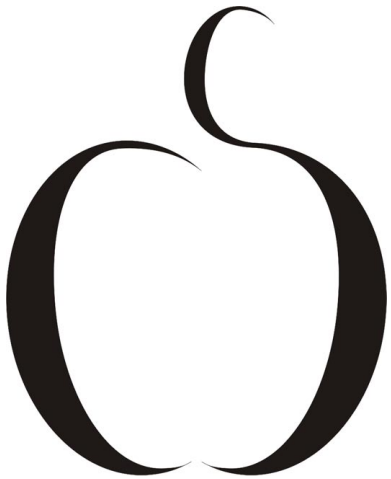
Алгоритмы для NP-трудных задач

Лекция 5: Приближенные алгоритмы

А. Куликов

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



План лекции

1. Определения

План лекции

1. Определения
2. Задача о покрытии множествами

План лекции

1. Определения
2. Задача о покрытии множествами
3. Кратчайшая надпоследовательность

План лекции

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

Приближенный алгоритм

Определение

Приближенный алгоритм

Определение

- Приближенным алгоритмом (approximation algorithm) для задачи П называется полиномиальный по времени алгоритм, возращающий для каждого входа I какое-то решение $x \in \text{sol}(I)$. 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 $x \in \text{sol}(I)$. 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 $|\text{OPT}(I) - A(I)| \leq c$.

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 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;

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 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.
- 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 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.
- 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

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.

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

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

Жадный приближенный алгоритм

Неформально

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

Формально

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

- $I := \emptyset$

Жадный приближенный алгоритм

Неформально

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

Формально

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

- $I := \emptyset$
- 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

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

- $I := \emptyset$
- 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

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

- $I := \emptyset$
- 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

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

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

Жадный приближенный алгоритм

Неформально

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

Формально

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

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

Трудность приближения

Факт

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

План лекции

Задача о кратчайшей общей надпоследовательности

Определение

Задача о кратчайшей общей надпоследовательности

Определение

- Дано множество строк $\{s_1, \ldots, s_n\}$.

Задача о кратчайшей общей надпоследовательности

Определение

- Дано множество строк $\{s_1, \ldots, s_n\}$.
- Задача о кратчайшей общей надпоследовательности (shortest common superstring) заключается в нахождении самой короткой строки u, которая содержит в качестве подстроки каждую из s_i.

Сведение к задаче о покрытии множествами

Сведение к задаче о покрытии множествами

- НУО, среди строк нет подстрок друг друга

Сведение к задаче о покрытии множествами

- НУО, среди строк нет подстрок друг друга
- для всех i, j, k, для которых суффикс строки s_i длины k совпадает с префиксом строки s_j длины k, построим строку $w_{ijk} = s_i \cdot s_j[k+1..|s_j|]$:

Сведение к задаче о покрытии множествами

- НУО, среди строк нет подстрок друг друга
- для всех i, j, k, для которых суффикс строки s_i длины k совпадает с префиксом строки s_j длины k, построим строку $w_{ijk} = s_i \cdot s_j[k+1..|s_j|]$:

$$s_6 = \underbrace{abcab}_{X}\underbrace{bbca}_{Y}, s_2 = \underbrace{bbca}_{Y}\underbrace{aaccbcac}_{Z}$$

Сведение к задаче о покрытии множествами

- НУО, среди строк нет подстрок друг друга
- для всех i, j, k, для которых суффикс строки s_i длины k совпадает с префиксом строки s_j длины k, построим строку $w_{ijk} = s_i \cdot s_j[k+1..|s_j|]$:

$$s_6 = \underbrace{abcab}_{X}\underbrace{bbca}_{Y}, s_2 = \underbrace{bbca}_{Y}\underbrace{aaccbcac}_{Z}$$

$$w_{6,2,4} = \underbrace{abcab}_{X}\underbrace{bbca}_{Y}\underbrace{aaccbcac}_{Z}$$

Сведение к задаче о покрытии множествами

- НУО, среди строк нет подстрок друг друга
- для всех i, j, k, для которых суффикс строки s_i длины k совпадает с префиксом строки s_j длины k, построим строку $w_{ijk} = s_i \cdot s_j[k+1..|s_j|]$:

$$s_6 = \underbrace{abcab}_{X}\underbrace{bbca}_{Y}, s_2 = \underbrace{bbca}_{Y}\underbrace{aaccbcac}_{Z}$$

$$w_{6,2,4} = \underbrace{abcab}_{X}\underbrace{bbca}_{Y}\underbrace{aaccbcac}_{Z}$$

