда-Фалкерсона заключалась в том *как наиболее эффективным образом нарушить железнодорожные перевозки Советского Союза* [А. Шрайвер, 2002].

Задача о максимальном потоке и задача о минимальном разрезе

Задача о максимальном потоке

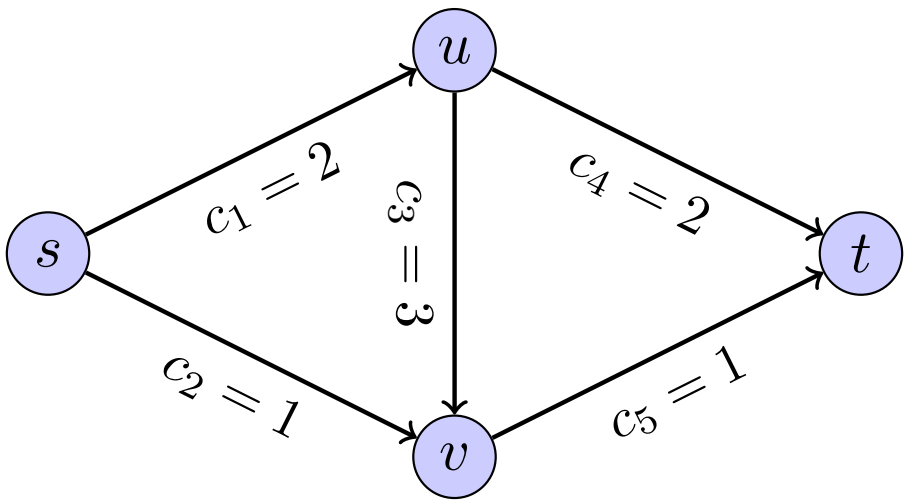
ВХОД:
Ориентированный граф $G = (V, E)$. Каждое ребро e имеет пропускную способность C_e. Два специальных узла: **источник** s и **сток** t;

ВЫХОД:
Для каждого ребра $e = (u, v)$ назначить поток $f(u, v) \leq C(u, v)$ такой, чтобы $\sum_{u,(s,u)\in E} f(s, u)$ была максимальной.

Задача о максимальном потоке

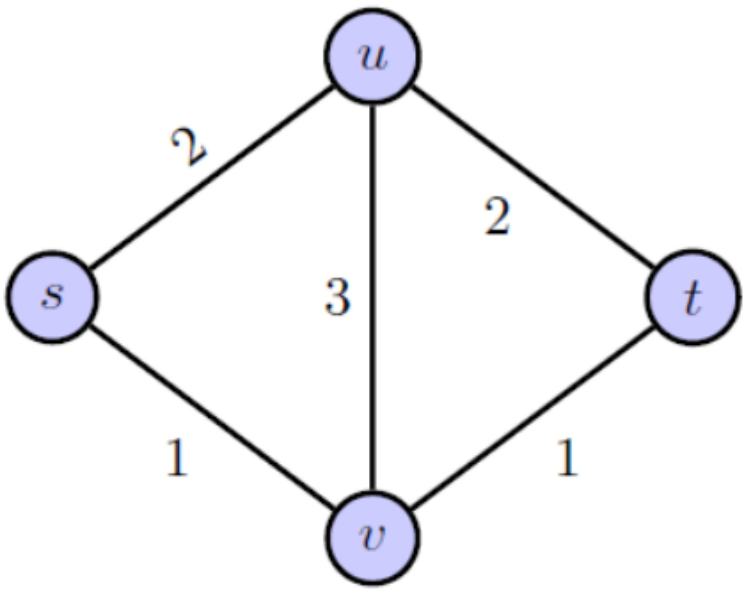
ВХОД:
Ориентированный граф $G = (V, E)$. Каждое ребро e имеет пропускную способность C_e. Два специальных узла: **источник** s и **сток** t;

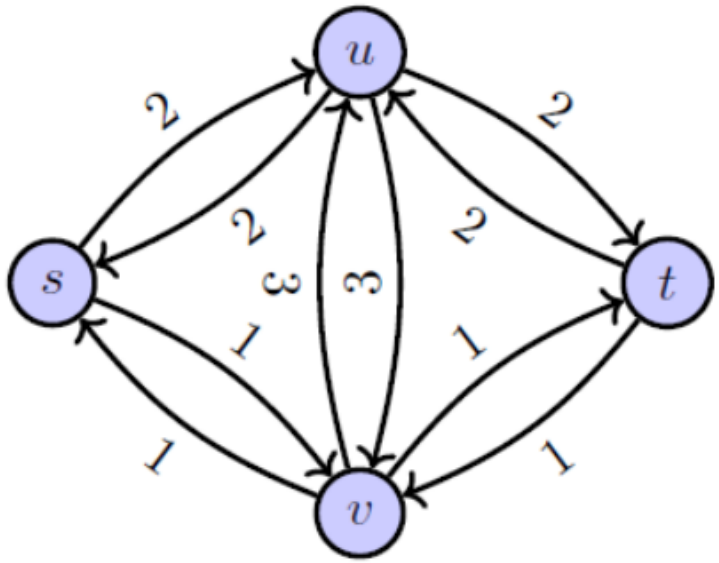
ВЫХОД:
Для каждого ребра $e = (u, v)$ назначить поток $f(u, v) \leq C(u, v)$ такой, чтобы $\sum_{u,(s,u)\in E} f(s, u)$ была максимальной.

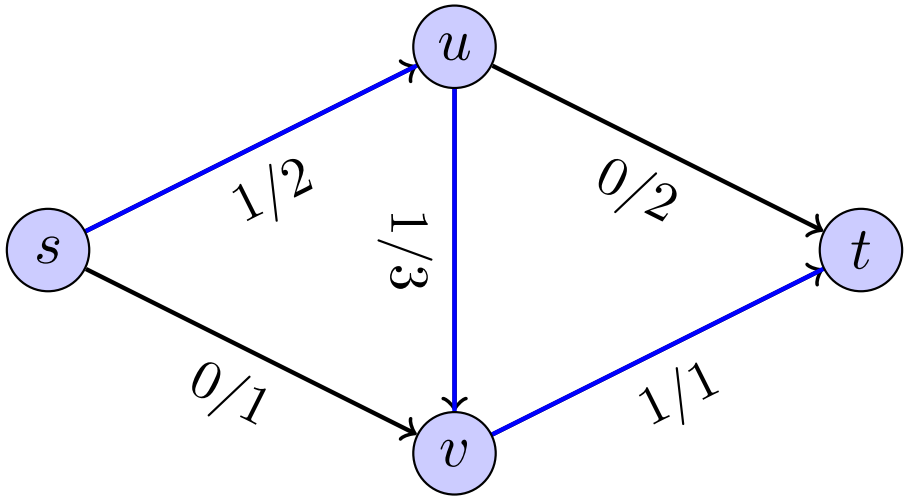


Цель: перевести как можно больше груза из **источника** s в **сток** t.

Преобразование неориентированного графа в сеть

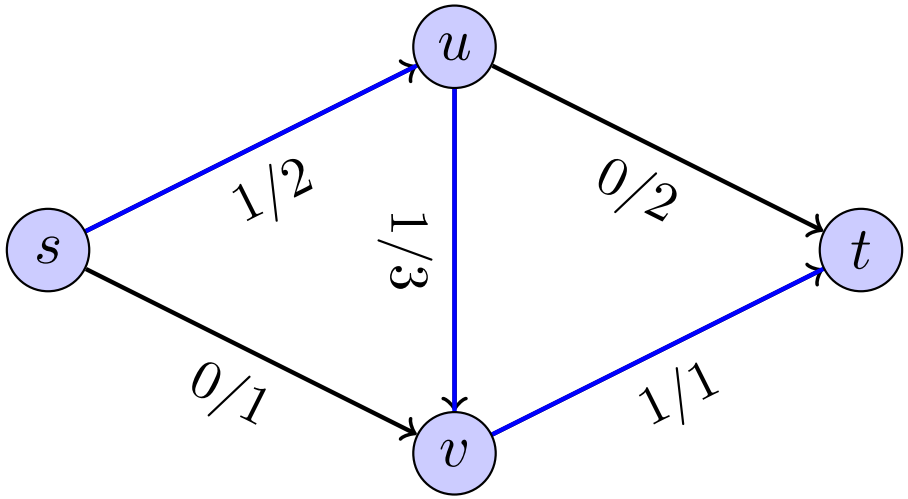






$s - t$ ПОТОК

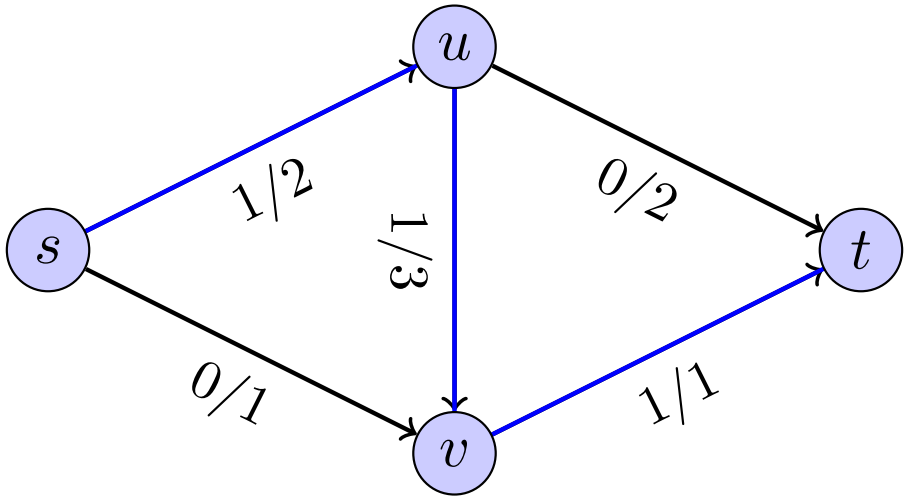
$f : E \to R^+$ называется $s - t$ **ПОТОКОМ** если:



$s - t$ ПОТОК

$f : E \to R^+$ называется $s - t$ **ПОТОКОМ** если:

1. (Ограничения по пропускной способности):
   $0 \leq f(e) \leq C_e$ для всех ребер e;

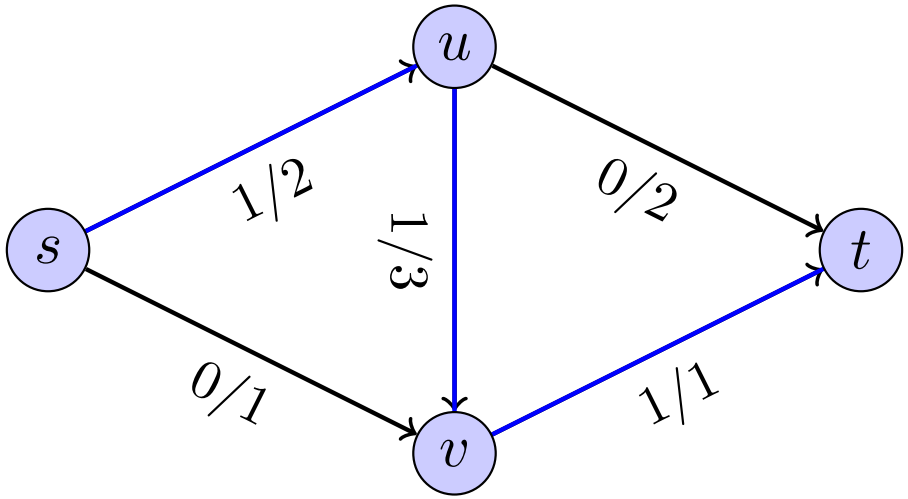


$s-t$ ПОТОК

$f : E \to R^+$ называется $s-t$ **ПОТОКОМ** если:

1. (Ограничения по пропускной способности): $0 \leq f(e) \leq C_e$ для всех ребер e;
2. (Сохранение потока): Для любой промежуточной вершины $v \in V \setminus \{s,t\}$, $f^{in}(v) = f^{out}(v)$, где $f^{in}(v) = \sum_{e \text{ into } v} f(e)$ и $f^{out}(v) = \sum_{e \text{ out of } v} f(e)$. (T.e. вход = выходу для любой внутренней вершины.)

$s-t$ ПОТОК



$s-t$ ПОТОК

$f : E \to R^+$ называется $s-t$ **потоком** если:

1. (Ограничения по пропускной способности): $0 \leq f(e) \leq C_e$ для всех ребер e;
2. (Сохранение потока): Для любой промежуточной вершины $v \in V \setminus \{s,t\}$, $f^{in}(v) = f^{out}(v)$, где $f^{in}(v) = \sum_{e \text{ into } v} f(e)$ и $f^{out}(v) = \sum_{e \text{ out of } v} f(e)$. (T.e. вход = выходу для любой внутренней вершины.)

Величина потока f равна $|f| = f^{out}(s)$.

$s-t$ ПОТОК

$f : V \times V \to R^+$ называется $s-t$ **потоком** если:

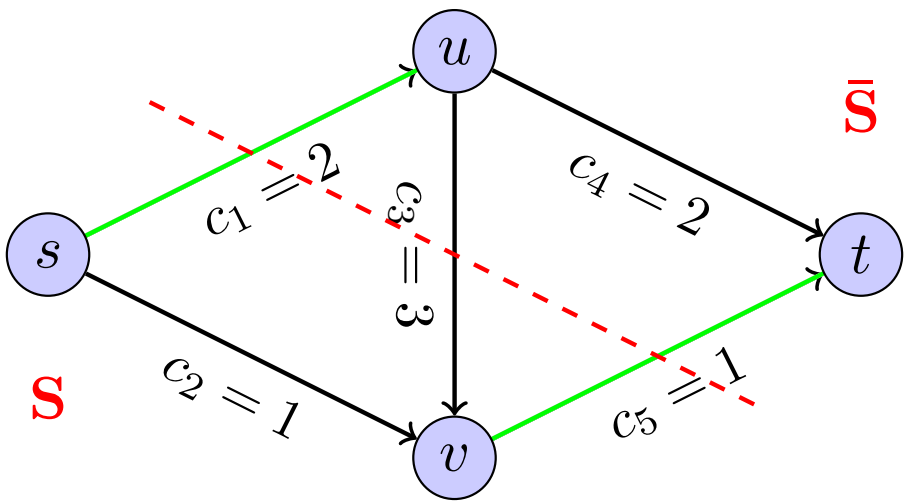
1. (Ограничения по пропускной способности):
   $0 \leq x_{ij} \leq c_{ij} \quad \forall (i,j) \in E$;
2. (Сохранение потока): Для любой промежуточной вершины $\sum_{(i,j)\in E} x_{ij} = \sum_{(j,k)\in E} x_{jk} \quad \forall j \in V, j \neq s,t$
3. $x_{ij} = 0 \quad (i,j) \notin E$

Величина потока f равна $|f| = \sum_{(s,i)\in E} x_{si} = \sum_{(j,s)\in E} x_{js}$.

Задача о минимальном разрезе

ВХОД:
Ориентированный граф $G = (V, E)$. Каждое ребро e имеет пропускную способность C_e. Два специальных узла: **источник** s и **сток** t;

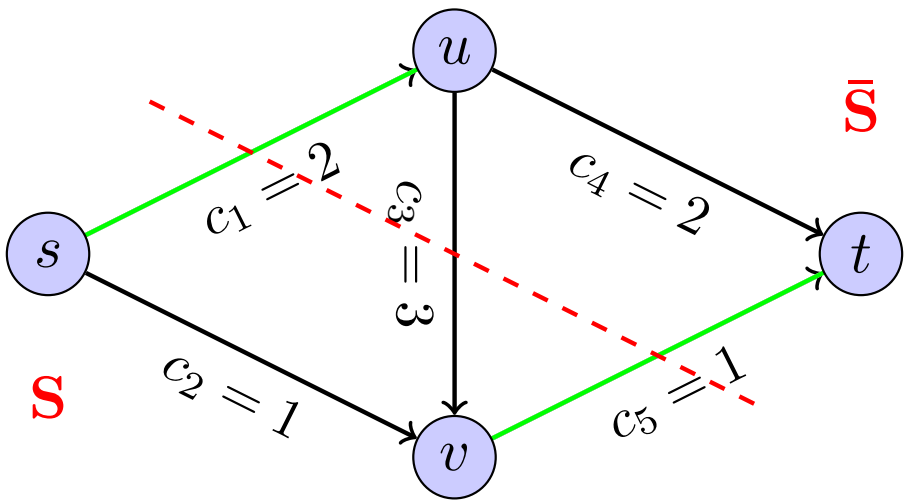
ВЫХОД:
Найти $s - t$ разрез, имеющий минимальную пропускную способность.



$$C(S, \bar{S}) = 3$$

$s-t$ РАЗРЕ

$s-t$ разрезом называется разбиение $(S, \bar{S})$ множества V такое, что $s \in S$ и $t \in \bar{S}$. **Пропускная способность разреза** $(S, \bar{S})$ определяется как $C(S, \bar{S}) = \sum_{e \text{ from } S \text{ to } \bar{S}} C(e)$.



$$C(S, \bar{S}) = 3$$

Некоторые алгоритмы решения задачи

Год	Разработчик	Трудоемкость
1956	L. R. Ford and D. R. Fulkerson	$O(mC)$
1970	Y. Dinitz	$O(mn^2)$
1972	J. Edmonds and R. Karp	$O(m^2n)$
1974	A. Karzanov	$O(n^3)$
1986	A. Goldberg and R. Tarjan	$O(mn^2)$, $O(n^3)$, $O(mn\log(\frac{n^2}{m}))$
2013	J. Orlin	$O(mn)$

Алгоритм Форда — Фалкерсона [1956]

Lester Randolph Ford Jr. and Delbert Ray Fulkerson

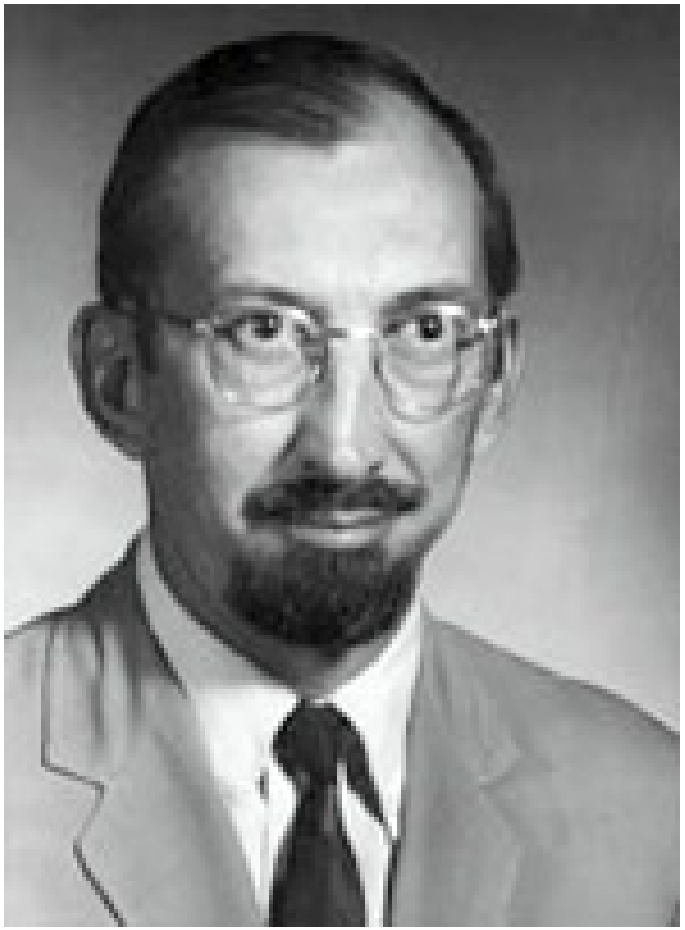


Рис.: Lester Randolph Ford Jr. and Delbert Ray Fulkerson

Замечание 1: Техника динамического программирования

- Динамическое программирование, кажется, не работает, поскольку нелегко определить соответствующие подзадачи. Фактически, не существует эффективного алгоритма, известного для Задачи о максимальном потоке, который действительно можно рассматривать как принадлежащий парадигме динамического программирования.

Замечание 1: Техника динамического программирования

- Динамическое программирование, кажется, не работает, поскольку нелегко определить соответствующие подзадачи. Фактически, не существует эффективного алгоритма, известного для Задачи о максимальном потоке, который действительно можно рассматривать как принадлежащий парадигме динамического программирования.
- Мы знаем, что Задача о максимальном потоке находится в классе **P**, поскольку ее можно сформулировать как задачу линейного программирования. Однако сетевая структура имеет свойство, позволяющее использовать более эффективный алгоритм, неофициально называемый **сетевым симплекс методом**. Кроме того, специальные алгоритмы более эффективны.

Замечание 2: Стратегия улучшения

- Рассмотрим общую Стратегию улучшения:
  Улудшение(f)

```
1: x = x_0; //начинаем с исходного решения;
2: while TRUE do
3:     x =Improve(x); //сделать один шаг к
       оптимальному;
4:     if Выполяется критекий остановки(x, f)
       then
5:         break;
6:     end if
7: end while
8: return x;
```

Три ключевых вопроса:

1. Как построить начальное решение?
   - Для Задачи о максимальном потоке, исходное решение может быть легко получено путем установки $f(e) = 0$ для каждого ребра e (называется 0−потоком). Легко проверить, что ограничения по пропускной способности и сохранению выполняются для 0-потока.

Три ключевых вопроса Стратегии улучшения

Три ключевых вопроса:

1. Как построить начальное решение?
   - Для Задачи о максимальном потоке, исходное решение может быть легко получено путем установки $f(e) = 0$ для каждого ребра e (называется 0−потоком). Легко проверить, что ограничения по пропускной способности и сохранению выполняются для 0-потока.
2. Как улучшить решение?

Три ключевых вопроса Стратегии улучшения

Три ключевых вопроса:

1. Как построить начальное решение?
   - Для Задачи о максимальном потоке, исходное решение может быть легко получено путем установки $f(e) = 0$ для каждого ребра e (называется 0−потоком). Легко проверить, что ограничения по пропускной способности и сохранению выполняются для 0-потока.
2. Как улучшить решение?
3. Когда остановиться?

Идея: увеличивать поток вдоль пути в исходном графе

- Пусть p — простой $s-t$ путь в сети G.

1: Инициализация $f(e) = 0$ для всех e.
2: **while** существует $s-t$ путь в графе G **do**
3: **Произвольно** выбрать $s-t$ путь p в графе G;
4: $f =$УВЕЛИЧЕНИЕ(p, f);
5: **end while**
6: **return** f;

Увеличение потока вдоль пути

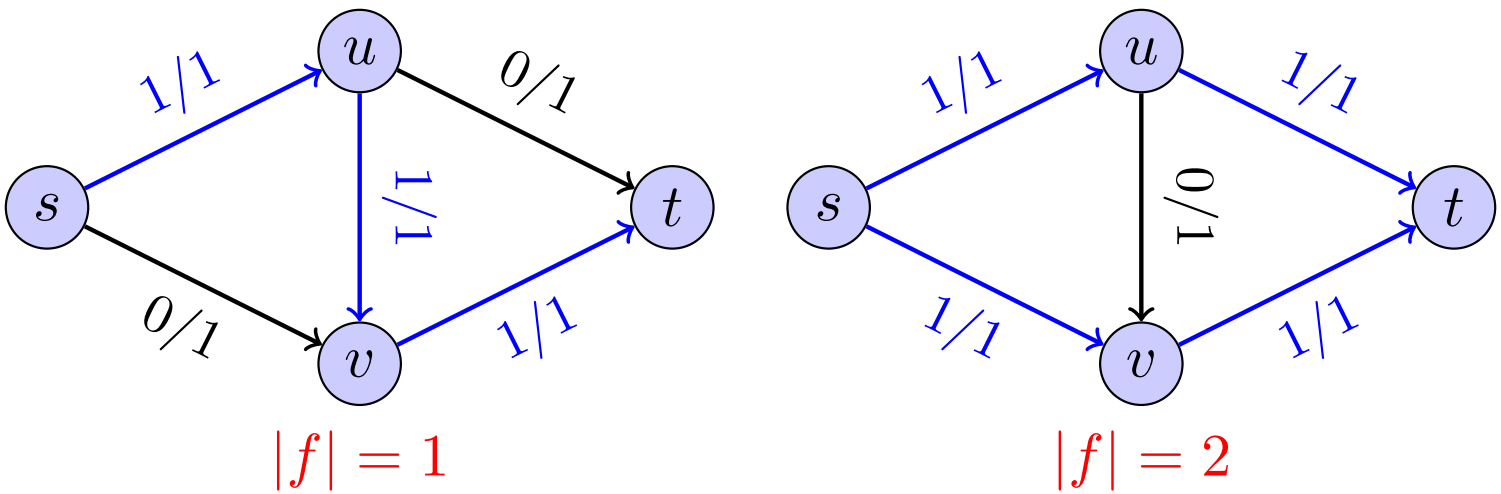
- Определим $bottleneck(p, f)$ как наименьшую остаточную пропускную способность ребер в пути p.

УВЕЛИЧЕНИЕ(p, f)

1: Пусть $b = bottleneck(p, f)$;
2: **for** каждого ребра $e = (u, v) \in p$ **do**
3: Увеличить $f(u, v)$ на b;
4: **end for**

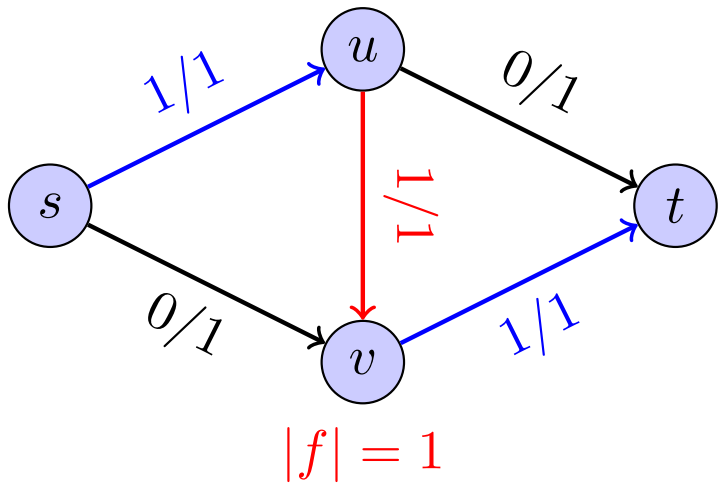
Почему вожможна неудача?

- Рассмотрим следующий пример. Мы начинаем с 0-потока и находим путь $s - t$ в G, скажем, $p = s \to u \to v \to t$, чтобы передать еще одну единицу товара для увеличения значения f.
- Однако мы не можем снова найти путь $s - t$ в G для дальнейшего увеличения f (левый рисунок), хотя максимальное значение потока составляет 2 (правый рисунок).



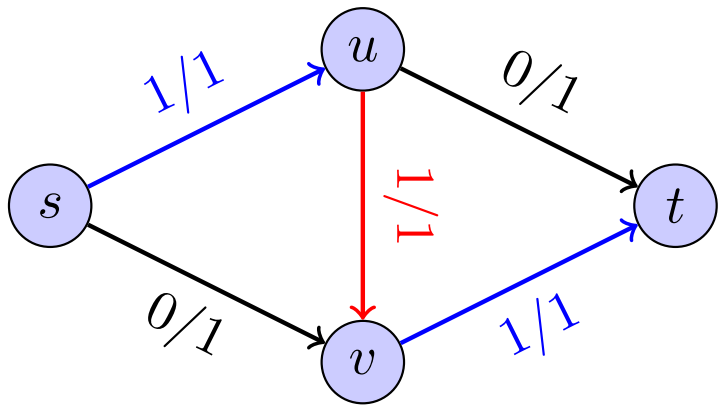
Алгоритм Форда — Фалкерсона: функциональность “откат”

- Ключевое наблюдение:
  - При построении потока f можно допустить ошибки на некоторых ребрах, то есть такие ребра не должны использоваться для продвижения потока. Например, ребро $u \to v$ не должно использоваться.



Алгоритм Форда — Фалкерсона: функциональность “откат”

- Ключевое наблюдение:
  - При построении потока f можно допустить ошибки на некоторых ребрах, то есть такие ребра не должны использоваться для продвижения потока. Например, ребро $u \to v$ не должно использоваться.



$|f| = 1$

  - Чтобы улучшить текущий поток f, мы должны разработать способ **исправлять такие ошибки**, то есть “отменять” передачу потока, назначенную для ребра.

