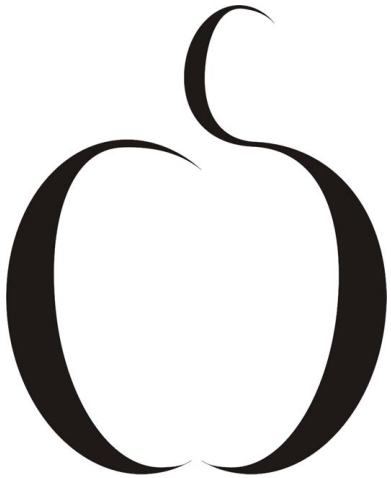
с/к “Эффективные алгоритмы”

Лекция 1: Приближенные алгоритмы

А. Куликов

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



План лекции

1. Что такое приближенные алгоритмы и для чего они нужны

План лекции

1. Что такое приближенные алгоритмы и для чего они нужны
2. Вершинное покрытие

План лекции

1. Что такое приближенные алгоритмы и для чего они нужны
2. Вершинное покрытие
3. Покрытие множествами

План лекции

1. Что такое приближенные алгоритмы и для чего они нужны
2. Вершинное покрытие
3. Покрытие множествами

Для чего нужны

Для чего нужны

- возникающие на практике оптимизационные задачи часто являются NP-трудными

Для чего нужны

- возникающие на практике оптимизационные задачи часто являются NP-трудными
- полиномиальный алгоритм для такой задачи вряд ли существует

Для чего нужны

- возникающие на практике оптимизационные задачи часто являются NP-трудными
- полиномиальный алгоритм для такой задачи вряд ли существует
- можно пытаться построить эвристический алгоритм, который будет решать задачу за разумное время на реальных данных

Для чего нужны

- возникающие на практике оптимизационные задачи часто являются NP-трудными
- полиномиальный алгоритм для такой задачи вряд ли существует
- можно пытаться построить эвристический алгоритм, который будет решать задачу за разумное время на реальных данных
- или же алгоритм, находящий решение, которое не сильно хуже оптимального;

Для чего нужны

- возникающие на практике оптимизационные задачи часто являются NP-трудными
- полиномиальный алгоритм для такой задачи вряд ли существует
- можно пытаться построить эвристический алгоритм, который будет решать задачу за разумное время на реальных данных
- или же алгоритм, находящий решение, которое не сильно хуже оптимального; такие алгоритмы называются приближенными

Оптимизационная задача

Определение

Оптимизационная задача

Определение

- Пусть дан вход I оптимизационной задачи Π.

Оптимизационная задача

Определение

- Пусть дан вход I оптимизационной задачи Π.
- Через $\mathrm{sol}(I)$ обозначим множество всех возможных решений. Будем считать, что $\mathrm{sol}(I) \neq \emptyset$ для любого входа I.

Оптимизационная задача

Определение

- Пусть дан вход I оптимизационной задачи Π.
- Через $\mathrm{sol}(I)$ обозначим множество всех возможных решений. Будем считать, что $\mathrm{sol}(I) \neq \emptyset$ для любого входа I.
- Каждому возможному решению $x \in \mathrm{sol}(I)$ сопоставлена стоимость $\mathrm{cost}(x)$.

Оптимизационная задача

Определение

- Пусть дан вход I оптимизационной задачи Π.
- Через $\mathrm{sol}(I)$ обозначим множество всех возможных решений. Будем считать, что $\mathrm{sol}(I) \neq \emptyset$ для любого входа I.
- Каждому возможному решению $x \in \mathrm{sol}(I)$ сопоставлена стоимость $\mathrm{cost}(x)$.
- Требуется найти $x \in \mathrm{sol}(I)$, на котором cost достигает своего оптимального (минимального или максимального) значения.

Оптимизационная задача

Определение

- Пусть дан вход I оптимизационной задачи Π.
- Через $\mathrm{sol}(I)$ обозначим множество всех возможных решений. Будем считать, что $\mathrm{sol}(I) \neq \emptyset$ для любого входа I.
- Каждому возможному решению $x \in \mathrm{sol}(I)$ сопоставлена стоимость $\mathrm{cost}(x)$.
- Требуется найти $x \in \mathrm{sol}(I)$, на котором cost достигает своего оптимального (минимального или максимального) значения.
- Через $\mathrm{OPT}(I)$ обозначим оптимальную стоимость для входа I.

Приближенный алгоритм

Определение

Приближенный алгоритм

Определение

- Приближенным алгоритмом (approximation algorithm) для задачи Π называется полиномиальный по времени алгоритм, возвращающий для каждого входа I какое-то решение $x \in \mathrm{sol}(I)$. Через $A(I)$ мы обозначаем стоимость решения, найденного алгоритмом A на входе I.

Приближенный алгоритм

Определение

- Приближенным алгоритмом (approximation algorithm) для задачи Π называется полиномиальный по времени алгоритм, возращающий для каждого входа I какое-то решение $x \in \mathrm{sol}(I)$. Через $A(I)$ мы обозначаем стоимость решения, найденного алгоритмом A на входе I.
- Приближенный алгоритм A имеет абсолютную оценку точности (absolute performance guarantee) c, если для любого входа I выполняется неравенство $|\mathrm{OPT}(I) - A(I)| \leq c$.

Приближенный алгоритм (продолжение)

Определение

Приближенный алгоритм (продолжение)

Определение

- Приближенный алгоритм A имеет относительную оценку точности (relative performance guarantee) α, если для любого входа I выполняется неравенство

Приближенный алгоритм (продолжение)

Определение

- Приближенный алгоритм A имеет относительную оценку точности (relative performance guarantee) α, если для любого входа I выполняется неравенство
  - $A(I)/\mathrm{OPT}(I) \geq \alpha$ для максимизационной задачи;

Приближенный алгоритм (продолжение)

Определение

- Приближенный алгоритм A имеет относительную оценку точности (relative performance guarantee) α, если для любого входа I выполняется неравенство
  - $A(I)/\mathrm{OPT}(I) \geq \alpha$ для максимизационной задачи;
  - $A(I)/\mathrm{OPT}(I) \leq \alpha$ для минимизационной задачи.