- для любой строки s определим $\mathrm{set}(s) = \{s_i \mid s_i$ — подстрока $s\}$

Сведение к задаче о покрытии множествами

- НУО, среди строк нет подстрок друг друга
- для всех i, j, k, для которых суффикс строки s_i длины k совпадает с префиксом строки s_j длины k, построим строку $w_{ijk} = s_i \cdot s_j[k+1..|s_j|]$:

$$s_6 = \underbrace{abcab}_{X}\underbrace{bbca}_{Y}, s_2 = \underbrace{bbca}_{Y}\underbrace{aaccbcac}_{Z}$$

$$w_{6,2,4} = \underbrace{abcab}_{X}\underbrace{bbca}_{Y}\underbrace{aaccbcac}_{Z}$$

- для любой строки s определим $\mathrm{set}(s) = \{s_i \mid s_i$ — подстрока $s\}$
- стоимость множества $\mathrm{set}(s)$ положим равной длине строки s

Сведение к задаче о покрытии множествами

- НУО, среди строк нет подстрок друг друга
- для всех i, j, k, для которых суффикс строки s_i длины k совпадает с префиксом строки s_j длины k, построим строку $w_{ijk} = s_i \cdot s_j[k+1..|s_j|]$:

$$s_6 = \underbrace{abcab}_{X}\underbrace{bbca}_{Y}, s_2 = \underbrace{bbca}_{Y}\underbrace{aaccbcac}_{Z}$$

$$w_{6,2,4} = \underbrace{abcab}_{X}\underbrace{bbca}_{Y}\underbrace{aaccbcac}_{Z}$$

- для любой строки s определим $\mathrm{set}(s) = \{s_i \mid s_i$ — подстрока $s\}$
- стоимость множества $\mathrm{set}(s)$ положим равной длине строки s
- вход задачи о покрытии множествами: $U = \{s_1, \ldots, s_n\}$, $\mathcal{F} = \{\mathrm{set}(s_i)\} \cup \{\mathrm{set}(w_{ijk})\}$

Сведение к задаче о покрытии множествами (продолжение)

Сведение к задаче о покрытии множествами (продолжение)

- итак, алгоритм возвращает нам некоторые множества

Сведение к задаче о покрытии множествами (продолжение)

- итак, алгоритм возвращает нам некоторые множества
- мы сливаем соответствующие им строки и получаем строку, содержащую все s_i

Сведение к задаче о покрытии множествами (продолжение)

- итак, алгоритм возвращает нам некоторые множества
- мы сливаем соответствующие им строки и получаем строку, содержащую все s_i
- осталось показать, что построенный нами алгоритм является $2H_n$-приближенным

Сведение к задаче о покрытии множествами (продолжение)

- итак, алгоритм возвращает нам некоторые множества
- мы сливаем соответствующие им строки и получаем строку, содержащую все s_i
- осталось показать, что построенный нами алгоритм является $2H_n$-приближенным
- для этого достаточно доказать приведенную ниже лемму

Сведение к задаче о покрытии множествами (продолжение)

- итак, алгоритм возвращает нам некоторые множества
- мы сливаем соответствующие им строки и получаем строку, содержащую все s_i
- осталось показать, что построенный нами алгоритм является $2H_n$-приближенным
- для этого достаточно доказать приведенную ниже лемму

Лемма

Стоимость оптимального решения полученной задачи о покрытии множествами не более чем в два раза хуже длины решения исходной задачи о кратчайшей надпоследовательности.

Доказательство леммы

Доказательство

Доказательство леммы

Доказательство

- перенумеруем строки s_i в порядке их вхождения в (какую-нибудь) оптимальную строку s

Доказательство леммы

Доказательство

- перенумеруем строки s_i в порядке их вхождения в (какую-нибудь) оптимальную строку s
- ясно, что вхождение каждой строки начинается и заканчивается строго позже предыдущей

Доказательство леммы

Доказательство

- перенумеруем строки s_i в порядке их вхождения в (какую-нибудь) оптимальную строку s
- ясно, что вхождение каждой строки начинается и заканчивается строго позже предыдущей
- разделим строки на блоки следующим образом: в первый блок берем s_1 и все s_i, первые вхождения которых начинаются до конца первого вхождения s_1