Реализация функциональности “откат”

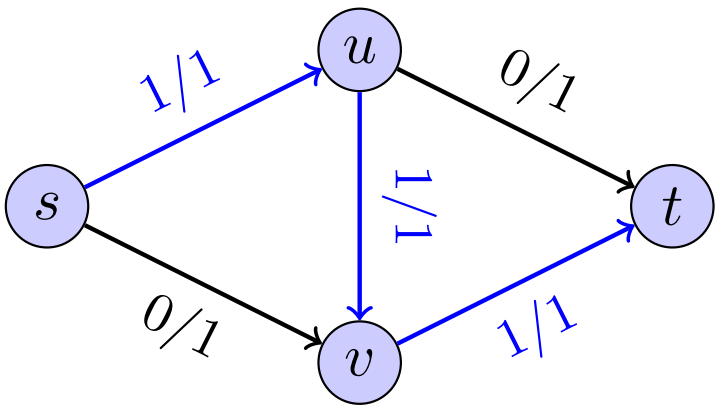
- Но как реализовать функциональность “откат”?

Реализация функциональности “откат”

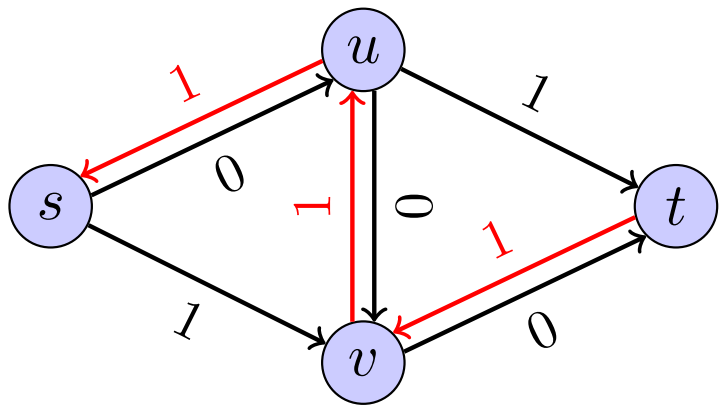
- Но как реализовать функциональность “откат”?
- **Добовлять обратные ребра!**

Реализация функциональности “откат”

- Но как реализовать функциональность “откат”?
- **Добовлять обратные ребра!**
- Предположим, мы добавляем **обратное** ребро $v \to u$ в исходный граф.
  Затем мы можем исправить поток, направив его от v до u.



Поток f



Обратные ребра

Остаточная сеть с “обратными” ребрами для коррекции ошибок

Definition

Для сети $G = (V, E)$ с потоком f определяем **Остаточный граф** $G_f = (V, E')$.

Остаточная сеть с “обратными” ребрами для коррекции ошибок

Definition

Для сети $G = (V, E)$ с потоком f определяем **Остаточный граф** $G_f = (V, E')$.
Для любого ребра $e = (u, v) \in E$ добавляются два ребра в E' следующим образом:

Остаточная сеть с “обратными” ребрами для коррекции ошибок

Definition

Для сети $G = (V, E)$ с потоком f определяем **Остаточный граф** $G_f = (V, E')$.
Для любого ребра $e = (u, v) \in E$ добавляются два ребра в E' следующим образом:

1. **Прямое ребро** (u, v) с остаточной пропускной способностью: Если $f(e) < C(e)$, то ребро $e = (u, v)$ добовляется в G' с пропускной способностью $C(e) = C(e) - f(e)$.

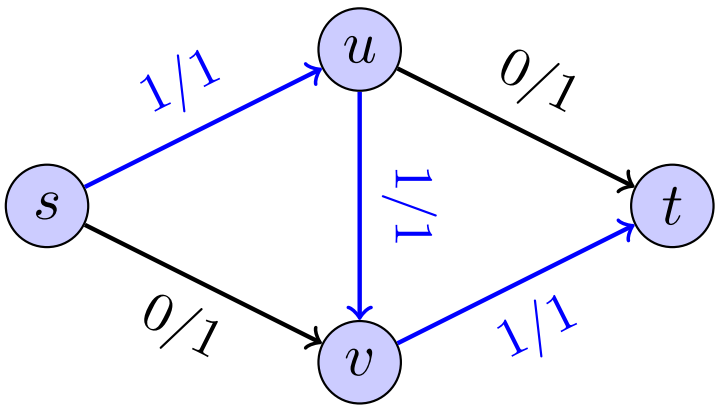
Остаточная сеть с “обратными” ребрами для коррекции ошибок

Definition

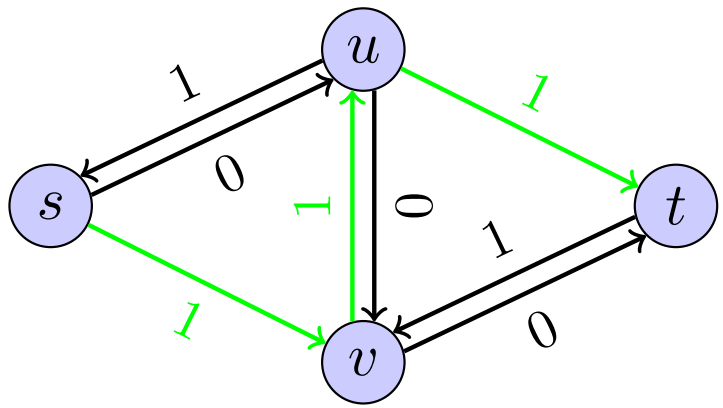
Для сети $G = (V, E)$ с потоком f определяем **Остаточный граф** $G_f = (V, E')$.
Для любого ребра $e = (u, v) \in E$ добавляются два ребра в E' следующим образом:

1. **Прямое ребро** (u, v) с остаточной пропускной способностью: Если $f(e) < C(e)$, то ребро $e = (u, v)$ добовляется в G' с пропускной способностью $C(e) = C(e) - f(e)$.
2. **Обратное ребро** (v, u) с откатной пропускной способностью: Если $f(e) > 0$, то ребро $e' = (v, u)$ добовляется в G' с пропускной способностью $C(e') = f(e)$.

Поиск $s - t$ пути в G_f вместо G



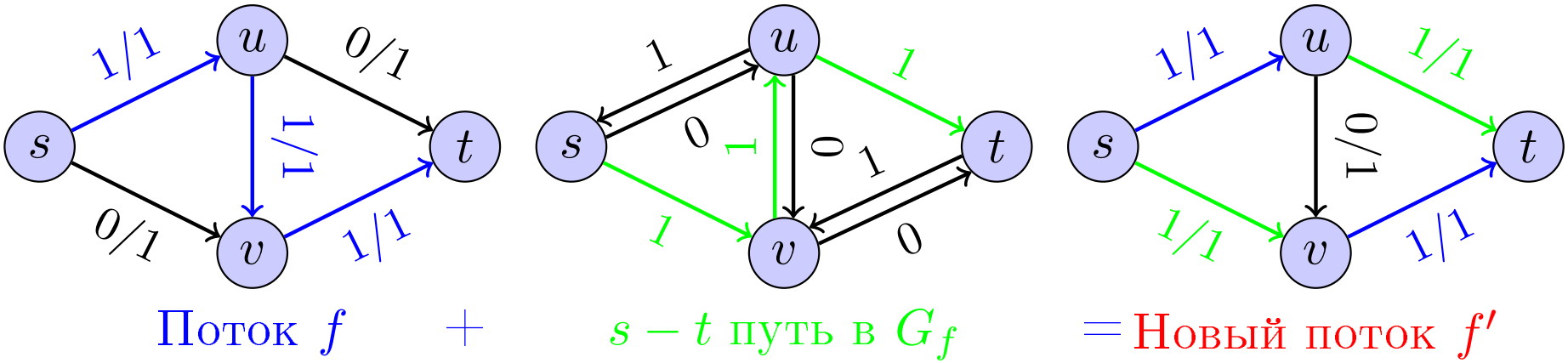
Поток f



Остаточная сеть G_f

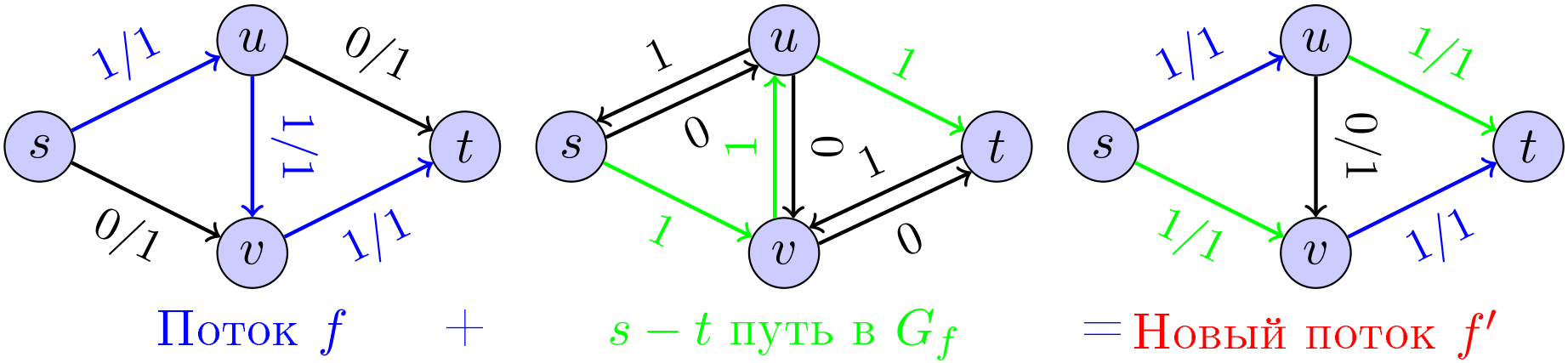
- Обратите внимание, что мы не можем найти путь $s - t$ в G; однако, мы можем найти $s - t$ path $s \to v \to u \to t$ в G_f, который содержит обратное ребро (v, u).

Увеличение потока f вдоль пути $s - t$ в G_f



- Используя обратное ребро $v \to u$, ранее принятая передача от u к v отбрасывается.

Увеличение потока f вдоль пути $s-t$ в G_f



- Используя обратное ребро $v \to u$, ранее принятая передача от u к v отбрасывается.
- Более конкретно, поток f, меняет свой путь (передача по $s \to u \to v \to t$ заменяется на $s \to u \to t$), кроме того используется путь $s \to v \to t$.

- Каждый простой $s - t$ путь p в G_f, называется **увеличивающим**. Пусть $bottleneck(p, f)$ — минимальная пропускная способность ребер в пути p.

Алгоритм Форда-Фалкерсона

- Каждый простой $s-t$ путь p в G_f, называется **увеличивающим**. Пусть $bottleneck(p, f)$ — минимальная пропускная способность ребер в пути p.

Ford-Fulkerson algorithm:

1: Инициализация: $f(e) = 0$ для всех e.
2: **while** существует $s-t$ путь в остаточной сети G_f **do**
3: **Произвольно** выбрать $s-t$ путь p в G_f;
4: $f =$AUGMENT(p, f);
5: **end while**
6: **return** f;

Корректность и трудоемкость алгоритма

Свойство 1: увеличение генерирует новый поток

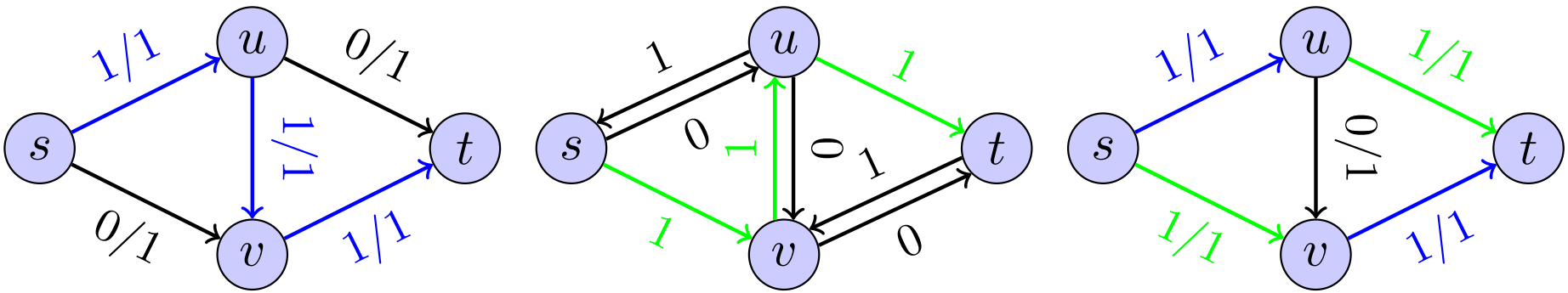
Lemma

Операция $f' = \text{AUGMENT}\ (p, f)$ *генерирует новый поток* f' *в* G.

Свойство 1: увеличение генерирует новый поток

Lemma

Операция $f' = \text{AUGMENT}\ (p, f)$ *генерирует новый поток* f' *в* G.



Доказательство.

- Проверим **ограничения по пропускной способности**: исследуем два возможных случая для ребра $e = (u, v)$ в пути p.

Доказательство.

- Проверим **ограничения по пропускной способности**: исследуем два возможных случая для ребра $e = (u, v)$ в пути p.
  1. (u, v) — прямое ребро, возникшее из $(u, v) \in E$: $0 \leq f(e) \leq f'(e) = f(e) + bottleneck(p, f) \leq f(e) + (C(e) - f(e)) \leq C(e)$.

Доказательство.

- Проверим **ограничения по пропускной способности**: исследуем два возможных случая для ребра $e = (u, v)$ в пути p.
  1. (u, v) — прямое ребро, возникшее из $(u, v) \in E$: $0 \le f(e) \le f'(e) = f(e) + bottleneck(p, f) \le f(e) + (C(e) - f(e)) \le C(e)$.
  2. (u, v) — обратное ребро, возникшее из $(v, u) \in E$: $C(e) \ge f(e) \ge f'(e) = f(e) - bottleneck(p, f) \ge f(e) - f(e) = 0$.

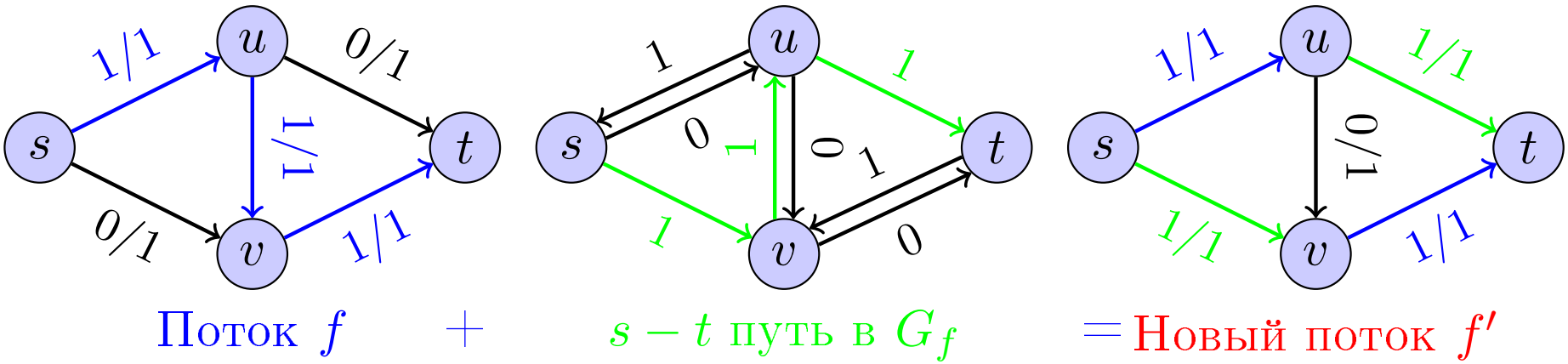
Доказательство.

- Проверим **ограничения по пропускной способности**: исследуем два возможных случая для ребра $e = (u, v)$ в пути p.
  1. (u, v) — прямое ребро, возникшее из $(u, v) \in E$: $0 \leq f(e) \leq f'(e) = f(e) + bottleneck(p, f) \leq f(e) + (C(e) - f(e)) \leq C(e)$.
  2. (u, v) — обратное ребро, возникшее из $(v, u) \in E$: $C(e) \geq f(e) \geq f'(e) = f(e) - bottleneck(p, f) \geq f(e) - f(e) = 0$.
- Провепим **ограничение по сохранению потока**: Для каждого узла v изменение количества потока, входящего в v, совпадает с изменением количества потока, выходящего из v.

Свойство 2: монотонное возрастание потока

Lemma

$|f'| > |f|$.



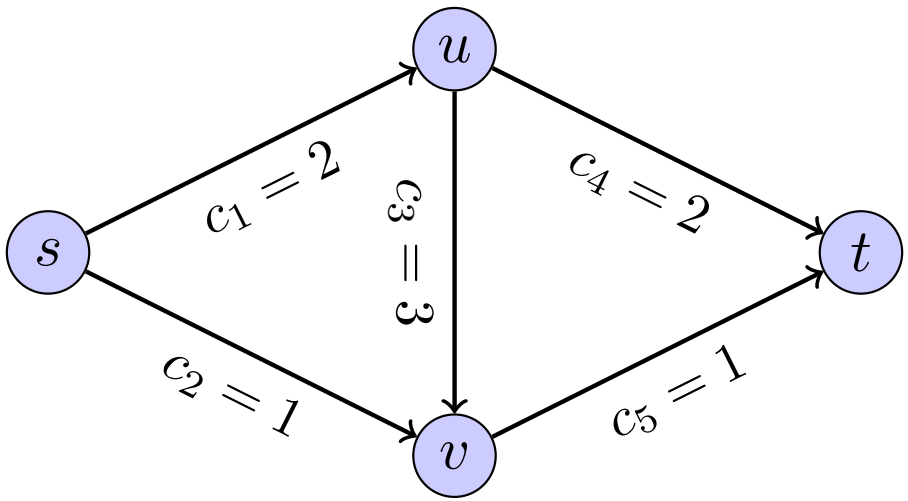
- Замечание: $|f'| = |f| + bottleneck(p, f) > |f|$ поскольку $bottleneck(p, f) > 0$.

Свойство 3: тривиальная верхняя граница потока

Lemma

$|f|$ *имеет верхнюю границу* $C = \sum_{e \text{ смежно с } s} C(e)$.

(Замечание: ребра из s полностью насыщены потоком f.)



Свойство 4: шаг увеличения

Theorem

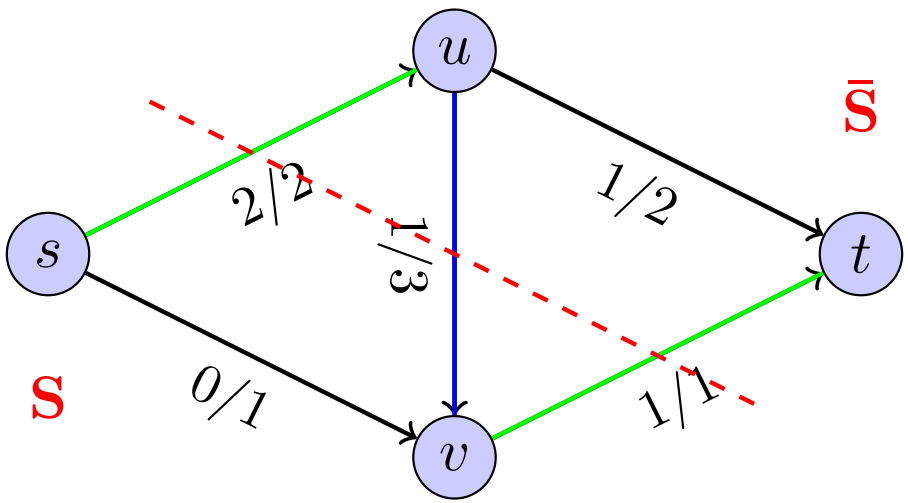
Если все ребра имеют целочисленные значения, то на каждом промежуточном этапе выполнения Алгоритма Форда—Фалкерсона *величина потока* $|f|$ *и остаточные пропускные способности являются целыми числами, а* $bottleneck(p, f) \geq 1$. *В цикле* `while` *не более* C *итераций.*

- Трудоемкость: $O(mC)$.
  - $O(C)$ итераций: при разумном предположении, что все емкости являются целыми числами, $bottleneck(p, f) \geq 1$ на каждой итерации и, таким образом, $|f'| \geq |f| + 1$.
  - На каждой итерации требуется выполнить $O(m + n)$ операций, чтобы найти путь $s - t$ в G_f, используя технику DFS или BFS.
- Обратите внимание, что оценка не является полиномиальной, так как C операций — это экспоненциально от размера входных данных задачи.

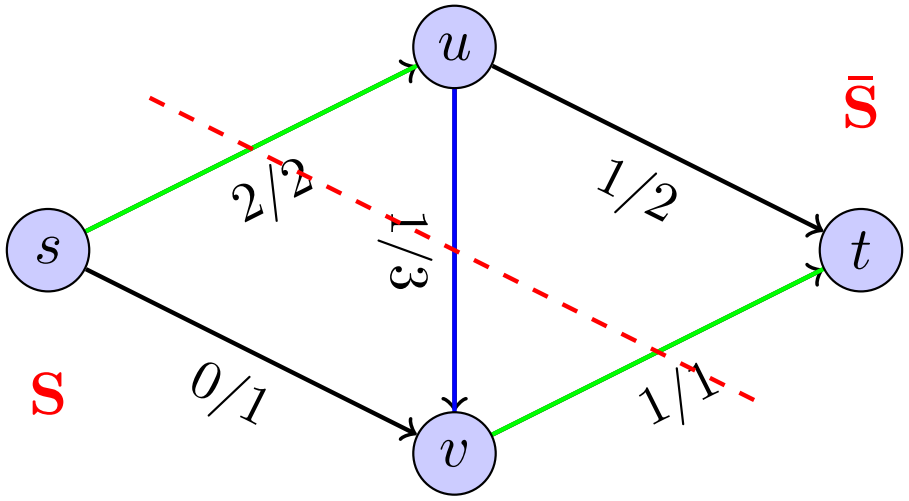
Свойство 5: более точная верхняя граница

Theorem

Для любого потока f и $s-t$ разреза $(S, \bar{S})$ выполняется $|f| \leq C(S, \bar{S})$.



$$|f| = 2 \leq C(S, \bar{S}) = 3$$

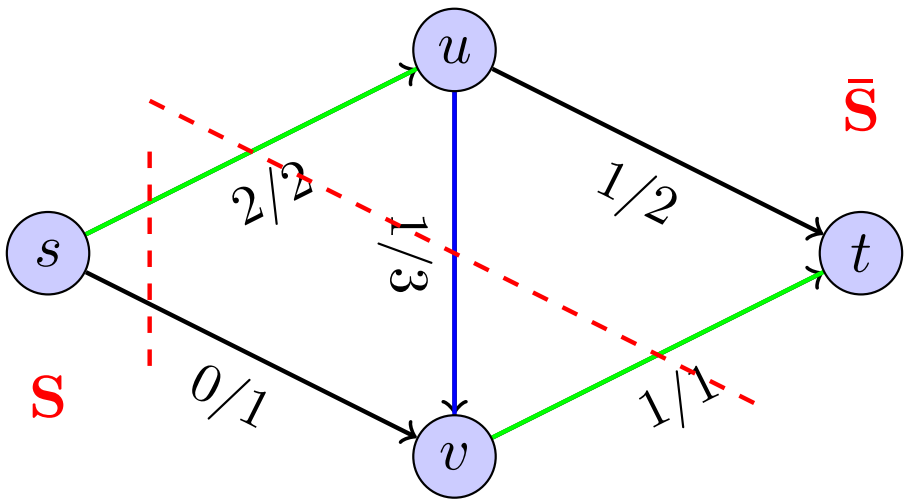


Доказательство.

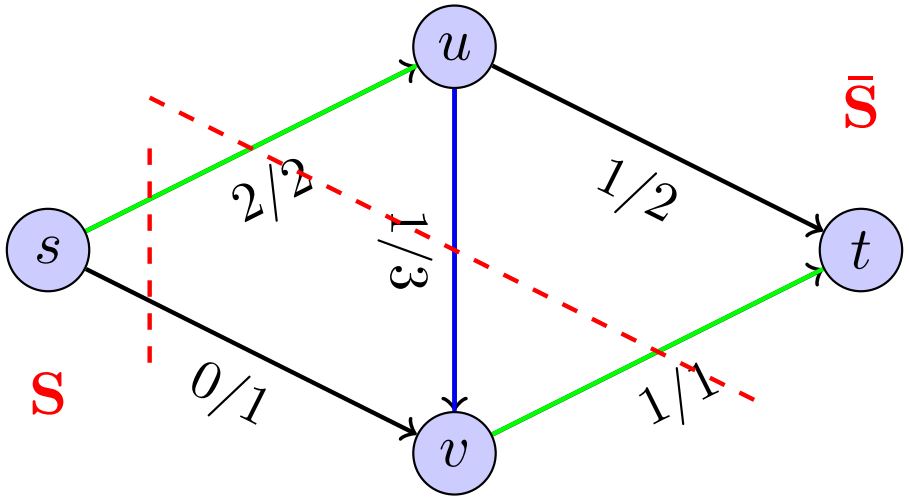
$$
\begin{aligned}
|f| &= f^{out}(S) - f^{in}(S) && \text{(по лемме о величине потока)} \\
&\leq f^{out}(S) && \text{(поскольку } f^{in}(S) \geq 0\text{)} \\
&= \sum_{e \in S \to \bar{S}} f(e) \\
&\leq \sum_{e \in S \to \bar{S}} C(e) && \text{(поскольку } f(e) \leq C(e)\text{)} \\
&= C(S, \bar{S})
\end{aligned}
$$

Lemma

Для любого $s-t$ потока f и любого $s-t$ разреза $(S, \bar{S})$ величина потока проходящего через разрез равна $|f|$. Формально, $|f| = f^{out}(S) - f^{in}(S)$.



$$|f| = 2 + 0 = 2$$
$$f^{out}(S) - f^{in}(S) = 2 + 1 - 1 = |f|$$



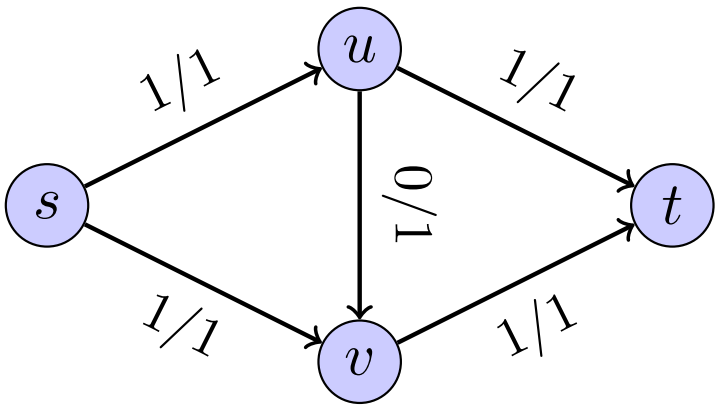
Доказательство.

- Для любого узла $v \neq s$ и $v \neq t$ выполняется $0 = f^{out}(v) - f^{in}(v)$
- Таким образом:

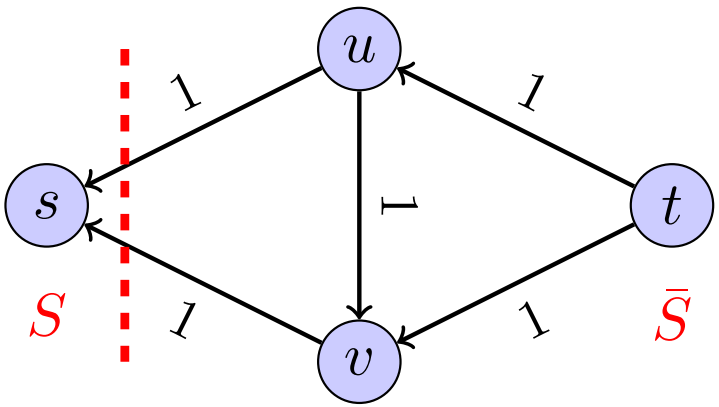
$$
\begin{aligned}
|f| &= f^{out}(s) - f^{in}(s) && // \text{ поскольку: } f^{in}(s) = 0 \\
&= \sum_{v \in S} (f^{out}(v) - f^{in}(v)) \\
&= (\sum_{e \in S \to \bar{S}} f(e) + \sum_{e \in S \to S} f(e)) \\
&\quad -(\sum_{e \in \bar{S} \to S} f(e) + \sum_{e \in S \to S} f(e)) \\
&= f^{out}(S) - f^{in}(S)
\end{aligned}
$$

Theorem

Алгоритм Форда — Фалкерсона *останавливается когда поток* f *наибольший и разрез* $(S, \bar{S})$ *— минимальный.*



Поток f



Остаточная сеть G_f

Доказательство.

