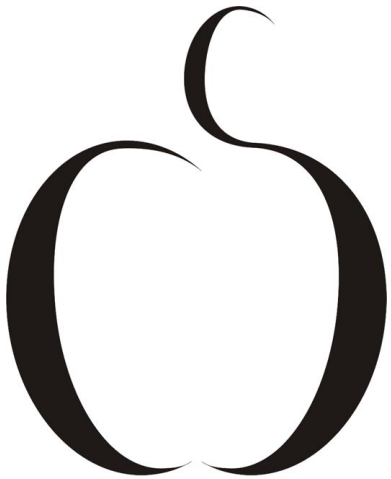
Алгоритмы для NP-трудных задач

Лекция 2: Обзор

А. Куликов

Computer Science клуб при ПОМИ
http://logic.pdmi.ras.ru/~infclub/



План лекции

1. FPT алгоритмы
   - Покрытие точек прямыми
   - Вершинное покрытие

План лекции

1. FPT алгоритмы
   - Покрытие точек прямыми
   - Вершинное покрытие
2. Приближённые алгоритмы
   - Задача о вершинном покрытии
   - Задача о коммивояжёре

Цель первых двух лекций

Привести несколько красивых алгоритмов, не особо вдаваясь в определения и доказательства. Все формальности будут дальше в курсе.

План лекции

1. FPT алгоритмы
   - Покрытие точек прямыми
   - Вершинное покрытие
2. Приближённые алгоритмы
   - Задача о вершинном покрытии
   - Задача о коммивояжёре

FPT алгоритмы для NP-трудных задач

Алгоритмы, находящие точное решение для данной задачи за время

$$f(k)n^c,$$

где

- n — **размер входа,**
- c — **константа,**
- k — **некоторый параметр,**
- f — **произвольная функция.**

Вершинное покрытие и доминирующее множество

Определение

Вершинное покрытие и доминирующее множество

Определение

- (Параметризованная) задача о вершинном покрытии (parameterized vertex cover problem) заключается в нахождении по данному графу $G = (V, E)$ и числу k такого множества вершин $V' \subseteq V$, что $|V'| \leq k$ и для любого ребра $(u, v) \in E$, хотя бы одна из вершин u и v содержится в V'.

Вершинное покрытие и доминирующее множество

Определение

- (Параметризованная) задача о вершинном покрытии (parameterized vertex cover problem) заключается в нахождении по данному графу $G = (V, E)$ и числу k такого множества вершин $V' \subseteq V$, что $|V'| \leq k$ и для любого ребра $(u, v) \in E$, хотя бы одна из вершин u и v содержится в V'.
- (Параметризованная) задача о доминирующем множестве (parameterzied dominating set problem) заключается в нахождении по данному графу $G = (V, E)$ и числу k такого множества вершин $V' \subseteq V$, что для любой вершины $u \in V$ найдётся такая вершина $v \in V'$, что $(u, v) \in E$.

Сравнение алгоритмов

Сравнение алгоритмов

- Наивный алгоритм для задачи о вершинном покрытии имеет время работы $2^k n$, для задачи о доминирующем множестве — n^{k+1}.

Сравнение алгоритмов

- Наивный алгоритм для задачи о вершинном покрытии имеет время работы $2^k n$, для задачи о доминирующем множестве — n^{k+1}.
- Отношение $n^{k+1}/2^k n$ для различных значений k и n.

	$n = 50$	$n = 100$	$n = 150$
$k = 2$	625	2 500	5 625
$k = 3$	15 625	125 000	421 875
$k = 5$	390 625	6 255 000	31 640 625
$k = 10$	1.9×10^{12}	9.8×10^{14}	3.7×10^{16}
$k = 20$	1.8×10^{26}	9.5×10^{31}	2.1×10^{35}

Fixed parameter tractability

Определение

Fixed parameter tractability

Определение

- Параметризованной задачей (parameterized problem) L называется подмножество $\Sigma^* \times N$ для некоторого конечного алфавита Σ. Входом задачи является пара (x, k), k называется параметром (parameter).

Fixed parameter tractability

Определение

- Параметризованной задачей (parameterized problem) L называется подмножество $\Sigma^* \times N$ для некоторого конечного алфавита Σ. Входом задачи является пара (x, k), k называется параметром (parameter).
- Параметризованная задача L называется fixed parameter tractable (FPT), если принадлежность пары (x, k) языку L может быть проверена за время $f(k)|x|^c$, где c — константа (не зависящая ни от k, ни от n), а f — произвольная функция.

Ядро

Определение

Ядром (kernel, kernelization algorithm) параметризованной задачи $L \subseteq \Sigma^* \times N$ называется алгоритм, который по входу $(x, k) \in \Sigma^* \times N$ выдаёт за полиномиальное от $|x| + k$ время $(x', k') \in \Sigma^* \times N$, так что

Ядро

Определение

Ядром (kernel, kernelization algorithm) параметризованной задачи $L \subseteq \Sigma^* \times N$ называется алгоритм, который по входу $(x, k) \in \Sigma^* \times N$ выдаёт за полиномиальное от $|x| + k$ время $(x', k') \in \Sigma^* \times N$, так что

- $(x, k) \in L$ тогда и только тогда, когда $(x', k') \in L$;

Ядро

Определение

Ядром (kernel, kernelization algorithm) параметризованной задачи $L \subseteq \Sigma^* \times N$ называется алгоритм, который по входу $(x, k) \in \Sigma^* \times N$ выдаёт за полиномиальное от $|x| + k$ время $(x', k') \in \Sigma^* \times N$, так что

- $(x, k) \in L$ тогда и только тогда, когда $(x', k') \in L$;
- $|x'| + k' \leq g(k)$, где g — произвольная вычислимая функция; g называется размером (size) ядра.

Ядро

Определение

Ядром (kernel, kernelization algorithm) параметризованной задачи $L \subseteq \Sigma^* \times N$ называется алгоритм, который по входу $(x, k) \in \Sigma^* \times N$ выдаёт за полиномиальное от $|x| + k$ время $(x', k') \in \Sigma^* \times N$, так что

- $(x, k) \in L$ тогда и только тогда, когда $(x', k') \in L$;
- $|x'| + k' \leq g(k)$, где g — произвольная вычислимая функция; g называется размером (size) ядра.

Замечание

Легко видеть, что если у параметризованной задачи есть ядро, то она принадлежат классу FPT.

Ядро

Определение

Ядром (kernel, kernelization algorithm) параметризованной задачи $L \subseteq \Sigma^* \times N$ называется алгоритм, который по входу $(x, k) \in \Sigma^* \times N$ выдаёт за полиномиальное от $|x| + k$ время $(x', k') \in \Sigma^* \times N$, так что

- $(x, k) \in L$ тогда и только тогда, когда $(x', k') \in L$;
- $|x'| + k' \leq g(k)$, где g — произвольная вычислимая функция; g называется размером (size) ядра.

Замечание

Легко видеть, что если у параметризованной задачи есть ядро, то она принадлежат классу FPT. На самом деле, верно и обратное.

FPT $\Rightarrow$ ядро

FPT $\Rightarrow$ ядро

- Итак, пусть существует алгоритм A, проверяющий принадлежность пары (x, k) языку L за время $f(k)n^c$ $(n = |x|)$.

FPT $\Rightarrow$ ядро

- Итак, пусть существует алгоритм A, проверяющий принадлежность пары (x, k) языку L за время $f(k)n^c$ $(n = |x|)$.
- Если $f(k) \leq n$, тогда проверим принадлежность алгоритмом A за время $f(k)n^c \leq n^{c+1}$ и вернём тривиальную пару (x', k') из или не из языка L.

FPT $\Rightarrow$ ядро

- Итак, пусть существует алгоритм A, проверяющий принадлежность пары (x, k) языку L за время $f(k)n^c$ $(n = |x|)$.
- Если $f(k) \leq n$, тогда проверим принадлежность алгоритмом A за время $f(k)n^c \leq n^{c+1}$ и вернём тривиальную пару (x', k') из или не из языка L.
- Если же $f(k) > n$, то просто вернём (x, k).

FPT $\Rightarrow$ ядро

- Итак, пусть существует алгоритм A, проверяющий принадлежность пары (x, k) языку L за время $f(k)n^c$ $(n = |x|)$.
- Если $f(k) \leq n$, тогда проверим принадлежность алгоритмом A за время $f(k)n^c \leq n^{c+1}$ и вернём тривиальную пару (x', k') из или не из языка L.
- Если же $f(k) > n$, то просто вернём (x, k).
- Получили ядро размера $f(k)$.

План лекции

Покрытие точек прямыми

Задача

Покрыть данное множество P из n точек k прямыми.

Покрытие точек прямыми

Задача

Покрыть данное множество P из n точек k прямыми.

Замечание

Можно считать, что каждая прямая из решения проходит хотя бы через две точки множества P, поэтому кандидатов не более n^2.

Покрытие точек прямыми

Задача

Покрыть данное множество P из n точек k прямыми.

Замечание

Можно считать, что каждая прямая из решения проходит хотя бы через две точки множества P, поэтому кандидатов не более n^2.

Правило упрощения

Если прямая проходит через множество S из более, чем k точек, $(P, k) \Rightarrow (P \setminus S, k - 1)$.

Покрытие точек прямыми

Задача

Покрыть данное множество P из n точек k прямыми.

Замечание

Можно считать, что каждая прямая из решения проходит хотя бы через две точки множества P, поэтому кандидатов не более n^2.

Правило упрощения

Если прямая проходит через множество S из более, чем k точек, $(P, k) \Rightarrow (P \setminus S, k - 1)$.

Анализ

Если данное правило не применимо, а точек все ещё больше, чем k^2, то решения, очевидно, нет. Получили ядро размера k^2.

План лекции

Ядро для вершинного покрытия

Общая стратегия

Привести несколько правил упрощения и показать, что если ни одно из них ко входному графу не применимо, а его размер всё ещё больше $f(k)$, то ответ уже очевиден.

Ядро для вершинного покрытия

Общая стратегия

Привести несколько правил упрощения и показать, что если ни одно из них ко входному графу не применимо, а его размер всё ещё больше $f(k)$, то ответ уже очевиден.

Правила упрощения для задачи о вершинном покрытии

Правило 1. Если v — изолированная вершина, $(G, k) \Rightarrow (G \setminus v, k)$.

Правило 2. Если $d(v) > k$, $(G, k) \Rightarrow (G \setminus v, k - 1)$.

Анализ

Анализ

Допустим, ни одно из двух правил не применимо.

Анализ

Анализ

Допустим, ни одно из двух правил не применимо.

- Если $|V(G)| > k(k+1)$, то решения нет (каждая вершина должна быть соседом хотя бы одной вершины из покрытия).

Анализ

Анализ

Допустим, ни одно из двух правил не применимо.

- Если $|V(G)| > k(k+1)$, то решения нет (каждая вершина должна быть соседом хотя бы одной вершины из покрытия).
- В противном случае $|V(G)| \leq k(k+1)$ и ядро построено.

Ядро

Ядро

- Можно рассматривать как полиномиальную по времени предобработку.

Ядро

- Можно рассматривать как полиномиальную по времени предобработку.
- Некоторые алгоритмы используют много простых правил упрощения. Анализ же размера ядра довольно сложен.

Ядро

- Можно рассматривать как полиномиальную по времени предобработку.
- Некоторые алгоритмы используют много простых правил упрощения. Анализ же размера ядра довольно сложен.
- В то время как есть алгоритмы, основанные на довольно элегантных идеях (crown decomposition, лемма о подсолнухе).

CD-разбиение

Определение

CD-разбиением (crown decomposition) графа называется такое разбиение вершин на три множества C, H, B, что

CD-разбиение

Определение

CD-разбиением (crown decomposition) графа называется такое разбиение вершин на три множества C, H, B, что

- C — независимое множество;

CD-разбиение

Определение

CD-разбиением (crown decomposition) графа называется такое разбиение вершин на три множества C, H, B, что

- C — независимое множество;
- между C и B нет рёбер;

CD-разбиение

Определение

CD-разбиением (crown decomposition) графа называется такое разбиение вершин на три множества C, H, B, что

- C — независимое множество;
- между C и B нет рёбер;
- между C и H есть паросочетание, покрывающее H.