Доказательство леммы

Доказательство

- перенумеруем строки s_i в порядке их вхождения в (какую-нибудь) оптимальную строку s
- ясно, что вхождение каждой строки начинается и заканчивается строго позже предыдущей
- разделим строки на блоки следующим образом: в первый блок берем s_1 и все s_i, первые вхождения которых начинаются до конца первого вхождения s_1
- сделаем из первой и последней строки этого блока общую строку — получим строку $w_{1,j,k}$ (или же просто s_1)

Доказательство леммы

Доказательство

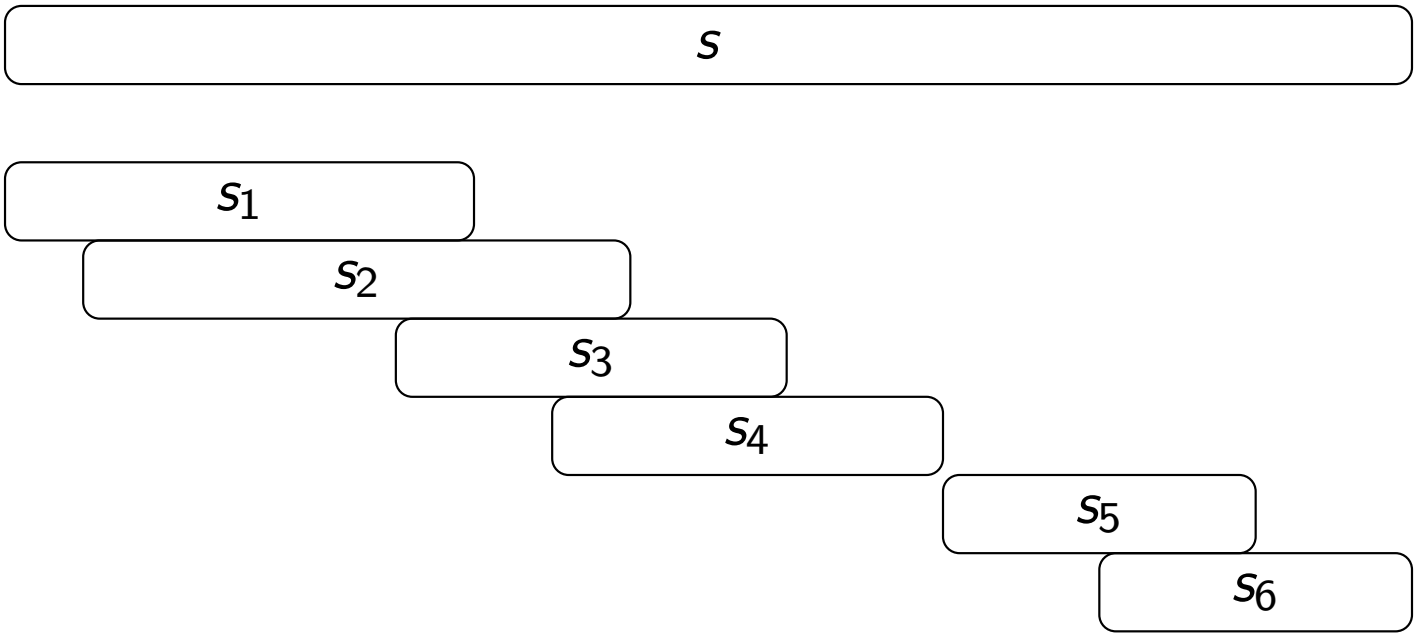
- перенумеруем строки s_i в порядке их вхождения в (какую-нибудь) оптимальную строку s
- ясно, что вхождение каждой строки начинается и заканчивается строго позже предыдущей
- разделим строки на блоки следующим образом: в первый блок берем s_1 и все s_i, первые вхождения которых начинаются до конца первого вхождения s_1
- сделаем из первой и последней строки этого блока общую строку — получим строку $w_{1,j,k}$ (или же просто s_1)
- следующий блок строим, начиная с первой строки, не попавшей в первый блок, и т.д.