- Алгоритм Форда — Фалкерсона останавливается, когда не существует $s-t$ пути в остаточной сети G_f. Пусть S — множество узлов достижимых из s в G_f, и $\bar{S} = V - S$. $(S, \bar{S})$ образует $s-t$ разрез поскольку $S \neq \phi$ и $\bar{S} \neq \phi$.

Доказательство.

- Алгоритм Форда — Фалкерсона останавливается, когда не существует $s-t$ пути в остаточной сети G_f. Пусть S — множество узлов достижимых из s в G_f, и $\bar{S} = V - S$. $(S, \bar{S})$ образует $s-t$ разрез поскольку $S \neq \phi$ и $\bar{S} \neq \phi$.
- Давайте рассмотрим два типа ребер $e = (u, v) \in E$ проходящих через разрез $(S, \bar{S})$:

Доказательство.

- Алгоритм Форда — Фалкерсона останавливается, когда не существует $s-t$ пути в остаточной сети G_f. Пусть S — множество узлов достижимых из s в G_f, и $\bar{S} = V - S$. $(S, \bar{S})$ образует $s-t$ разрез поскольку $S \neq \phi$ и $\bar{S} \neq \phi$.
- Давайте рассмотрим два типа ребер $e = (u, v) \in E$ проходящих через разрез $(S, \bar{S})$:
  1. $u \in S, v \in \bar{S}$: мы имеем $f(e) = C(e)$. (Иначе, S можно расширить, включив v, так как (u, v) находится в G_f.)

Доказательство.

- Алгоритм Форда — Фалкерсона останавливается, когда не существует $s-t$ пути в остаточной сети G_f. Пусть S — множество узлов достижимых из s в G_f, и $\bar{S} = V - S$. $(S, \bar{S})$ образует $s-t$ разрез поскольку $S \neq \phi$ и $\bar{S} \neq \phi$.
- Давайте рассмотрим два типа ребер $e = (u, v) \in E$ проходящих через разрез $(S, \bar{S})$:
  1. $u \in S, v \in \bar{S}$: мы имеем $f(e) = C(e)$. (Иначе, S можно расширить, включив v, так как (u, v) находится в G_f.)
  2. $u \in \bar{S}, v \in S$: мы имеем $f(e) = 0$. (Иначе, S можно расширить, включив u так как (v, u) находится в G_f.)

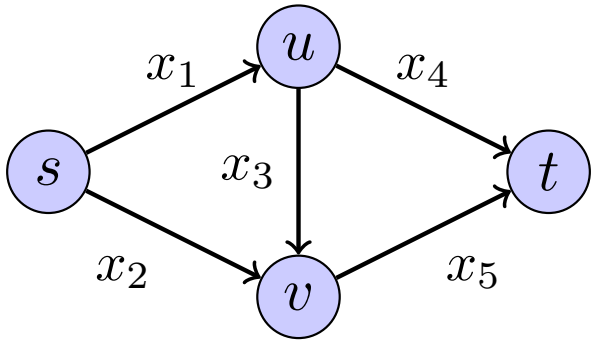
Доказательство.

- Алгоритм Форда — Фалкерсона останавливается, когда не существует $s-t$ пути в остаточной сети G_f. Пусть S — множество узлов достижимых из s в G_f, и $\bar{S} = V - S$. $(S, \bar{S})$ образует $s-t$ разрез поскольку $S \neq \phi$ и $\bar{S} \neq \phi$.
- Давайте рассмотрим два типа ребер $e = (u, v) \in E$ проходящих через разрез $(S, \bar{S})$:
  1. $u \in S, v \in \bar{S}$: мы имеем $f(e) = C(e)$. (Иначе, S можно расширить, включив v, так как (u, v) находится в G_f.)
  2. $u \in \bar{S}, v \in S$: мы имеем $f(e) = 0$. (Иначе, S можно расширить, включив u так как (v, u) находится в G_f.)
- Т.о., мы имеем

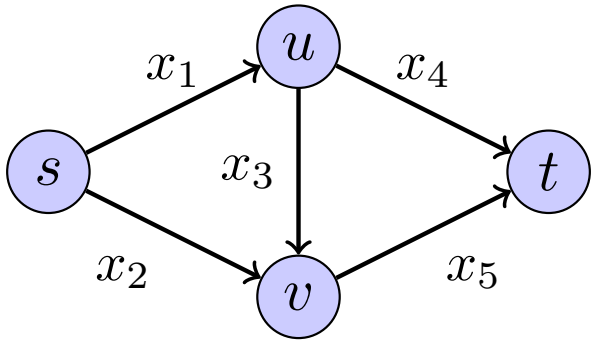
$$
\begin{aligned}
|f| &= f^{out}(S) - f^{in}(S) \\
&= f^{out}(S) && \text{(поскольку } f^{in}(S) = 0\text{)} \\
&= \sum_{e \in S \to \bar{S}} f(e) \\
&= \sum_{e \in S \to \bar{S}} C(e) && \text{(поскольку } f(e) = C(e)\text{)} \\
&= C(S, \bar{S})
\end{aligned}
$$

Рассмотрим Алгоритм Форда — Фалкерсона с точки зрения теории двойственности

Объяснение двойственности MaxFlow-MinCut: двойственные задачи

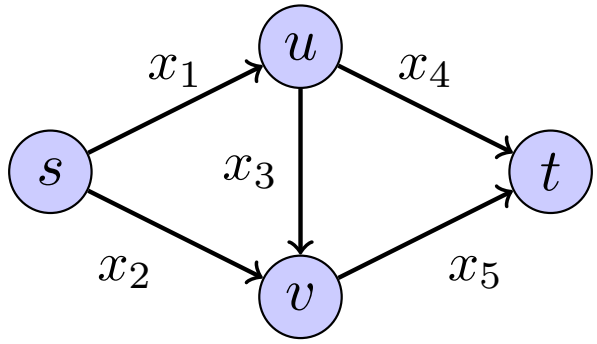


Объяснение двойственности MaxFlow-MinCut: двойственные задачи



Пусть x_i обозначает поток через ребро i.

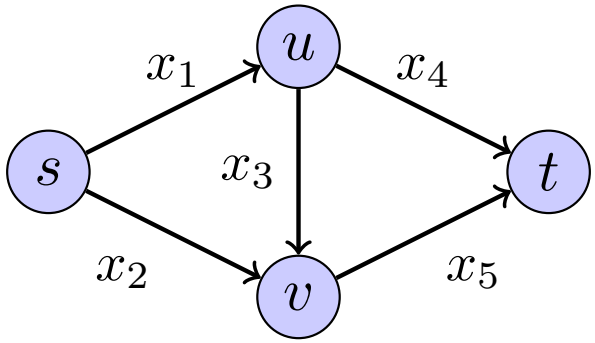
Объяснение двойственности MaxFlow-MinCut: двойственные задачи



Пусть x_i обозначает поток через ребро i.

$$
\begin{array}{lccccccl}
\max & & & & & & f & \\
s.t. & x_1 & +x_2 & & & -f & = 0 & \text{vertex } s \\
 & & & -x_4 & -x_5 & +f & = 0 & \text{vertex } t \\
 & -x_1 & & +x_3 & +x_4 & & = 0 & \text{vertex } u \\
 & & -x_2 & -x_3 & & +x_5 & = 0 & \text{vertex } v \\
 & x_1 & & & & & \leq C_1 & \\
 & & x_2 & & & & \leq C_2 & \\
 & & & x_3 & & & \leq C_3 & \\
 & & & & x_4 & & \leq C_4 & \\
 & & & & & x_5 & \leq C_5 & \\
 & x_1, & x_2, & x_3, & x_4, & x_5 & \geq 0 &
\end{array}
$$

Эквивалентная версия



$$
\begin{array}{lccccccl}
\max & & & & & & f & \\
s.t. & x_1 & +x_2 & & & & -f & \leq 0 \text{ vertex } s \\
 & & & & -x_4 & -x_5 & +f & \leq 0 \text{ vertex } t \\
 & -x_1 & & +x_3 & +x_4 & & & \leq 0 \text{ vertex } u \\
 & & -x_2 & -x_3 & & +x_5 & & \leq 0 \text{ vertex } v \\
 & x_1 & & & & & & \leq C_1 \\
 & & x_2 & & & & & \leq C_2 \\
 & & & x_3 & & & & \leq C_3 \\
 & & & & x_4 & & & \leq C_4 \\
 & & & & & x_5 & & \leq C_5 \\
 & x_1, & x_2, & x_3, & x_4, & x_5 & & \geq 0
\end{array}
$$

Отметим, что: из ограничений (1), (2), (3), и (4) следует равенство $-x_2 - x_3 + x_5 = 0$. Таким же образом получим другие равенства.

Основная задача: множество переменных z для узлов.

$$
\begin{array}{lccccccccccc}
\min & & & & & & C_1 z_1 & +C_2 z_2 & +C_3 z_3 & +C_4 z_4 & +C_5 z_5 & \\
s.t. & y_s & & -y_u & & & +z_1 & & & & & \geq 0 \\
& y_s & & & -y_v & & & +z_2 & & & & \geq 0 \\
& & & y_u & -y_v & & & & +z_3 & & & \geq 0 \\
& & -y_t & +y_u & & & & & & +z_4 & & \geq 0 \\
& & -y_t & & +y_v & & & & & & +z_5 & \geq 0 \\
& -y_s & +y_t & & & & & & & & & \geq 1 \\
& y_s, & y_t, & y_u, & y_v, & & z_1, & z_2, & z_3, & z_4, & z_5 & \geq 0
\end{array}
$$

Основная задача: множество переменных z для узлов.

$$
\begin{array}{lccccccccccc}
\min & & & & & & C_1 z_1 & +C_2 z_2 & +C_3 z_3 & +C_4 z_4 & +C_5 z_5 & \\
s.t. & y_s & & -y_u & & & +z_1 & & & & & \geq 0 \\
 & y_s & & & -y_v & & & +z_2 & & & & \geq 0 \\
 & & & y_u & -y_v & & & & +z_3 & & & \geq 0 \\
 & & -y_t & +y_u & & & & & & +z_4 & & \geq 0 \\
 & & -y_t & & +y_v & & & & & & +z_5 & \geq 0 \\
 & -y_s & +y_t & & & & & & & & & \geq 1 \\
 & y_s, & y_t, & y_u, & y_v, & & z_1, & z_2, & z_3, & z_4, & z_5 & \geq 0
\end{array}
$$

Отметим, что:

Основная задача: множество переменных z для узлов.

$$
\begin{array}{lccccccccccc}
\min & & & & & & C_1 z_1 & +C_2 z_2 & +C_3 z_3 & +C_4 z_4 & +C_5 z_5 & \\
s.t. & y_s & & -y_u & & & +z_1 & & & & & \geq 0 \\
& y_s & & & -y_v & & & +z_2 & & & & \geq 0 \\
& & & y_u & -y_v & & & & +z_3 & & & \geq 0 \\
& & -y_t & +y_u & & & & & & +z_4 & & \geq 0 \\
& & -y_t & & +y_v & & & & & & +z_5 & \geq 0 \\
& -y_s & +y_t & & & & & & & & & \geq 1 \\
& y_s, & y_t, & y_u, & y_v, & & z_1, & z_2, & z_3, & z_4, & z_5 & \geq 0
\end{array}
$$

Отметим, что:

1. Поскольку ограничения связаны с разницей между y_s, y_u, y_v и y_t, одно из них можно изменить без последствий.
   Зафиксируем $y_s = 0$. Получим $y_t \geq 1$ (из ограничения (6).

Основная задача: множество переменных z для узлов.

$$
\begin{array}{lccccccccccc}
\min & & & & & & C_1 z_1 & +C_2 z_2 & +C_3 z_3 & +C_4 z_4 & +C_5 z_5 & \\
s.t. & y_s & & -y_u & & & +z_1 & & & & & \geq 0 \\
& y_s & & & -y_v & & & +z_2 & & & & \geq 0 \\
& & & y_u & -y_v & & & & +z_3 & & & \geq 0 \\
& & -y_t & +y_u & & & & & & +z_4 & & \geq 0 \\
& & -y_t & & +y_v & & & & & & +z_5 & \geq 0 \\
& -y_s & +y_t & & & & & & & & & \geq 1 \\
& y_s, & y_t, & y_u, & y_v, & & z_1, & z_2, & z_3, & z_4, & z_5 & \geq 0
\end{array}
$$

Отметим, что:

1. Поскольку ограничения связаны с разницей между y_s, y_u, y_v и y_t, одно из них можно изменить без последствий. Зафиксируем $y_s = 0$. Получим $y_t \geq 1$ (из ограничения (6).
2. Ограничение (4) требует $z_4 \geq y_t - y_u$, и цель состоит в том, чтобы минимизировать функцию, содержащую $C_4 z_4$, потребуем $y_t = 1$.

Основная задача: множество переменных z для узлов.

$$
\begin{array}{lccccccccccc}
\min & & & & & & C_1 z_1 & +C_2 z_2 & +C_3 z_3 & +C_4 z_4 & +C_5 z_5 & \\
s.t. & y_s & & -y_u & & & +z_1 & & & & & \geq 0 \\
& y_s & & & -y_v & & & +z_2 & & & & \geq 0 \\
& & & y_u & -y_v & & & & +z_3 & & & \geq 0 \\
& & -y_t & +y_u & & & & & & +z_4 & & \geq 0 \\
& & -y_t & & +y_v & & & & & & +z_5 & \geq 0 \\
& -y_s & +y_t & & & & & & & & & \geq 1 \\
& y_s, & y_t, & y_u, & y_v, & & z_1, & z_2, & z_3, & z_4, & z_5 & \geq 0
\end{array}
$$

Отметим, что:

1. Поскольку ограничения связаны с разницей между y_s, y_u, y_v и y_t, одно из них можно изменить без последствий. Зафиксируем $y_s = 0$. Получим $y_t \geq 1$ (из ограничения (6).
2. Ограничение (4) требует $z_4 \geq y_t - y_u$, и цель состоит в том, чтобы минимизировать функцию, содержащую $C_4 z_4$, потребуем $y_t = 1$.
3. Ограничение (1) требует $z_1 \geq y_u$, и цель состоит в том, чтобы минимизировать функцию, содержащую $C_1 z_1$, потребуем $z_1 = y_u$. Так же поступим с ограничением (2).

Эквивалентная LP модель

Прямая: множество переменных для узлов.

$$
\begin{array}{lccccccccccc}
\min & & & & & C_1 z_1 & +C_2 z_2 & +C_3 z_3 & +C_4 z_4 & +C_5 z_5 & & \\
s.t. & & -y_u & & & +z_1 & & & & & & = 0 \\
& & & -y_v & & & +z_2 & & & & & = 0 \\
& & y_u & -y_v & & & & +z_3 & & & & \geq 0 \\
& & y_u & & & & & & +z_4 & & & \geq 1 \\
& & & y_v & & & & & & +z_5 & & \geq 1 \\
& y_s & & & & & & & & & & = 0 \\
& & y_t & & & & & & & & & = 1 \\
& & & y_u, & y_v, & z_1, & z_2, & z_3, & z_4, & z_5 & & \geq 0
\end{array}
$$

Отметим, что: коэффициенты ограничений (3), (4) и (5) образуют вполне унимодулярную матрицу, поэтому оптимальное решение является целочисленным.

Эквивалентная ILP модель

Прямая: множество переменных для узлов.

$$
\begin{array}{lrrrrrrrrrl}
\min & & & & C_1 z_1 & +C_2 z_2 & +C_3 z_3 & +C_4 z_4 & +C_5 z_5 & \\
s.t. & & -y_u & & +z_1 & & & & & = 0 \\
 & & & -y_v & & +z_2 & & & & = 0 \\
 & & y_u & -y_v & & & +z_3 & & & \geq 0 \\
 & & y_u & & & & & +z_4 & & \geq 1 \\
 & & & y_v & & & & & +z_5 & \geq 1 \\
 & y_s & & & & & & & & = 0 \\
 & & y_t & & & & & & & = 1 \\
 & & y_u, & y_v, & z_1, & z_2, & z_3, & z_4, & z_5 & = 0/1
\end{array}
$$

MaxFlow-MinCut: двойственность

$$
\begin{array}{lccccccccccc}
\min & & & & & C_1 z_1 & +C_2 z_2 & +C_3 z_3 & +C_4 z_4 & +C_5 z_5 & & \\
s.t. & & -y_u & & & +z_1 & & & & & & = 0 \\
& & & -y_v & & & +z_2 & & & & & = 0 \\
& & y_u & -y_v & & & & +z_3 & & & & \geq 0 \\
& & y_u & & & & & & +z_4 & & & \geq 1 \\
& & & y_v & & & & & & +z_5 & & \geq 1 \\
& y_s & & & & & & & & & & = 0 \\
& & y_t & & & & & & & & & = 1 \\
& & & y_u, & y_v, & z_1, & z_2, & z_3, & z_4, & z_5 & & = 0/1
\end{array}
$$

- Предположим, мы объясняем первичные переменные как:
  - y_i представляет находится ли узел i в S или $\bar{S}$: если узел i в S, то $y_i = 0$, и $y_i = 1$ иначе.
  - z_i представляет находится ли ребро в разрезе: например, $z_1 = 1$ если $y_s = 0$ и $y_u = 1$, т.е., ребро (s, u) разрезается.
- В основной задаче нужно найти минимальный разрез.
- Слабая двойственность дает $f \leq c$, а сильная двойственность эквивалентна теореме о максимальном потоке и минимальном разрезе.

Алгоритм Форда — Фалкерсона является прямо-двойственным алгоритмом

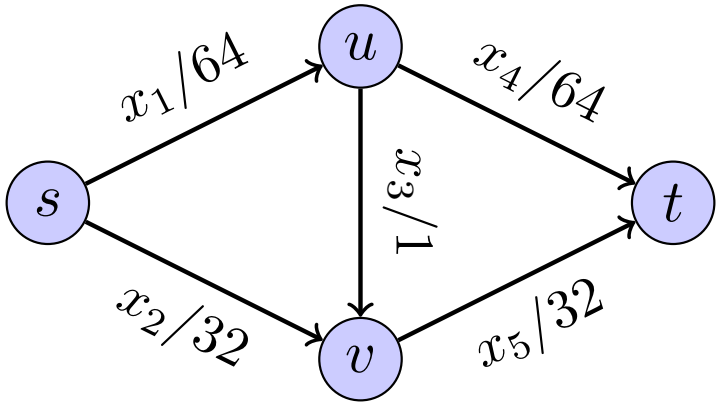
Прямо-двойственный алгоритм

- Напомним, что общий прямо-двойственный алгоритм можно описать следующим образом.

```
1: Инициализируем x как двойственное допустимое
   решение;
2: while TRUE do
3:    Построить DRP соответствующее x;
4:    Пусть ω_opt — оптимальное решение DRP;
5:    if ω_opt = 0 then
6:       return x;
7:    else
8:       Исправить x в соответствии с оптимальным
         решением DRP;
9:    end if
10: end while
```

- Мы покажем, что решение DRP эквивалентно нахождению увеличивающего пути в остаточной сети.

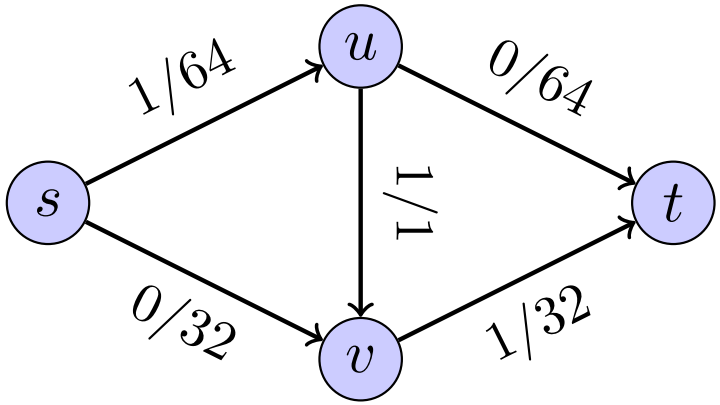
Двойственная задача и DRP I



- Двойственная D: множество переменных для ребер;

$$
\begin{array}{lccccccl}
\max & & & & & & f & \\
s.t. & x_1 & +x_2 & & & & -f & \leq 0 \text{ vertex } s \\
 & & & & -x_4 & -x_5 & +f & \leq 0 \text{ vertex } t \\
 & -x_1 & & +x_3 & +x_4 & & & \leq 0 \text{ vertex } u \\
 & & -x_2 & -x_3 & & +x_5 & & \leq 0 \text{ vertex } v \\
 & x_1 & & & & & & \leq 64 \\
 & & x_2 & & & & & \leq 32 \\
 & & & x_3 & & & & \leq 1 \\
 & & & & x_4 & & & \leq 64 \\
 & & & & & x_5 & & \leq 32 \\
 & x_1, & x_2, & x_3, & x_4, & x_5 & & \geq 0
\end{array}
$$

Двойственная задача и DRP II



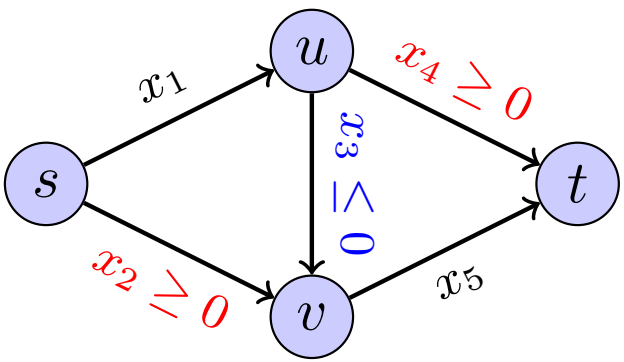
- Давайте рассмотрим двойное допустимое решение $\mathbf{x} = (1, 0, 1, 0, 1)$. Напомним, как формулируется DRP из D:
  - Замена правой стороны C_i на 0;
  - Добовляются ограничения: $x_i \leq 1$, $f \leq 1$;
  - Сохраняем только жесткие ограничения J. Делим J на два множества, т.е. $J = J^S \cup J^E$, где J^S содержит насыщенные ребра $J^S = \{i | x_i = C_i\}$, и J^E содержит пустые ребра $J^E = \{i | x_i = 0\}$. В приведенном примере, $J_S = \{3\}$, and $J_E = \{2, 4\}$.

DRP соответствует нахождению увеличивающего пути

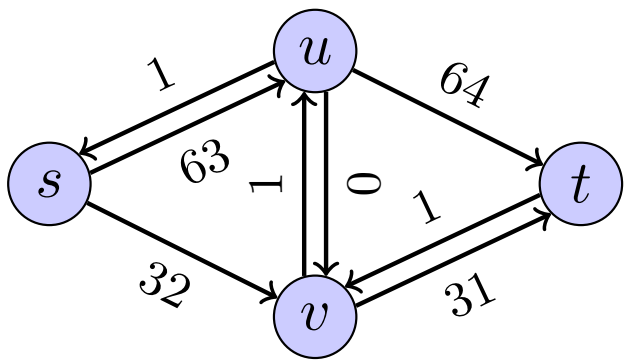
- DRP:

$$
\begin{array}{lrrrrrrl}
\max & & & & & & f & \\
s.t. & x_1 & +x_2 & & & & -f & = 0 \text{ vertex } s \\
 & & & & -x_4 & -x_5 & +f & = 0 \text{ vertex } t \\
 & -x_1 & & +x_3 & +x_4 & & & = 0 \text{ vertex } u \\
 & & -x_2 & -x_3 & & +x_5 & & = 0 \text{ vertex } v \\
 & & & x_i & & & & \leq 0 \quad i \in J^S \\
 & & & x_j & & & & \geq 0 \quad j \in J^E \\
 & x_1, & x_2, & x_3, & x_4, & x_5, & f & \leq 1
\end{array}
$$

- $\omega_{OPT} = 0$ подразумевает, что найдено оптимальное решение. Наоборот, $\omega_{OPT} = 1$ подразумевает, что найден увеличивающий $s - t$ путь (с единичным потоком) в G_f.

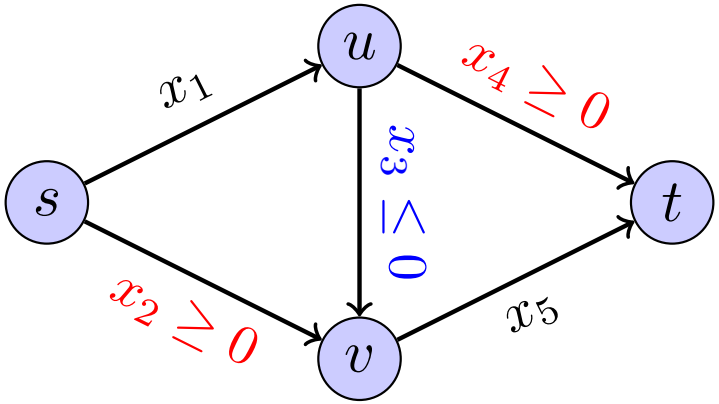


DRP

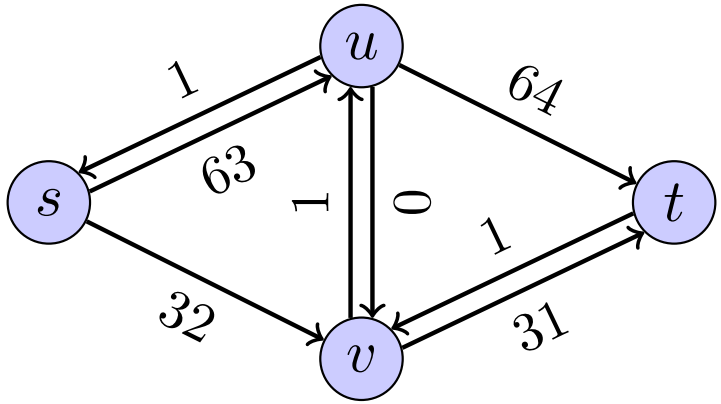


Остаточная сеть G_f

DRP и увеличивающий путь в остаточной сети



DRP



Остаточная сеть G_f

- Обратите внимание, что DRP соответствует поиску увеличивающего пути в остаточной сети G_f.
  - $x_i \leq 0, i \in J^S$, например, x_3, обозначает обратное ребро.
  - $x_j \geq 0, j \in J^E$, например, x_2, обозначает прямое ребро,
  - и для других ребер, не существует ограничений x_i, например, x_1.
- Т.о., Алгоритм Форда — Фалкерсона является по существу прямо-двойственным алгоритмом.

Плохой пример для алгоритма Ford-Fulkerson

Важность целочисленных ограничений

- В анализе Алгоритм Форда — Фалкерсона, целочисленность ограничений важно: узкое место даст увеличение по меньшей мере на 1.

Важность целочисленных ограничений

- В анализе Алгоритм Форда — Фалкерсона, целочисленность ограничений важно: узкое место даст увеличение по меньшей мере на 1.
- Анализ не корректен, если пропускные способности могут быть нерациональными.

Важность целочисленных ограничений

- В анализе Алгоритм Форда — Фалкерсона, целочисленность ограничений важно: узкое место даст увеличение по меньшей мере на 1.
- Анализ не корректен, если пропускные способности могут быть нерациональными.

  На самом деле поток может увеличиваться на все меньшее и меньшее число, и итераций будет бесконечно много.

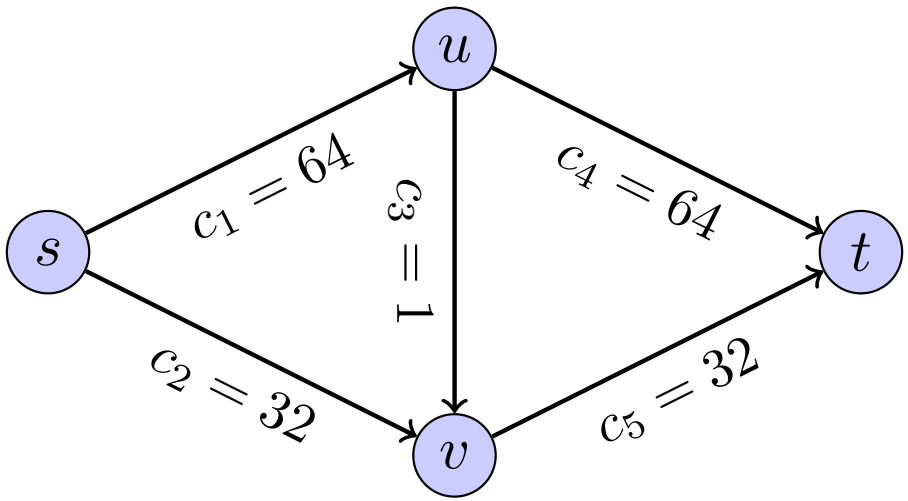
Важность целочисленных ограничений

- В анализе Алгоритм Форда — Фалкерсона, целочисленность ограничений важно: узкое место даст увеличение по меньшей мере на 1.
- Анализ не корректен, если пропускные способности могут быть нерациональными.