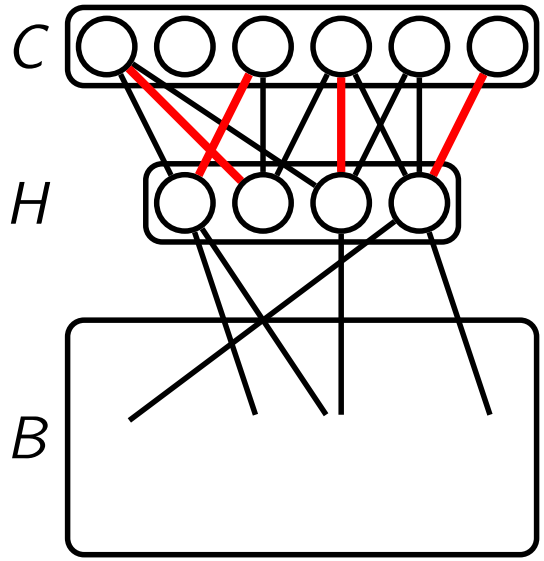
CD-разбиение

Определение

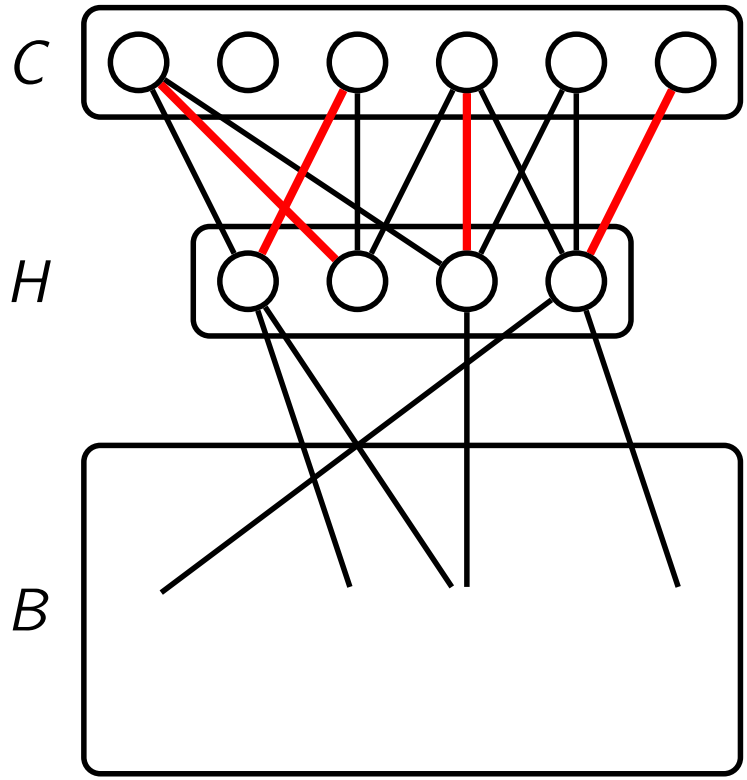
CD-разбиением (crown decomposition) графа называется такое разбиение вершин на три множества C, H, B, что

- C — независимое множество;
- между C и B нет рёбер;
- между C и H есть паросочетание, покрывающее H.

Пример



Правило упрощения для задачи о вершинном покрытии



Правило упрощения для задачи о вершинном покрытии

Паросочетание должно быть покрыто и для этой цели нет смысла использовать вершины множества C. Поэтому можно считать, что оно покрыто вершинами H, и $(G, k) \Rightarrow (G \setminus (H \cup C), k - |H|)$.

Основная лемма

Лемма

По данному графу G, не содержащему изолированных вершин, и параметру k можно за полиномиальное время

Основная лемма

Лемма

По данному графу G, не содержащему изолированных вершин, и параметру k можно за полиномиальное время

- либо найти паросочетание размера $k+1$($\Rightarrow$ решения нет);

Основная лемма

Лемма

По данному графу G, не содержащему изолированных вершин, и параметру k можно за полиномиальное время

- либо найти паросочетание размера $k+1$($\Rightarrow$ решения нет);
- либо найти CD-разбиение ($\Rightarrow$ упрощение);

Основная лемма

Лемма

По данному графу G, не содержащему изолированных вершин, и параметру k можно за полиномиальное время

- либо найти паросочетание размера $k+1$($\Rightarrow$ решения нет);
- либо найти CD-разбиение ($\Rightarrow$ упрощение);
- или заключить, что граф содержит не более, чем $4k$ вершин ($\Rightarrow$ ядро).

Основная лемма

Лемма

По данному графу G, не содержащему изолированных вершин, и параметру k можно за полиномиальное время

- либо найти паросочетание размера $k+1$($\Rightarrow$ решения нет);
- либо найти CD-разбиение ($\Rightarrow$ упрощение);
- или заключить, что граф содержит не более, чем $4k$ вершин ($\Rightarrow$ ядро).

Лемма

Это даёт ядро размера $4k$ для задачи о вершинном покрытии.

Доказательство леммы

Доказательство леммы

- Найдём (жадно) максимальное по включению паросочетание в графе G.

Доказательство леммы

- Найдём (жадно) максимальное по включению паросочетание в графе G.
- Если его размер хотя бы $k + 1$, то решения, очевидно, нет.

Доказательство леммы

- Найдём (жадно) максимальное по включению паросочетание в графе G.
- Если его размер хотя бы $k + 1$, то решения, очевидно, нет.
- Пусть X есть множество вершин найденного паросочетания, I — оставшиеся вершины.

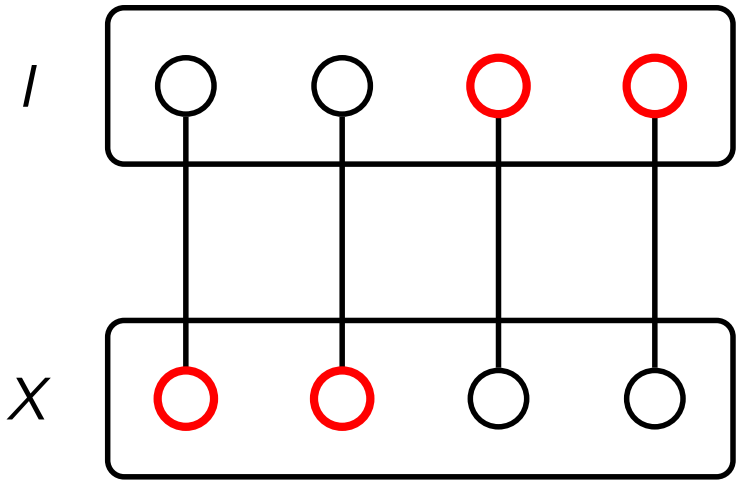
Доказательство леммы

- Найдём (жадно) максимальное по включению паросочетание в графе G.
- Если его размер хотя бы $k + 1$, то решения, очевидно, нет.
- Пусть X есть множество вершин найденного паросочетания, I — оставшиеся вершины.
- Тогда I — независимое множество.

Доказательство леммы

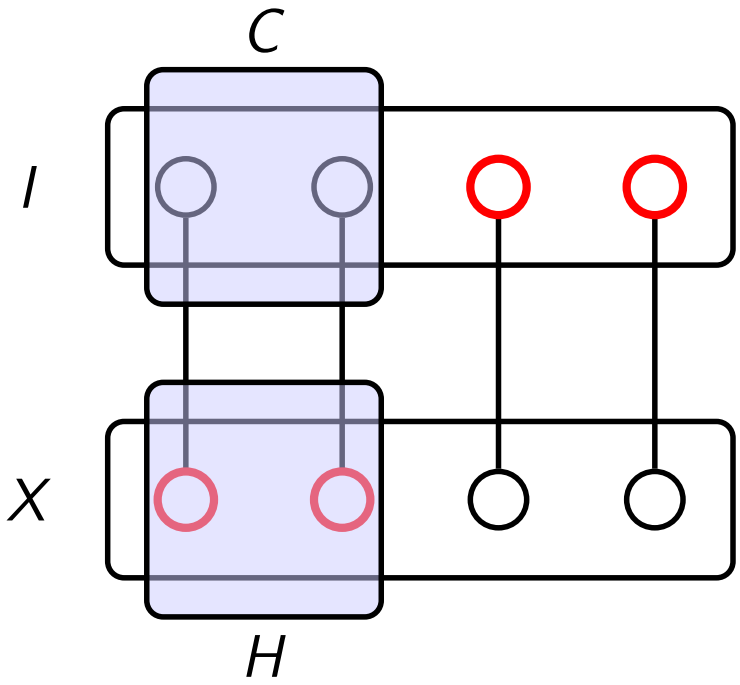
- Найдём (жадно) максимальное по включению паросочетание в графе G.
- Если его размер хотя бы $k + 1$, то решения, очевидно, нет.
- Пусть X есть множество вершин найденного паросочетания, I — оставшиеся вершины.
- Тогда I — независимое множество.
- Найдём максимальное паросочетание/минимальное вершинное покрытие в двудольном графе между X и I.

Первый случай



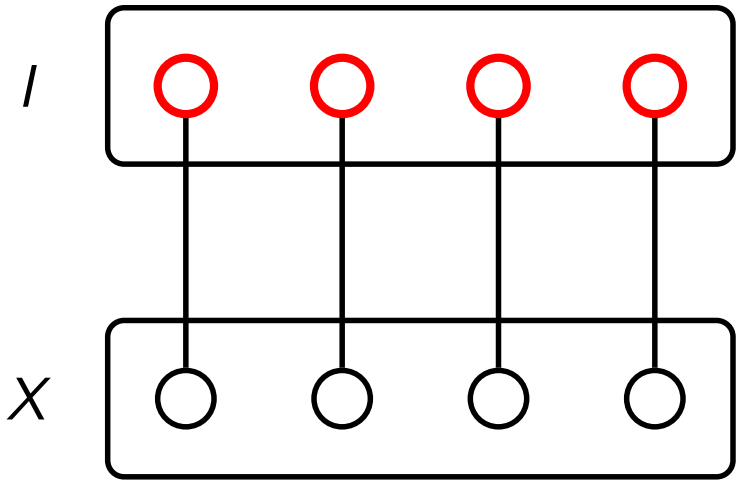
- Если найденное вершинное покрытие содержит хотя бы одну вершину множества X,

Первый случай



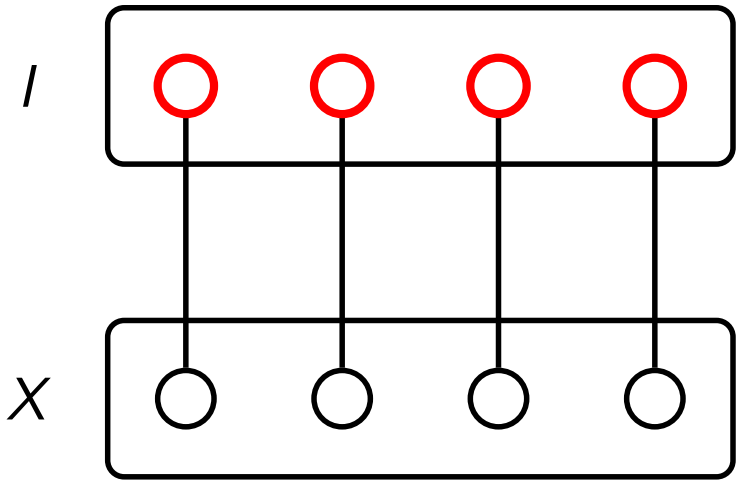
- Если найденное вершинное покрытие содержит хотя бы одну вершину множества X, то у графа есть CD-разбиение.

Первый случай



- Если найденное вершинное покрытие содержит хотя бы одну вершину множества X, то у графа есть CD-разбиение.
- Если вершинное покрытие содержит только вершины множества I,

Первый случай



- Если найденное вершинное покрытие содержит хотя бы одну вершину множества X, то у графа есть CD-разбиение.
- Если вершинное покрытие содержит только вершины множества I, то оно содержит все вершины множества I и всего в графе не более $4k$ вершин.

