

с/к “Эффективные алгоритмы”

Лекция 3: Приближенные алгоритмы

А. Куликов

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



План лекции

1 Задача о коммивояжере

План лекции

1 Задача о коммивояжере

2 Задача о рюкзаке

Приближенный алгоритм

Определение

Приближенный алгоритм

Определение

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

Приближенный алгоритм

Определение

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

Приближенный алгоритм

Определение

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

Приближенный алгоритм

Определение

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

Приближенный алгоритм

Определение

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

Полиномиальная приближенная схема

Определение

- Пусть дана максимизационная задача Π .
- Π 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 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.

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 Задача о коммивояжере

2 Задача о рюкзаке

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:

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

NP-трудность

NP-трудность

- Задача о коммивояжере является NP-трудной.

NP-трудность

- Задача о коммивояжере является NP-трудной.
- Тривиальный алгоритм перебирает все возможные пути за время $n!$.

NP-трудность

- Задача о коммивояжере является NP-трудной.
- Тривиальный алгоритм перебирает все возможные пути за время $n!$.
- Наилучший известный на данный момент алгоритм основан на динамическом программировании и имеет время работы (а также память) 2^n .

NP-трудность

- Задача о коммивояжере является NP-трудной.
- Тривиальный алгоритм перебирает все возможные пути за время $n!$.
- Наилучший известный на данный момент алгоритм основан на динамическом программировании и имеет время работы (а также память) 2^n .
- Задача о коммивояжере в метрическом пространстве тоже является NP-трудной.

NP-трудность

- Задача о коммивояжере является NP-трудной.
- Тривиальный алгоритм перебирает все возможные пути за время $n!$.
- Наилучший известный на данный момент алгоритм основан на динамическом программировании и имеет время работы (а также память) 2^n .
- Задача о коммивояжере в метрическом пространстве тоже является NP-трудной.
 - ▶ допустим, мы умеем решать эту задачу за полиномиальное время

NP-трудность

- Задача о коммивояжере является NP-трудной.
- Тривиальный алгоритм перебирает все возможные пути за время $n!$.
- Наилучший известный на данный момент алгоритм основан на динамическом программировании и имеет время работы (а также память) 2^n .
- Задача о коммивояжере в метрическом пространстве тоже является NP-трудной.
 - ▶ допустим, мы умеем решать эту задачу за полиномиальное время
 - ▶ возьмем произвольный граф и присвоим всем его ребрам вес 1

NP-трудность

- Задача о коммивояжере является NP-трудной.
- Тривиальный алгоритм перебирает все возможные пути за время $n!$.
- Наилучший известный на данный момент алгоритм основан на динамическом программировании и имеет время работы (а также память) 2^n .
- Задача о коммивояжере в метрическом пространстве тоже является NP-трудной.
 - ▶ допустим, мы умеем решать эту задачу за полиномиальное время
 - ▶ возьмем произвольный граф и присвоим всем его ребрам вес 1
 - ▶ между любыми двумя не соединенными ребром вершинами добавим ребро веса 2

NP-трудность

- Задача о коммивояжере является NP-трудной.
- Тривиальный алгоритм перебирает все возможные пути за время $n!$.
- Наилучший известный на данный момент алгоритм основан на динамическом программировании и имеет время работы (а также память) 2^n .
- Задача о коммивояжере в метрическом пространстве тоже является NP-трудной.
 - ▶ допустим, мы умеем решать эту задачу за полиномиальное время
 - ▶ возьмем произвольный граф и присвоим всем его ребрам вес 1
 - ▶ между любыми двумя не соединенными ребром вершинами добавим ребро веса 2
 - ▶ тогда построение минимального гамильтонова цикла в полученном графе эквивалентно ответу на вопрос, существует ли в исходном графе гамильтонов цикл

2-оптимальный алгоритм

Алгоритм

APPROX-TSP(G)

2-оптимальный алгоритм

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G

2-оптимальный алгоритм

Алгоритм

APPROX-TSP(G)

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

2-оптимальный алгоритм

Алгоритм

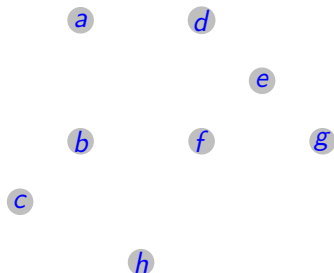
APPROX-TSP(G)

- строим минимальное 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)