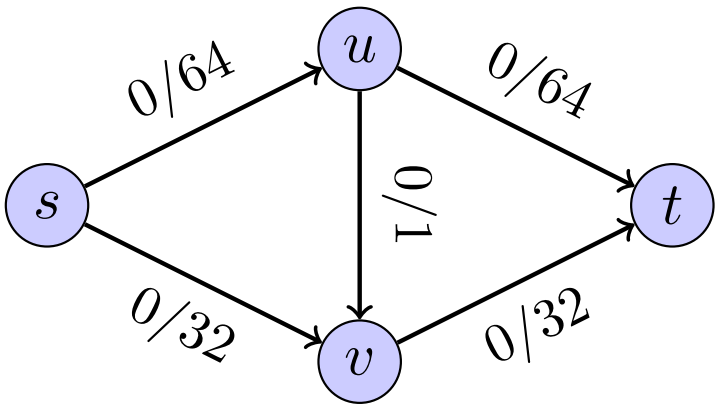
  На самом деле поток может увеличиваться на все меньшее и меньшее число, и итераций будет бесконечно много.

  Хуже того, эти бесконечные итерации могут не сходиться к максимальному потоку.

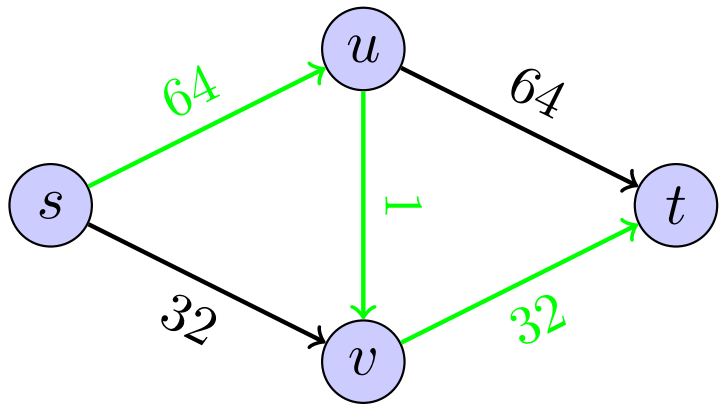
Алгоритм Форда — Фалкерсона: плохой пример 2



Плохой пример для алгоритма Форда — Фалкерсона: Шаг 1

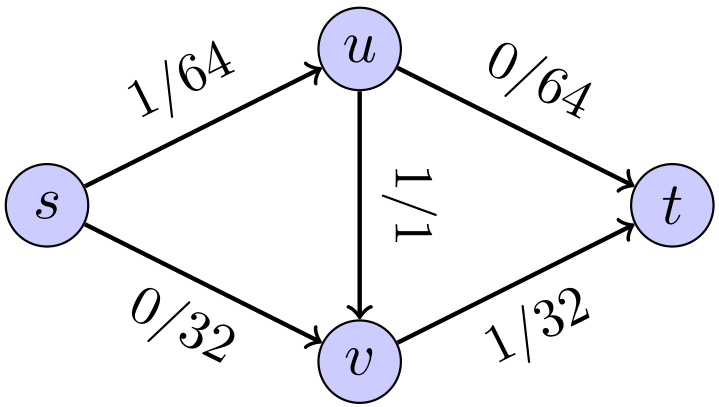


Поток $f : |f| = 0$

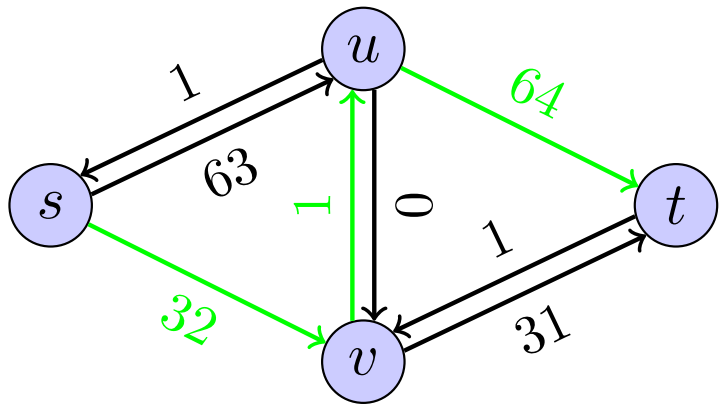


$s - t$ путь в G_f

Плохой пример для алгоритма Форда — Фалкерсона: Шаг 2

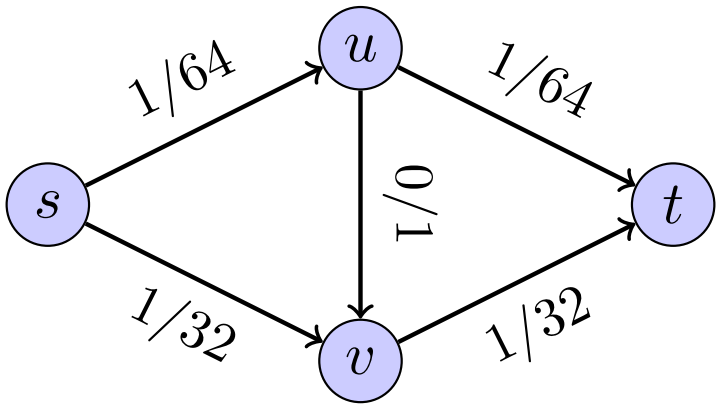


Поток $f : |f| = 1$



$s - t$ путь в G_f

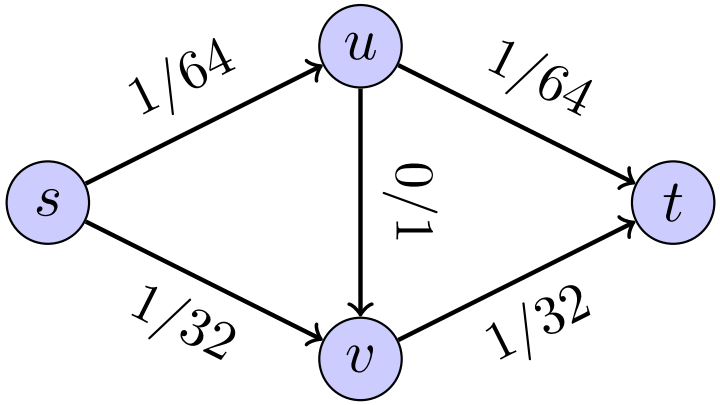
Плохой пример для алгоритма Форда — Фалкерсона: Шаг 3



Поток $f : |f| = 2$

- Обратите внимание, что после двух итераций эта проблема аналогична исходной, за исключением того, что пропусктные способности на $(s, u), (s, v), (u, t), (v, t)$ уменьшаются на 1.

Плохой пример для алгоритма Форда — Фалкерсона: Шаг 3



Поток $f : |f| = 2$

- Обратите внимание, что после двух итераций эта проблема аналогична исходной, за исключением того, что пропусктные способности на $(s, u), (s, v), (u, t), (v, t)$ уменьшаются на 1.
- Т.о., алгоритм Форда — Фалкерсона остановится после выполнения $64 + 32$ итераций, поскольку $bottleneck = 1$ на всех итерациях.

Слабость Алгоритма Форда — Фалкерсона

- Произвольный выбор увеличивающих путей приводит к следующим слабостям:

Слабость Алгоритма Форда — Фалкерсона

- Произвольный выбор увеличивающих путей приводит к следующим слабостям:
  - Путь с небольшой пропускной способностью выбирается как увеличивающий путь

Слабость Алгоритма Форда — Фалкерсона

- Произвольный выбор увеличивающих путей приводит к следующим слабостям:
  - Путь с небольшой пропускной способностью выбирается как увеличивающий путь
  - Мы увеличиваем поток за много большее число шагов, чем в лудшем случае.

Слабость Алгоритма Форда — Фалкерсона

- Произвольный выбор увеличивающих путей приводит к следующим слабостям:
  - Путь с небольшой пропускной способностью выбирается как увеличивающий путь
  - Мы увеличиваем поток за много большее число шагов, чем в лудшем случае.
- В оригинальной работе Форда и Фалкерсона было рассмотрено несколько эвристик для улучшения.

- Различные стратегии выбора увеличивающего пути в G_f:

- Различные стратегии выбора увеличивающего пути в G_f:
  1. Толстые трубы:

- Различные стратегии выбора увеличивающего пути в G_f:
  1. Толстые трубы:
     - Выбрать увеличивающий путь с **наибольшей пропускной способность** или использовать **масштабирование**.

- Различные стратегии выбора увеличивающего пути в G_f:
  1. Толстые трубы:
     - Выбрать увеличивающий путь с **наибольшей пропускной способность** или использовать **масштабирование**.
  2. Короткие трубы:

- Различные стратегии выбора увеличивающего пути в G_f:
  1. Толстые трубы:
     - Выбрать увеличивающий путь с **наибольшей пропускной способность** или использовать **масштабирование**.
  2. Короткие трубы:
     - Алгоритм Эдмонса-Карпа находит **кратчайший увеличивающий путь**.

- Различные стратегии выбора увеличивающего пути в G_f:
  1. Толстые трубы:
     - Выбрать увеличивающий путь с **наибольшей пропускной способность** или использовать **масштабирование**.
  2. Короткие трубы:
     - Алгоритм Эдмонса-Карпа находит **кратчайший увеличивающий путь**.
     - Алгоритм Диница: расширяет **BFS дерево** чтобы построить **уровневую сеть** находит увеличение потока в уровневой сети. Для оценки выполняется **аматризационный анализ**.

- Различные стратегии выбора увеличивающего пути в G_f:
  1. Толстые трубы:
     - Выбрать увеличивающий путь с **наибольшей пропускной способность** или использовать **масштабирование**.
  2. Короткие трубы:
     - Алгоритм Эдмонса-Карпа находит **кратчайший увеличивающий путь**.
     - Алгоритм Диница: расширяет **BFS дерево** чтобы построить **уровневую сеть** находит увеличение потока в уровневой сети. Для оценки выполняется **аматризационный анализ**.
     - Алгоритм Диница: выполняет **DFS** в **уровневой сети** чтобы найти **блокирующий поток**, который насыщает **все кратчайшие увеличивающие пути**.

- Различные стратегии выбора увеличивающего пути в G_f:
  1. Толстые трубы:
     - Выбрать увеличивающий путь с **наибольшей пропускной способность** или использовать **масштабирование**.
  2. Короткие трубы:
     - Алгоритм Эдмонса-Карпа находит **кратчайший увеличивающий путь**.
     - Алгоритм Диница: расширяет **BFS дерево** чтобы построить **уровневую сеть** находит увеличение потока в уровневой сети. Для оценки выполняется **аматризационный анализ**.
     - Алгоритм Диница: выполняет **DFS** в **уровневой сети** чтобы найти **блокирующий поток**, который насыщает **все кратчайшие увеличивающие пути**.
     - Алгоритм Карзанова: **насыщает ребра** когда строит блокирующий поток. Используется идея **предпотока**.

- Различные стратегии выбора увеличивающего пути в G_f:
  1. Толстые трубы:
     - Выбрать увеличивающий путь с **наибольшей пропускной способность** или использовать **масштабирование**.
  2. Короткие трубы:
     - Алгоритм Эдмонса-Карпа находит **кратчайший увеличивающий путь**.
     - Алгоритм Диница: расширяет **BFS дерево** чтобы построить **уровневую сеть** находит увеличение потока в уровневой сети. Для оценки выполняется **аматризационный анализ**.
     - Алгоритм Диница: выполняет **DFS** в **уровневой сети** чтобы найти **блокирующий поток**, который насыщает **все кратчайшие увеличивающие пути**.
     - Алгоритм Карзанова: **насыщает ребра** когда строит блокирующий поток. Используется идея **предпотока**.
     - Алгоритм «поднять в начало»: использует идею предпотока; однако, предпоток строится не в **уровневой сети**, а в остаточной сети. Используются **метки расстояний чтобы оценить расстояние от узла до** t.

Улучшение 1: Техника масштабирования (Диниц)

Техника масштабирования

- Вопрос: можем ли мы выбрать увеличивающий путь с **большой пропускной способностью**? Если $bottleneck(p, f)$ большое, то, возможно, потребуется меньше итераций.

Техника масштабирования

- Вопрос: можем ли мы выбрать увеличивающий путь с **большой пропускной способностью**? Если $bottleneck(p, f)$ большое, то, возможно, потребуется меньше итераций.
- $s - t$ путь p в G_f с **наибольшим** $bottleneck(p, f)$ можно найти, используя бинарный поиск, или немного измененным алгоритмом Дейкстры, за время $O(m + n \log n)$; однако, это все еще не совсем эффективно.

Техника масштабирования

- Вопрос: можем ли мы выбрать увеличивающий путь с **большой пропускной способностью**? Если $bottleneck(p, f)$ большое, то, возможно, потребуется меньше итераций.
- $s - t$ путь p в G_f с **наибольшим** $bottleneck(p, f)$ можно найти, используя бинарный поиск, или немного измененным алгоритмом Дейкстры, за время $O(m + n \log n)$; однако, это все еще не совсем эффективно.
- Идея: ослабить требование **“наибольшей”** на **“достаточно большой”**. В частности, мы можем установить нижнюю границу Δ для $bottleneck(P, f)$ посредством **удаления “малых” ребер**, т.е. ребра с пропускной способностью меньше чем Δ удаляются из $G(f)$. Такая остаточная сеть обозначается как $G_f(\Delta)$ и Δ будет уменьшаться по мере продолжения итераций.

- Масштабирование-Форда - Фалкерсона(G)

1: Инициализация: $f(e) = 0$ для всех e.
2: Пусть $\Delta = C$;
3: **while** $\Delta \geq 1$ **do**
4: **while** существует $s - t$ путь в $G_f(\Delta)$ **do**
5: Выбрать $s - t$ путь p;
6: $f =$AUGMENT(p, f);
7: **end while**
8: $\Delta = \frac{\Delta}{2}$;
9: **end while**
10: **return** f;

- Масштабирование-Форда - Фалкерсона(G)

1: Инициализация: $f(e) = 0$ для всех e.
2: Пусть $\Delta = C$;
3: **while** $\Delta \geq 1$ **do**
4: **while** существует $s - t$ путь в $G_f(\Delta)$ **do**
5: Выбрать $s - t$ путь p;
6: $f =$AUGMENT(p, f);
7: **end while**
8: $\Delta = \frac{\Delta}{2}$;
9: **end while**
10: **return** f;

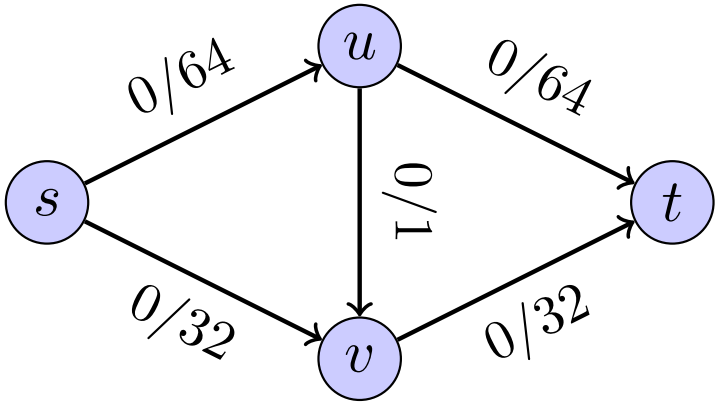
- Замечание: поток увеличивается с большим шагом по мере возможности; в противном случае размер шага уменьшается. Размер шага контролируется путем удаления «маленьких» ребер из остаточной сети.

- Масштабирование-Форда - Фалкерсона(G)

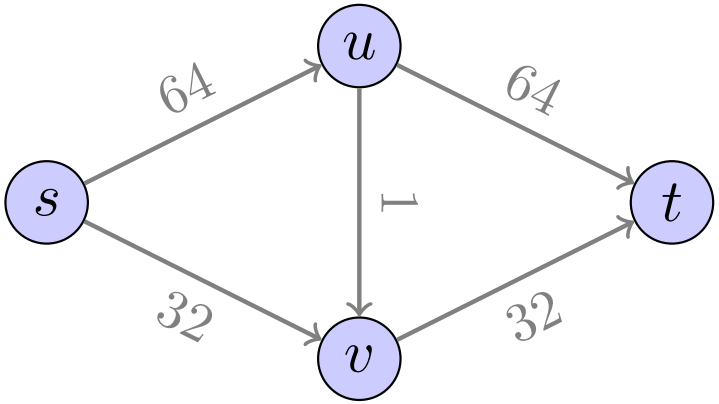
```
1: Инициализация: f(e) = 0 для всех e.
2: Пусть Δ = C;
3: while Δ ≥ 1 do
4:     while существует s − t путь в G_f(Δ) do
5:         Выбрать s − t путь p;
6:         f = AUGMENT(p, f);
7:     end while
8:     Δ = Δ/2;
9: end while
10: return f;
```

- Замечание: поток увеличивается с большим шагом по мере возможности; в противном случае размер шага уменьшается. Размер шага контролируется путем удаления «маленьких» ребер из остаточной сети.
- Отметим, что Δ окончательно станет равной 1; таким образом, никакие ребра в остаточном графе не будут игнорироваться.

Пример: Шаг 1



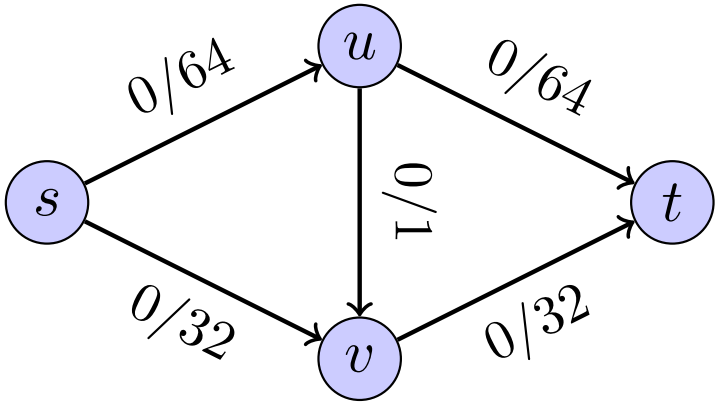
Поток $f : |f| = 0$



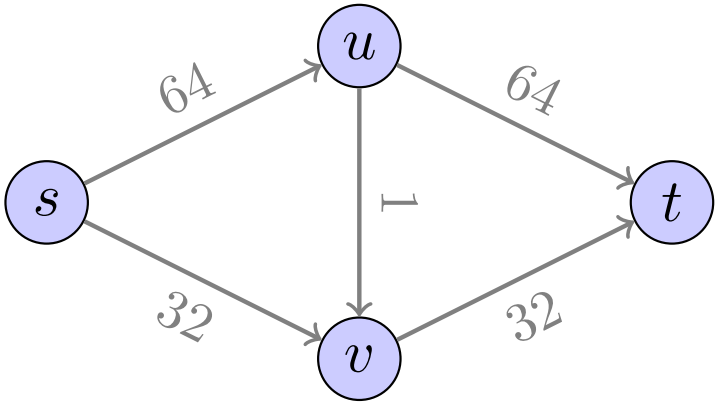
Нет $s - t$ путей в G_f

- Поток: нулевой поток;

Пример: Шаг 1



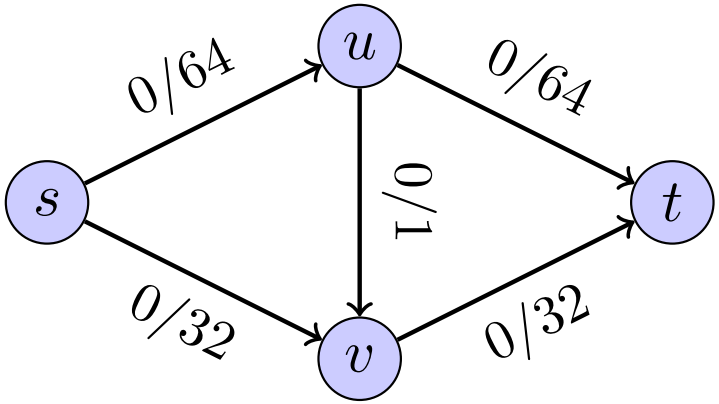
Поток $f : |f| = 0$



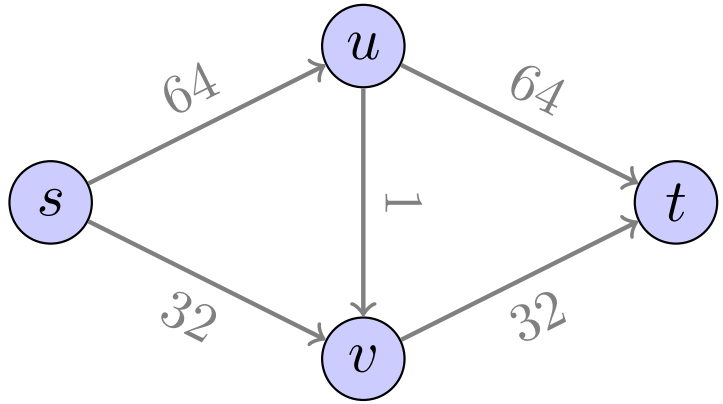
Нет $s - t$ путей в G_f

- Поток: нулевой поток;
- Δ: $\Delta = 96$;

Пример: Шаг 1



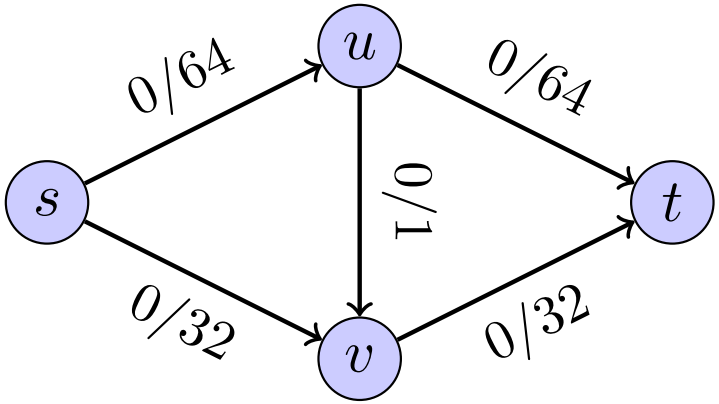
Поток $f : |f| = 0$



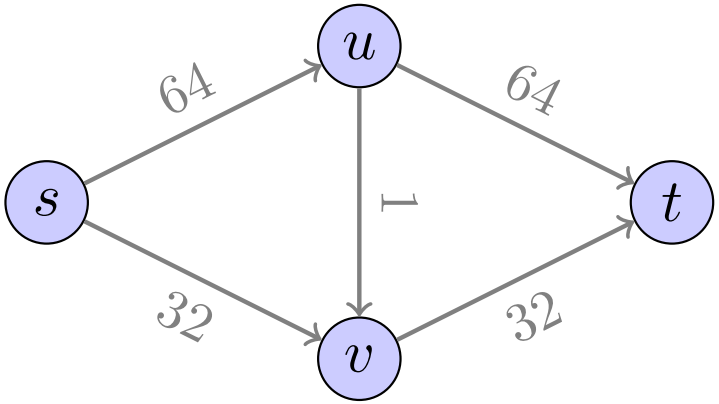
Нет $s - t$ путей в G_f

- Поток: нулевой поток;
- Δ: $\Delta = 96$;
- $G_f(\Delta)$: все ребра удаляются, поскольку их пропускные способности меньше 96.

Пример: Шаг 1



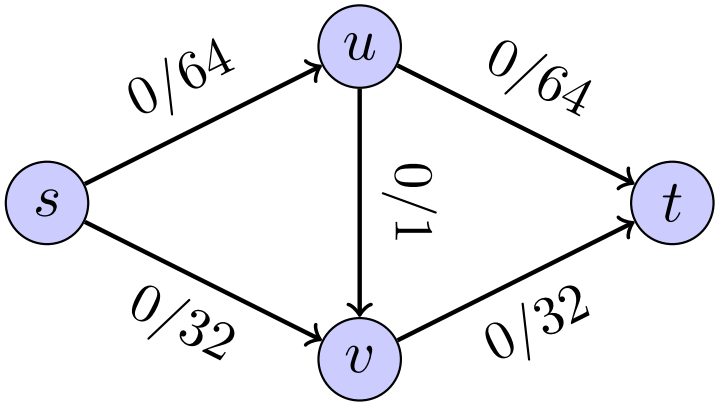
Поток $f : |f| = 0$



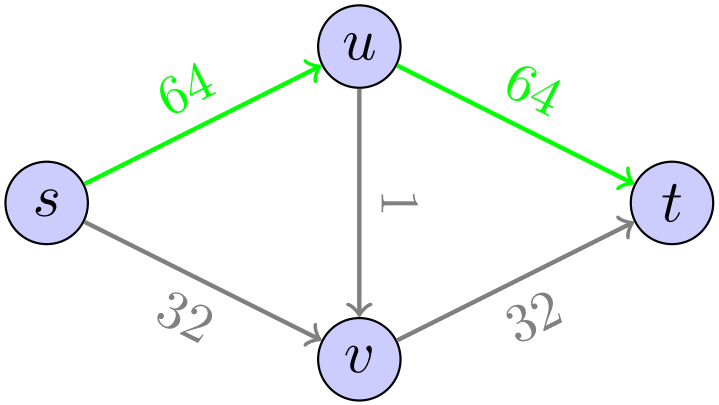
Нет $s - t$ путей в G_f

- Поток: нулевой поток;
- Δ: $\Delta = 96$;
- $G_f(\Delta)$: все ребра удаляются, поскольку их пропускные способности меньше 96.
- $s - t$ пути нет. Следовательно Δ масштабируется: $\Delta = \frac{\Delta}{2} = 48$.