План лекции

1. FPT алгоритмы
   - Покрытие точек прямыми
   - Вершинное покрытие
2. Приближённые алгоритмы
   - Задача о вершинном покрытии
   - Задача о коммивояжёре

Приближённые алгоритмы для NP-трудных задач

Алгоритмы, находящие за полиномиальное время для данной оптимизационной задачи решение, которое гарантированно не сильно хуже оптимального.

План лекции

1. FPT алгоритмы
   - Покрытие точек прямыми
   - Вершинное покрытие
2. Приближённые алгоритмы
   - Задача о вершинном покрытии
   - Задача о коммивояжёре

2-приближенный алгоритм

Алгоритм

Approx-Vertex-Cover(G)

2-приближенный алгоритм

Алгоритм

Approx-Vertex-Cover(G)

- $C = \emptyset$

2-приближенный алгоритм

Алгоритм

Approx-Vertex-Cover(G)

- $C = \emptyset$
- $E' = E[G]$

2-приближенный алгоритм

Алгоритм

Approx-Vertex-Cover(G)

- $C = \emptyset$
- $E' = E[G]$
- пока $E' \neq \emptyset$

2-приближенный алгоритм

Алгоритм

Approx-Vertex-Cover(G)

- $C = \emptyset$
- $E' = E[G]$
- пока $E' \neq \emptyset$
  - берем произвольное ребро $(u, v) \in E'$

2-приближенный алгоритм

Алгоритм

Approx-Vertex-Cover(G)

- $C = \emptyset$
- $E' = E[G]$
- пока $E' \neq \emptyset$
  - берем произвольное ребро $(u, v) \in E'$
  - $C := C \cup \{u, v\}$

2-приближенный алгоритм

Алгоритм

Approx-Vertex-Cover(G)

- $C = \emptyset$
- $E' = E[G]$
- пока $E' \neq \emptyset$
  - берем произвольное ребро $(u, v) \in E'$
  - $C := C \cup \{u, v\}$
  - удалим из E' все покрытые ребра

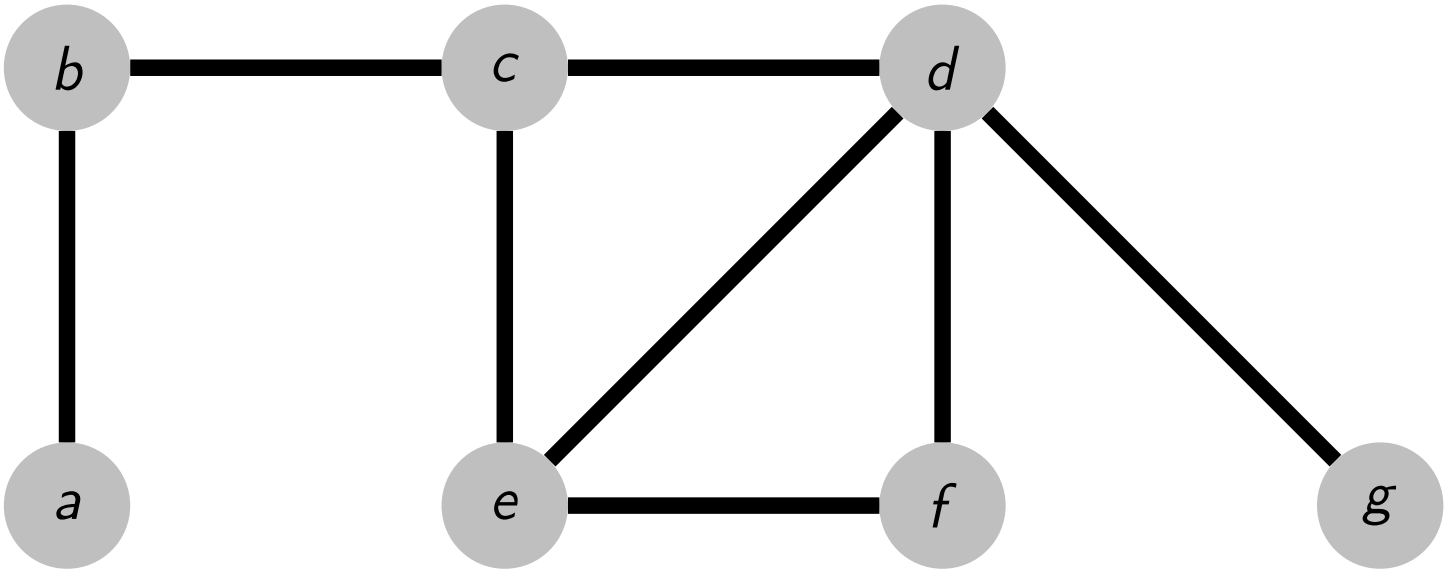
2-приближенный алгоритм

Алгоритм

Approx-Vertex-Cover(G)

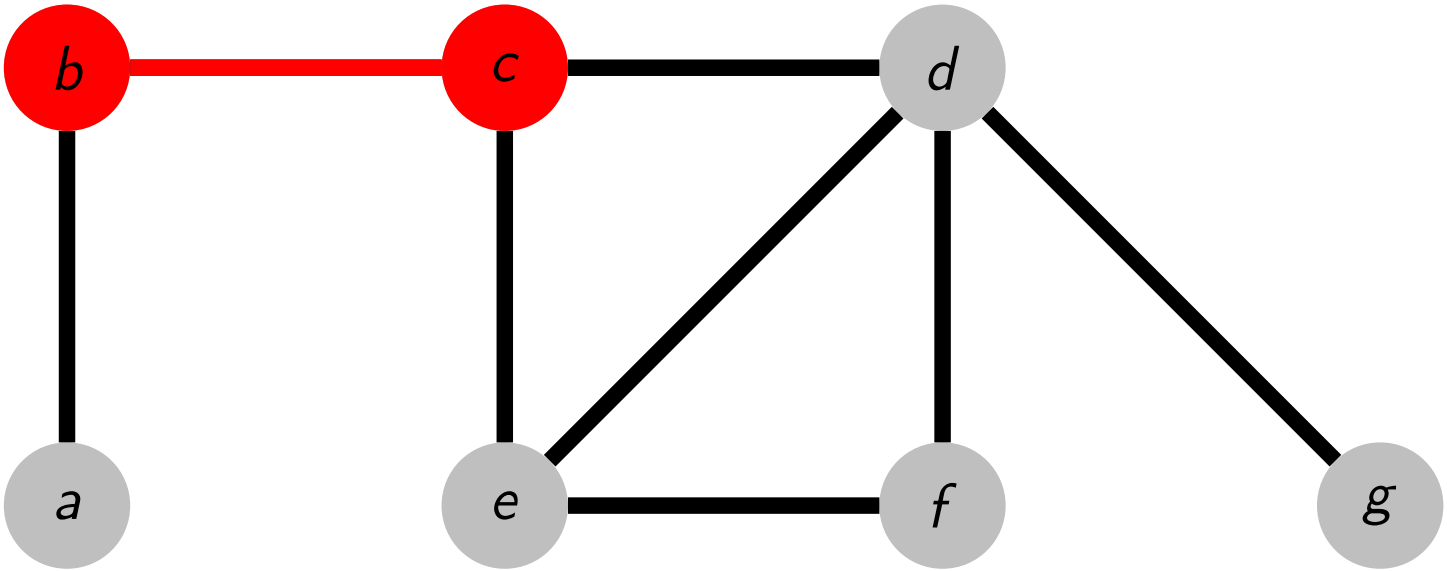
- $C = \emptyset$
- $E' = E[G]$
- пока $E' \neq \emptyset$
  - берем произвольное ребро $(u, v) \in E'$
  - $C := C \cup \{u, v\}$
  - удалим из E' все покрытые ребра
- вернуть C

Пример работы алгоритма



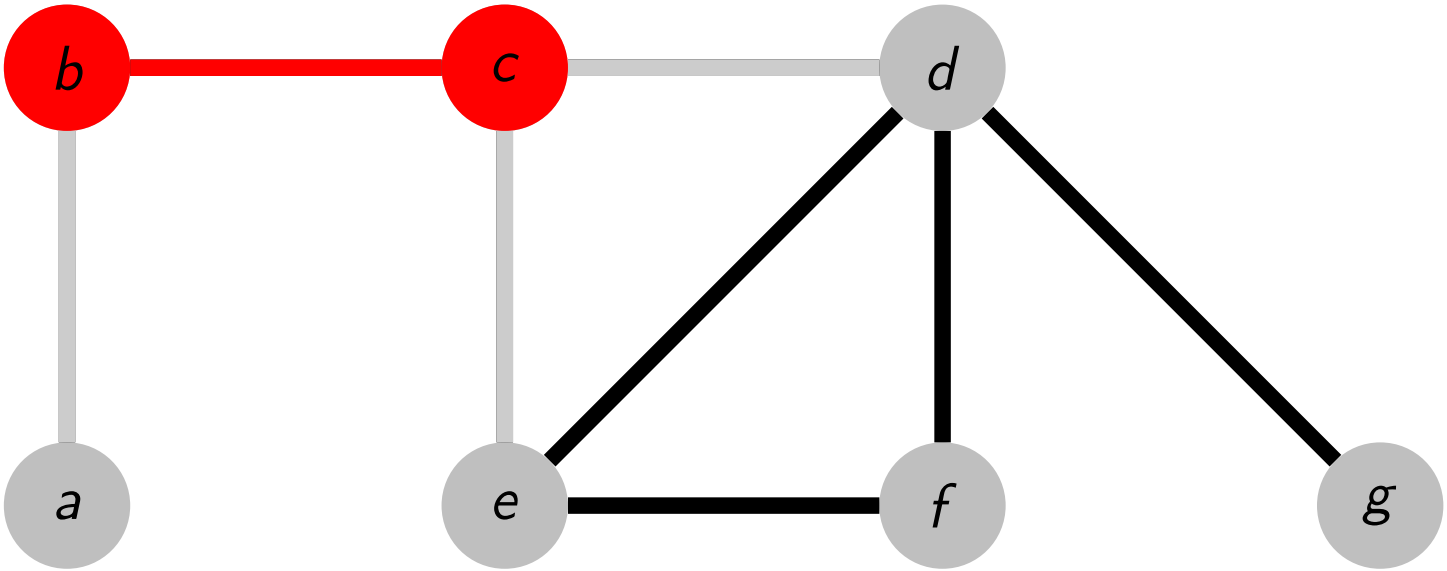
дан граф на семи вершинах

Пример работы алгоритма



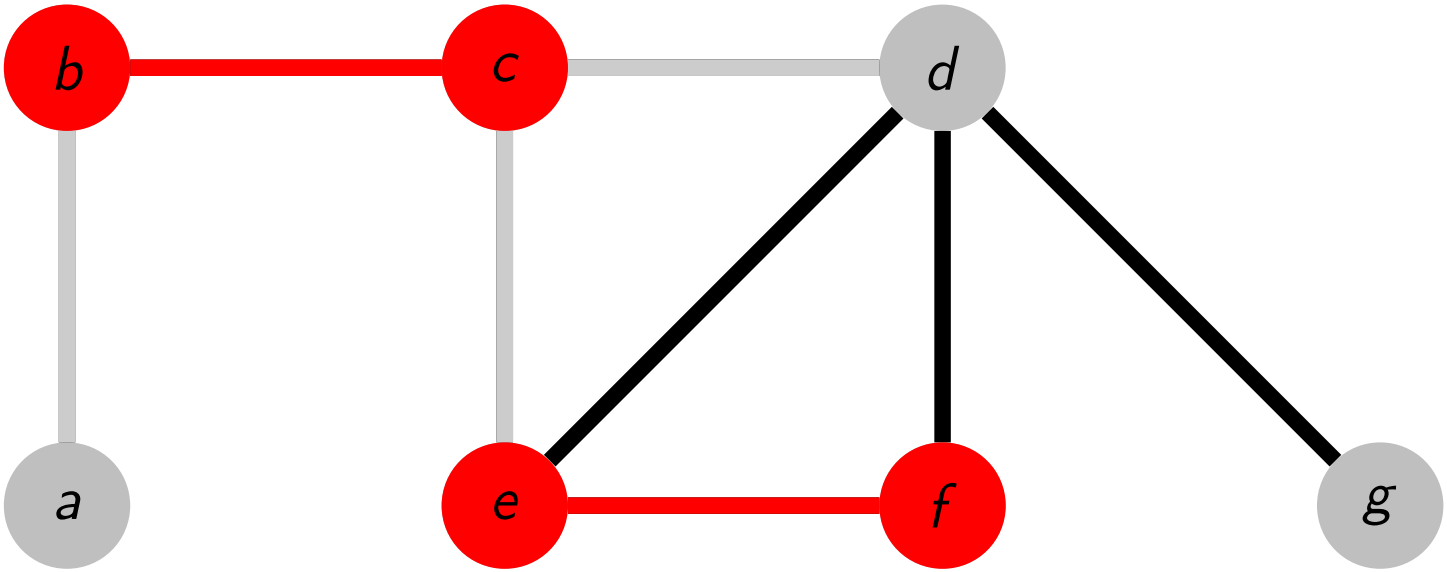
выбираем ребро (b, c)