Приближенный алгоритм (продолжение)

Определение

- Приближенный алгоритм A имеет относительную оценку точности (relative performance guarantee) α, если для любого входа I выполняется неравенство
  - $A(I)/\mathrm{OPT}(I) \geq \alpha$ для максимизационной задачи;
  - $A(I)/\mathrm{OPT}(I) \leq \alpha$ для минимизационной задачи.
- Такие алгоритмы мы называем α-приближенными.

Приближенный алгоритм (продолжение)

Определение

- Приближенный алгоритм A имеет относительную оценку точности (relative performance guarantee) α, если для любого входа I выполняется неравенство
  - $A(I)/\mathrm{OPT}(I) \geq \alpha$ для максимизационной задачи;
  - $A(I)/\mathrm{OPT}(I) \leq \alpha$ для минимизационной задачи.
- Такие алгоритмы мы называем α-приближенными.

Замечание

Для оптимизационных задач $\alpha \leq 1$, для минимизационных — $\alpha \geq 1$.
Алгоритм тем лучше, чем ближе α к 1.

Полиномиальная приближенная схема

Определение

Полиномиальная приближенная схема

Определение

- Пусть дана максимизационная задача П.

Полиномиальная приближенная схема

Определение

- Пусть дана максимизационная задача П.
- П имеет полиномиальную приближенную схему (polynomial time approximation scheme, PTAS), если

Полиномиальная приближенная схема

Определение

- Пусть дана максимизационная задача Π.
- Π имеет полиномиальную приближенную схему (polynomial time approximation scheme, PTAS), если
  1. для любого $\epsilon \geq 0$ существует $(1 - \epsilon)$-приближенный алгоритм для Π;

Полиномиальная приближенная схема

Определение

- Пусть дана максимизационная задача П.
- П имеет полиномиальную приближенную схему (polynomial time approximation scheme, PTAS), если
  1. для любого $\epsilon \geq 0$ существует $(1 - \epsilon)$-приближенный алгоритм для П;
  2. для любого $\epsilon > 0$ время работы такого алгоритма зависит от n полиномиально.

Полиномиальная приближенная схема

Определение

- Пусть дана максимизационная задача Π.
- Π имеет полиномиальную приближенную схему (polynomial time approximation scheme, PTAS), если
  1. для любого $\epsilon \geq 0$ существует $(1-\epsilon)$-приближенный алгоритм для Π;
  2. для любого $\epsilon > 0$ время работы такого алгоритма зависит от n полиномиально.
- Если же время работы такого алгоритма полиномиально не только по n, но и по $1/\epsilon$, то полученная схема называется полностью полиномиальной приближенной схемой (fully polynomial time approximation scheme, FPTAS).

Полиномиальная приближенная схема

Определение

- Пусть дана максимизационная задача Π.
- Π имеет полиномиальную приближенную схему (polynomial time approximation scheme, PTAS), если
  1. для любого $\epsilon \geq 0$ существует $(1-\epsilon)$-приближенный алгоритм для Π;
  2. для любого $\epsilon > 0$ время работы такого алгоритма зависит от n полиномиально.
- Если же время работы такого алгоритма полиномиально не только по n, но и по $1/\epsilon$, то полученная схема называется полностью полиномиальной приближенной схемой (fully polynomial time approximation scheme, FPTAS).
- Определение для минимизационной задачи полностью аналогично.

План лекции

1. Что такое приближенные алгоритмы и для чего они нужны
2. Вершинное покрытие
3. Покрытие множествами

Задача о вершинном покрытии

Определение

Задача о вершинном покрытии

Определение

- Вершинным покрытием (vertex cover) неориентированного графа $G = (V, E)$ называется такое подмножество его вершин $V' \subseteq V$, что для любого ребра $(u, v) \in E$ хотя бы одна из вершин u и v содержится в V'.

Задача о вершинном покрытии

Определение

- Вершинным покрытием (vertex cover) неориентированного графа $G = (V, E)$ называется такое подмножество его вершин $V' \subseteq V$, что для любого ребра $(u, v) \in E$ хотя бы одна из вершин u и v содержится в V'.
- Под размером покрытия понимается его мощность.

Задача о вершинном покрытии

Определение

- Вершинным покрытием (vertex cover) неориентированного графа $G = (V, E)$ называется такое подмножество его вершин $V' \subseteq V$, что для любого ребра $(u, v) \in E$ хотя бы одна из вершин u и v содержится в V'.
- Под размером покрытия понимается его мощность.
- Задача о вершинном покрытии (vertex cover problem) заключается в нахождении покрытия минимальной мощности.

Задача о вершинном покрытии

Определение

- Вершинным покрытием (vertex cover) неориентированного графа $G = (V, E)$ называется такое подмножество его вершин $V' \subseteq V$, что для любого ребра $(u, v) \in E$ хотя бы одна из вершин u и v содержится в V'.
- Под размером покрытия понимается его мощность.
- Задача о вершинном покрытии (vertex cover problem) заключается в нахождении покрытия минимальной мощности.
- Decision problem: по графу и числу k определить, существует ли покрытие мощности не более k.

NP-трудность

Факт

NP-трудность

Факт

- Задача о вершинном покрытии является NP-трудной, а ее decision version – NP-полной.