Пример: Шаг 2



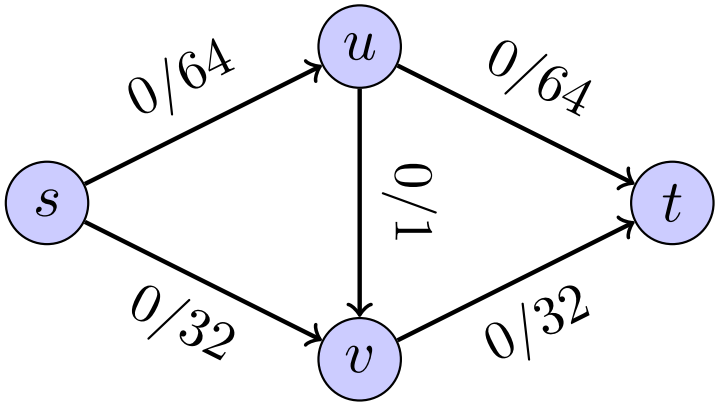
Поток $f : |f| = 0$



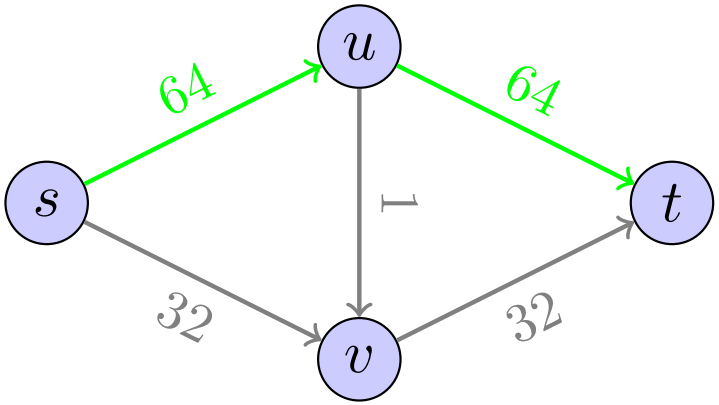
$s - t$ путь в G_f

- Поток: нулевой поток;

Пример: Шаг 2



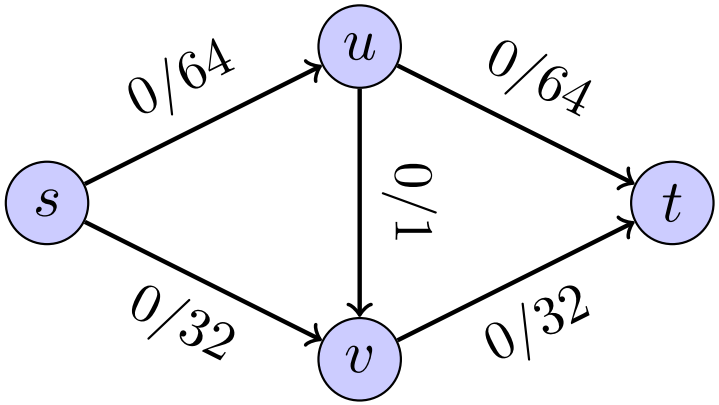
Поток $f : |f| = 0$



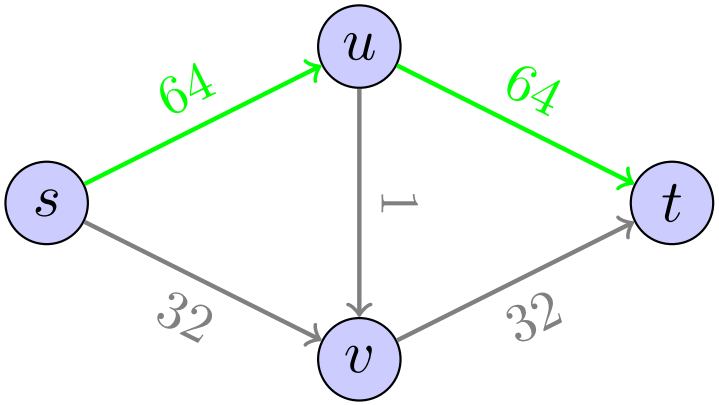
$s - t$ путь в G_f

- Поток: нулевой поток;
- Δ: $\Delta = 48$;

Пример: Шаг 2



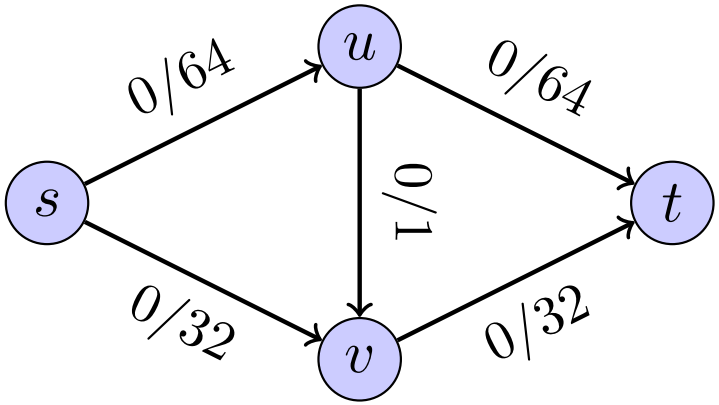
Поток $f : |f| = 0$



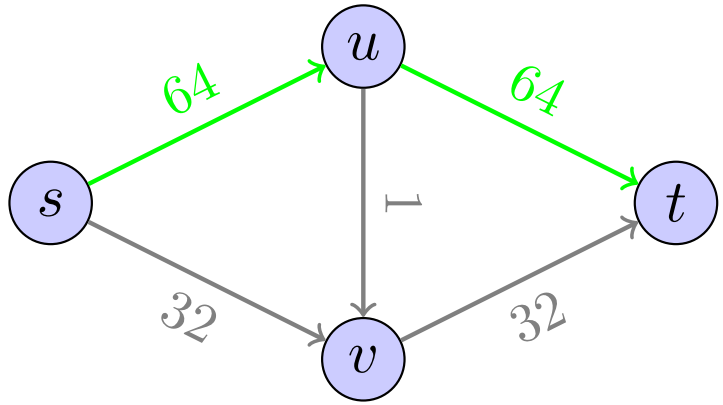
$s - t$ путь в G_f

- Поток: нулевой поток;
- Δ: $\Delta = 48$;
- $G_f(\Delta)$: три ребра удаляются, поскольку их пропускные способности меньше 48.

Пример: Шаг 2



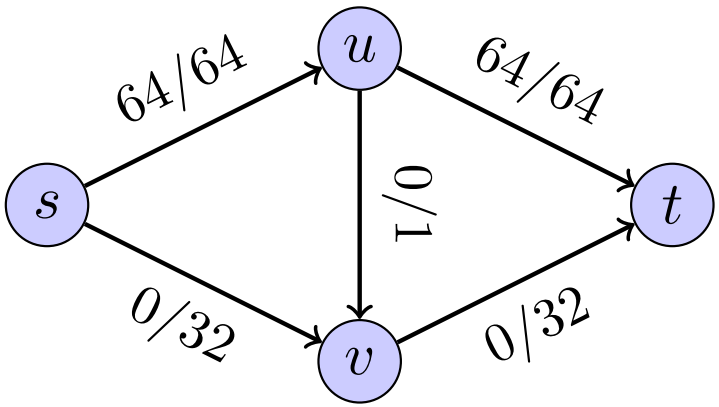
Поток $f : |f| = 0$



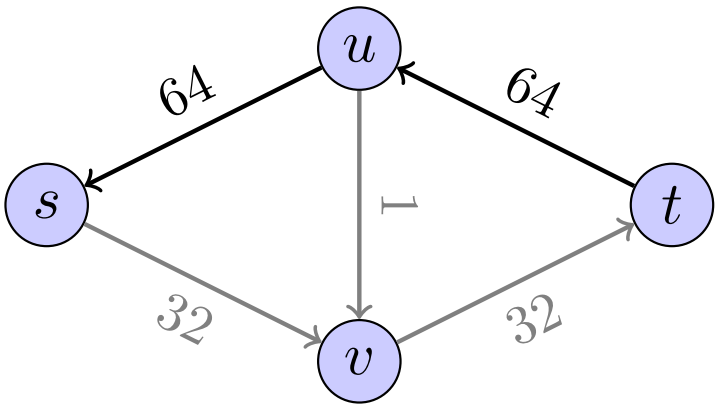
$s - t$ путь в G_f

- Поток: нулевой поток;
- Δ: $\Delta = 48$;
- $G_f(\Delta)$: три ребра удаляются, поскольку их пропускные способности меньше 48.
- $s - t$ путь: есть путь $s - u - t$. Выполняется операция увеличения потока.

Пример: Шаг 3



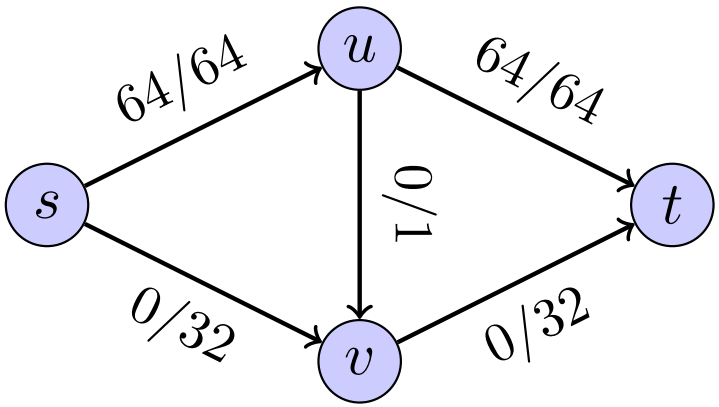
Поток $f : |f| = 64$



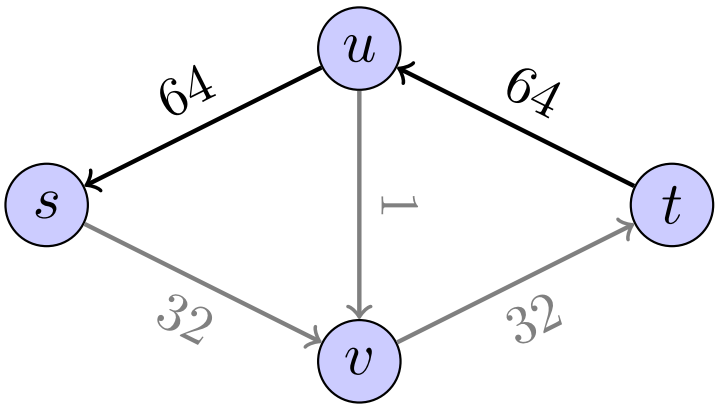
$s - t$ пути в G_f нет

- Поток: 64;

Пример: Шаг 3



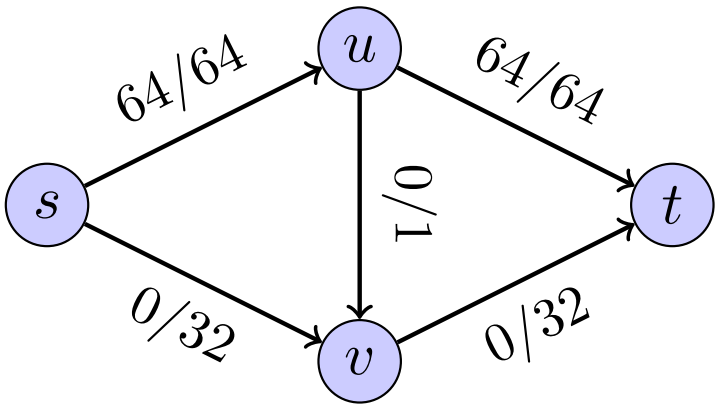
Поток $f : |f| = 64$



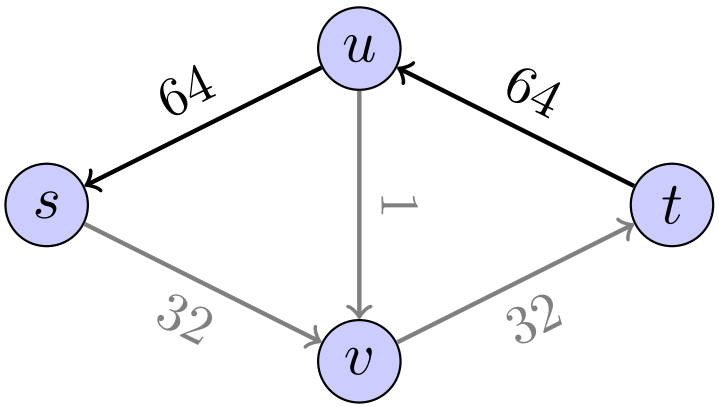
$s - t$ пути в G_f нет

- Поток: 64;
- Δ: $\Delta = 48$;

Пример: Шаг 3



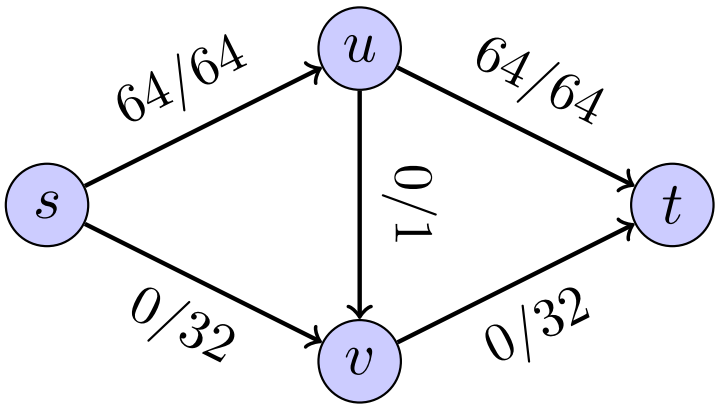
Поток $f : |f| = 64$



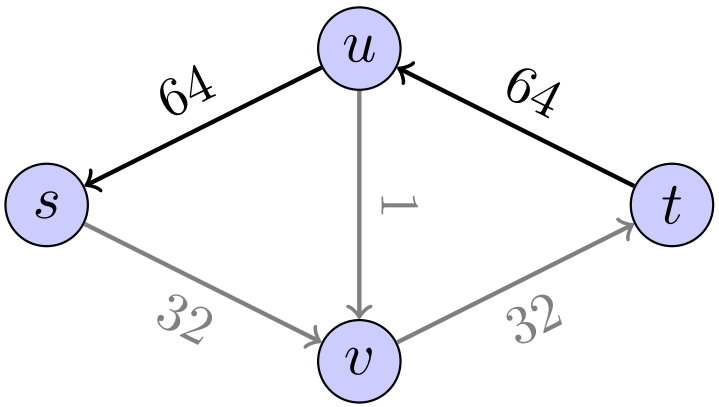
$s - t$ пути в G_f нет

- Поток: 64;
- Δ: $\Delta = 48$;
- $G_f(\Delta)$: три ребра удаляются, поскольку их пропускные способности меньше 48.

Пример: Шаг 3



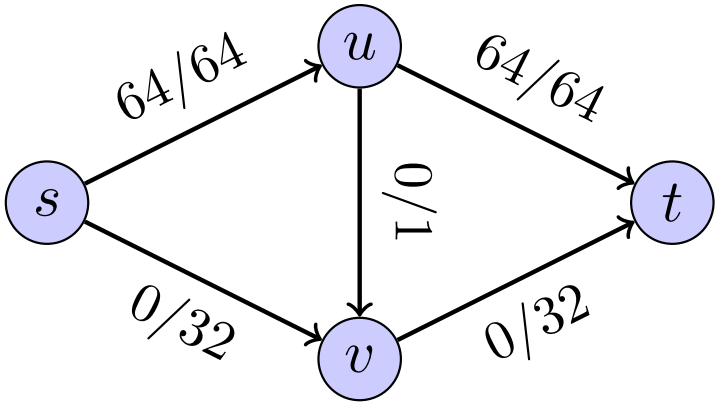
Поток $f : |f| = 64$



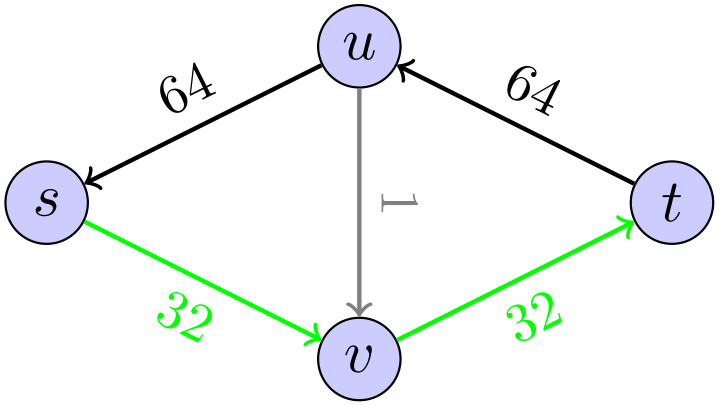
$s - t$ пути в G_f нет

- Поток: 64;
- Δ: $\Delta = 48$;
- $G_f(\Delta)$: три ребра удаляются, поскольку их пропускные способности меньше 48.
- $s - t$ путь: пути нет. Выполняется масштабирование: $\Delta = \frac{\Delta}{2} = 24$.

Пример: Шаг 4



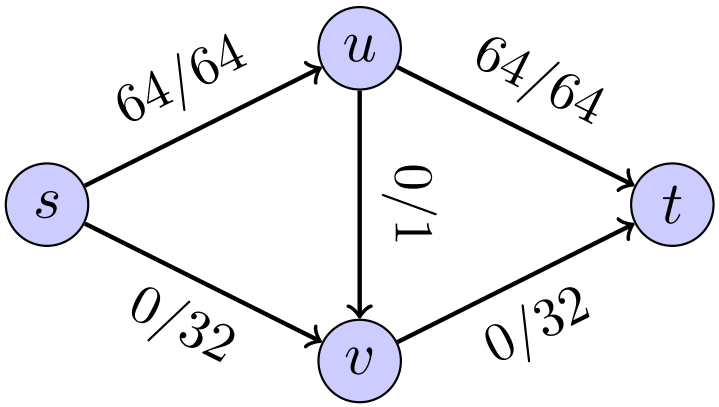
Поток $f : |f| = 64$



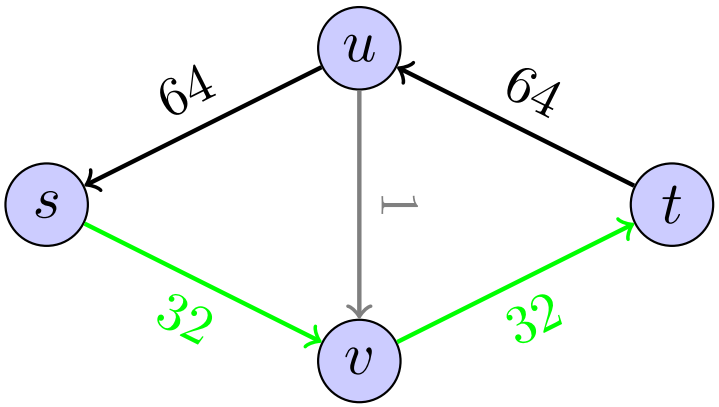
$s - t$ путь в G_f

- Поток: 64;

Пример: Шаг 4



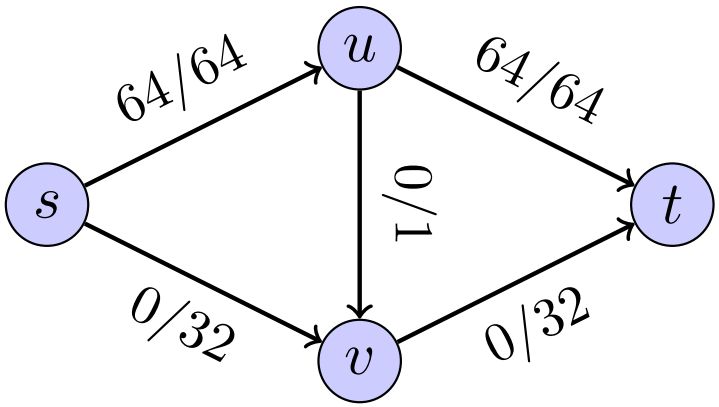
Поток $f : |f| = 64$



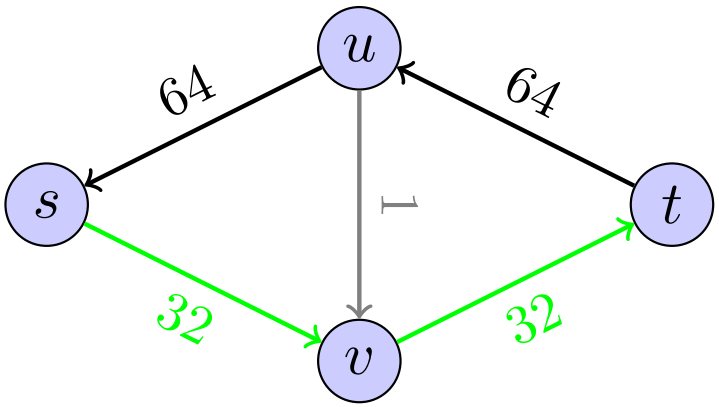
$s - t$ путь в G_f

- Поток: 64;
- Δ: $\Delta = 24$;

Пример: Шаг 4



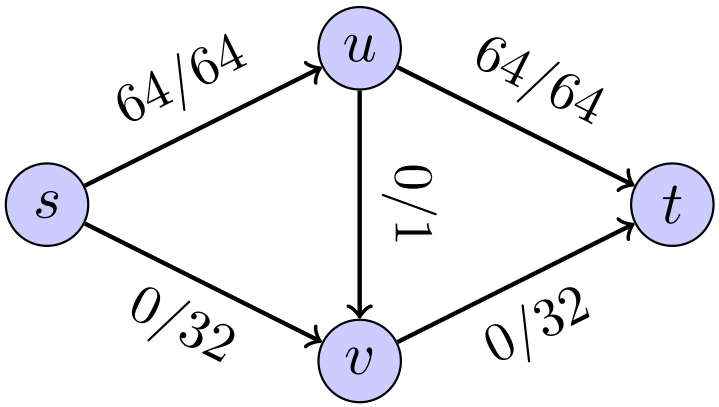
Поток $f : |f| = 64$



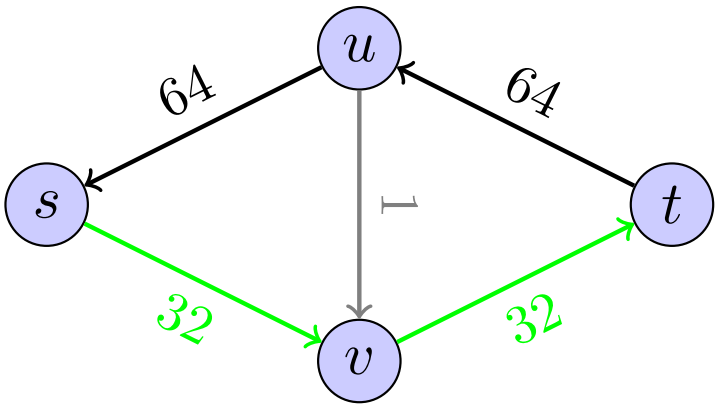
$s - t$ путь в G_f

- Поток: 64;
- Δ: $\Delta = 24$;
- $G_f(\Delta)$: одно ребро удаляется, поскольку его пропускная способность меньше 24.

Пример: Шаг 4



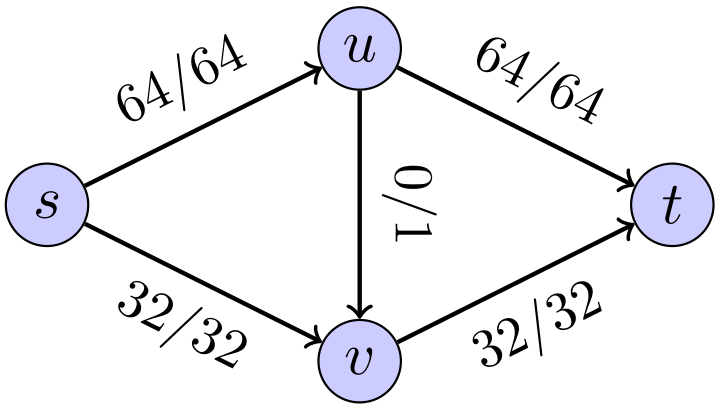
Поток $f : |f| = 64$



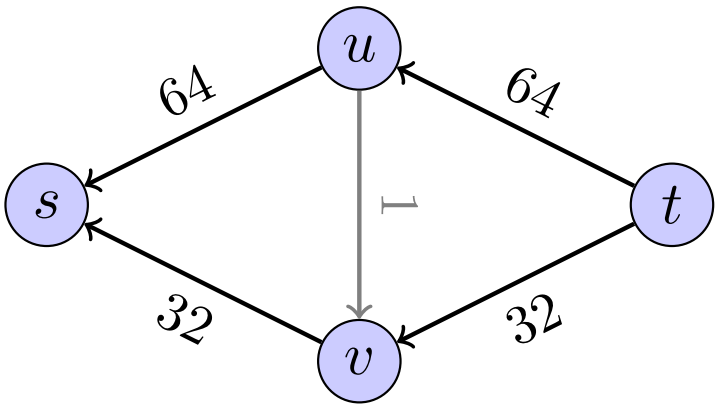
$s - t$ путь в G_f

- Поток: 64;
- Δ: $\Delta = 24$;
- $G_f(\Delta)$: одно ребро удаляется, поскольку его пропускная способность меньше 24.
- $s - t$ путь: найден путь: $s - v - t$. Выполняется операция увеличения потока.

Пример: Шаг 5



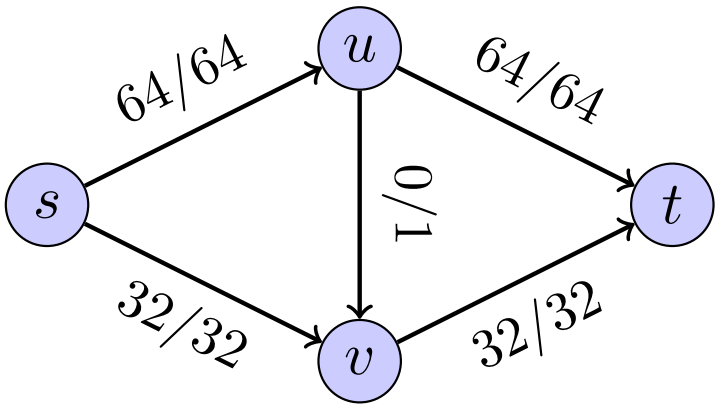
Поток $f : |f| = 96$



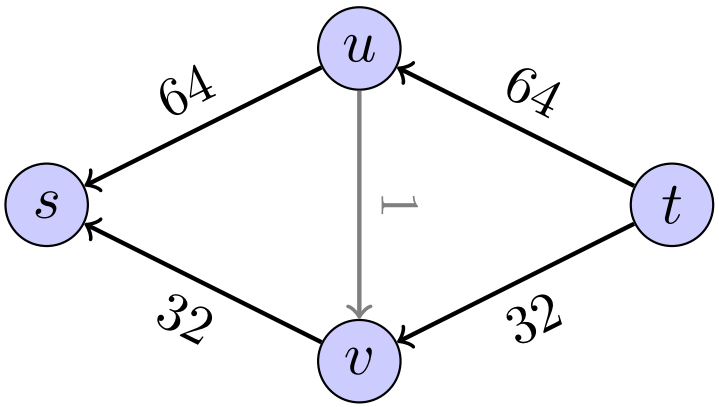
$s - t$ пути в G_f нет

- Поток: 96. Получен максимальный поток.

Пример: Шаг 5



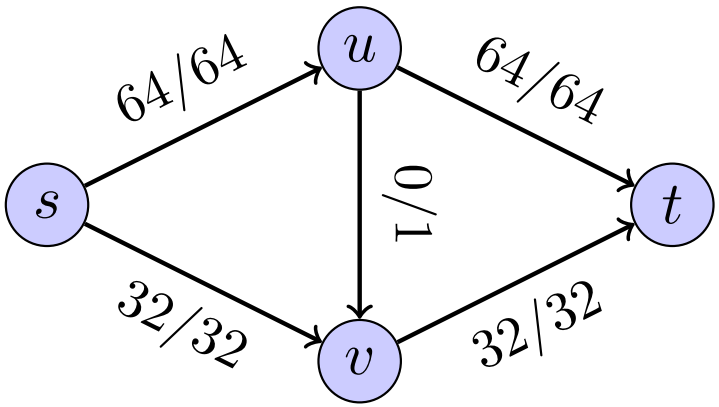
Поток $f : |f| = 96$



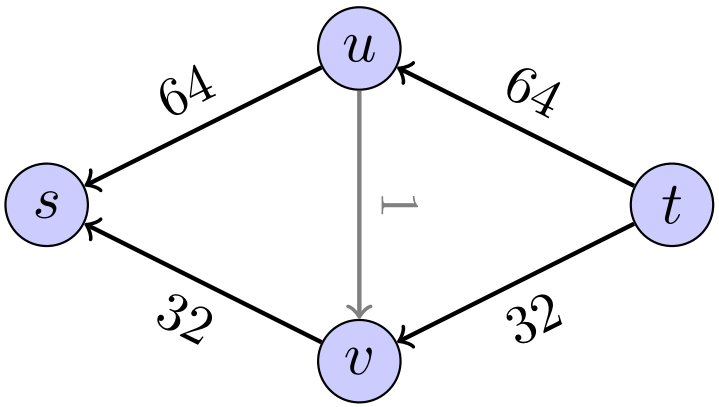
$s - t$ пути в G_f нет

- Поток: 96. Получен максимальный поток.
- Δ: $\Delta = 24$;

Пример: Шаг 5



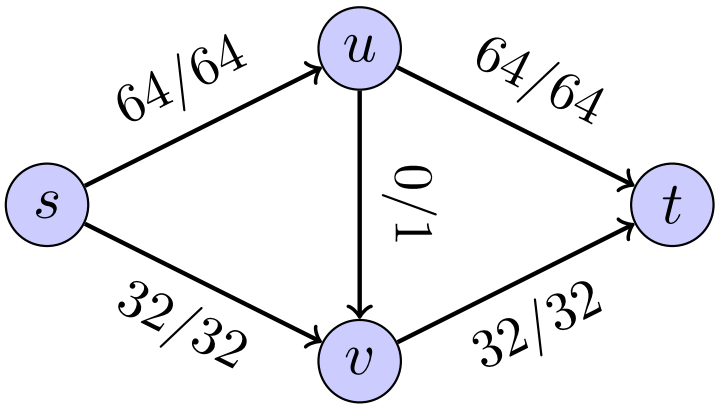
Поток $f : |f| = 96$



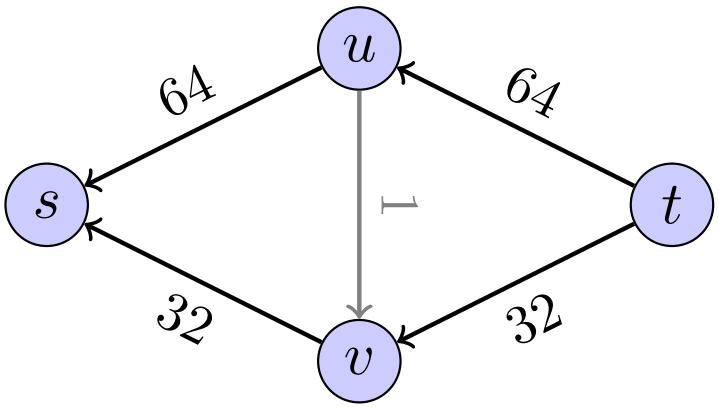
$s - t$ пути в G_f нет

- Поток: 96. Получен максимальный поток.
- Δ: $\Delta = 24$;
- $G_f(\Delta)$: одно ребро удаляется, поскольку его пропускная способность меньше 24.