Пример работы алгоритма



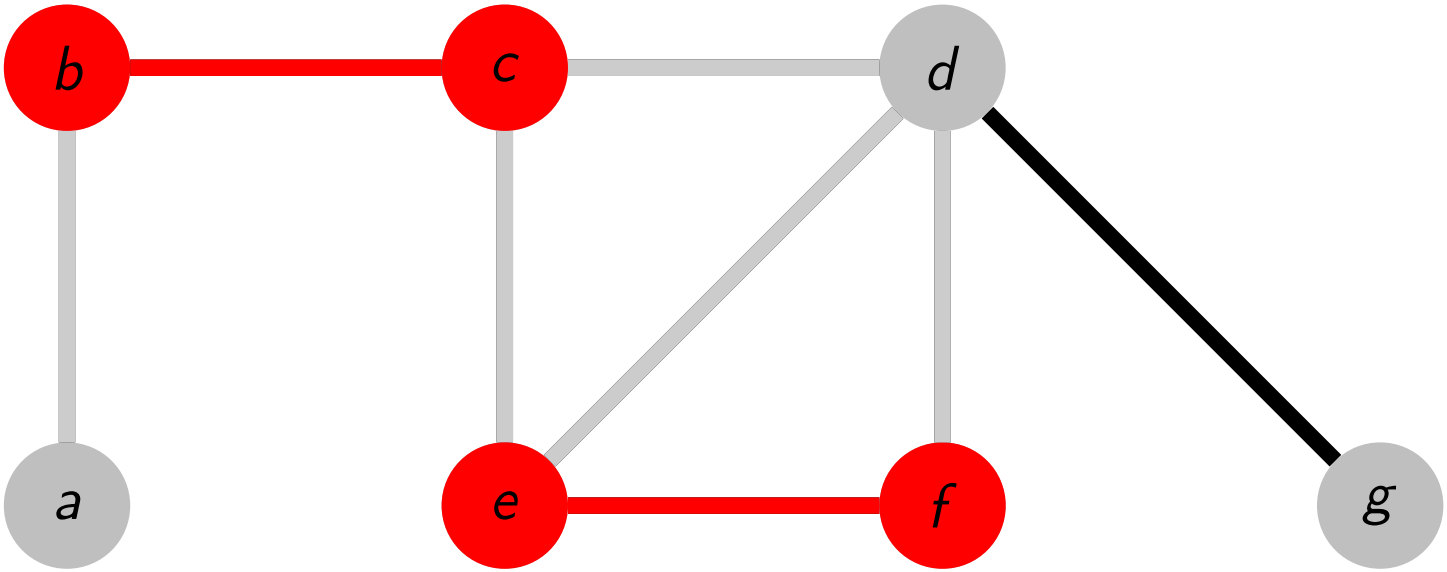
покрытые ребра удаляем

Пример работы алгоритма



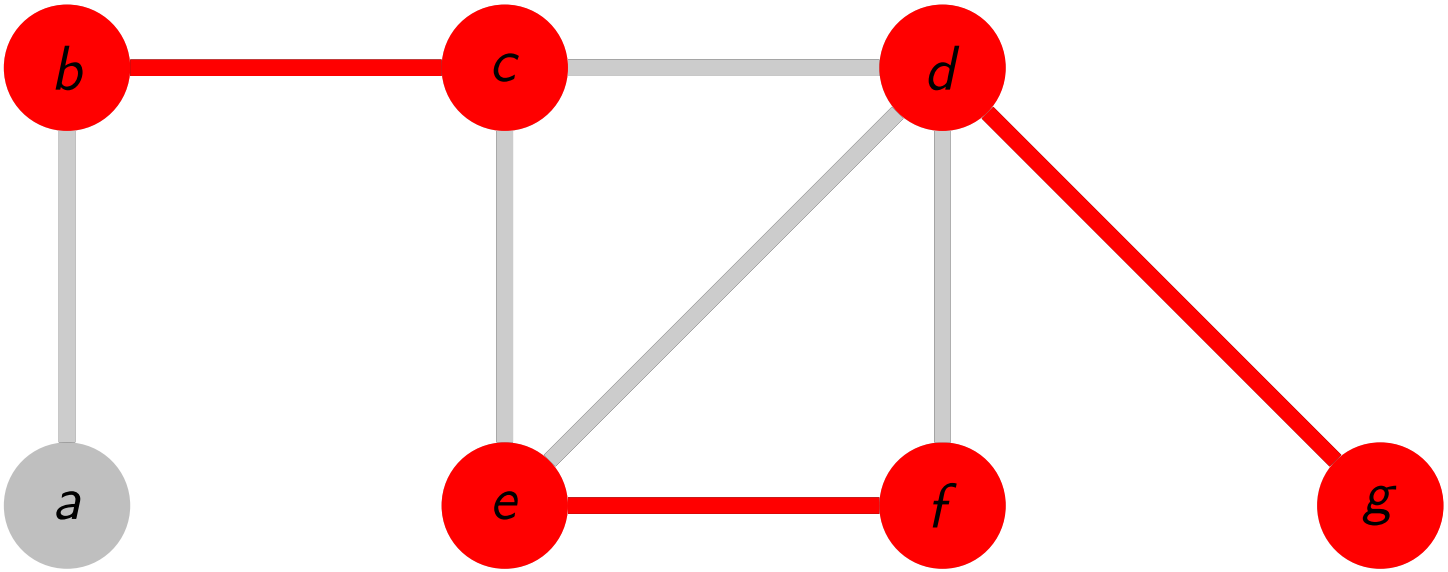
выбираем ребро (e, f)

Пример работы алгоритма



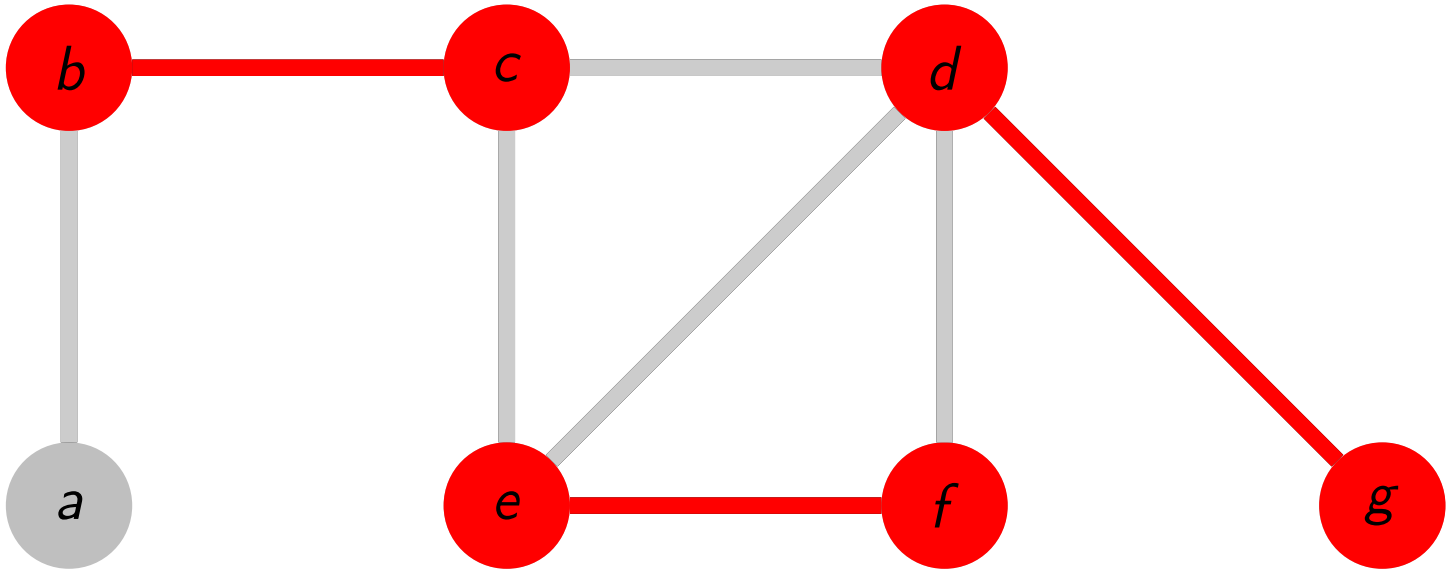
покрытые ребра удаляем

Пример работы алгоритма



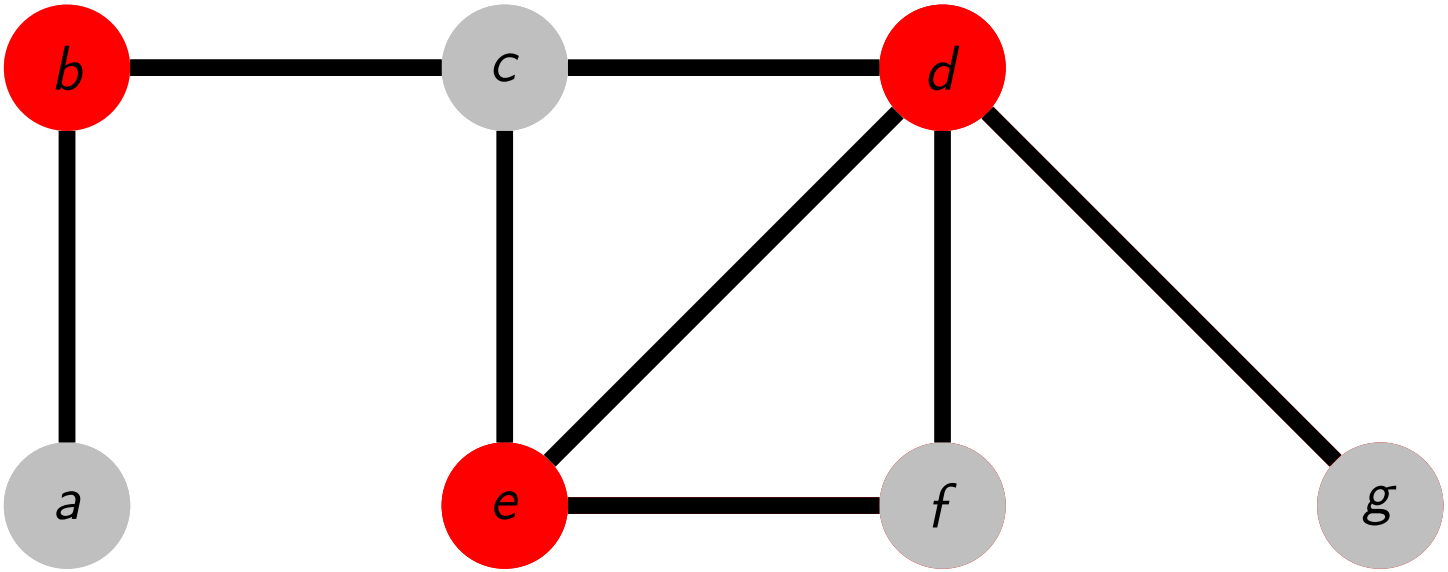
выбираем ребро (d, g)

Пример работы алгоритма



построенное покрытие: $\{b, c, d, e, f, g\}$

Пример работы алгоритма



оптимальное покрытие: $\{b, d, e\}$

Анализ алгоритма

Теорема

Алгоритм Approx-Vertex-Cover является 2-оптимальным.