NP-трудность

Факт

- Задача о вершинном покрытии является NP-трудной, а ее decision version – NP-полной.
- Одна из знаменитых 21-й NP-полной задачи Карпа.

NP-трудность

Факт

- Задача о вершинном покрытии является NP-трудной, а ее decision version – NP-полной.
- Одна из знаменитых 21-й NP-полной задачи Карпа.
- Остается NP-полной даже на графах степени 3.

NP-трудность

Факт

- Задача о вершинном покрытии является NP-трудной, а ее decision version – NP-полной.
- Одна из знаменитых 21-й NP-полной задачи Карпа.
- Остается NP-полной даже на графах степени 3.
- NP-полнота может быть доказана сведением от 3-выполнимости (3-satisfiability) или проблемы нахождения максимальной клики (clique problem).

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)

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

2-приближенный алгоритм

Алгоритм

APPROX-VERTEX-COVER(G)

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

2-приближенный алгоритм

Алгоритм

Approx-Vertex-Cover(G)

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

2-приближенный алгоритм

Алгоритм

Approx-Vertex-Cover(G)

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

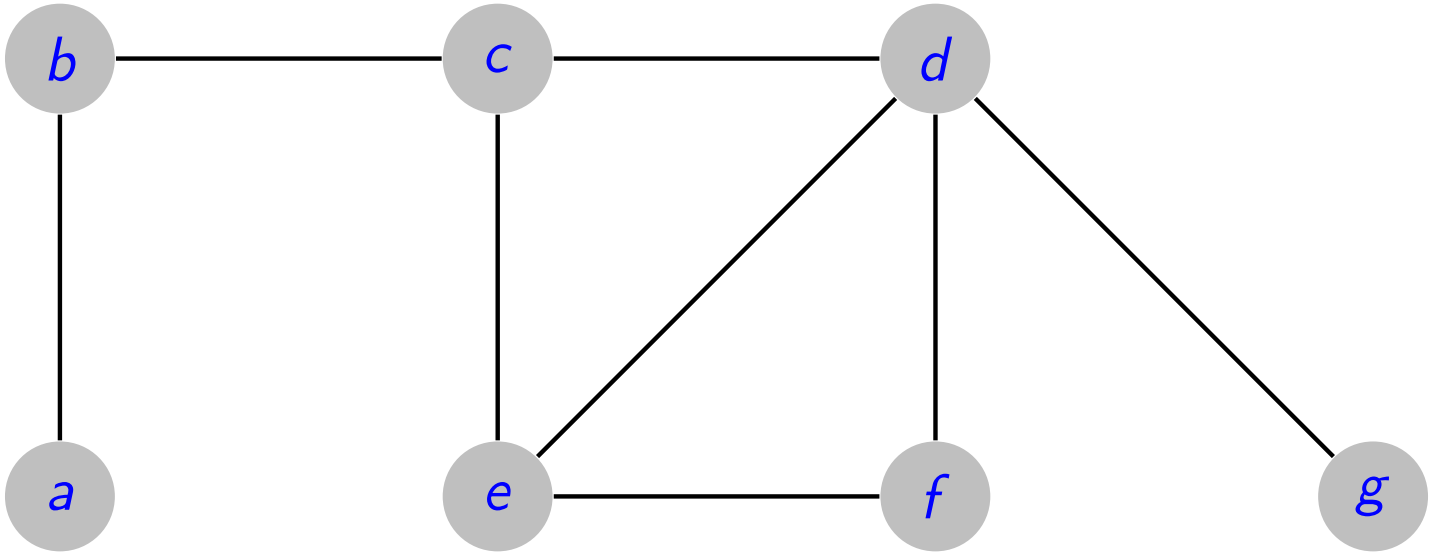
2-приближенный алгоритм

Алгоритм

Approx-Vertex-Cover(G)

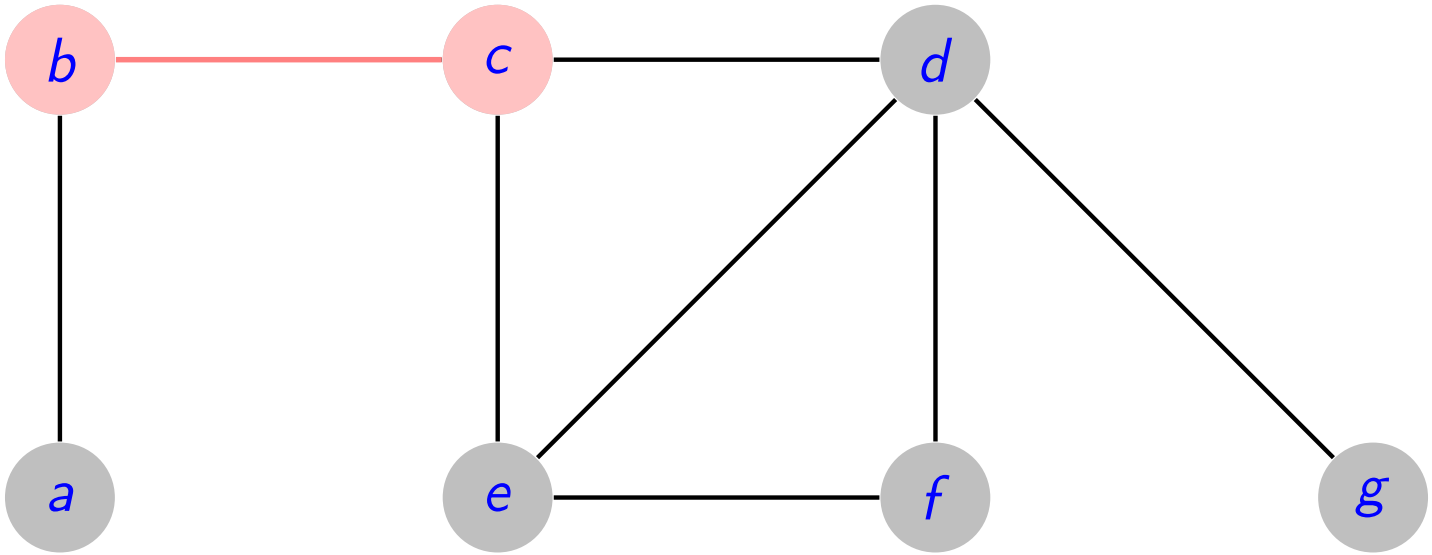
- $C := \emptyset$
- $E' := E[G]$
- while $E' \neq \emptyset$
  - берем произвольное ребро $(u, v) \in E'$
  - $C := C \cup \{u, v\}$
  - удалим из E' все покрытые ребра
- return C