Пример: Шаг 5



Поток $f : |f| = 96$



$s - t$ пути в G_f нет

- Поток: 96. Получен максимальный поток.
- Δ: $\Delta = 24$;
- $G_f(\Delta)$: одно ребро удаляется, поскольку его пропускная способность меньше 24.
- $s - t$ путь: $s - t$ пути нет.

Анализ: Вшешний цикл `while`

Lemma

(Вшешний цикл `while`*) Число итераций во внешнем цикле* `while` *не превосходит* $1 + \log_2 C$.

Алгоритм Форда - Фалкерсона с масштабированием:

1: Инициализация: $f(e) = 0$ для всех e.
2: Пусть $\Delta = C$;
3: **while** $\Delta \geq 1$ **do**
4: **while** существует $s - t$ путь в $G_f(\Delta)$ **do**
5: Выбрать $s - t$ путь p;
6: $f =$AUGMENT(p, f);
7: **end while**
8: $\Delta = \Delta/2$;
9: **end while**
10: **return** f;

Анализ: внутренний цикл `while`

Theorem

(внутренний цикл `while` *) При фиксированном* Δ*, число увеличений потока не превосходит* $2m$*.*

Алгоритм Форда - Фалкерсона с масштабированием:

1: Инициализация: $f(e) = 0$ для всех e.
2: Пусть $\Delta = C$;
3: **while** $\Delta \geq 1$ **do**
4: **while** существует $s - t$ путь в $G_f(\Delta)$ **do**
5: Выбрать $s - t$ путь p;
6: $f =$AUGMENT(p, f);
7: **end while**
8: $\Delta = \Delta/2$;
9: **end while**
10: **return** f;

Доказательство.

1. Пусть f — поток, полученный когда фаза с масштабом Δ завершилась, и f^* — максимальный поток. Мы имеем $|f| \geq |f^*| - m\Delta$.

Анализ: внутренний цикл `while`

Доказательство.

1. Пусть f — поток, полученный когда фаза с масштабом Δ завершилась, и f^* — максимальный поток. Мы имеем $|f| \geq |f^*| - m\Delta$.
2. На последующей фазе с масштабом $\frac{\Delta}{2}$, после каждого шага увеличения величина потока $|f|$ увеличится по крайней мере на $\frac{\Delta}{2}$.

Доказательство.

1. Пусть f — поток, полученный когда фаза с масштабом Δ завершилась, и f^* — максимальный поток. Мы имеем $|f| \geq |f^*| - m\Delta$.
2. На последующей фазе с масштабом $\frac{\Delta}{2}$, после каждого шага увеличения величина потока $|f|$ увеличится по крайней мере на $\frac{\Delta}{2}$.

Т.о., будет выполнено не более чем $2m$ увеличений потока на фазе с масштабом $\frac{\Delta}{2}$. □

Анализ: внутренний цикл `while`

Доказательство.

1. Пусть f — поток, полученный когда фаза с масштабом Δ завершилась, и f^* — максимальный поток. Мы имеем $|f| \geq |f^*| - m\Delta$.
2. На последующей фазе с масштабом $\frac{\Delta}{2}$, после каждого шага увеличения величина потока $|f|$ увеличится по крайней мере на $\frac{\Delta}{2}$.

Т.о., будет выполнено не более чем $2m$ увеличений потока на фазе с масштабом $\frac{\Delta}{2}$. □

- Трудоемкость: $O(m^2 \log_2 C)$.

Анализ: внутренний цикл `while`

Доказательство.

1. Пусть f — поток, полученный когда фаза с масштабом Δ завершилась, и f^* — максимальный поток. Мы имеем $|f| \geq |f^*| - m\Delta$.
2. На последующей фазе с масштабом $\frac{\Delta}{2}$, после каждого шага увеличения величина потока $|f|$ увеличится по крайней мере на $\frac{\Delta}{2}$.

Т.о., будет выполнено не более чем $2m$ увеличений потока на фазе с масштабом $\frac{\Delta}{2}$. □

- Трудоемкость: $O(m^2 \log_2 C)$.
  - $O(\log_2 C)$ раз выполняется внешний цикл `while`;

Анализ: внутренний цикл `while`

Доказательство.

1. Пусть f — поток, полученный когда фаза с масштабом Δ завершилась, и f^* — максимальный поток. Мы имеем $|f| \geq |f^*| - m\Delta$.
2. На последующей фазе с масштабом $\frac{\Delta}{2}$, после каждого шага увеличения величина потока $|f|$ увеличится по крайней мере на $\frac{\Delta}{2}$.

Т.о., будет выполнено не более чем $2m$ увеличений потока на фазе с масштабом $\frac{\Delta}{2}$. □

- Трудоемкость: $O(m^2 \log_2 C)$.
  - $O(\log_2 C)$ раз выполняется внешний цикл `while`;
  - $O(m)$ раз выполняется внутренний цикл;

Анализ: внутренний цикл `while`

Доказательство.

1. Пусть f — поток, полученный когда фаза с масштабом Δ завершилась, и f^* — максимальный поток. Мы имеем $|f| \geq |f^*| - m\Delta$.
2. На последующей фазе с масштабом $\frac{\Delta}{2}$, после каждого шага увеличения величина потока $|f|$ увеличится по крайней мере на $\frac{\Delta}{2}$.

Т.о., будет выполнено не более чем $2m$ увеличений потока на фазе с масштабом $\frac{\Delta}{2}$. □

- Трудоемкость: $O(m^2 \log_2 C)$.
  - $O(\log_2 C)$ раз выполняется внешний цикл `while`;
  - $O(m)$ раз выполняется внутренний цикл;
  - Каждый шаг увеличения требует $O(m)$ времени.

Анализ: внутренний цикл `while`

Доказательство.

1. Пусть f — поток, полученный когда фаза с масштабом Δ завершилась, и f^* — максимальный поток. Мы имеем $|f| \geq |f^*| - m\Delta$.
2. На последующей фазе с масштабом $\frac{\Delta}{2}$, после каждого шага увеличения величина потока $|f|$ увеличится по крайней мере на $\frac{\Delta}{2}$.

Т.о., будет выполнено не более чем $2m$ увеличений потока на фазе с масштабом $\frac{\Delta}{2}$. □

- Трудоемкость: $O(m^2 \log_2 C)$.
  - $O(\log_2 C)$ раз выполняется внешний цикл `while`;
  - $O(m)$ раз выполняется внутренний цикл;
  - Каждый шаг увеличения требует $O(m)$ времени.
- Масштабирование — это один из способов сделать алгоритм расширения потока по путям полиномиальным по времени, если пропускные способности являются целыми числами.

Почему $|f| \geq |f^*| - m\Delta$?

Доказательство.

- Пусть S — множество узлов достижимых из s в остаточной сети $G_f(\Delta)$, и $\bar{S} = V - S$. Т.о., $(S, \bar{S})$ — разрез, поскольку $S \neq \phi$ и $\bar{S} \neq \phi$.

Почему $|f| \geq |f^*| - m\Delta$?

Доказательство.

- Пусть S — множество узлов достижимых из s в остаточной сети $G_f(\Delta)$, и $\bar{S} = V - S$. Т.о., $(S, \bar{S})$ — разрез, поскольку $S \neq \phi$ и $\bar{S} \neq \phi$.
- Исследуем два типа разрезаемых ребер $e = (u, v) \in E$.

Почему $|f| \geq |f^*| - m\Delta$?

Доказательство.

- Пусть S — множество узлов достижимых из s в остаточной сети $G_f(\Delta)$, и $\bar{S} = V - S$. Т.о., $(S, \bar{S})$ — разрез, поскольку $S \neq \phi$ и $\bar{S} \neq \phi$.
- Исследуем два типа разрезаемых ребер $e = (u, v) \in E$.
  1. $u \in S, v \in \bar{S}$: мы имеем $f(e) \geq C(e) - \Delta$. (Иначе, S должно быть расширино узлом v так как (u, v) в $G_f(\Delta)$.)

Почему $|f| \geq |f^*| - m\Delta$?

Доказательство.

- Пусть S — множество узлов достижимых из s в остаточной сети $G_f(\Delta)$, и $\bar{S} = V - S$. Т.о., $(S, \bar{S})$ — разрез, поскольку $S \neq \phi$ и $\bar{S} \neq \phi$.
- Исследуем два типа разрезаемых ребер $e = (u, v) \in E$.
  1. $u \in S, v \in \bar{S}$: мы имеем $f(e) \geq C(e) - \Delta$. (Иначе, S должно быть расширино узлом v так как (u, v) в $G_f(\Delta)$.)
  2. $u \in \bar{S}, v \in S$: мы имеем $f(e) \leq \Delta$. (Иначе, S должно быть расширино узлом v так как (u, v) в $G_f(\Delta)$.)

Почему $|f| \geq |f^*| - m\Delta$?

Доказательство.

- Пусть S — множество узлов достижимых из s в остаточной сети $G_f(\Delta)$, и $\bar{S} = V - S$. Т.о., $(S, \bar{S})$ — разрез, поскольку $S \neq \phi$ и $\bar{S} \neq \phi$.
- Исследуем два типа разрезаемых ребер $e = (u, v) \in E$.
  1. $u \in S, v \in \bar{S}$: мы имеем $f(e) \geq C(e) - \Delta$. (Иначе, S должно быть расширино узлом v так как (u, v) в $G_f(\Delta)$.)
  2. $u \in \bar{S}, v \in S$: мы имеем $f(e) \leq \Delta$. (Иначе, S должно быть расширино узлом v так как (u, v) в $G_f(\Delta)$.)
- Следовательно:

$$\begin{aligned} |f| &= \sum_{e \in S \to \bar{S}} f(e) - \sum_{e \in \bar{S} \to S} f(e) \\ &\geq \sum_{e \in S \to \bar{S}} (C(e) - \Delta) - \sum_{e \in \bar{S} \to S} \Delta \\ &\geq \sum_{e \in S \to \bar{S}} C(e) - m\Delta \\ &= C(S, \bar{S}) - m\Delta \\ &\geq |f^*| - m\Delta \end{aligned}$$

Улудшение 2: Алгоритм Эдмонса - Карпа.
Использование **кратчайших увеличивающих путей**

Алгоритм Эдмонса - Карпа [1972]

Рис.: Jack Edmonds, and Richard Karp

- Алгоритм был опубликован Диницем в 1970 и независимо Эдмонсом и Карпом в 1972.

Алгоритм Эдмонса - Карпа

Edmonds-Karp(G)

1: Инициализация: $f(e) = 0$ для всех e.
2: **while** существует $s - t$ путь в G_f **do**
3: Найти **кратчайший** $s - t$ путь p в G_f используя BFS;
4: f =augment(p, f);
5: **end while**
6: **return** f;

Theorem

Алгоритм Эдмонса - Карпа *имеет трудоемкость* $O(m^2n)$.

Доказательство.

- Во время выполнения Алгоритма Эдмонса - Карпа, ребро $e = (u, v)$ может быть в качестве **bottleneck** не более чем $\frac{n}{2}$ раз.

Theorem

Алгоритм Эдмонса - Карпа *имеет трудоемкость* $O(m^2 n)$.

Доказательство.

- Во время выполнения Алгоритма Эдмонса - Карпа, ребро $e = (u, v)$ может быть в качестве **bottleneck** не более чем $\frac{n}{2}$ раз.
- Т.о., цикл `while` будет выполняться $\frac{n}{2}m$ раз.

Theorem

Алгоритм Эдмонса - Карпа *имеет трудоемкость* $O(m^2 n)$.

Доказательство.

- Во время выполнения Алгоритма Эдмонса - Карпа, ребро $e = (u, v)$ может быть в качестве **bottleneck** не более чем $\frac{n}{2}$ раз.
- Т.о., цикл `while` будет выполняться $\frac{n}{2}m$ раз.
- Потребуется $O(m)$ времени чтобы найти кратчайший путь, используя BFS, а затем увеличить поток вдоль пути.

□

Theorem

Алгоритм Эдмонса - Карпа *имеет трудоемкость* $O(m^2 n)$.

Доказательство.

- Во время выполнения Алгоритма Эдмонса - Карпа, ребро $e = (u, v)$ может быть в качестве **bottleneck** не более чем $\frac{n}{2}$ раз.
- Т.о., цикл `while` будет выполняться $\frac{n}{2}m$ раз.
- Потребуется $O(m)$ времени чтобы найти кратчайший путь, используя BFS, а затем увеличить поток вдоль пути.

□

- Алгоритм Эдмонса - Карпа — полиномиальный: трудоемкость — полином от n и m, даже, если пропускные способности являются вещественными числами, и предполагается, что операция над вещественными числами требует единицы времени. Строгая полиномиальность является более естественой с комбинаторной точки зрения.

Theorem

Любое ребро (u, v) в G может быть в качестве bottleneck не более чем $\frac{n}{2}$ раз

Theorem

Любое ребро (u, v) в G может быть в качестве bottleneck не более чем $\frac{n}{2}$ раз

Доказательство.

- Для остаточной сети G_f, разобьем узлы по уровням $L_0, L_1, ...$, где $L_0 = \{s\}$, и L_i содержит все узлы v такие, что кратчайший путь из s в v имеет длину i. Пусть $d_f(u)$ обозначает номер уровня узла u, т.е. расстояние от s до u в G_f.

Theorem

Любое ребро (u, v) в G может быть в качестве bottleneck не более чем $\frac{n}{2}$ раз

Доказательство.

- Для остаточной сети G_f, разобьем узлы по уровням $L_0, L_1, ...$, где $L_0 = \{s\}$, и L_i содержит все узлы v такие, что кратчайший путь из s в v имеет длину i. Пусть $d_f(u)$ обозначает номер уровня узла u, т.е. расстояние от s до u в G_f.
- Рассмотрим два последовательных появления ребра (u, v) в качестве узкого места, пусть это произошло на шаге k и на шаге k'''.

Theorem

Любое ребро (u, v) в G может быть в качестве bottleneck не более чем $\frac{n}{2}$ раз

Доказательство.

- Для остаточной сети G_f, разобьем узлы по уровням $L_0, L_1, ...$, где $L_0 = \{s\}$, и L_i содержит все узлы v такие, что кратчайший путь из s в v имеет длину i. Пусть $d_f(u)$ обозначает номер уровня узла u, т.е. расстояние от s до u в G_f.
- Рассмотрим два последовательных появления ребра (u, v) в качестве узкого места, пусть это произошло на шаге k и на шаге k'''.
  - На шаге k, мы имеем $d_f(v) = d_f(u) + 1$. Отметим, что после увеличения потока, ребро $e = (u, v)$ изменит свое направление.

Theorem

Любое ребро (u, v) в G может быть в качестве bottleneck не более чем $\frac{n}{2}$ раз

Доказательство.

- Для остаточной сети G_f, разобьем узлы по уровням $L_0, L_1, ...$, где $L_0 = \{s\}$, и L_i содержит все узлы v такие, что кратчайший путь из s в v имеет длину i. Пусть $d_f(u)$ обозначает номер уровня узла u, т.е. расстояние от s до u в G_f.
- Рассмотрим два последовательных появления ребра (u, v) в качестве узкого места, пусть это произошло на шаге k и на шаге k'''.
  - На шаге k, мы имеем $d_f(v) = d_f(u) + 1$. Отметим, что после увеличения потока, ребро $e = (u, v)$ изменит свое направление.
  - На шаге k''', $e = (u, v)$ становится BOTTLENECK-ребром снова, поэтому ребро $e' = (v, u)$ изменит направление, которое она имела на шаге предшествующем k''', скажем это шаг k''.

Theorem

Любое ребро (u, v) в G может быть в качестве bottleneck не более чем $\frac{n}{2}$ раз

Доказательство.

- Для остаточной сети G_f, разобьем узлы по уровням $L_0, L_1, ...,$ где $L_0 = \{s\}$, и L_i содержит все узлы v такие, что кратчайший путь из s в v имеет длину i. Пусть $d_f(u)$ обозначает номер уровня узла u, т.е. расстояние от s до u в G_f.
- Рассмотрим два последовательных появления ребра (u, v) в качестве узкого места, пусть это произошло на шаге k и на шаге k'''.
  - На шаге k, мы имеем $d_f(v) = d_f(u) + 1$. Отметим, что после увеличения потока, ребро $e = (u, v)$ изменит свое направление.
  - На шаге k''', $e = (u, v)$ становится BOTTLENECK-ребром снова, поэтому ребро $e' = (v, u)$ изменит направление, которое она имела на шаге предшествующем k''', скажем это шаг k''.
  - На шаге k'', мы имеем $d_{f''}(u) = d_{f''}(v) + 1$.

Theorem

Любое ребро (u, v) в G может быть в качестве bottleneck не более чем $\frac{n}{2}$ раз

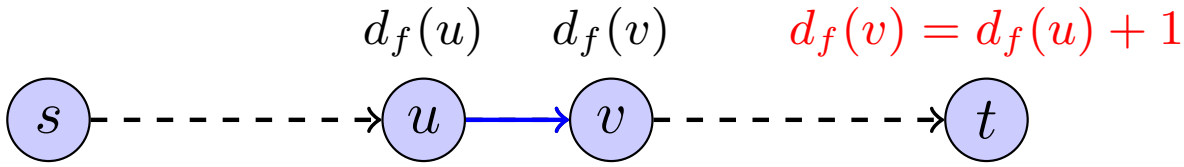
Доказательство.

- Для остаточной сети G_f, разобьем узлы по уровням $L_0, L_1, ...$, где $L_0 = \{s\}$, и L_i содержит все узлы v такие, что кратчайший путь из s в v имеет длину i. Пусть $d_f(u)$ обозначает номер уровня узла u, т.е. расстояние от s до u в G_f.
- Рассмотрим два последовательных появления ребра (u, v) в качестве узкого места, пусть это произошло на шаге k и на шаге k'''.
  - На шаге k, мы имеем $d_f(v) = d_f(u) + 1$. Отметим, что после увеличения потока, ребро $e = (u, v)$ изменит свое направление.
  - На шаге k''', $e = (u, v)$ становится BOTTLENECK-ребром снова, поэтому ребро $e' = (v, u)$ изменит направление, которое она имела на шаге предшествующем k''', скажем это шаг k''.
  - На шаге k'', мы имеем $d_{f''}(u) = d_{f''}(v) + 1$.

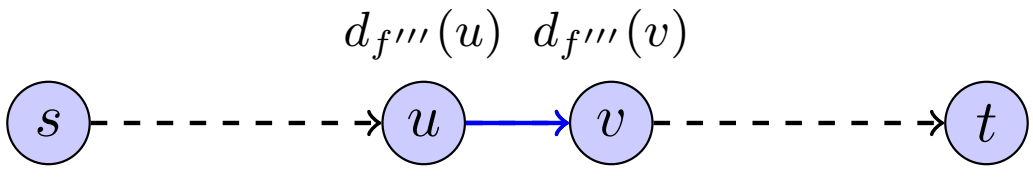
- T.o., $d_{f''}(u) = d_{f''}(v) + 1 \geq d_{f'}(v) + 1 \geq d_f(u) + 2$.

Анализ Алгоритм Эдмонса - Карпа

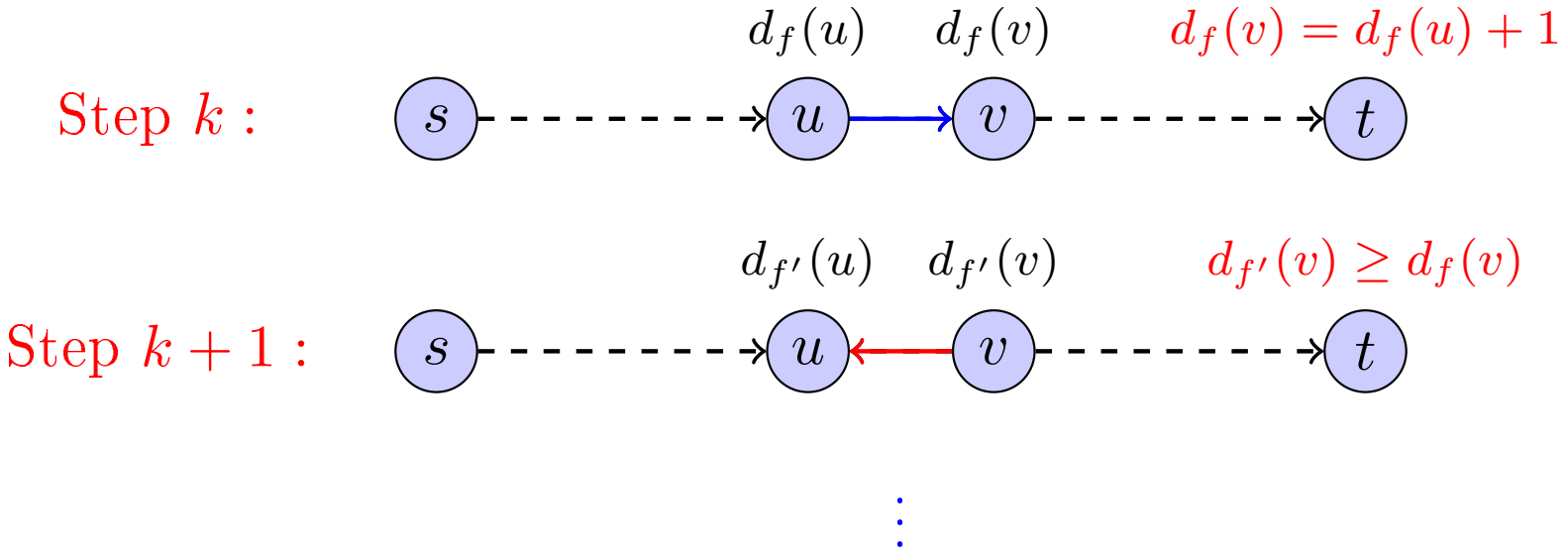
Step k :

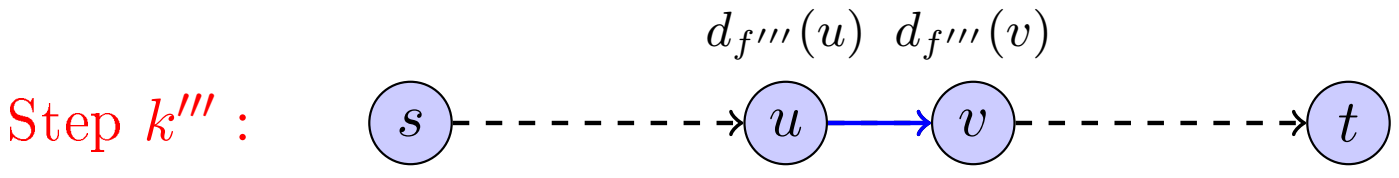


Step k''' :

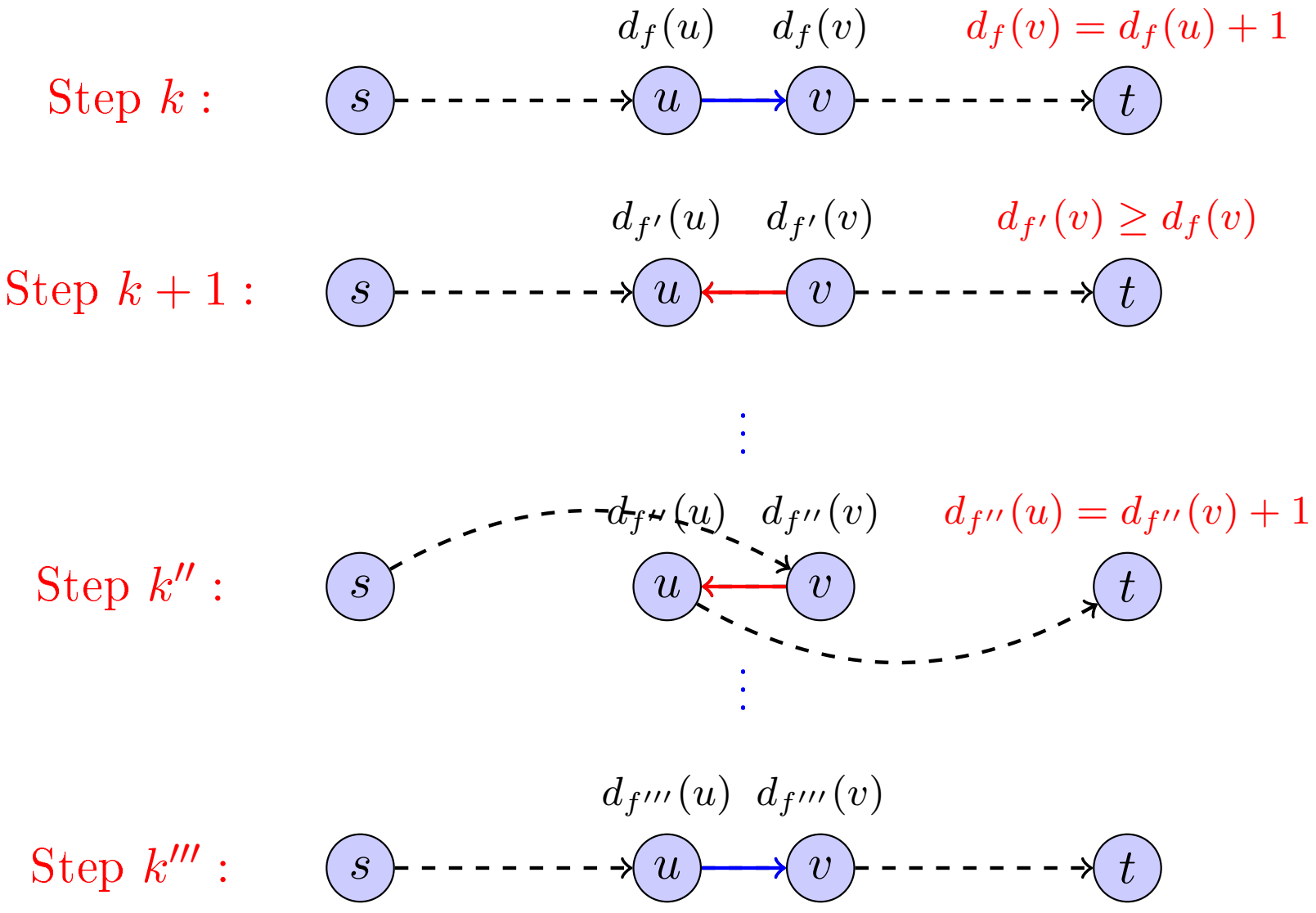


Анализ Алгоритм Эдмонса - Карпа

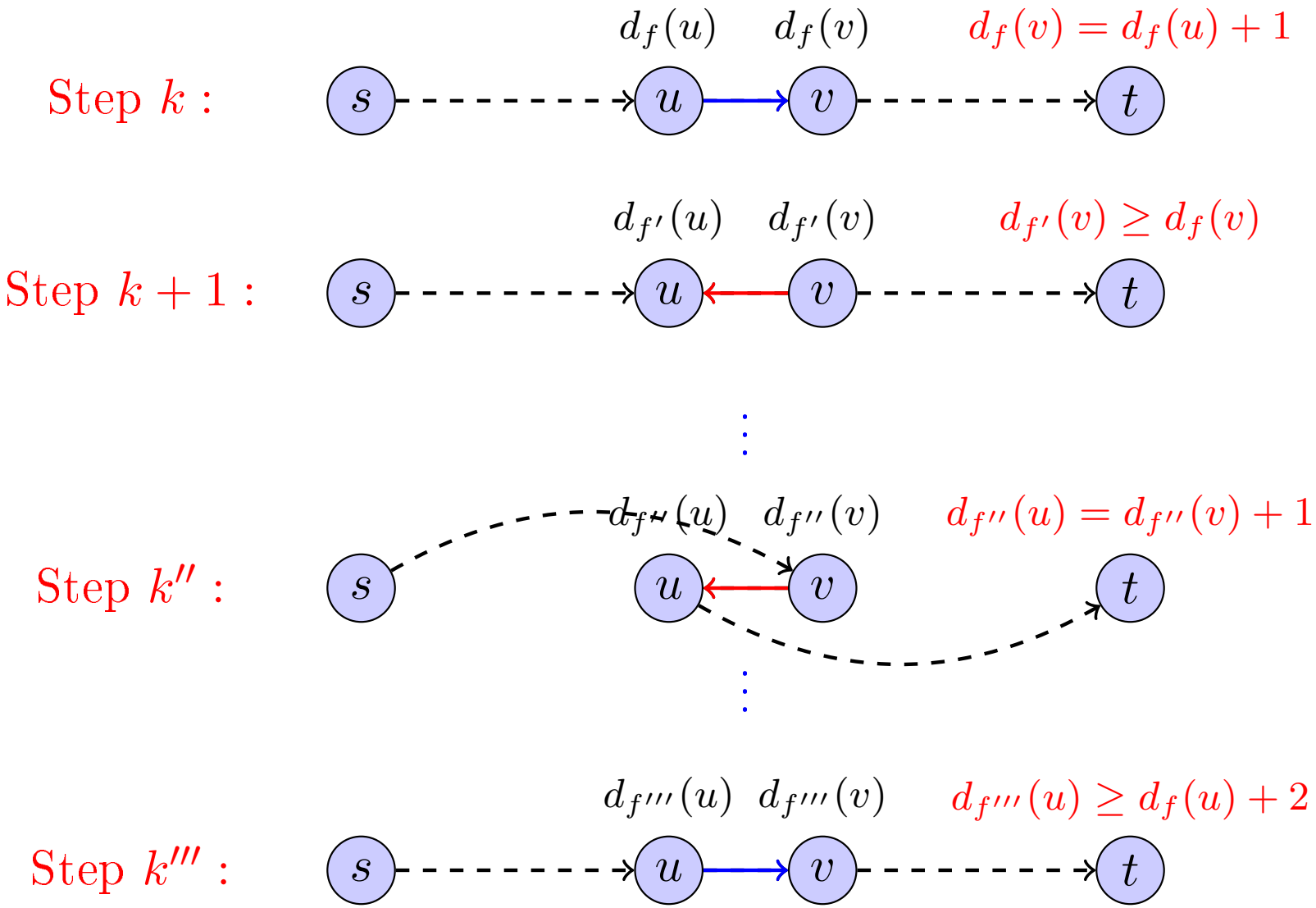




Анализ Алгоритм Эдмонса - Карпа



Анализ Алгоритм Эдмонса - Карпа



Номер уровня узла никогда не уменьшается

Theorem

Рассмотрим поток f и соответствующую остаточную сеть G_f.