Анализ алгоритма

Теорема

Алгоритм Approx-Vertex-Cover является 2-оптимальным.

Доказательство

Анализ алгоритма

Теорема

Алгоритм Approx-Vertex-Cover является 2-оптимальным.

Доказательство

- ясно, что выдаваемое множество C покрытием является

Анализ алгоритма

Теорема

Алгоритм Approx-Vertex-Cover является 2-оптимальным.

Доказательство

- ясно, что выдаваемое множество C покрытием является
- пусть A – множество ребер, выбираемых алгоритмом

Анализ алгоритма

Теорема

Алгоритм Approx-Vertex-Cover является 2-оптимальным.

Доказательство

- ясно, что выдаваемое множество C покрытием является
- пусть A – множество ребер, выбираемых алгоритмом
- видно, что никакие два из них не имеют общего конца

Анализ алгоритма

Теорема

Алгоритм Approx-Vertex-Cover является 2-оптимальным.

Доказательство

- ясно, что выдаваемое множество C покрытием является
- пусть A – множество ребер, выбираемых алгоритмом
- видно, что никакие два из них не имеют общего конца
- следовательно, $2|A| = |C|$

Анализ алгоритма

Теорема

Алгоритм Approx-Vertex-Cover является 2-оптимальным.

Доказательство

- ясно, что выдаваемое множество C покрытием является
- пусть A – множество ребер, выбираемых алгоритмом
- видно, что никакие два из них не имеют общего конца
- следовательно, $2|A| = |C|$
- далее, если C' – покрытие, то $|A| \leq |C'|$, так как C' обязано покрывать хотя бы по одному концу каждого ребра из A

Анализ алгоритма

Теорема

Алгоритм Approx-Vertex-Cover является 2-оптимальным.

Доказательство

- ясно, что выдаваемое множество C покрытием является
- пусть A – множество ребер, выбираемых алгоритмом
- видно, что никакие два из них не имеют общего конца
- следовательно, $2|A| = |C|$
- далее, если C' – покрытие, то $|A| \leq |C'|$, так как C' обязано покрывать хотя бы по одному концу каждого ребра из A
- таким образом, $|C| = 2|A| \leq 2|C_{\text{opt}}|$

План лекции

Задача о коммивояжере

Определение

Задача о коммивояжере

Определение

- Дан полный неориентированный граф $G = (V, E)$, каждому ребру (u, v) которого приписана некоторая стоимость $c(u, v)$.

Задача о коммивояжере

Определение

- Дан полный неориентированный граф $G = (V, E)$, каждому ребру (u, v) которого приписана некоторая стоимость $c(u, v)$.
- Задача о коммивояжере (travelling salesman problem, TSP) заключается в нахождении в графе гамильтонова цикла минимальной стоимости.

Задача о коммивояжере

Определение

- Дан полный неориентированный граф $G = (V, E)$, каждому ребру (u, v) которого приписана некоторая стоимость $c(u, v)$.
- Задача о коммивояжере (travelling salesman problem, TSP) заключается в нахождении в графе гамильтонова цикла минимальной стоимости.
- Задача о коммивояжере в метрическом пространстве (metric TSP) есть частный случай задачи о коммивояжере, где расстояния входного графа удовлетворяют неравенству треугольника:

$$c(u, w) \leq c(u, v) + c(v, w) \quad \forall u, v, w \in V.$$

2-оптимальный алгоритм

Алгоритм

Approx-TSP(G)

2-оптимальный алгоритм

Алгоритм

Approx-TSP(G)

- строим минимальное покрывающее дерево T графа G

2-оптимальный алгоритм

Алгоритм

APPROX-TSP(G)

- строим минимальное покрывающее дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

2-оптимальный алгоритм

Алгоритм

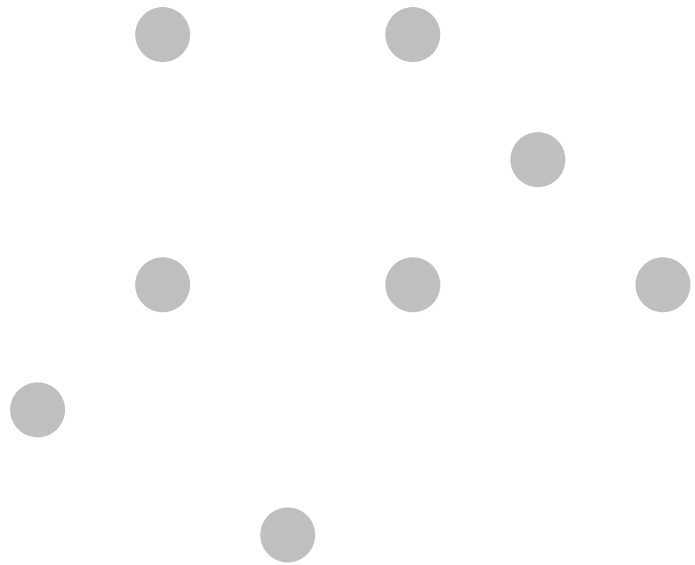
Approx-TSP(G)

- строим минимальное покрывающее дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