Пример работы алгоритма



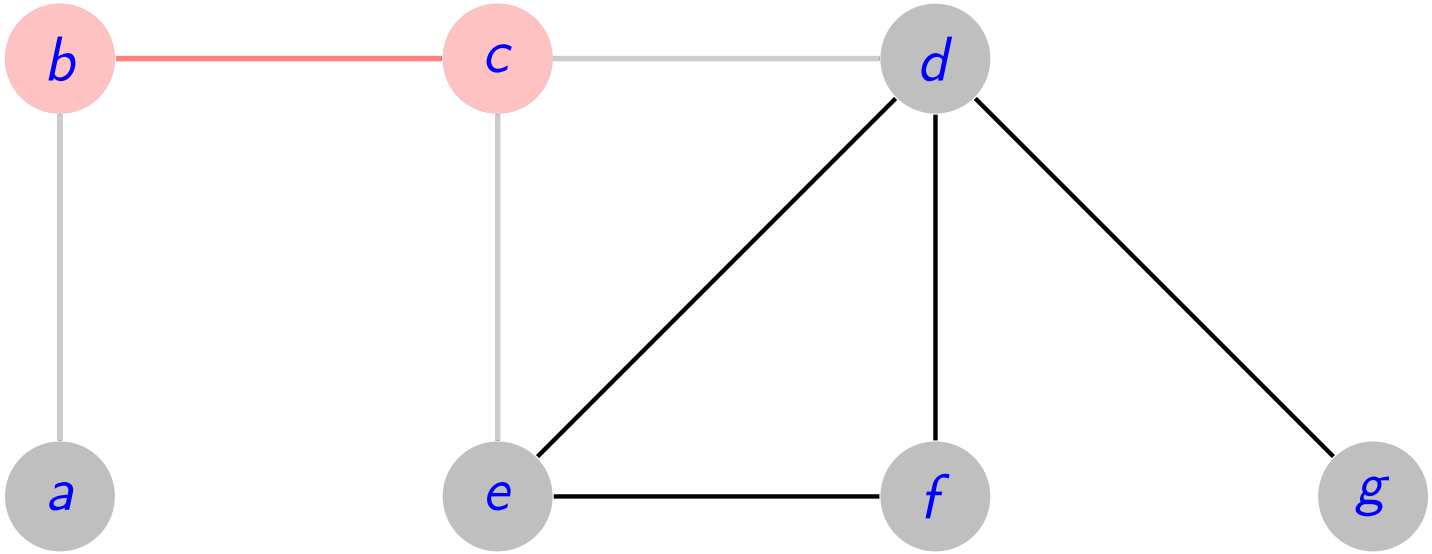
дан граф на семи вершинах

Пример работы алгоритма



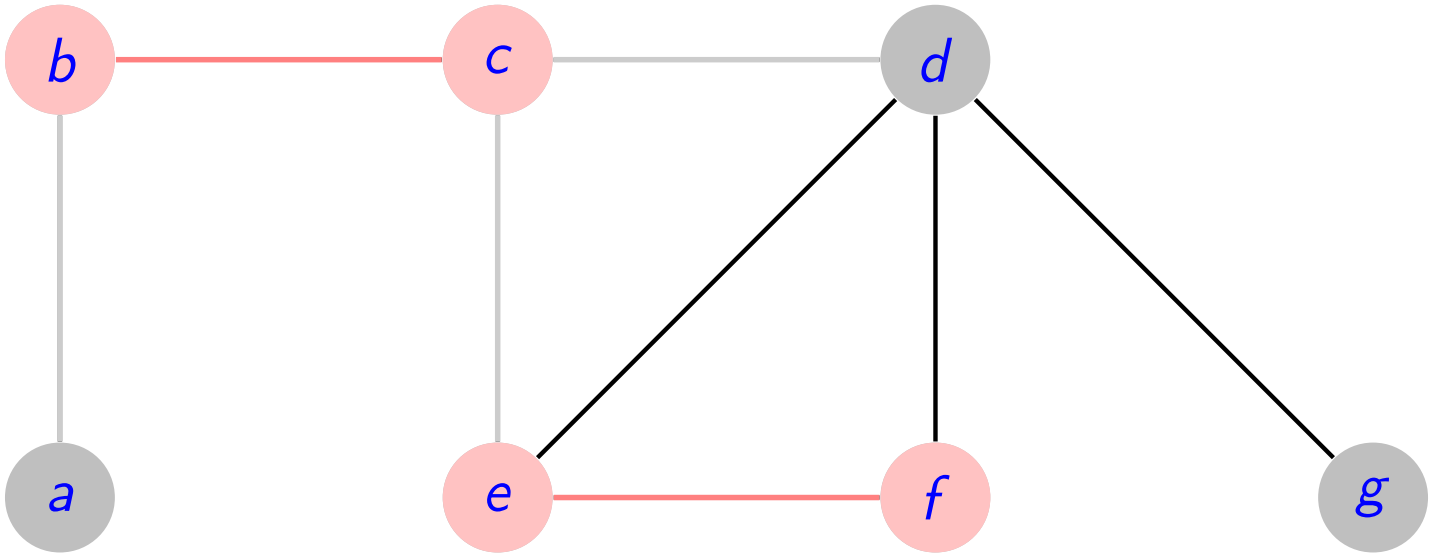
выбираем ребро (b, c)