Доказательство леммы

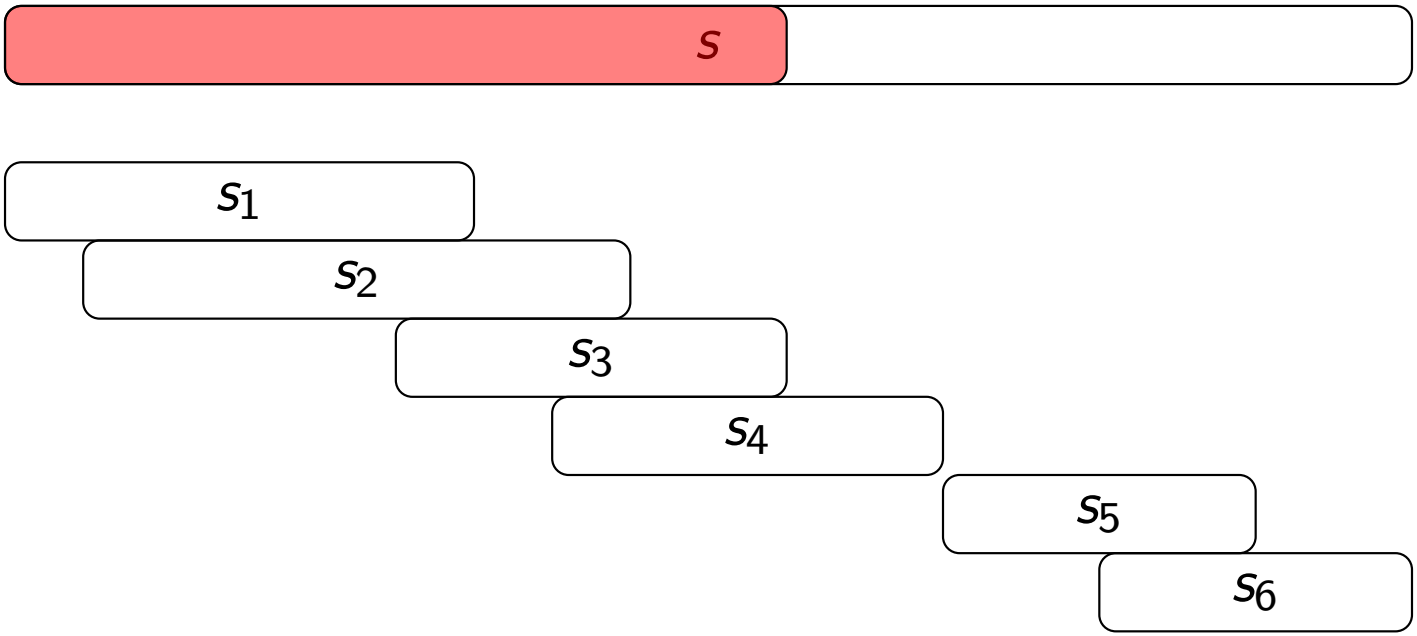
Доказательство

- перенумеруем строки s_i в порядке их вхождения в (какую-нибудь) оптимальную строку s
- ясно, что вхождение каждой строки начинается и заканчивается строго позже предыдущей
- разделим строки на блоки следующим образом: в первый блок берем s_1 и все s_i, первые вхождения которых начинаются до конца первого вхождения s_1
- сделаем из первой и последней строки этого блока общую строку — получим строку $w_{1,j,k}$ (или же просто s_1)
- следующий блок строим, начиная с первой строки, не попавшей в первый блок, и т.д.
- в итоге, получаем некоторое решение задачи о покрытии множествами