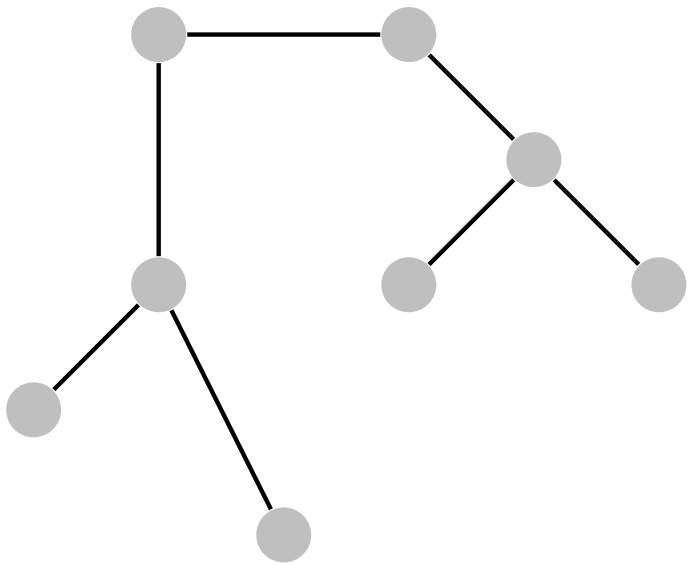


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G

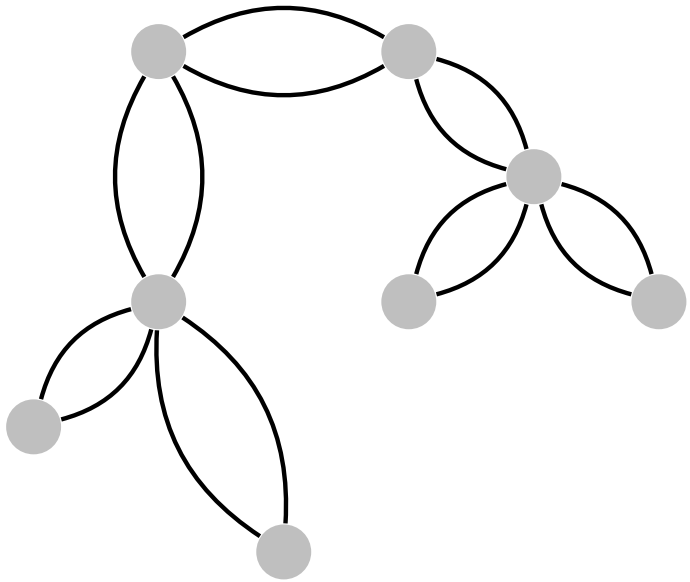


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

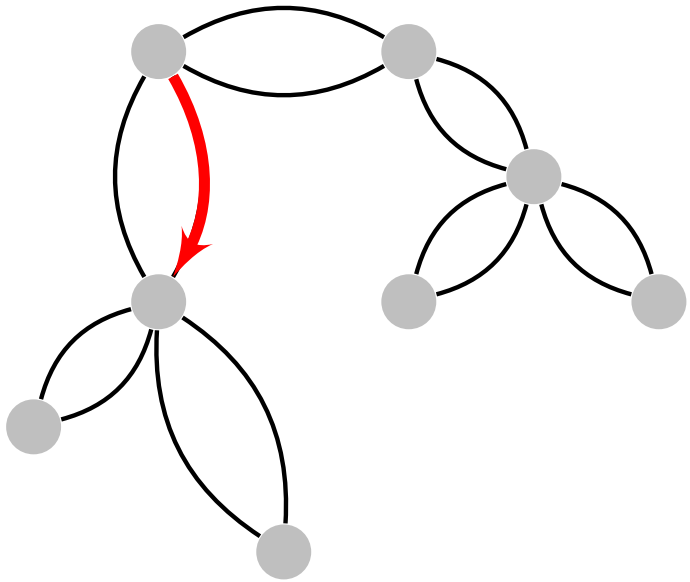


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

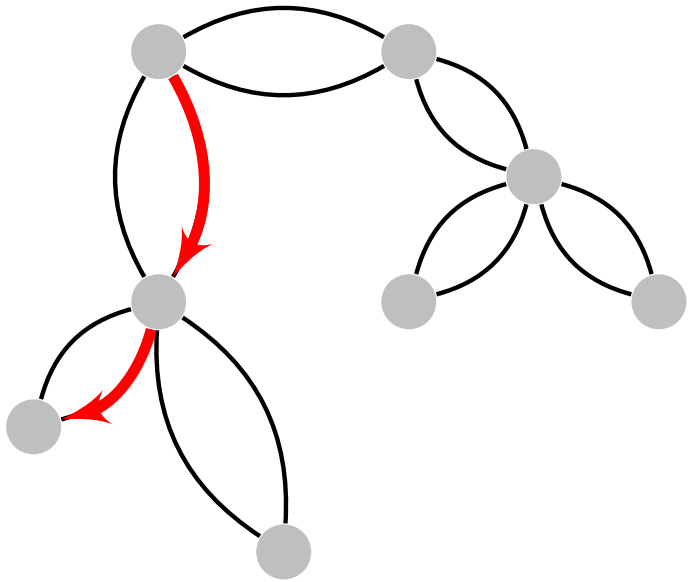


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

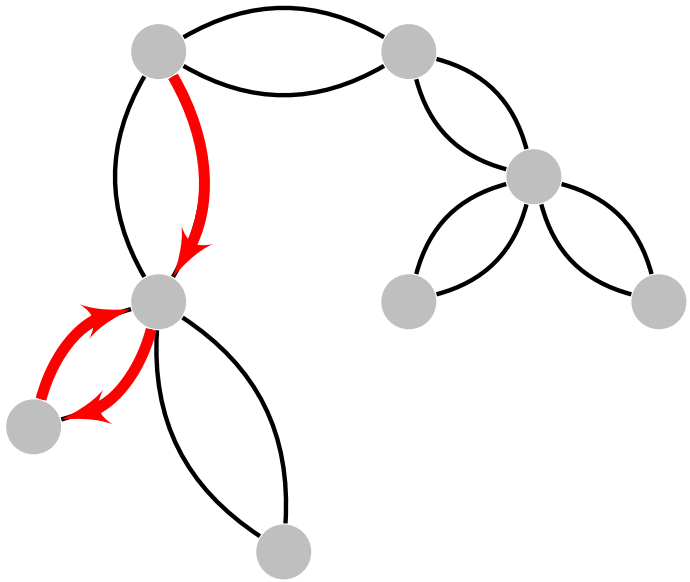


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

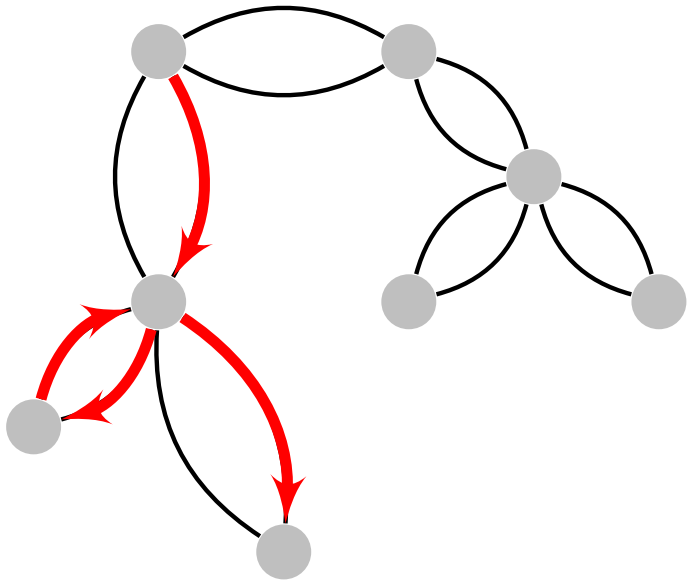


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

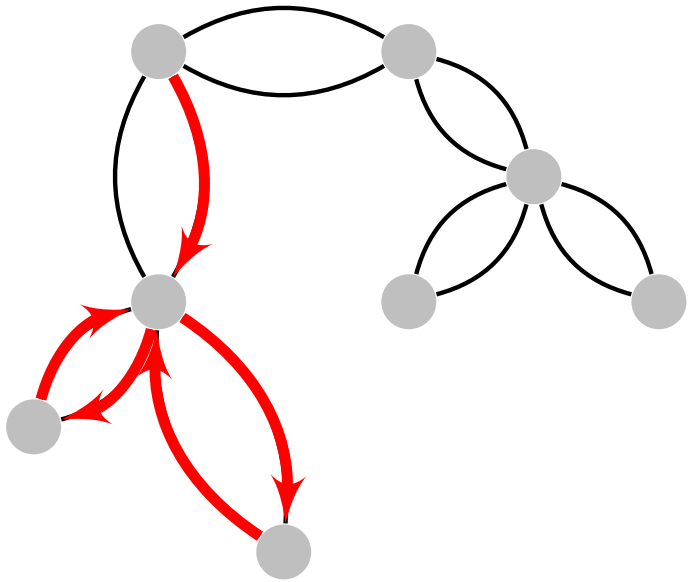


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

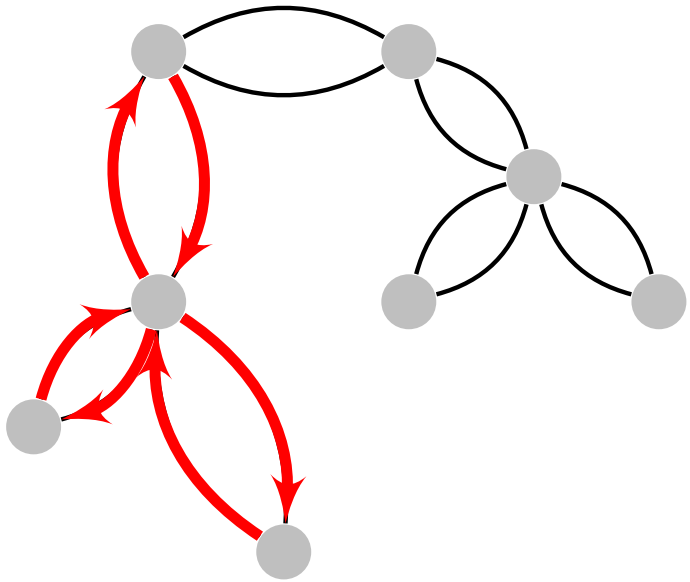


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

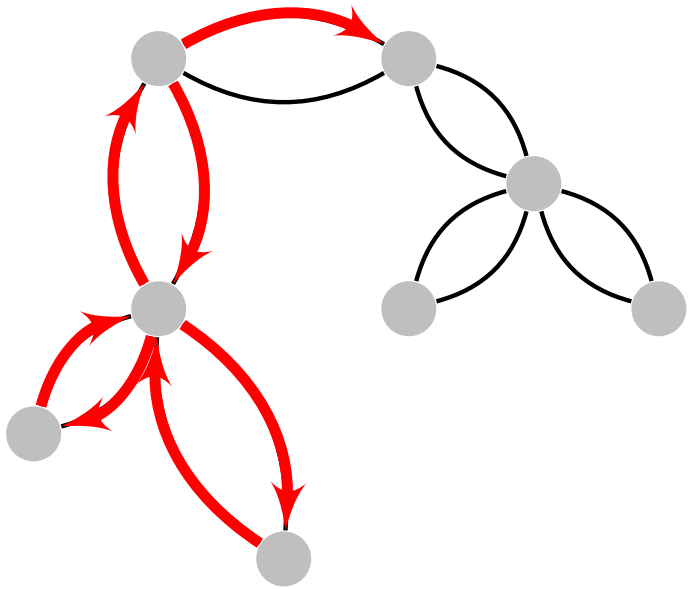


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

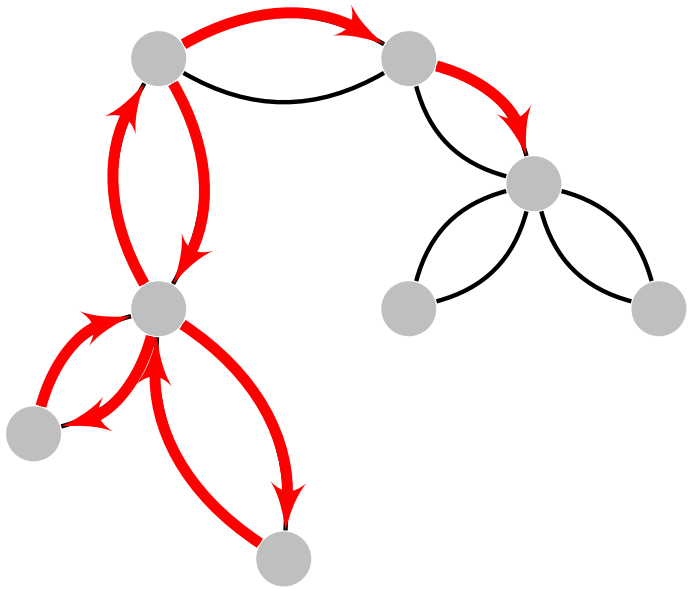


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

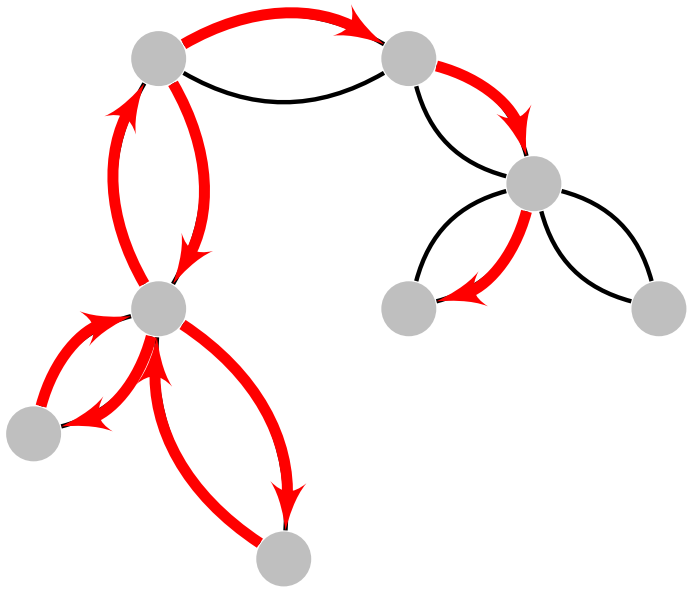


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

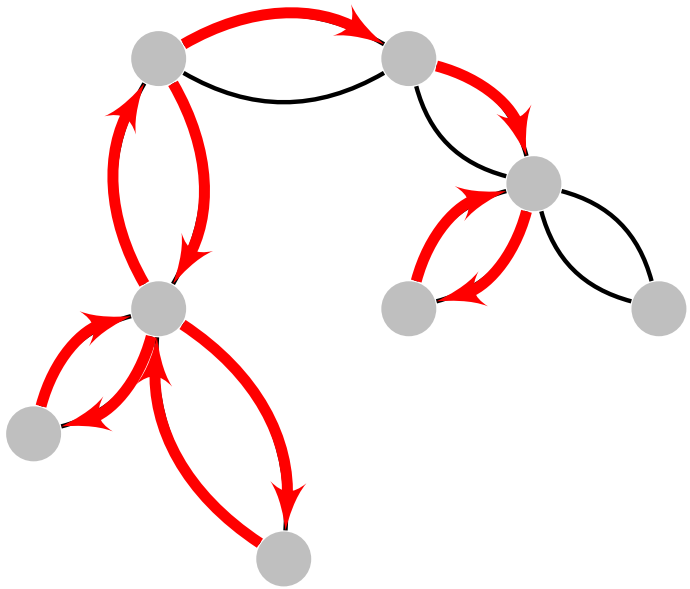


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

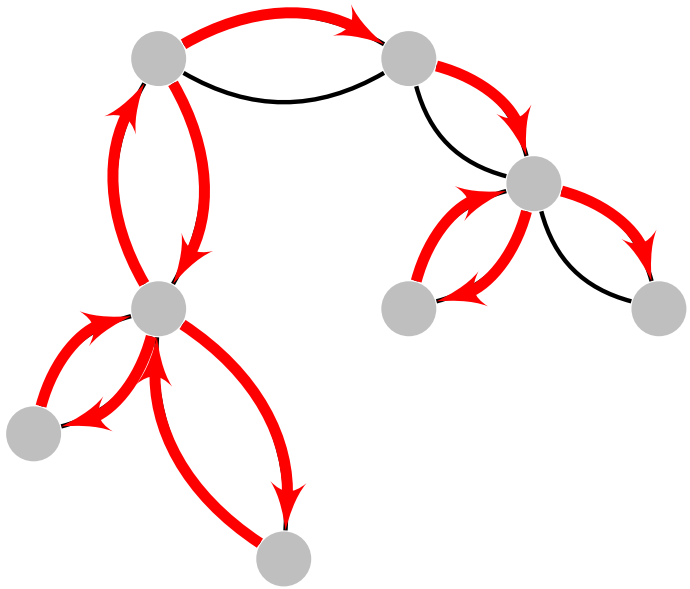


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

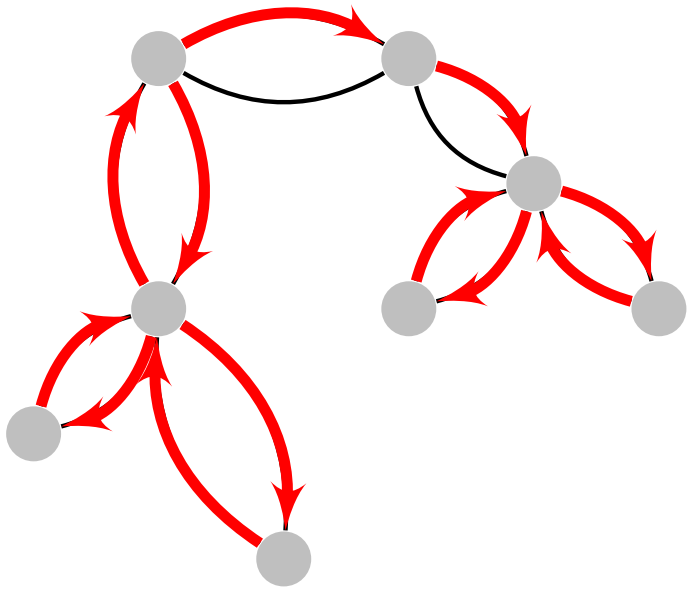


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

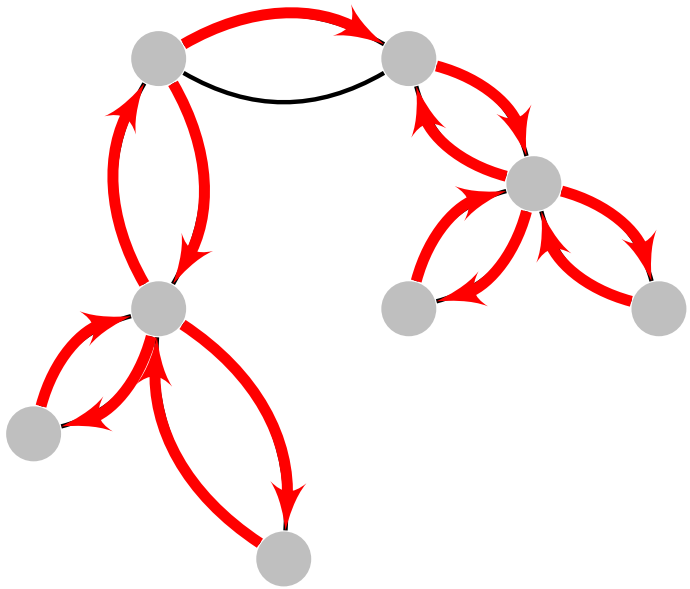


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

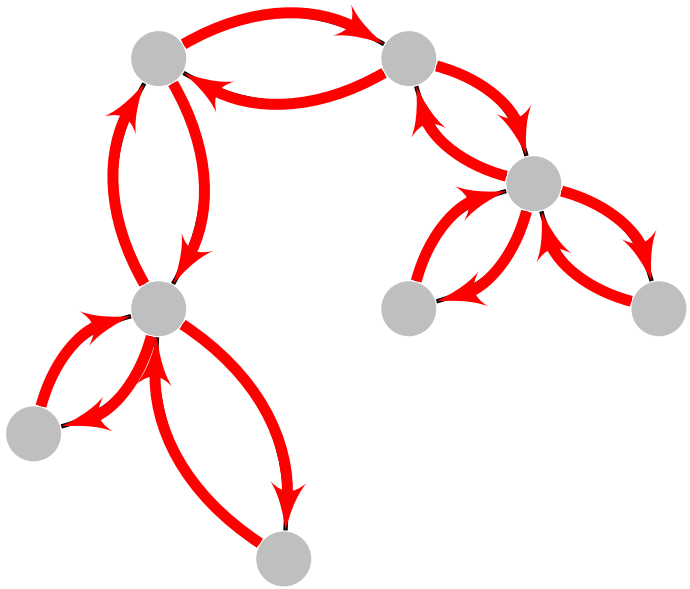


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл

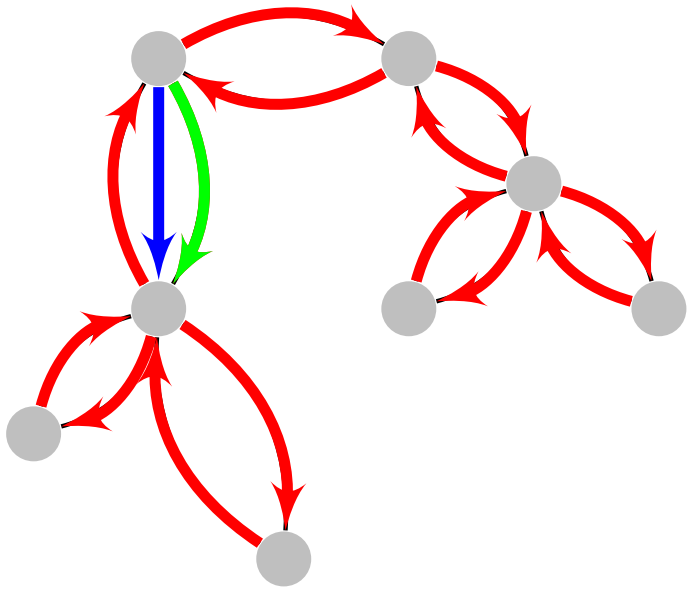