Пример работы алгоритма



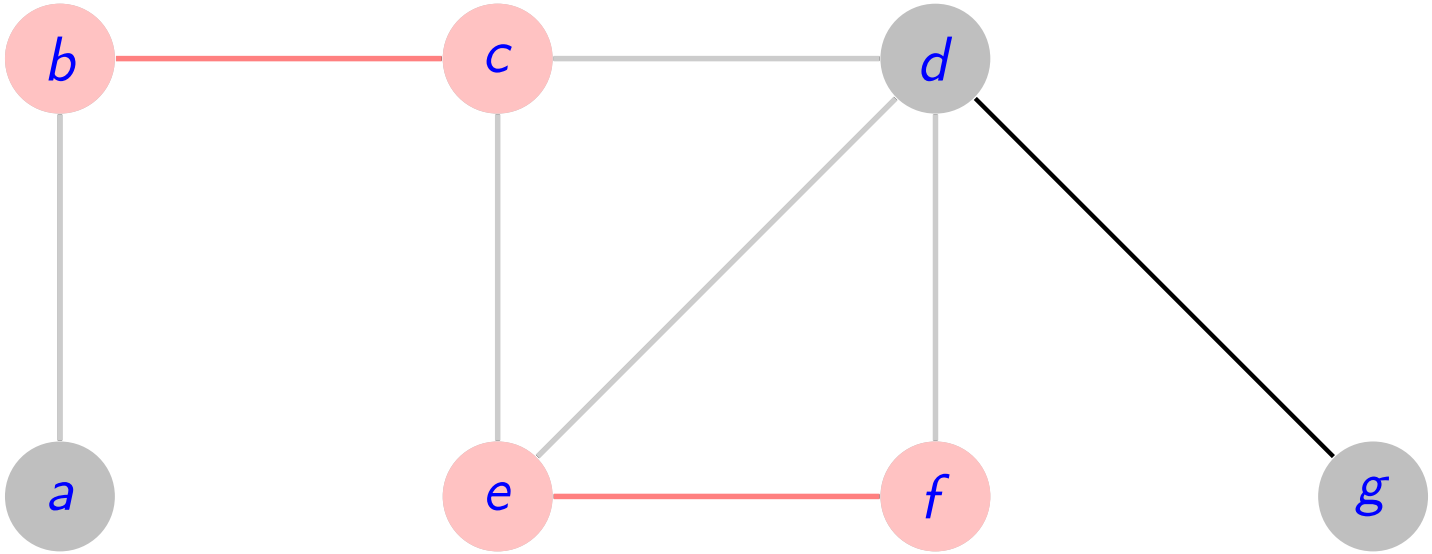
покрытые ребра удаляем

Пример работы алгоритма



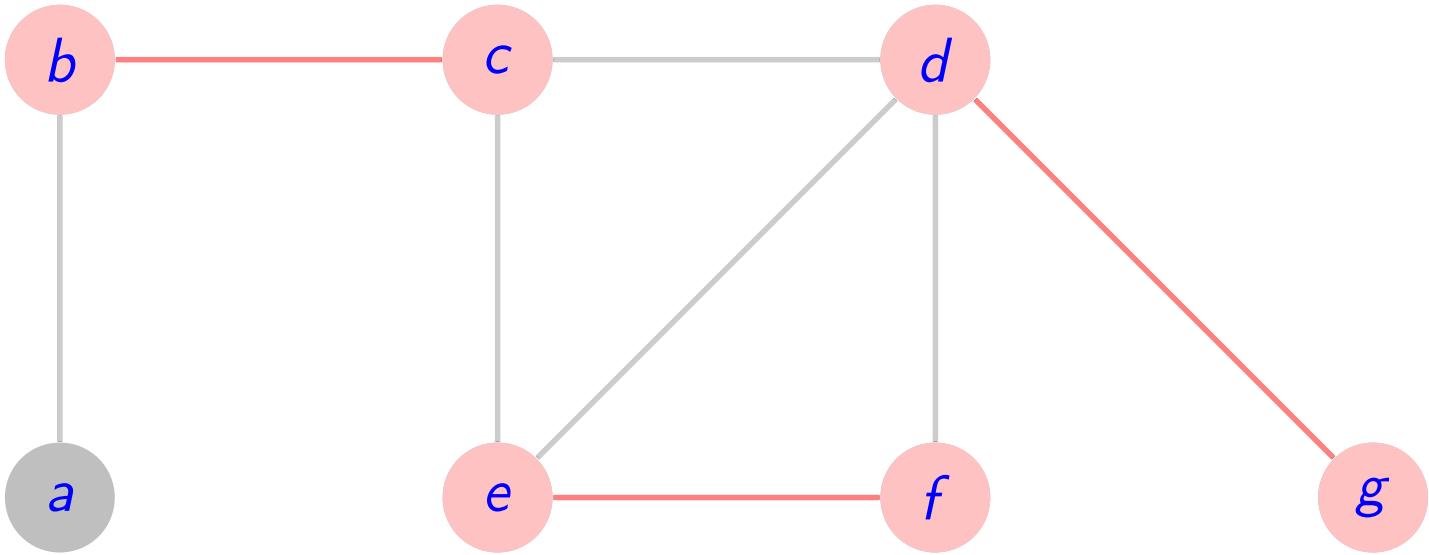
выбираем ребро (e, f)