Номер уровня узла никогда не уменьшается

Theorem

Рассмотрим поток f и соответствующую остаточную сеть G_f. Предположим, что кратчайший путь p из s в t в сети G_f был выбран в качестве увеличивающего.

Номер уровня узла никогда не уменьшается

Theorem

Рассмотрим поток f и соответствующую остаточную сеть G_f. Предположим, что кратчайший путь p из s в t в сети G_f был выбран в качестве увеличивающего. После увеличения получен поток f'. Тогда для любого узла v, $d_f(v) \leq d_{f'}(v)$.

Номер уровня узла никогда не уменьшается

Theorem

Рассмотрим поток f и соответствующую остаточную сеть G_f. Предположим, что кратчайший путь p из s в t в сети G_f был выбран в качестве увеличивающего. После увеличения получен поток f'. Тогда для любого узла v, $d_f(v) \leq d_{f'}(v)$.

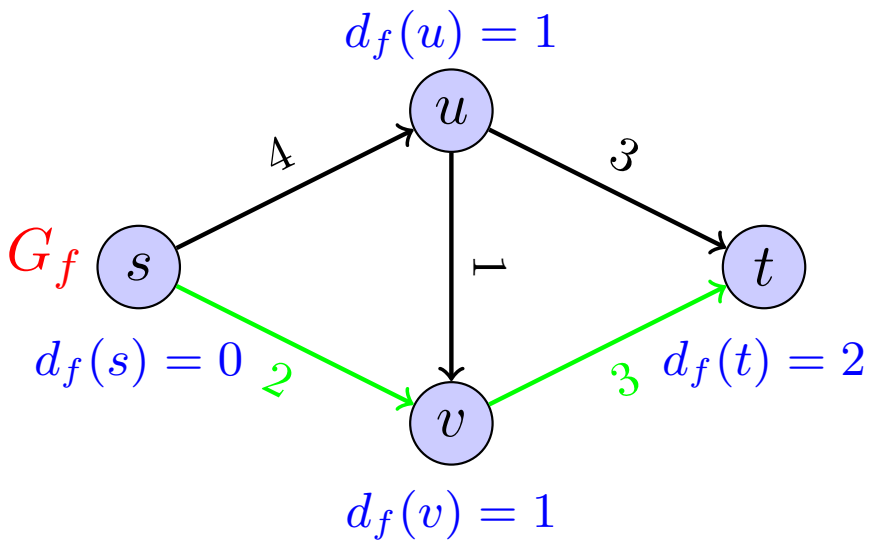
Др. словами: Для любого узла v, расстояние $d_f(v)$ в остаточной сети G_f никогда не уменьшается, если кратчайший увеличивающий путь выбран для увеличения потока.

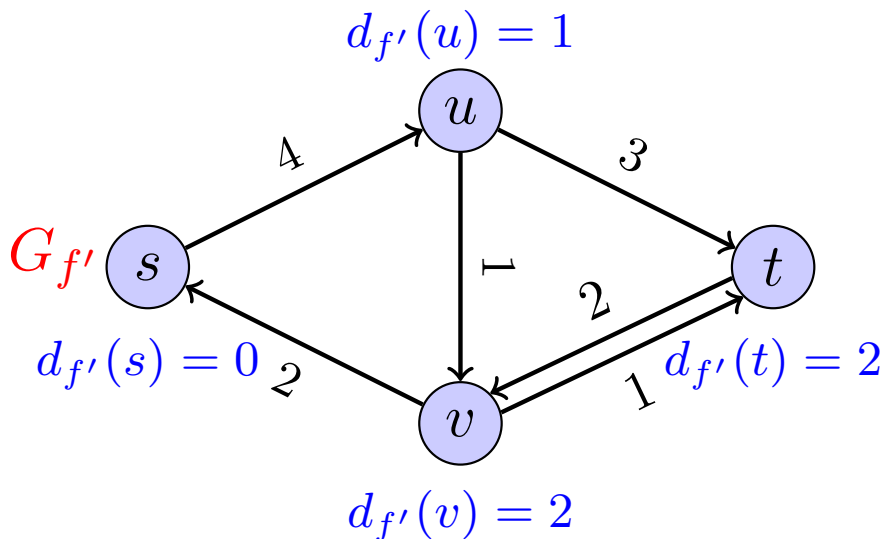
Номер уровня узла никогда не уменьшается

Theorem

Рассмотрим поток f и соответствующую остаточную сеть G_f. Предположим, что кратчайший путь p из s в t в сети G_f был выбран в качестве увеличивающего. После увеличения получен поток f'. Тогда для любого узла v, $d_f(v) \leq d_{f'}(v)$.

Др. словами: Для любого узла v, расстояние $d_f(v)$ в остаточной сети G_f никогда не уменьшается, если кратчайший увеличивающий путь выбран для увеличения потока.





Доказательство.

- Вопервых мы утверждаем, что для любого ребра (v_i, v_j) в $G_{f'}$, $d_f(v_j) \leq d_f(v_i) + 1$.

Доказательство.

- Вопервых мы утверждаем, что для любого ребра (v_i, v_j) в $G_{f'}$, $d_f(v_j) \leq d_f(v_i) + 1$.
  - Случай 1: (v_i, v_j) в G_f, например (u, v): Очевидно.

Доказательство.

- Вопервых мы утверждаем, что для любого ребра (v_i, v_j) в $G_{f'}$, $d_f(v_j) \leq d_f(v_i) + 1$.
  - Случай 1: (v_i, v_j) в G_f, например (u, v): Очевидно.
  - Случай 2: (v_i, v_j) не принадлежит G_f: В качестве примера возьмем (u, s). (s, u) должна быть в увеличивающем (кратчайшем) пути в G_f и следовательно $d_f(u) = d_f(s) + 1$.

Доказательство.

- Вопервых мы утверждаем, что для любого ребра (v_i, v_j) в $G_{f'}$, $d_f(v_j) \leq d_f(v_i) + 1$.
  - Случай 1: (v_i, v_j) в G_f, например (u, v): Очевидно.
  - Случай 2: (v_i, v_j) не принадлежит G_f: В качестве примера возьмем (u, s). (s, u) должна быть в увеличивающем (кратчайшем) пути в G_f и следовательно $d_f(u) = d_f(s) + 1$.
- Далее, предположим $d_{f'}(v) = r$. Пусть $(s, v_1, ..., v_{r-1}, v)$ — кратчайший путь до v в $G_{f'}$.

Доказательство.

- Вопервых мы утверждаем, что для любого ребра (v_i, v_j) в $G_{f'}$, $d_f(v_j) \leq d_f(v_i) + 1$.
  - Случай 1: (v_i, v_j) в G_f, например (u, v): Очевидно.
  - Случай 2: (v_i, v_j) не принадлежит G_f: В качестве примера возьмем (u, s). (s, u) должна быть в увеличивающем (кратчайшем) пути в G_f и следовательно $d_f(u) = d_f(s) + 1$.
- Далее, предположим $d_{f'}(v) = r$. Пусть $(s, v_1, ..., v_{r-1}, v)$ — кратчайший путь до v в $G_{f'}$.
  Мы имеем:

$$\begin{aligned} d_f(v) &\leq d_f(v_{r-1}) + 1 \\ &\leq d_f(v_{r-2}) + 2 \\ &..... \\ &\leq d_f(s) + r \\ &= r \end{aligned}$$

Улудшение 3: Алгоритм Диница и его варианты

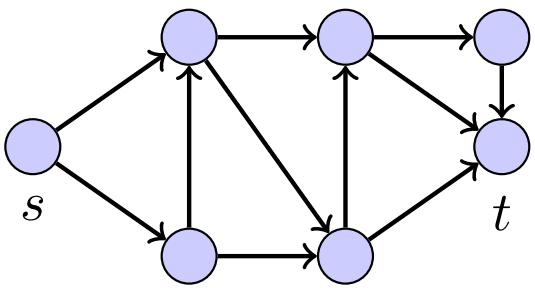
- Идея:
  - Ускорить алгоритм Форда - Фалкерсона с помощью структуры данных.

- Идея:
  - Ускорить алгоритм Форда - Фалкерсона с помощью структуры данных.
  - Отметим, что поиск увеличивающего пути занимает $O(m)$ времени. В дереве BFS, есть ребра, которые можно насытить и они попадут в разрез между s и t. Т. о., весьма ценно сохранить **такую информацию**, для последующих итераций.

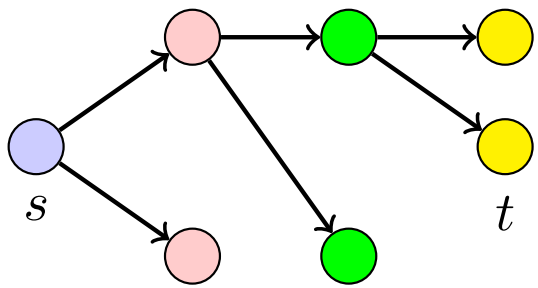
- Идея:
  - Ускорить алгоритм Форда - Фалкерсона с помощью структуры данных.
  - Отметим, что поиск увеличивающего пути занимает $O(m)$ времени. В дереве BFS, есть ребра, которые можно насытить и они попадут в разрез между s и t. Т. о., весьма ценно сохранить **такую информацию**, для последующих итераций.
  - **BFS дерево** дополняется до **уровневой сети**:

- Идея:
  - Ускорить алгоритм Форда - Фалкерсона с помощью структуры данных.
  - Отметим, что поиск увеличивающего пути занимает $O(m)$ времени. В дереве BFS, есть ребра, которые можно насытить и они попадут в разрез между s и t. Т. о., весьма ценно сохранить **такую информацию**, для последующих итераций.
  - **BFS дерево** дополняется до **уровневой сети**:
    - BFS дерево: содержит **одно ребро, ведущее в узел v;**

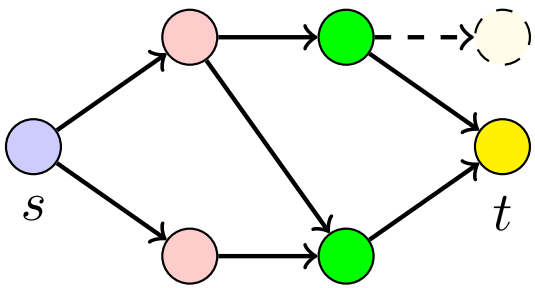
- Идея:
  - Ускорить алгоритм Форда - Фалкерсона с помощью структуры данных.
  - Отметим, что поиск увеличивающего пути занимает $O(m)$ времени. В дереве BFS, есть ребра, которые можно насытить и они попадут в разрез между s и t. Т. о., весьма ценно сохранить **такую информацию**, для последующих итераций.
  - **BFS дерево** дополняется до **уровневой сети**:
    - BFS дерево: содержит **одно ребро, ведущее в узел v;**
    - Уровневая сеть: содержит **все ребра, находящиеся на кратчайших $s-t$ путях в остаточной сети**. Теперь, самый короткий $s-t$ путь может быть найден за время $O(n)$.



Остаточная сеть G_f



BFS дерево



Уровневая сеть N_f

1. **Блокирующий поток** так же известен как **кратчайший насыщающий поток** насыщает все кратчайшие $s - t$ пути в остаточной сети. После увеличения потока блокирующим потоком, число уровней увеличивается и уровень узла t увеличивается **по крайней мере на 1**.

1. **Блокирующий поток** так же известен как **кратчайший насыщающий поток** насыщает все кратчайшие $s - t$ пути в остаточной сети. После увеличения потока блокирующим потоком, число уровней увеличивается и уровень узла t увеличивается **по крайней мере на 1**.
2. **DFS**: Алгоритмом DFS ищется путь в уровневой сети. На это тратится $O(n)$ времени (ипользуются номера уровней узлов). Наоборот, алгоритм Эдмонса - Карпа использует BFS чтобы найти кратчайший путь в остаточной сети, что требует $O(m)$ времени.

1. **Блокирующий поток** так же известен как **кратчайший насыщающий поток** насыщает все кратчайшие $s - t$ пути в остаточной сети. После увеличения потока блокирующим потоком, число уровней увеличивается и уровень узла t увеличивается **по крайней мере на 1**.
2. **DFS**: Алгоритмом DFS ищется путь в уровневой сети. На это тратится $O(n)$ времени (ипользуются номера уровней узлов). Наоборот, алгоритм Эдмонса - Карпа использует BFS чтобы найти кратчайший путь в остаточной сети, что требует $O(m)$ времени.

Отметим, что: при работе на двудольном графе алгоритм Диница совпадает с алгоритмом Хопкрофта-Карпа.

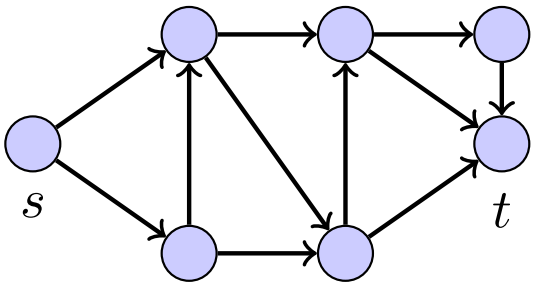
Алгоритм Диница

Dinic's-Max-Flow(G)

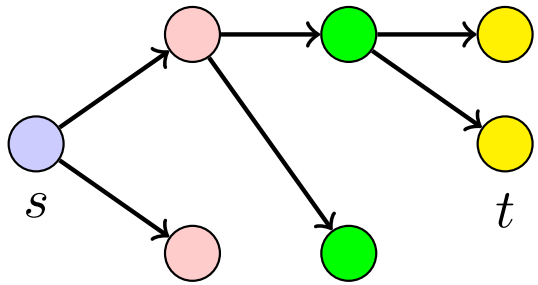
1: Инициализация: $f(e) = 0$ для всех e.
2: **while** `TRUE` **do**
3: Построить **уровневую сеть** N_f из **остаточной сети** G_f, используя BFS;
4: **if** t не достижим из s в G_f **then**
5: break;
6: **end if**
7: Найти **блокирующий поток** b_f в N_f, используя **DFS**;
8: Увеличить поток $f = f + b_f$;
9: **end while**
10: **return** f;

Construct-Layered-Network(G_f)

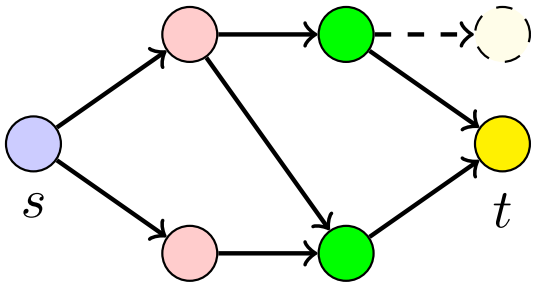
1: Положить $d_f(s) = 0$, $d_f(v) = \infty$ для всех узлов $v \neq s$, и добавить s в очередь Q;
2: Построить уровневую сеть $N_f = (V_f, E_f)$, положив $V_f = \{s\}$ и $E_f = \{\}$;
3: **while** Q не пустая **do**
4: $v = Q.dequeue()$;
5: **for** каждого ребра (v, w) в G_f **do**
6: **if** $d_f(w) = \infty$ **then**
7: $Q.enqueue(w)$; $d_f(w) = d_f(v) + 1$;
8: $V_f = V_f \cup \{w\}$; $E_f = E_f \cup \{(v, w)\}$;
9: **end if**
10: **if** $d_f(w) = d_f(v) + 1$ **then**
11: $E_f = E_f \cup \{(v, w)\}$;
12: **end if**
13: **end for**
14: **end while**
15: Выполнить BFS в N_f из узла t, двигаясь по ребрам в обратном направлении, и удалить узел v из N_f, если v не достижим;
16: **return** N_f;



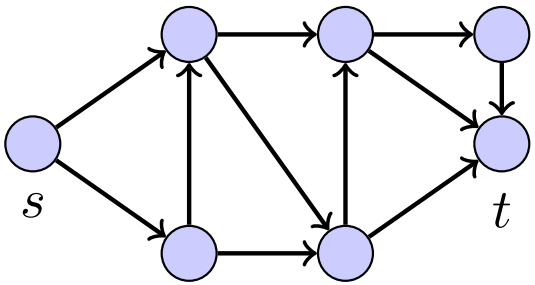
Остаточная сеть G_f



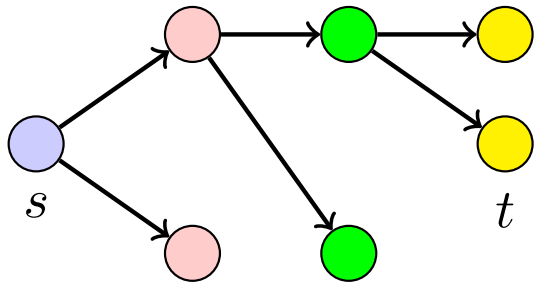
дерево BFS



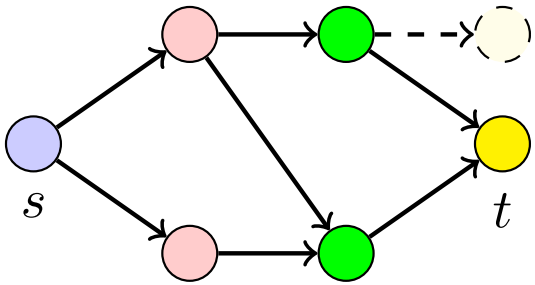
Уровневая сеть N_f



Остаточная сеть G_f

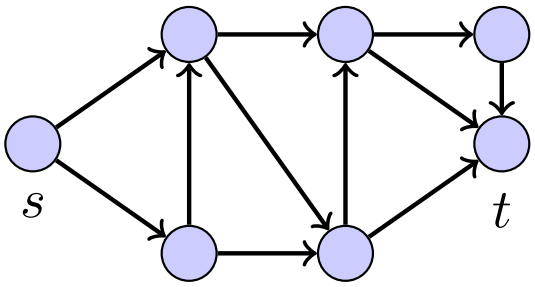


дерево BFS

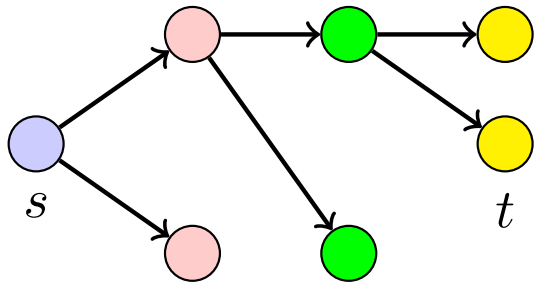


Уровневая сеть N_f

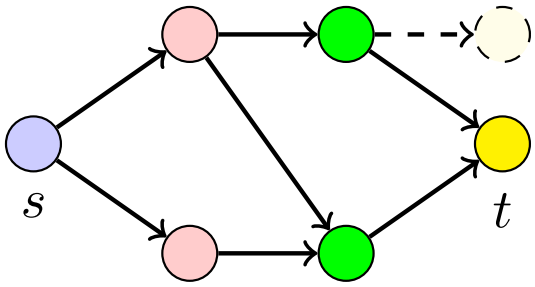
- Отличие от стандартной процедуры BFS в том, что каждое ребро (v, w) с $d_f(w) = d_f(v) + 1$ добавляется в N_f даже, если w уже был добавлен в очередь Q. Т.о., для каждой вершины v, все ребра из крадчайших путей от s до v добавляются в N_f.



Остаточная сеть G_f



дерево BFS



Уровневая сеть N_f

- Отличие от стандартной процедуры BFS в том, что каждое ребро (v, w) с $d_f(w) = d_f(v) + 1$ добавляется в N_f даже, если w уже был добавлен в очередь Q. Т.о., для каждой вершины v, все ребра из крадчайших путей от s до v добавляются в N_f.
- Узлы (и инцидентные им ребра), не находящиеся на кратчайших путях от s до t, будут удалены из N_f, например, узел и ребра нарисованные пунктиром.

Нахождение блокирующего потока в уровневой сети N_f

DINIC-BLOCKING-FLOW(N_f)

1: Установить b_f как 0-поток;
2: **while** существует ребро исходящее из s в N_f **do**
3: Найти путь p из s, имеющий максимальную длину в N_f;
4: **if** p доходит до t **then**
5: b_f =AUGMENT(p, b_f);
6: Удалить из N_f насыщенные ребра пути p;
7: **else**
8: Удалить узел, являющийся последним в p (и инцидентные ребра);
9: **end if**
10: **end while**
11: **return** b_f;

Алгоритм Диница

- Выполнение алгоритма можно разделить на **фазы**, каждая фаза состоит из построения уровневой сети и построения в ней блокирующего потока.

Алгоритм Диница

- Выполнение алгоритма можно разделить на **фазы**, каждая фаза состоит из построения уровневой сети и построения в ней блокирующего потока.
- **Блокирующий поток** содержит **набор** кратчайших $s-t$ путей в G_f. После насыщения этих путей, узел t становится недостижимым из s.

Алгоритм Диница

- Выполнение алгоритма можно разделить на **фазы**, каждая фаза состоит из построения уровневой сети и построения в ней блокирующего потока.
- **Блокирующий поток** содержит **набор** кратчайших $s-t$ путей в G_f. После насыщения этих путей, узел t становится недостижимым из s.
- Замечание: после построения уровневой сети за время $O(m)$, находится блокирующий поток. На построение увеличивающего пути тратится $O(n)$ времени. Напротив алгоритм Эдмонса - Карпа производит увеличение **только по одному** $s-t$ пути после выполнения BFS. На это тратится время $O(m)$.

Трудоемкость: $O(mn^2)$

- $\#\texttt{WHILE} = O(n)$. (Причина: После увеличения потока блокирующим потоком, расстояние $d_f(t)$ должно возрасти по крайней мере на 1.)

Трудоемкость: $O(mn^2)$

- #`WHILE` $= O(n)$. (Причина: После увеличения потока блокирующим потоком, расстояние $d_f(t)$ должно возрасти по крайней мере на 1.)
- На каждой итерации тратится $O(m)$ времени на построение уровневой сети, используя BFS, и тратится $O(mn)$ времени на нахождение блокирующего потока, поскольку:

Трудоемкость: $O(mn^2)$

- #`WHILE` $= O(n)$. (Причина: После увеличения потока блокирующим потоком, расстояние $d_f(t)$ должно возрасти по крайней мере на 1.)
- На каждой итерации тратится $O(m)$ времени на построение уровневой сети, используя BFS, и тратится $O(mn)$ времени на нахождение блокирующего потока, поскольку:
  1. Тратится $O(n)$ времени на нахождение $s-t$ пути в уровневой сети N_f, используя DFS.

Трудоемкость: $O(mn^2)$

- #`WHILE` $= O(n)$. (Причина: После увеличения потока блокирующим потоком, расстояние $d_f(t)$ должно возрасти по крайней мере на 1.)
- На каждой итерации тратится $O(m)$ времени на построение уровневой сети, используя BFS, и тратится $O(mn)$ времени на нахождение блокирующего потока, поскольку:
  1. Тратится $O(n)$ времени на нахождение $s-t$ пути в уровневой сети N_f, используя DFS.
  2. По крайней мере одно ребро в увеличивающем пути будет насыщено и, поэтому удалено из N_f.

Трудоемкость: $O(mn^2)$

- #`WHILE` $= O(n)$. (Причина: После увеличения потока блокирующим потоком, расстояние $d_f(t)$ должно возрасти по крайней мере на 1.)
- На каждой итерации тратится $O(m)$ времени на построение уровневой сети, используя BFS, и тратится $O(mn)$ времени на нахождение блокирующего потока, поскольку:
  1. Тратится $O(n)$ времени на нахождение $s-t$ пути в уровневой сети N_f, используя DFS.
  2. По крайней мере одно ребро в увеличивающем пути будет насыщено и, поэтому удалено из N_f.
  3. Т.о., потребуется не более, чем m итераций для вычисления блокирующего потока.

Расстояние $d_f(t)$ возрастает по крайней мере на 1 на каждой фазе

Theorem

Рассмотрим поток f и соответствующую уровневую сеть N_f. Предположим, что блокирующий поток b_f найден в N_f и после увеличения построен новый поток f'. Тогда $d_{f'}(t) \geq d_f(t) + 1$.