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

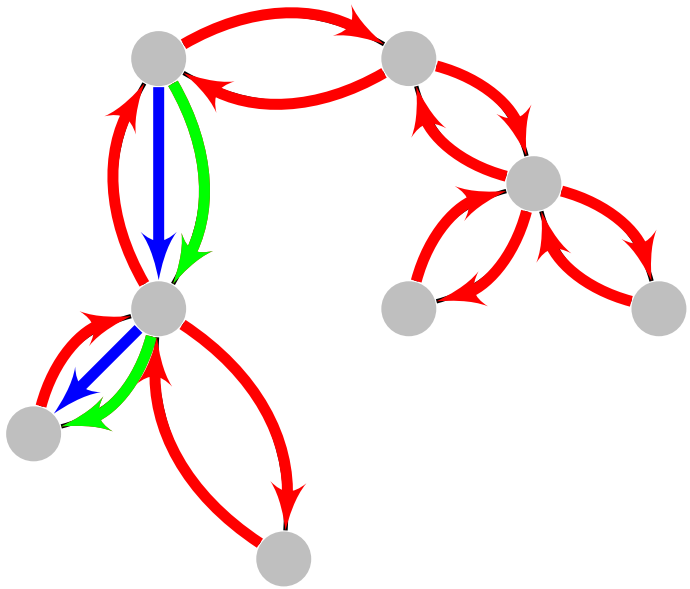


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

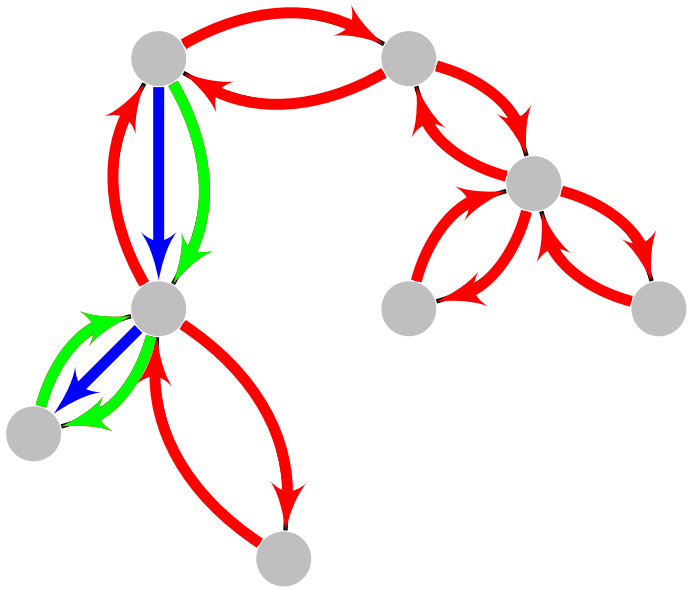


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

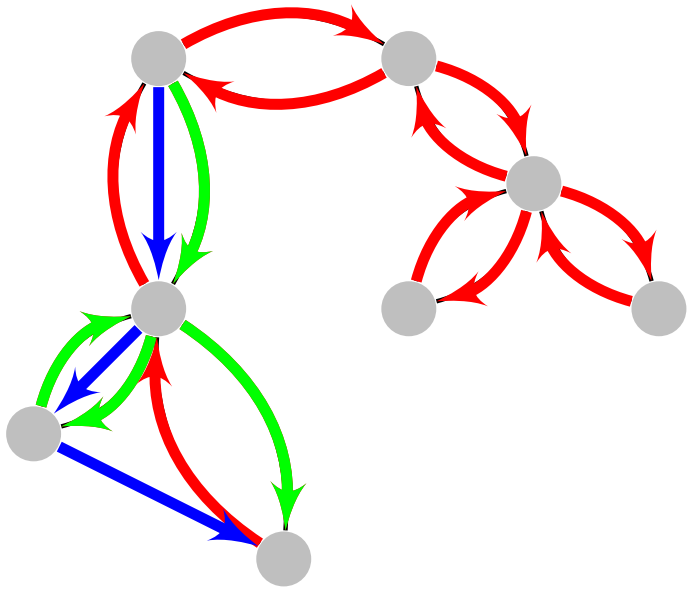


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

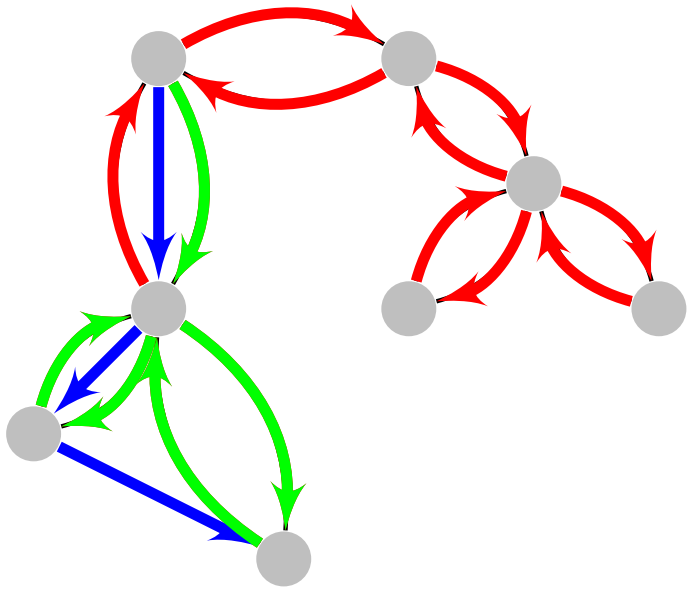


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

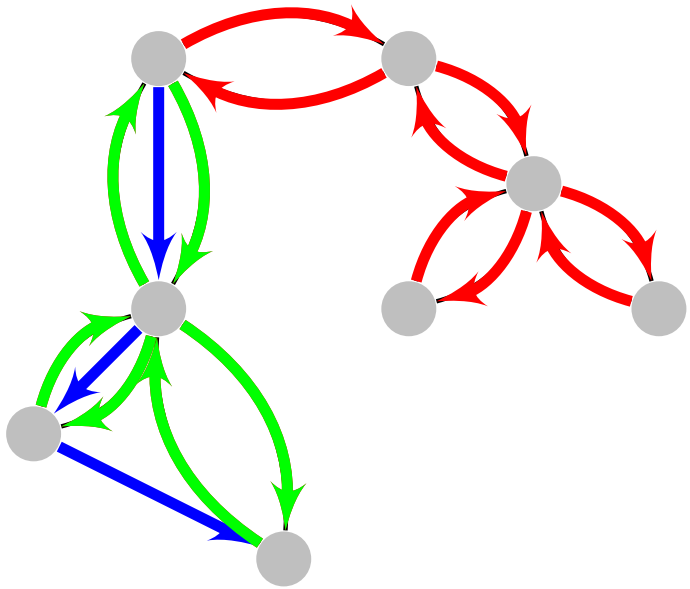


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

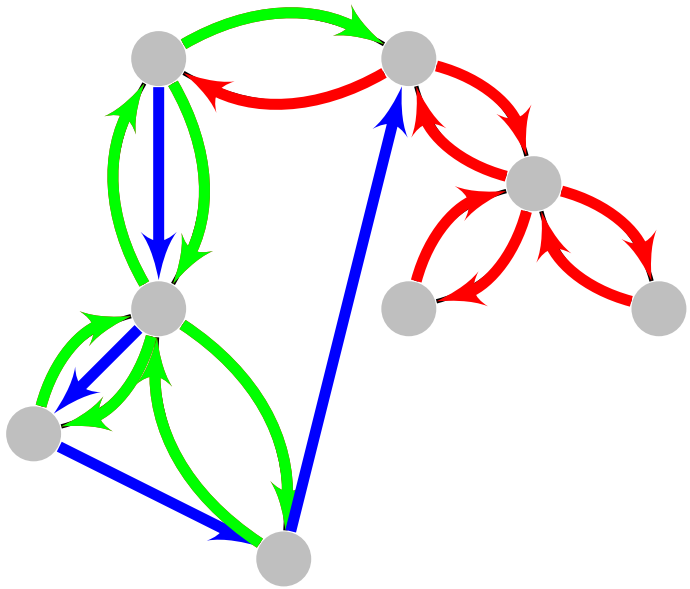


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

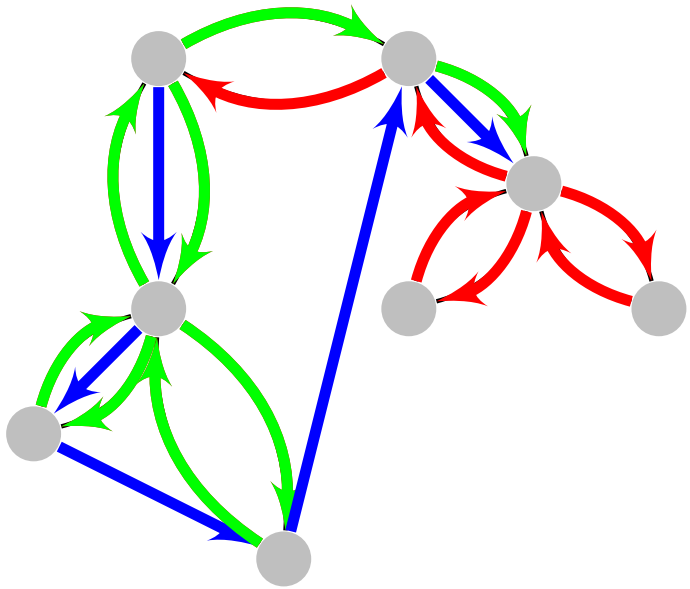


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

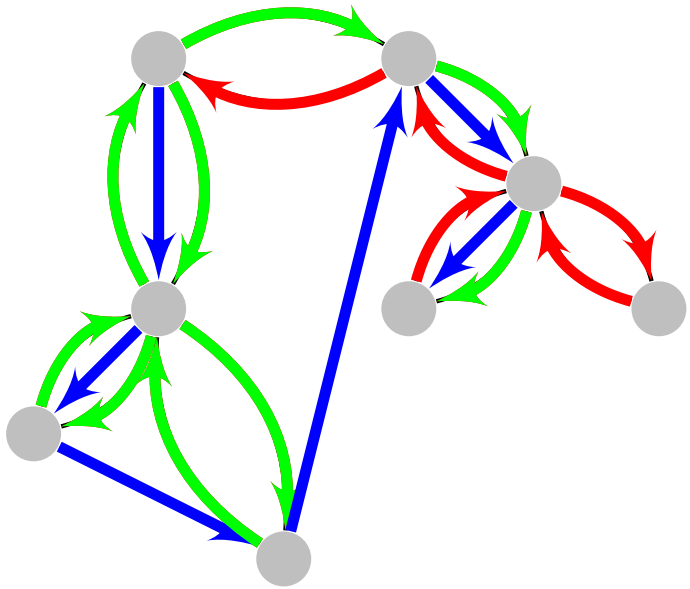


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

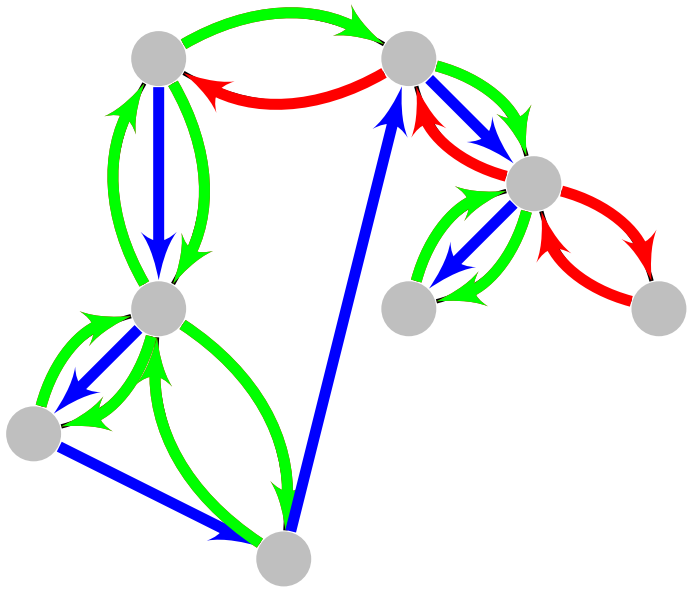


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

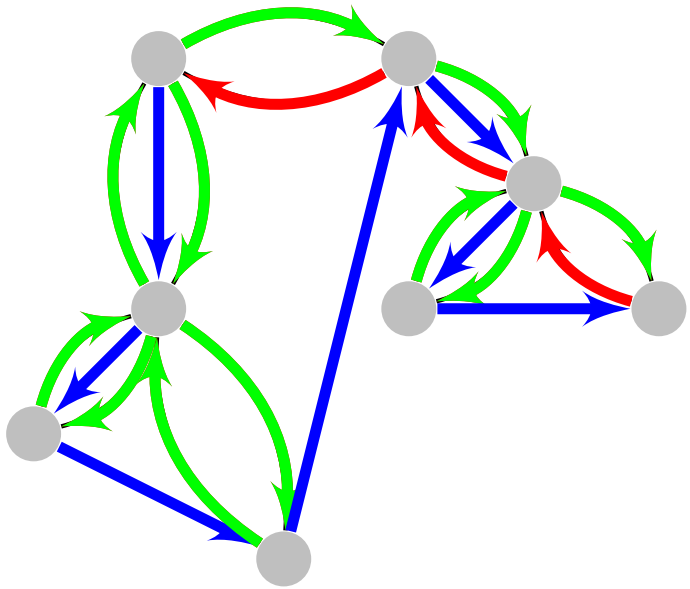


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

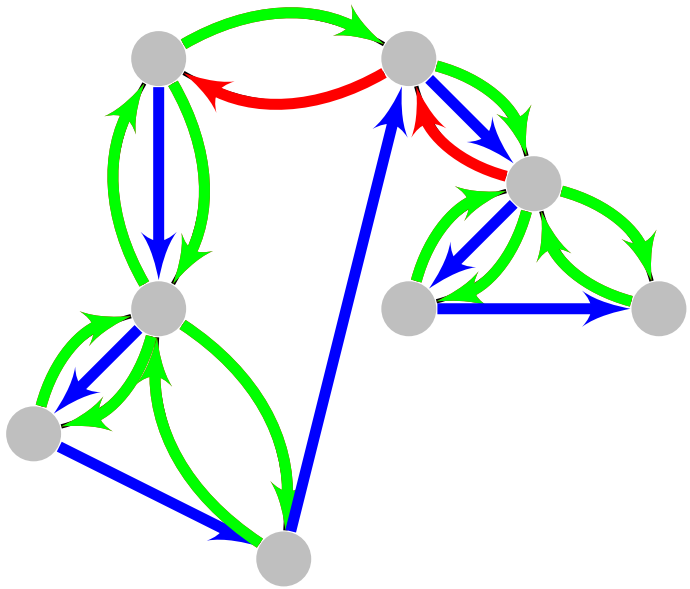


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

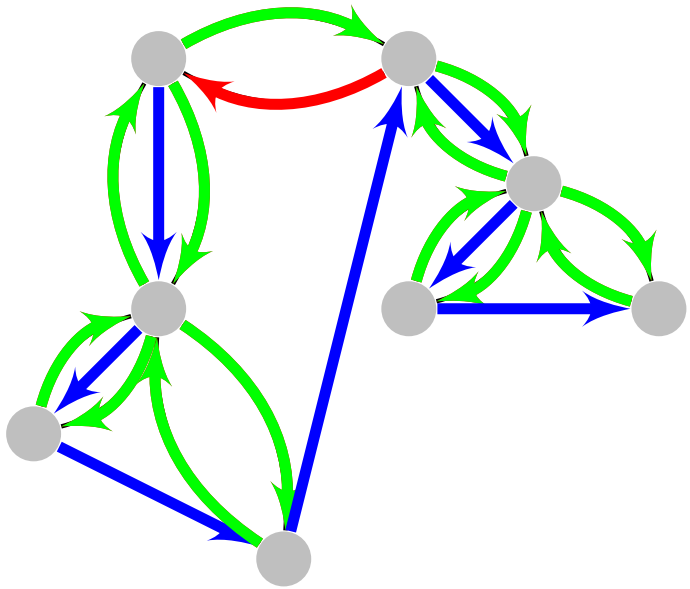


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

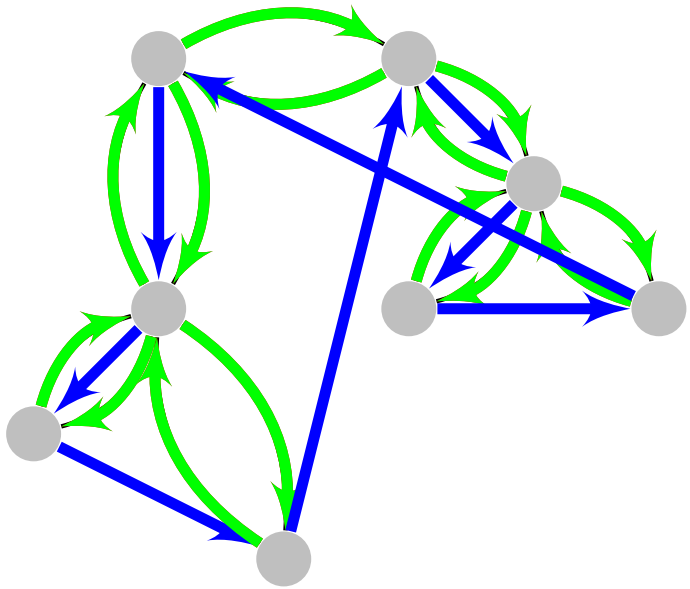


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

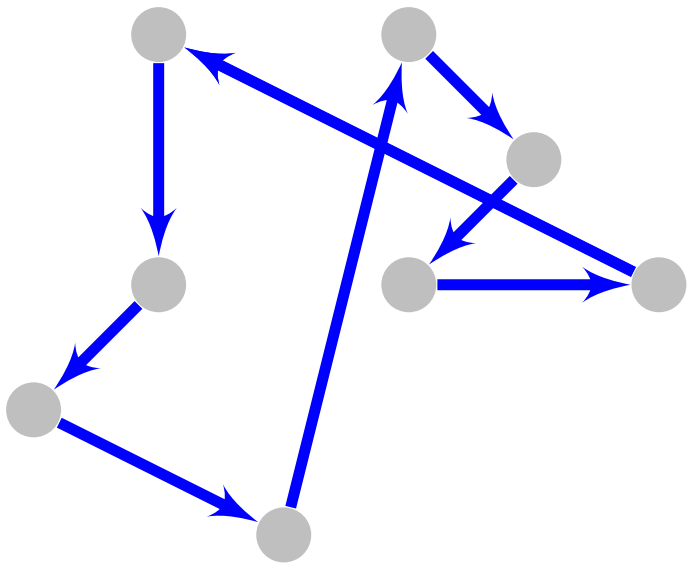


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G
- продублируем каждое ребро дерева T и в полученном графе найдем эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл



Анализ алгоритма

Теорема

Алгоритм APPROX-TSP является 2-приближенным.