Пример работы алгоритма



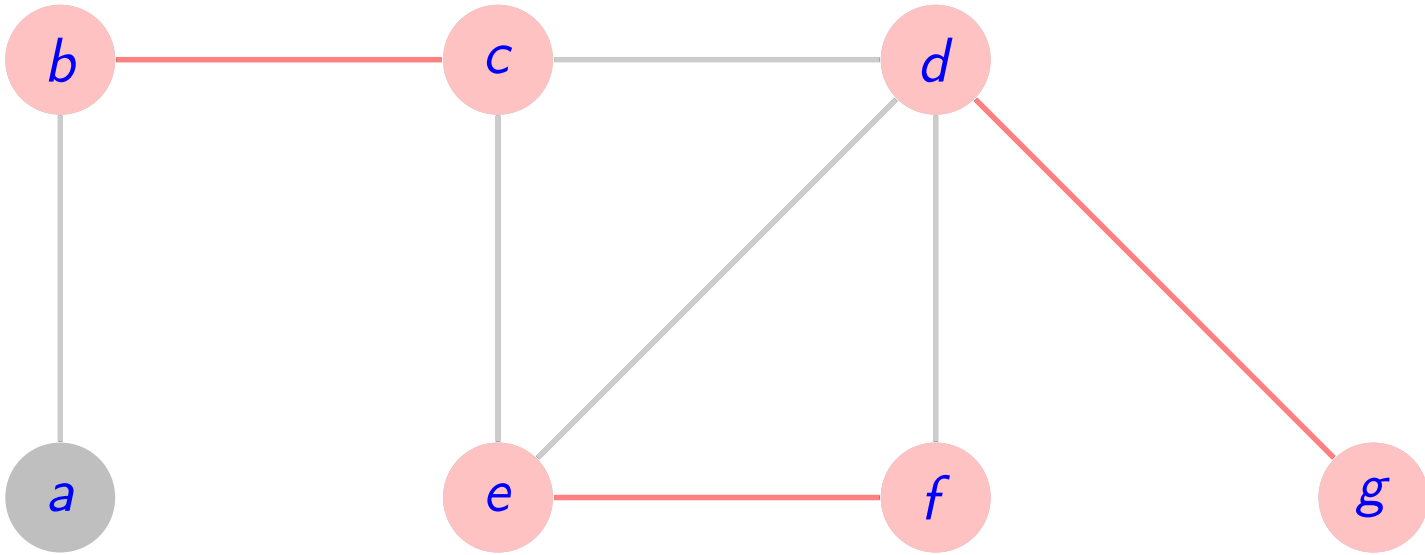
покрытые ребра удаляем

Пример работы алгоритма



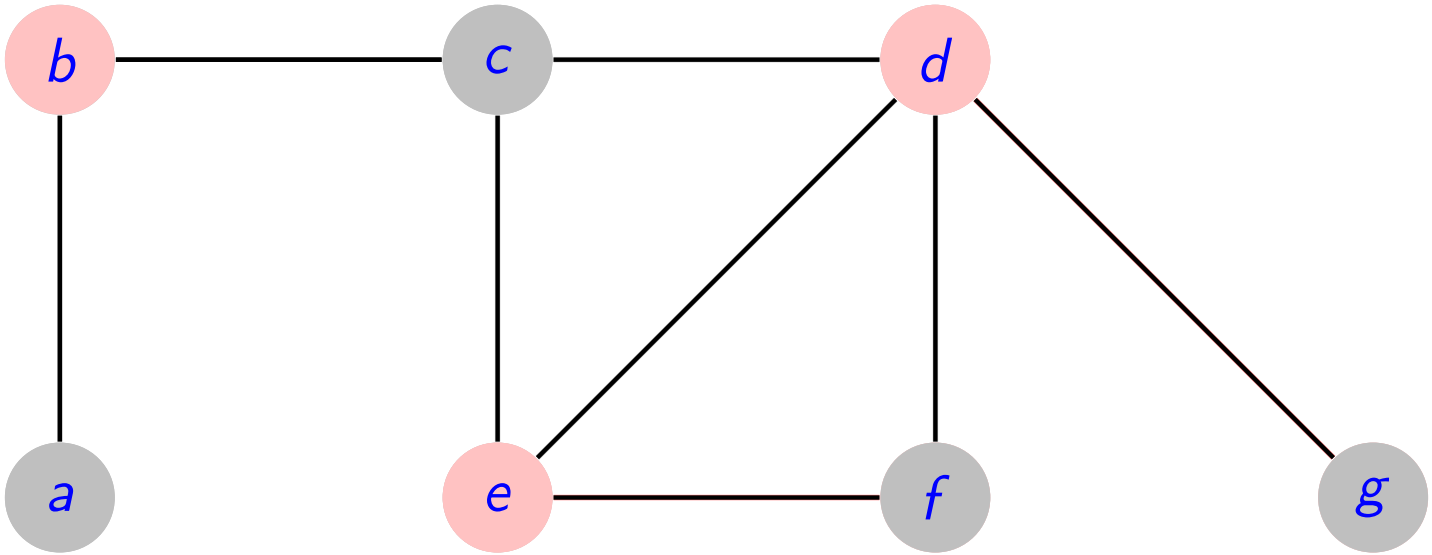
выбираем ребро (d, g)

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, $|C| = 2|A| \leq 2|C_{\text{opt}}|$

Факт

Лучшей константы для задачи о вершинном покрытии неизвестно, но известно, что из $P \neq NP$ следует, что не существует 1.3606-оптимального алгоритма.

План лекции

Задача о покрытии множествами

Определение

Задача о покрытии множествами

Определение

- Дано множество $U = \{u_1, \ldots, u_n\}$ и семейство его подмножеств $\mathcal{F} = \{S_1, \ldots, S_k\}$, $S_i \subseteq U$.