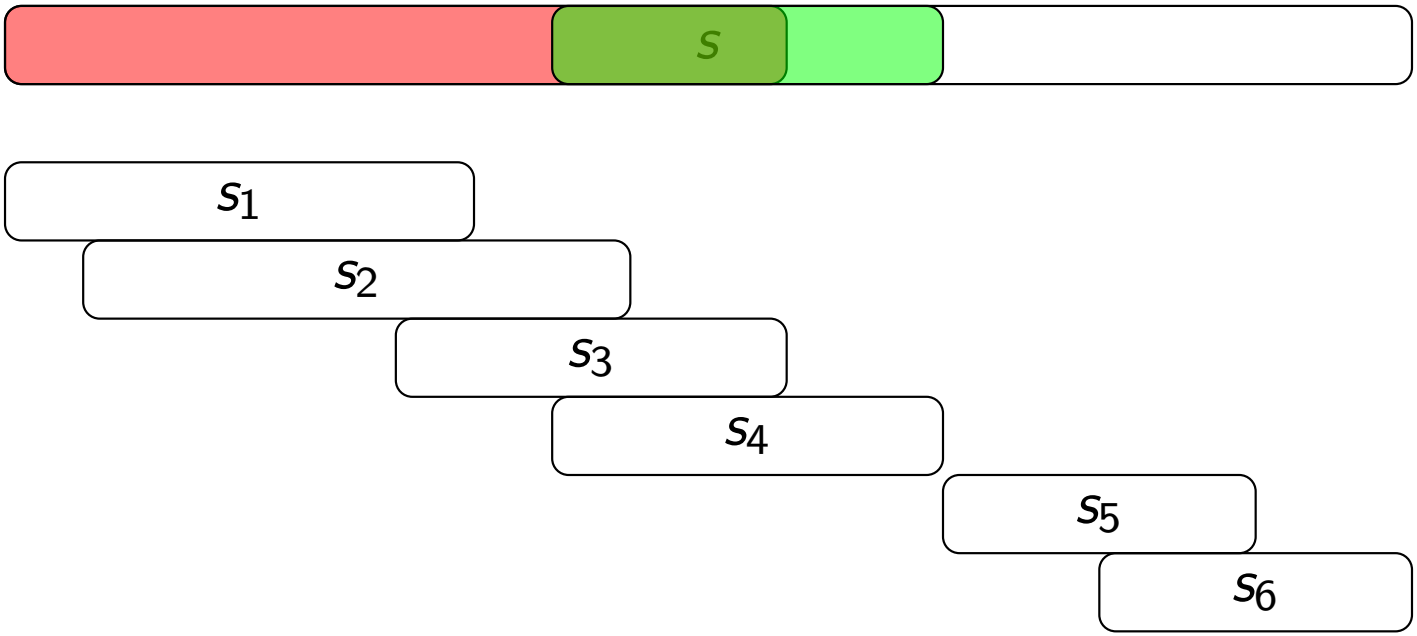
Пример



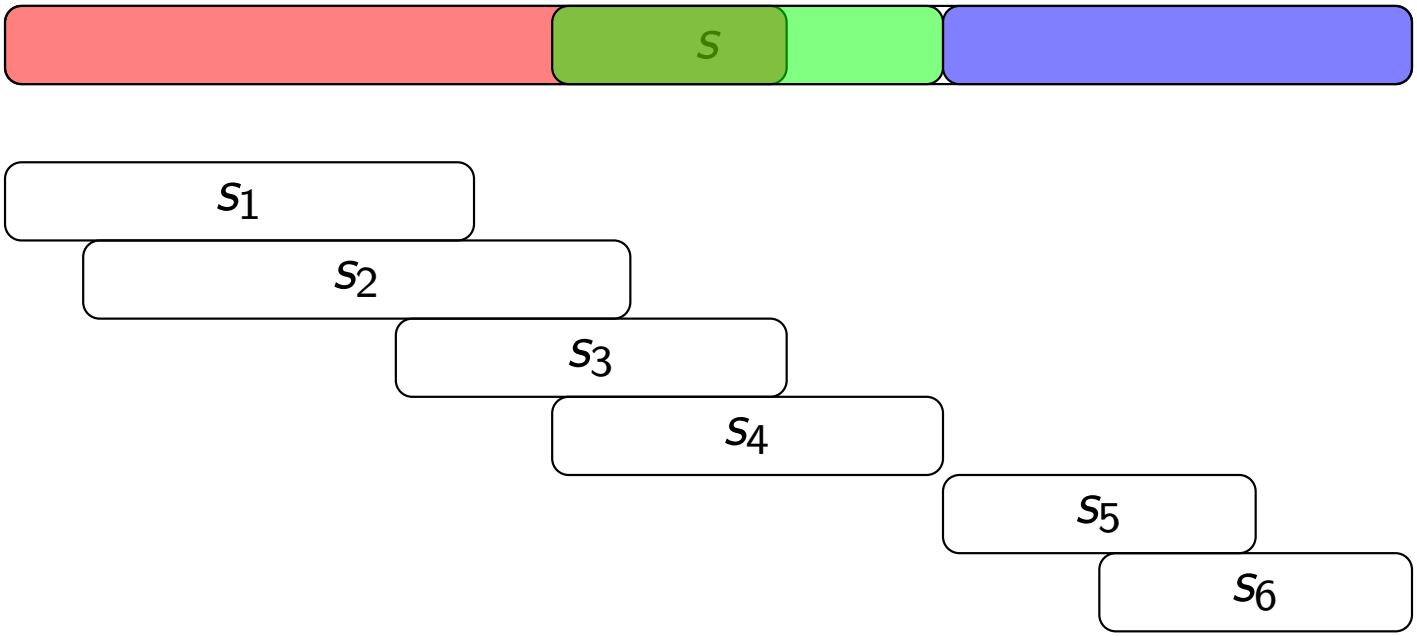
Пример



Пример



Пример



Доказательство леммы (продолжение)

Доказательство

Доказательство леммы (продолжение)

Доказательство

- достаточно показать, что стоимость данного конкретного решения не превосходит удвоенной длины s

Доказательство леммы (продолжение)

Доказательство

- достаточно показать, что стоимость данного конкретного решения не превосходит удвоенной длины s
- это верно, поскольку каждый символ строки s входит не более чем в два блока

□

Пример работы алгоритма

Пример работы алгоритма

- допустим, на вход даны следующие строки:
  $s_1 = abc$, $s_2 = bac$, $s_3 = bcd$, $s_4 = cde$

Пример работы алгоритма

- допустим, на вход даны следующие строки:
  $s_1 = abc$, $s_2 = bac$, $s_3 = bcd$, $s_4 = cde$
- вычисляем w_{ijk}:
  $w_{132} = abcd$, $w_{141} = abcde$, $w_{241} = bacde$, $w_{342} = bcde$

Пример работы алгоритма

- допустим, на вход даны следующие строки:
  $s_1 = abc$, $s_2 = bac$, $s_3 = bcd$, $s_4 = cde$
- вычисляем w_{ijk}:
  $w_{132} = abcd$, $w_{141} = abcde$, $w_{241} = bacde$, $w_{342} = bcde$
- вычисляем $\text{set}(s_i)$ и $\text{set}(w_{ijk})$, а также стоимости:
  $S_1 = \text{set}(s_1) = \{s_1\}$, $p_1 = 3$,
  $S_2 = \text{set}(s_2) = \{s_2\}$, $p_2 = 3$,
  $S_3 = \text{set}(s_3) = \{s_3\}$, $p_3 = 3$,
  $S_4 = \text{set}(s_4) = \{s_4\}$, $p_4 = 3$,
  $S_5 = \text{set}(w_{132}) = \{s_1, s_3\}$, $p_5 = 4$