Анализ алгоритма

Теорема

Алгоритм APPROX-TSP является 2-приближенным.

Доказательство

Анализ алгоритма

Теорема

Алгоритм Approx-TSP является 2-приближенным.

Доказательство

- пусть W_T — вес минимального остовного дерева, а W_{opt} — вес оптимального гамильтонова цикла

Анализ алгоритма

Теорема

Алгоритм Approx-TSP является 2-приближенным.

Доказательство

- пусть W_T — вес минимального остовного дерева, а W_{opt} — вес оптимального гамильтонова цикла
- $W_T \leq W_{\text{opt}}$, поскольку при выкидывании ребра из гамильтонва цикла получается остовное дерево

Анализ алгоритма

Теорема

Алгоритм Approx-TSP является 2-приближенным.

Доказательство

- пусть W_T — вес минимального остовного дерева, а W_{opt} — вес оптимального гамильтонова цикла
- $W_T \leq W_{\text{opt}}$, поскольку при выкидывании ребра из гамильтонва цикла получается остовное дерево
- каждое ребро построенного гамильтонова цикла заменяет какой-то путь эйлерова цикла, длина которого по неравенству треугольника не менее длины этого ребра

Анализ алгоритма

Теорема

Алгоритм APPROX-TSP является 2-приближенным.

Доказательство

- пусть W_T — вес минимального остовного дерева, а W_{opt} — вес оптимального гамильтонова цикла
- $W_T \leq W_{\text{opt}}$, поскольку при выкидывании ребра из гамильтонва цикла получается остовное дерево
- каждое ребро построенного гамильтонова цикла заменяет какой-то путь эйлерова цикла, длина которого по неравенству треугольника не менее длины этого ребра
- значит, длина найденного пути не превосходит $2W_T$, а следовательно, и $2W_{\text{opt}}$

Спасибо за внимание!