Задача о покрытии множествами

Определение

- Дано множество $U = \{u_1, \ldots, u_n\}$ и семейство его подмножеств $\mathcal{F} = \{S_1, \ldots, S_k\}$, $S_i \subseteq U$.
- В сумме все подмножества покрывают U: $U = \bigcup_{S \in \mathcal{F}} S$.

Задача о покрытии множествами

Определение

- Дано множество $U = \{u_1, \ldots, u_n\}$ и семейство его подмножеств $\mathcal{F} = \{S_1, \ldots, S_k\}$, $S_i \subseteq U$.
- В сумме все подмножества покрывают U: $U = \bigcup_{S \in \mathcal{F}} S$.
- Каждому подмножеству S_i сопоставлена некоторая неотрицательная стоимость $p_i \geq 0$.

Задача о покрытии множествами

Определение

- Дано множество $U = \{u_1, \ldots, u_n\}$ и семейство его подмножеств $\mathcal{F} = \{S_1, \ldots, S_k\}$, $S_i \subseteq U$.
- В сумме все подмножества покрывают U: $U = \bigcup_{S \in \mathcal{F}} S$.
- Каждому подмножеству S_i сопоставлена некоторая неотрицательная стоимость $p_i \geq 0$.
- Задача о покрытии множествами (set cover problem) заключается в нахождении набора подмножеств, покрывающего все множество U и имеющего минимальный возможный вес.

Замечание

Замечание

- Задача о вершинном покрытии является частным случаем задачи о покрытии множествами, где каждое подмножество содержит ровно два элемента.

Замечание

- Задача о вершинном покрытии является частным случаем задачи о покрытии множествами, где каждое подмножество содержит ровно два элемента.
- Одна из знаменитых 21-й NP-полной задачи Карпа.

Жадный приближенный алгоритм

Неформально

На каждом шаге выбираем множество с минимальной удельной стоимостью покрываемых им элементов

Жадный приближенный алгоритм

Неформально

На каждом шаге выбираем множество с минимальной удельной стоимостью покрываемых им элементов

Формально

Greedy-Set-Cover($X, \mathcal{F}$)

Жадный приближенный алгоритм

Неформально

На каждом шаге выбираем множество с минимальной удельной стоимостью покрываемых им элементов

Формально

Greedy-Set-Cover$(X, \mathcal{F})$

- $I := \emptyset$

Жадный приближенный алгоритм

Неформально

На каждом шаге выбираем множество с минимальной удельной стоимостью покрываемых им элементов

Формально

GREEDY-SET-COVER$(X, \mathcal{F})$

- $I := \emptyset$
- while $\bigcup_{j \in I} S_j \neq X$

Жадный приближенный алгоритм

Неформально

На каждом шаге выбираем множество с минимальной удельной стоимостью покрываемых им элементов

Формально

GREEDY-SET-COVER$(X, \mathcal{F})$

- $I := \emptyset$
- while $\bigcup_{j \in I} S_j \neq X$
  - $\forall i \notin I, \text{cost}[i] := p_i / |S_i \setminus \bigcup_{j \in I} S_j|$

Жадный приближенный алгоритм

Неформально

На каждом шаге выбираем множество с минимальной удельной стоимостью покрываемых им элементов

Формально

GREEDY-SET-COVER$(X, \mathcal{F})$

- $I := \emptyset$
- while $\bigcup_{j \in I} S_j \neq X$
  - $\forall i \notin I, \text{cost}[i] := p_i / |S_i \setminus \bigcup_{j \in I} S_j|$
  - выберем i_0, такое что $\text{cost}[i_0] = \min_{i \notin I} \text{cost}[i]$

Жадный приближенный алгоритм

Неформально

На каждом шаге выбираем множество с минимальной удельной стоимостью покрываемых им элементов

Формально

GREEDY-SET-COVER$(X, \mathcal{F})$

- $I := \emptyset$
- while $\bigcup_{j \in I} S_j \neq X$
  - $\forall i \notin I, \text{cost}[i] := p_i / |S_i \setminus \bigcup_{j \in I} S_j|$
  - выберем i_0, такое что $\text{cost}[i_0] = \min_{i \notin I} \text{cost}[i]$
  - $I := I \cup \{i_0\}$

Жадный приближенный алгоритм

Неформально

На каждом шаге выбираем множество с минимальной удельной стоимостью покрываемых им элементов

Формально

GREEDY-SET-COVER$(X, \mathcal{F})$

- $I := \emptyset$
- while $\bigcup_{j \in I} S_j \neq X$
  - $\forall i \notin I, \text{cost}[i] := p_i / |S_i \setminus \bigcup_{j \in I} S_j|$
  - выберем i_0, такое что $\text{cost}[i_0] = \min_{i \notin I} \text{cost}[i]$
  - $I := I \cup \{i_0\}$
- return I

Анализ алгоритма

Теорема

Алгоритм Greedy-Set-Cover является H_n-приближенным, где $H_n = 1 + 1/2 + \cdots + 1/n$ – сумма первых n членов гармонического ряда.

Доказательство.

Анализ алгоритма

Теорема

Алгоритм Greedy-Set-Cover является H_n-приближенным, где $H_n = 1 + 1/2 + \cdots + 1/n$ – сумма первых n членов гармонического ряда.

Доказательство.

- перенумеруем все элементы множества U в том порядке, как мы их покрывали

Анализ алгоритма

Теорема

Алгоритм Greedy-Set-Cover является H_n-приближенным, где $H_n = 1 + 1/2 + \cdots + 1/n$ – сумма первых n членов гармонического ряда.

Доказательство.