  $S_6 = \text{set}(w_{141}) = \{s_1, s_3, s_4\}$, $p_6 = 5$,
  $S_7 = \text{set}(w_{241}) = \{s_2, s_4\}$, $p_7 = 5$,
  $S_8 = \text{set}(w_{342}) = \{s_3, s_4\}$, $p_8 = 4$

Пример работы алгоритма

Пример работы алгоритма

- итак, входными данными задачи о покрытии множествами являются универсальное множество $U = \{s_i\}_{i \in [1..4]}$, множество его подмножеств $\mathcal{F} = \{S_i\}_{i \in [1..8]}$, где каждому подмножеству S_i сопоставлена стоимость p_i

Пример работы алгоритма

- итак, входными данными задачи о покрытии множествами являются универсальное множество $U = \{s_i\}_{i \in [1..4]}$, множество его подмножеств $\mathcal{F} = \{S_i\}_{i \in [1..8]}$, где каждому подмножеству S_i сопоставлена стоимость p_i
- алгоритм последовательно добавляет в покрытие множества S_6 и S_2

Пример работы алгоритма

- итак, входными данными задачи о покрытии множествами являются универсальное множество $U = \{s_i\}_{i \in [1..4]}$, множество его подмножеств $\mathcal{F} = \{S_i\}_{i \in [1..8]}$, где каждому подмножеству S_i сопоставлена стоимость p_i
- алгоритм последовательно добавляет в покрытие множества S_6 и S_2
- поскольку $S_6 = \text{set}(w_{141})$ и $S_2 = \text{set}(s_2)$, выходом нашего алгоритма будет конкатенация строк w_{141} и s_2, то есть строка *abcdebac*, которая в данном конкретном примере является оптимальной

Известно лучшее приближение

Факт

Существует 2.75-приближённый алгоритм.

Спасибо за внимание!