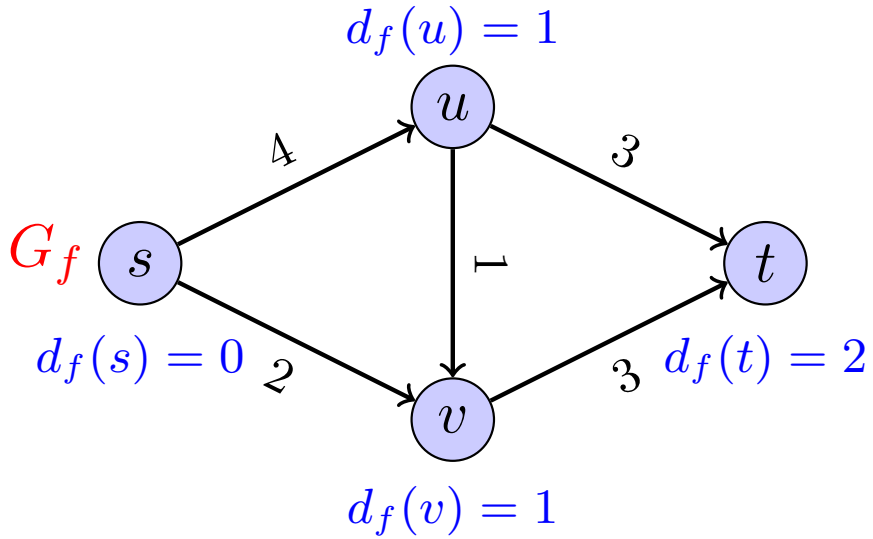
Расстояние $d_f(t)$ возрастает по крайней мере на 1 на каждой фазе

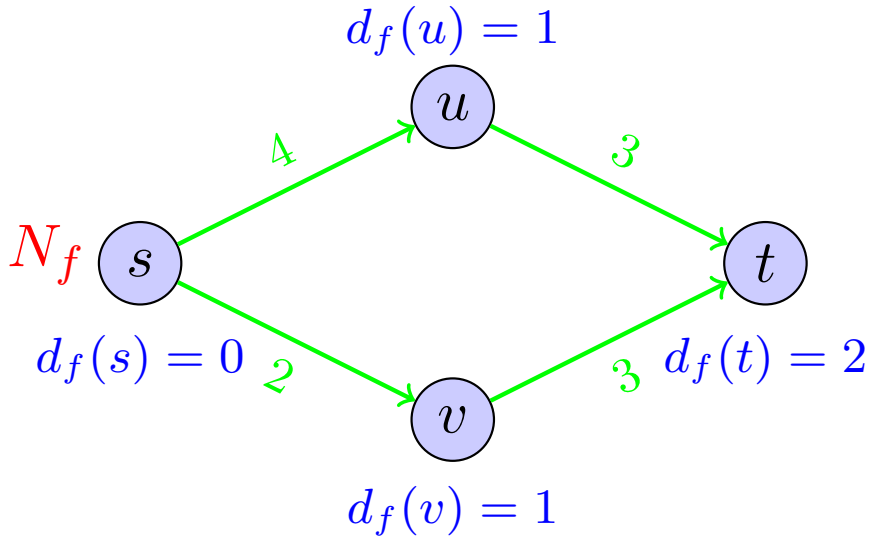
Theorem

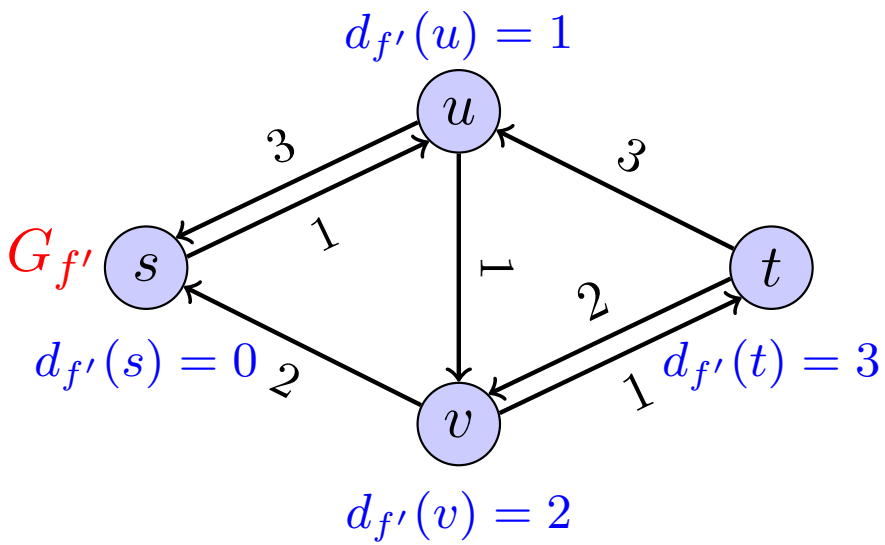
Рассмотрим поток f и соответствующую уровневую сеть N_f. Предположим, что блокирующий поток b_f найден в N_f и после увеличения построен новый поток f'. Тогда $d_{f'}(t) \geq d_f(t) + 1$.

- Отметим: Если увеличение потока f осуществляется только вдоль одного кратчайшего пути, скажем, $s \to v \to t$ в следующем примере, то $d_{f'}(t) = d_f(t) = 2$. Напротив, когда увеличение потока f осуществляется вдоль всех кратчайших путей, то $d_{f'}(t) \geq d_f(t) + 1$.

Пример


$d_f(u) = 1$
u
4
3
G_f
s
1
t
$d_f(s) = 0$
2
3
$d_f(t) = 2$
v
$d_f(v) = 1$


$d_f(u) = 1$
u
4
3
N_f
s
t
$d_f(s) = 0$
2
3
$d_f(t) = 2$
v
$d_f(v) = 1$


$d_{f'}(u) = 1$
u
3
3
$G_{f'}$
s
1
1
2
t
$d_{f'}(s) = 0$
2
1
$d_{f'}(t) = 3$
v
$d_{f'}(v) = 2$

Доказательство.

- Предположим противное, что выполняется $d_{f'}(t) = d_f(t) = r$. Пусть $p = (s, v_1, ..., v_{r-1}, t)$ кратчайший путь до t в $G_{f'}$. Тогда

$$\begin{aligned} d_f(t) &\leq d_f(v_{r-1}) + 1 \\ &..... \\ &\leq d_f(s) + r = r \end{aligned}$$

Доказательство.

- Предположим противное, что выполняется $d_{f'}(t) = d_f(t) = r$. Пусть $p = (s, v_1, ..., v_{r-1}, t)$ кратчайший путь до t в $G_{f'}$. Тогда

$$\begin{aligned} d_f(t) &\leq d_f(v_{r-1}) + 1 \\ &..... \\ &\leq d_f(s) + r = r \end{aligned}$$

- По нашему предположению $d_f(t) = r$, все "$\leq$" в формуле выше должны быть "$=$". Из равенства $d_f(v_{i+1}) = d_f(v_i) + 1$ следует, что ребро (v_i, v_{i+1}) должно присутствовать в G_f (Иначе (v_i, v_{i+1}) должно генерироваться как обратное к (v_{i+1}, v_i), и т.о., $d_f(v_i) = d_f(v_{i+1}) + 1$.).

Доказательство.

- Предположим противное, что выполняется $d_{f'}(t) = d_f(t) = r$. Пусть $p = (s, v_1, ..., v_{r-1}, t)$ кратчайший путь до t в $G_{f'}$. Тогда

$$\begin{aligned} d_f(t) &\leq d_f(v_{r-1}) + 1 \\ &..... \\ &\leq d_f(s) + r = r \end{aligned}$$

- По нашему предположению $d_f(t) = r$, все "$\leq$" в формуле выше должны быть "$=$". Из равенства $d_f(v_{i+1}) = d_f(v_i) + 1$ следует, что ребро (v_i, v_{i+1}) должно присутствовать в G_f (Иначе (v_i, v_{i+1}) должно генерироваться как обратное к (v_{i+1}, v_i), и т.о., $d_f(v_i) = d_f(v_{i+1}) + 1$.).
- Т.о., p является также путем в G_f. Более того, p должен быть кратчайшим путем в G_f так как p имеет длину r и $d_f(t) = r$.

Доказательство.

- Предположим противное, что выполняется $d_{f'}(t) = d_f(t) = r$. Пусть $p = (s, v_1, ..., v_{r-1}, t)$ кратчайший путь до t в $G_{f'}$. Тогда

$$\begin{aligned} d_f(t) &\leq d_f(v_{r-1}) + 1 \\ &..... \\ &\leq d_f(s) + r = r \end{aligned}$$

- По нашему предположению $d_f(t) = r$, все "$\leq$" в формуле выше должны быть "$=$". Из равенства $d_f(v_{i+1}) = d_f(v_i) + 1$ следует, что ребро (v_i, v_{i+1}) должно присутствовать в G_f (Иначе (v_i, v_{i+1}) должно генерироваться как обратное к (v_{i+1}, v_i), и т.о., $d_f(v_i) = d_f(v_{i+1}) + 1$.).
- Т.о., p является также путем в G_f. Более того, p должен быть кратчайшим путем в G_f так как p имеет длину r и $d_f(t) = r$.
- Напомним, что $G_{f'}$ построена из G_f посредством насыщения всех кратчайших путей (включая p) в G_f. Т.о., по крайней мере одно ребро в p полностью насыщается и не должно присутствовать в $G_{f'}$. Получили противоречие с предположением, что p является путем в $G_{f'}$.

Обобщения задачи о максимальном потоке

1. Максимальный поток в сети с неориентированными ребрами;

1. Максимальный поток в сети с неориентированными ребрами;
2. Произвольное число источников и/или стоков;

1. Максимальный поток в сети с неориентированными ребрами;
2. Произвольное число источников и/или стоков;
3. Нижняя граница на пропускную способность;

Обобщения задачи о максимальном потоке

1. Максимальный поток в сети с неориентированными ребрами;
2. Произвольное число источников и/или стоков;
3. Нижняя граница на пропускную способность;
4. Ограничение пропускной способности вершин

Обобщение 1: Неориентированные рёбра

- Превращаем неориентированный граф G в ориентированный граф G':

Обобщение 1: Неориентированные рёбра

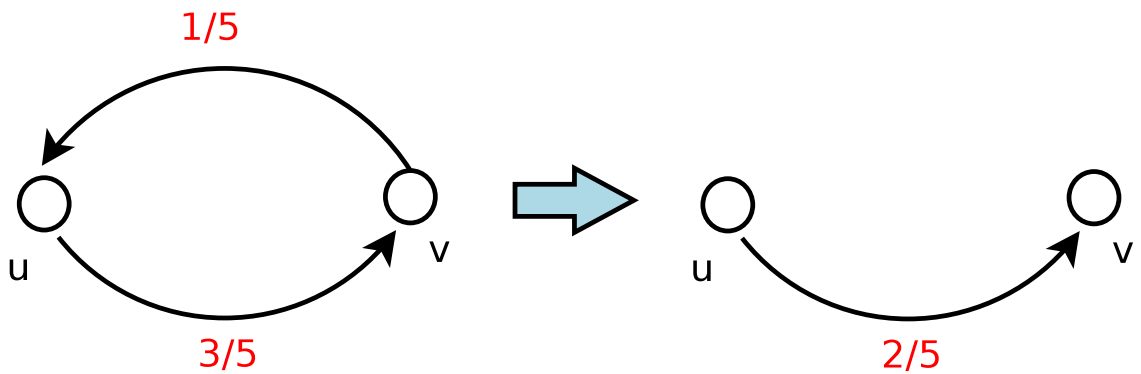
- Превращаем неориентированный граф G в ориентированный граф G':
  1. ребра: для каждого ребра (u, v) из G, создается пара ориентированных рёбер $e = (u, v)$ и $e' = (v, u)$ в G';

Обобщение 1: Неориентированные рёбра

- Превращаем неориентированный граф G в ориентированный граф G':
  1. ребра: для каждого ребра (u, v) из G, создается пара ориентированных рёбер $e = (u, v)$ и $e' = (v, u)$ в G';
  2. пропускная способность: полагаем $C(e') = C(e)$.

Обобщение 1: Неориентированные рёбра

- Превращаем неориентированный граф G в ориентированный граф G':
  1. ребра: для каждого ребра (u, v) из G, создается пара ориентированных рёбер $e = (u, v)$ и $e' = (v, u)$ в G';
  2. пропускная способность: полагаем $C(e') = C(e)$.
- Алгоритм: сначала вычисляем максимальный поток f' для сети G'; затем преобразовываем f' в f.



Note: a/b means f(e) = a, and capacity C(e)=b.

Обобщение 1: Неориентированные рёбра

Ключевое наблюдение: существует максимальный поток, который использует не более одного из двух противоположных ребер. Формально имеем:

Обобщение 1: Неориентированные рёбра

Ключевое наблюдение: существует максимальный поток, который использует не более одного из двух противоположных ребер. Формально имеем:

Theorem

Существует максимальный поток f для сети G, где $f(u, v) = 0$ или $f(v, u) = 0$.

Обобщение 1: Неориентированные рёбра

Ключевое наблюдение: существует максимальный поток, который использует не более одного из двух противоположных ребер. Формально имеем:

Theorem

Существует максимальный поток f для сети G, где $f(u,v) = 0$ или $f(v,u) = 0$.

Доказательство.

- Предположим f' — это максимальный поток для сети G', где $f'(u,v) > 0$ и $f'(v,u) > 0$. Заменим f' на f следующим образом:

Обобщение 1: Неориентированные рёбра

Ключевое наблюдение: существует максимальный поток, который использует не более одного из двух противоположных ребер. Формально имеем:

Theorem

Существует максимальный поток f для сети G, где $f(u,v) = 0$ или $f(v,u) = 0$.

Доказательство.

- Предположим f' — это максимальный поток для сети G', где $f'(u,v) > 0$ и $f'(v,u) > 0$. Заменим f' на f следующим образом:
- Пусть $\delta = \min\{f'(u,v), f'(v,u)\}$.

Обобщение 1: Неориентированные рёбра

Ключевое наблюдение: существует максимальный поток, который использует не более одного из двух противоположных ребер. Формально имеем:

Theorem

Существует максимальный поток f для сети G, где $f(u,v) = 0$ или $f(v,u) = 0$.

Доказательство.

- Предположим f' — это максимальный поток для сети G', где $f'(u,v) > 0$ и $f'(v,u) > 0$. Заменим f' на f следующим образом:
- Пусть $\delta = \min\{f'(u,v), f'(v,u)\}$.
- Положим $f(u,v) = f'(u,v) - \delta$, и $f(v,u) = f'(v,u) - \delta$.

Обобщение 1: Неориентированные рёбра

Ключевое наблюдение: существует максимальный поток, который использует не более одного из двух противоположных ребер. Формально имеем:

Theorem

Существует максимальный поток f для сети G, где $f(u,v) = 0$ или $f(v,u) = 0$.

Доказательство.

- Предположим f' — это максимальный поток для сети G', где $f'(u,v) > 0$ и $f'(v,u) > 0$. Заменим f' на f следующим образом:
- Пусть $\delta = \min\{f'(u,v), f'(v,u)\}$.
- Положим $f(u,v) = f'(u,v) - \delta$, и $f(v,u) = f'(v,u) - \delta$.
- f имеет то же значение, что и f', и, таким образом, оптимально. В тоже время $f(u,v) = 0$ или $f(v,u) = 0$.

Обобщение 2: Произвольное число источников и/или стоков

Циркуляция

ВХОД: сеть $G =< V, E >$, каждое ребро e имеет пропускную способность $C(e) > 0$, несколько источников s_i и стоков t_j. Сток t_j имеет спрос $d_j > 0$, а источник s_i имеет запас (обозначаемый как $d_i < 0$). Для простоты мы определим $d_v = 0$ для других узлов. Таким образом, мы имеем $\sum_i d_i = 0$. Пусть $D = \sum_{d_v>0} d_v$ — *суммарный спрос.*

Обобщение 2: Произвольное число источников и/или стоков

Циркуляция

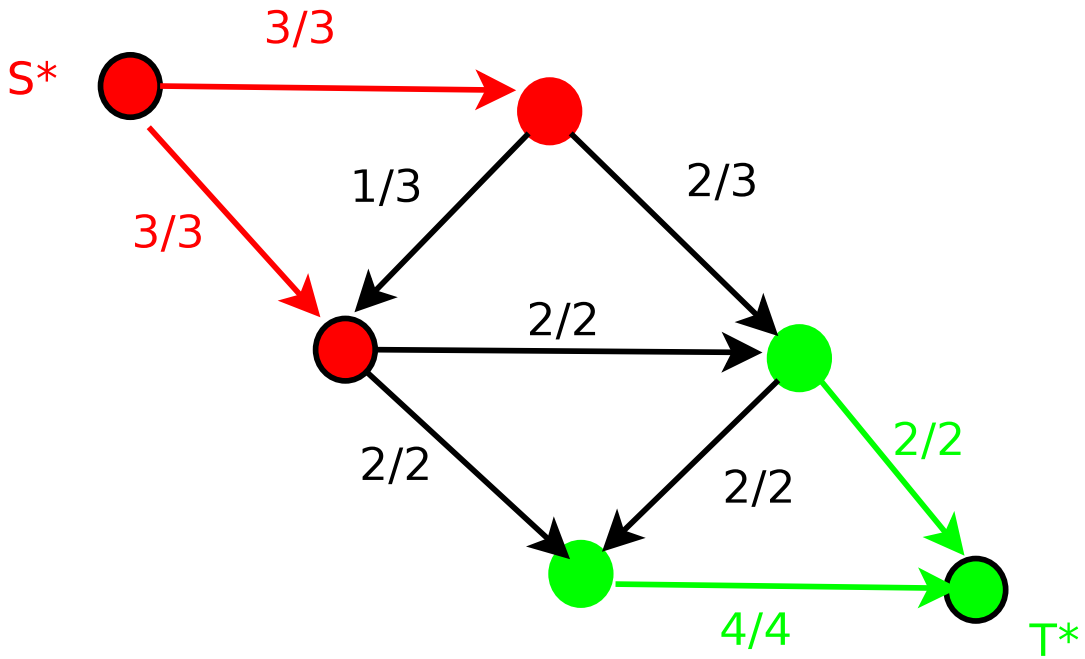
ВХОД: сеть $G =< V, E >$, каждое ребро e имеет пропускную способность $C(e) > 0$, несколько источников s_i и стоков t_j. Сток t_j имеет спрос $d_j > 0$, а источник s_i имеет запас (обозначаемый как $d_i < 0$). Для простоты мы определим $d_v = 0$ для других узлов. Таким образом, мы имеем $\sum_i d_i = 0$. Пусть $D = \sum_{d_v>0} d_v$ — *суммарный спрос.*
ВЫХОД: циркуляция f, удовлетворяющая всех потребителей с использованием доступного предложения, т. е.

1. (Ограничение о пропускной способности): $0 \leq f(e) \leq C(e)$;
2. (Ограничение по спросу): $f^{in}(v) - f^{out}(v) = d_v$;

Отметим, что: Циркуляция отличается от Многопродуктовой задачи тем, что существует ТОЛЬКО один товар. Другими словами, приемник может принять товар из любого источника. И наоборот, в Многопродуктовой задаче t_i принимает только товар k_i из s_i.

Отметим, что: Циркуляция отличается от Многопродуктовой задачи тем, что существует ТОЛЬКО один товар. Другими словами, приемник может принять товар из любого источника. И наоборот, в Многопродуктовой задаче t_i принимает только товар k_i из s_i.

Сведение: строится сеть G' путем добавления супериточника s^*, из которого идут ребра в каждый источник s_i с емкостью $C(s^*, s_i) = -d_i$. Аналогично, добавляется суперсток t^*, к которому идут ребра из каждого стока t_j с емкостью $C(t_j, t^*) = d(t_j)$.



Note: a/b means f(e) = a, and capacity C(e)=b.

Theorem

Существует допустимое решение задачи Циркуляция, *если величина максимального потока из* s^* *в* t^* *в* G' *совпадает с* D.

Theorem

Существует допустимое решение задачи Циркуляция, *если величина максимального потока из* s^* *в* t^* *в* G' *совпадает с* D.

Доказательство.

- $\Leftarrow$:

  Просто удалим все ребра (s^*, s_i) и (t_j, t^*).

Theorem

Существует допустимое решение задачи Циркуляция, *если величина максимального потока из* s^* *в* t^* *в* G' *совпадает с* D.

Доказательство.

- $\Leftarrow$:

  Просто удалим все ребра (s^*, s_i) и (t_j, t^*).

- $\Rightarrow$:

  1. Построим поток f так: $f(s^*, s_i) = -d_i$ и $f(t_j, t^*) = d_j$.

Theorem

Существует допустимое решение задачи Циркуляция, *если величина максимального потока из* s^* *в* t^* *в* G' *совпадает с* D.

Доказательство.

- $\Leftarrow$:

  Просто удалим все ребра (s^*, s_i) и (t_j, t^*).

- $\Rightarrow$:

  1. Построим поток f так: $f(s^*, s_i) = -d_i$ и $f(t_j, t^*) = d_j$.
  2. Рассмотрим разрез $(S, \bar{S})$, где $S = \{s^*\}$.

Theorem

Существует допустимое решение задачи Циркуляция, *если величина максимального потока из* s^* *в* t^* *в* G' *совпадает с* D.

Доказательство.

- $\Leftarrow$:

  Просто удалим все ребра (s^*, s_i) и (t_j, t^*).

- $\Rightarrow$:

  1. Построим поток f так: $f(s^*, s_i) = -d_i$ и $f(t_j, t^*) = d_j$.
  2. Рассмотрим разрез $(S, \bar{S})$, где $S = \{s^*\}$.
  3. Имеем $C(S, \bar{S}) = D$. Таким образом, f является максимальным потоком, поскольку он достигает максимального значения.

Обобщение 3: Нижняя граница на пропускную способность

ВХОД: сеть $G = (V, E)$, каждое ребро e имеет верхнюю границу на пропускную способность $C(e) > 0$, и нижнюю границу на пропускную способность $L(e) > 0$, несколько источников s_i и стоков t_j. Сток t_j имеет спрос $d_j > 0$, а источник s_i имеет запас (обозначаемый как $d_i < 0$). Для простоты мы определим $d_v = 0$ для других узлов. Таким образом, мы имеем $\sum_i d_i = 0$. Пусть $D = \sum_{d_v > 0} d_v$ — *суммарный спрос.*

Обобщение 3: Нижняя граница на пропускную способность

ВХОД: сеть $G = (V, E)$, каждое ребро e имеет верхнюю границу на пропускную способность $C(e) > 0$, и нижнюю границу на пропускную способность $L(e) > 0$, несколько источников s_i и стоков t_j. Сток t_j имеет спрос $d_j > 0$, а источник s_i имеет запас (обозначаемый как $d_i < 0$). Для простоты мы определим $d_v = 0$ для других узлов. Таким образом, мы имеем $\sum_i d_i = 0$. Пусть $D = \sum_{d_v > 0} d_v$ — *суммарный спрос.*

ВЫХОД: циркуляция f удовлетворяющая всех потребителей с использованием доступного предложения, т. е.,

Обобщение 3: Нижняя граница на пропускную способность

ВХОД: сеть $G = (V, E)$, каждое ребро e имеет верхнюю границу на пропускную способность $C(e) > 0$, и нижнюю границу на пропускную способность $L(e) > 0$, несколько источников s_i и стоков t_j. Сток t_j имеет спрос $d_j > 0$, а источник s_i имеет запас (обозначаемый как $d_i < 0$). Для простоты мы определим $d_v = 0$ для других узлов. Таким образом, мы имеем $\sum_i d_i = 0$. Пусть $D = \sum_{d_v > 0} d_v$ — *суммарный спрос.*

ВЫХОД: циркуляция f удовлетворяющая всех потребителей с использованием доступного предложения, т. е.,

1. (Ограничение о пропускной способности): $L(e) \leq f(e) \leq C(e)$;

Обобщение 3: Нижняя граница на пропускную способность

ВХОД: сеть $G = (V, E)$, каждое ребро e имеет верхнюю границу на пропускную способность $C(e) > 0$, и нижнюю границу на пропускную способность $L(e) > 0$, несколько источников s_i и стоков t_j. Сток t_j имеет спрос $d_j > 0$, а источник s_i имеет запас (обозначаемый как $d_i < 0$). Для простоты мы определим $d_v = 0$ для других узлов. Таким образом, мы имеем $\sum_i d_i = 0$. Пусть $D = \sum_{d_v > 0} d_v$ — *суммарный спрос.*

ВЫХОД: циркуляция f удовлетворяющая всех потребителей с использованием доступного предложения, т. е.,

1. (Ограничение о пропускной способности): $L(e) \leq f(e) \leq C(e)$;
2. (Ограничение по спросу): $f^{in}(v) - f^{out}(v) = d_v$;

Обобщение 3: Нижняя граница на пропускную способность

ВХОД: сеть $G = (V, E)$, каждое ребро e имеет верхнюю границу на пропускную способность $C(e) > 0$, и нижнюю границу на пропускную способность $L(e) > 0$, несколько источников s_i и стоков t_j. Сток t_j имеет спрос $d_j > 0$, а источник s_i имеет запас (обозначаемый как $d_i < 0$). Для простоты мы определим $d_v = 0$ для других узлов. Таким образом, мы имеем $\sum_i d_i = 0$. Пусть $D = \sum_{d_v > 0} d_v$ — *суммарный спрос.*

ВЫХОД: циркуляция f удовлетворяющая всех потребителей с использованием доступного предложения, т. е.,

1. (Ограничение о пропускной способности): $L(e) \leq f(e) \leq C(e)$;
2. (Ограничение по спросу): $f^{in}(v) - f^{out}(v) = d_v$;

Замечание: для каждого ребра e, мы требуем чтобы

Обобщение 3: Нижняя граница на пропускную способность

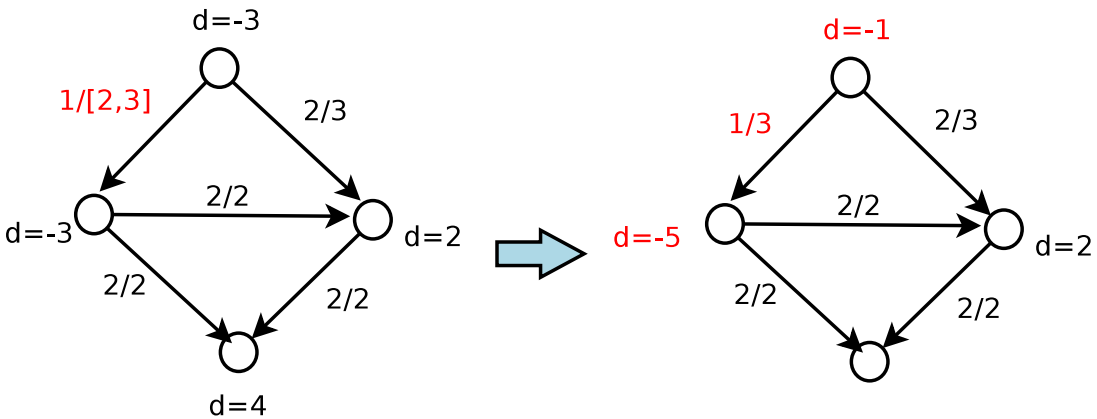
Сведение:

1. Построим исходную циркуляцию f_0, установив $f_0(e) = L(e)$.

Обобщение 3: Нижняя граница на пропускную способность

Сведение:

1. Построим исходную циркуляцию f_0, установив $f_0(e) = L(e)$.
2. Улудшим f_0 до циркуляции f'. Для этого решим еще одну задачу Циркуляция: создадим сеть G' без нижних границ на пропускную способность и со спросами: $d'_v = d_v - L(w, v)$.



Note: a/[l,b] means f(e) = a, and capacity L(e)=l, and C(e)=b.

Обобщение 3: Нижняя граница на пропускную способность

Theorem

Существует циркуляция f в G (с нижними границами) тогда и только тогда, когда существует циркуляция f' в G'.

Обобщение 3: Нижняя граница на пропускную способность

Theorem

Существует циркуляция f в G (с нижними границами) тогда и только тогда, когда существует циркуляция f' в G'.

Доказательство.

Построим f': $f'(e) = f(e) + L_e$. Легко проверить, что все ограничения выполняются. $\square$

Обобщение 4: Ограничение пропускной способности вершин

Каждую вершину v с ограниченной пропускной способностью c_v расщепляем на две вершины v_{in} и v_{out}. Все рёбра, до расщепления входящие в v, теперь входят в v_{in}. Все рёбра, до расщепления исходящие из v, теперь исходят из v_{out}. Добавляем ребро (v_{in}, v_{out}) с пропускной способностью c_v.

Приложения Задачи о максимальном потоке

Формулировка задачи:

1. Как определить s и t? Иногда требуются супериcточник s^* и суперсток t^*.

Формулировка задачи:

1. Как определить s и t? Иногда требуются супериcточник s^* и суперсток t^*.
2. Как определить пропускные способности L_e и C_e?

Формулировка задачи:

1. Как определить s и t? Иногда требуются супериcточник s^* и суперсток t^*.
2. Как определить пропускные способности L_e и C_e?
3. Иногда минимальный разрез более подходит в качестве критерия, чем максимальный поток.

Формулировка задачи:

1. Как определить s и t? Иногда требуются супериcточник s^* и суперсток t^*.
2. Как определить пропускные способности L_e и C_e?
3. Иногда минимальный разрез более подходит в качестве критерия, чем максимальный поток.
4. Необходимо доказать, что максимальный поток соответствует решению исходной задачи.

Формулировка задачи:

1. Как определить s и t? Иногда требуются супериcточник s^* и суперсток t^*.
2. Как определить пропускные способности L_e и C_e?
3. Иногда минимальный разрез более подходит в качестве критерия, чем максимальный поток.
4. Необходимо доказать, что максимальный поток соответствует решению исходной задачи.

Примечание: большинство проблем используют свойство, что существует максимальный целочисленный поток, если существует максимальный поток.

Обобщение: задача с несколькими оптимальными решениями

Несколько оптимальных решений

- В некоторых случаях может быть несколько оптимальных решений, то есть несколько потоков с одинаковым максимальным значением потока.

Несколько оптимальных решений

- В некоторых случаях может быть несколько оптимальных решений, то есть несколько потоков с одинаковым максимальным значением потока.
- Кроме того, эти максимальные потоки могут иметь совпадающие потоки по некоторым ребрам. Другими словами, эти ребра являются «необходимыми» ребрами.

Несколько оптимальных решений

- В некоторых случаях может быть несколько оптимальных решений, то есть несколько потоков с одинаковым максимальным значением потока.
- Кроме того, эти максимальные потоки могут иметь совпадающие потоки по некоторым ребрам. Другими словами, эти ребра являются «необходимыми» ребрами.
- Иногда нам нужно перечислить все эти оптимальные решения или получить небольшую выборку из этих оптимальных решений.