- перенумеруем все элементы множества U в том порядке, как мы их покрывали
- каждому x_i присвоим стоимость c_i, равную стоимости множества, которым его впервые покрыли

Анализ алгоритма

Теорема

Алгоритм Greedy-Set-Cover является H_n-приближенным, где $H_n = 1 + 1/2 + \cdots + 1/n$ – сумма первых n членов гармонического ряда.

Доказательство.

- перенумеруем все элементы множества U в том порядке, как мы их покрывали
- каждому x_i присвоим стоимость c_i, равную стоимости множества, которым его впервые покрыли
- как будет показано чуть позже, $c_i \leq P_{\text{opt}}/(n - i + 1)$

Анализ алгоритма

Теорема

Алгоритм Greedy-Set-Cover является H_n-приближенным, где $H_n = 1 + 1/2 + \cdots + 1/n$ – сумма первых n членов гармонического ряда.

Доказательство.

- перенумеруем все элементы множества U в том порядке, как мы их покрывали
- каждому x_i присвоим стоимость c_i, равную стоимости множества, которым его впервые покрыли
- как будет показано чуть позже, $c_i \leq P_{\text{opt}}/(n - i + 1)$
- $\sum_{i \in I} p_i = \sum_{j=1}^{n} c_j$, поскольку сумма стоимостей всех элементов, покрываемых при добавлении множества S_i равна как раз p_i

Анализ алгоритма

Теорема

Алгоритм Greedy-Set-Cover является H_n-приближенным, где $H_n = 1 + 1/2 + \cdots + 1/n$ – сумма первых n членов гармонического ряда.

Доказательство.

- перенумеруем все элементы множества U в том порядке, как мы их покрывали
- каждому x_i присвоим стоимость c_i, равную стоимости множества, которым его впервые покрыли
- как будет показано чуть позже, $c_i \leq P_{\text{opt}}/(n - i + 1)$
- $\sum_{i \in I} p_i = \sum_{j=1}^{n} c_j$, поскольку сумма стоимостей всех элементов, покрываемых при добавлении множества S_i равна как раз p_i
- $\sum_{j=1}^{n} c_j \leq P_{\text{opt}}(1 + 1/2 + \cdots + 1/n)$

□

Доказательство леммы

Лемма

$$c_i \leq P_{\text{opt}}/(n - i + 1)$$

Доказательство.

Доказательство леммы

Лемма

$$c_i \leq P_{\text{opt}}/(n - i + 1)$$

Доказательство.

- рассмотрим момент, в которой мы собираемся покрыть элемент u_i

Доказательство леммы

Лемма

$$c_i \leq P_{\text{opt}}/(n - i + 1)$$

Доказательство.

- рассмотрим момент, в которой мы собираемся покрыть элемент u_i
- ясно, что непокрытые элементы можно покрыть набором множеств общей стоимостью не более P_{opt}

Доказательство леммы

Лемма

$$c_i \leq P_{\text{opt}}/(n - i + 1)$$

Доказательство.

- рассмотрим момент, в которой мы собираемся покрыть элемент u_i
- ясно, что непокрытые элементы можно покрыть набором множеств общей стоимостью не более P_{opt}
- значит, хотя бы для одного из не выбранных пока множеств S_k выполняется неравенство $\text{cost}[S_k] \leq P_{\text{opt}}/|U \setminus \bigcup_{j \in I} S_j|$

Доказательство леммы

Лемма

$$c_i \leq P_{\text{opt}}/(n - i + 1)$$

Доказательство.

- рассмотрим момент, в которой мы собираемся покрыть элемент u_i
- ясно, что непокрытые элементы можно покрыть набором множеств общей стоимостью не более P_{opt}
- значит, хотя бы для одного из не выбранных пока множеств S_k выполняется неравенство $\text{cost}[S_k] \leq P_{\text{opt}}/|U \setminus \bigcup_{j \in I} S_j|$
- знаменатель последней дроби $\geq n - i + 1$, так как мы в данный момент покрываем u_i

Факт

Из $P \neq NP$ следует, что для задачи о покрытии множествами не существует $c \log n$-приближенного алгоритма (c — некоторая константа).

Что мы узнали за сегодня?

Что мы узнали за сегодня?

Что мы узнали за сегодня?

Что мы узнали за сегодня?

- Что такое приближенные алгоритмы и для чего они нужны.

Что мы узнали за сегодня?

Что мы узнали за сегодня?

- Что такое приближенные алгоритмы и для чего они нужны.
- α-оптимальный алгоритм гарантированно выдает решение, которое не более чем в α раз хуже, чем оптимальное.

Что мы узнали за сегодня?

Что мы узнали за сегодня?

- Что такое приближенные алгоритмы и для чего они нужны.
- α-оптимальный алгоритм гарантированно выдает решение, которое не более чем в α раз хуже, чем оптимальное.
- α может быть как константой, так и функцией от n.

Что мы узнали за сегодня?

Что мы узнали за сегодня?

- Что такое приближенные алгоритмы и для чего они нужны.
- α-оптимальный алгоритм гарантированно выдает решение, которое не более чем в α раз хуже, чем оптимальное.
- α может быть как константой, так и функцией от n.
- Если $P \neq NP$, то для некоторых задач не существует c-приближенных алгоритмов.

Что мы узнали за сегодня?

Что мы узнали за сегодня?

- Что такое приближенные алгоритмы и для чего они нужны.
- α-оптимальный алгоритм гарантированно выдает решение, которое не более чем в α раз хуже, чем оптимальное.
- α может быть как константой, так и функцией от n.
- Если $P \neq NP$, то для некоторых задач не существует c-приближенных алгоритмов.
- В то же время для некоторых задач известны алгоритмы, выдающие тем лучшие приближения, чем больше времени им дали.

Спасибо за внимание!