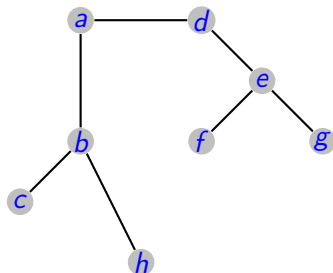


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

- строим минимальное остовное дерево T графа G

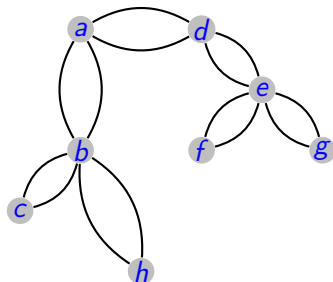


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

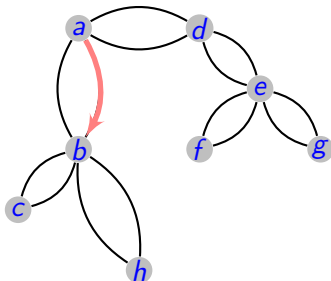


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

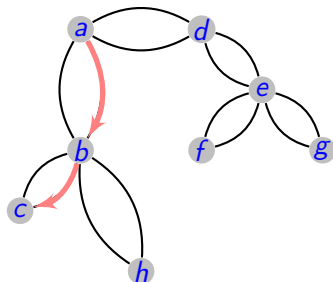


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

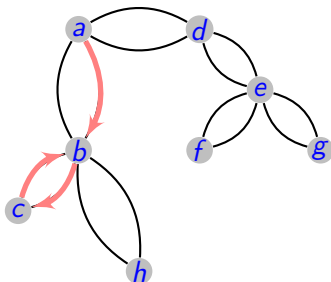


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

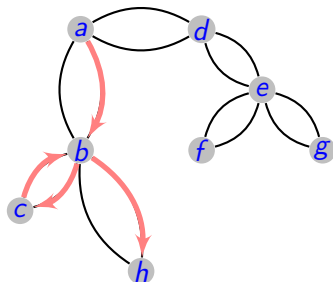


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

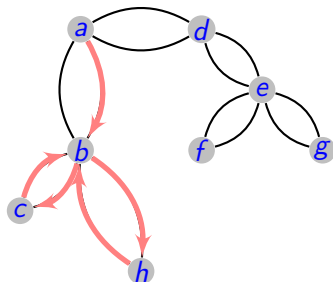


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

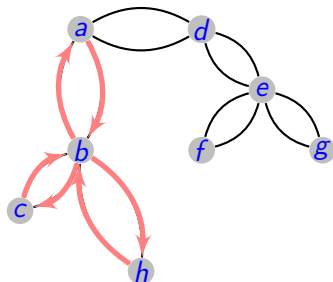


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

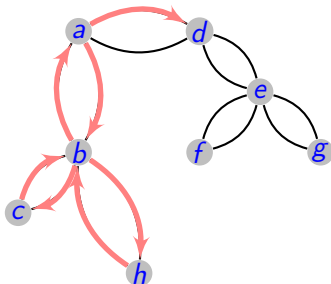


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

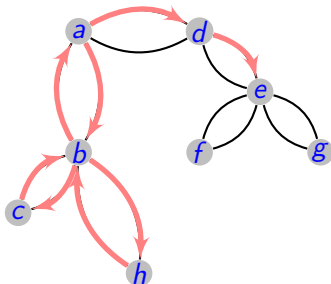


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

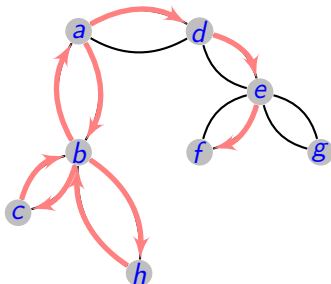


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

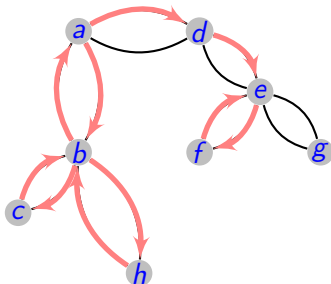


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

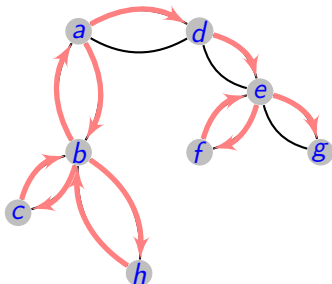


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

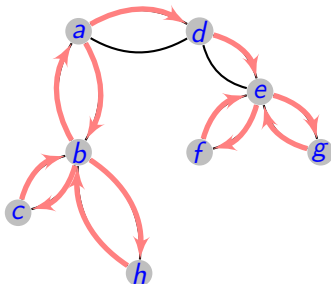


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

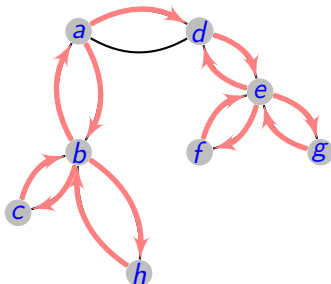


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

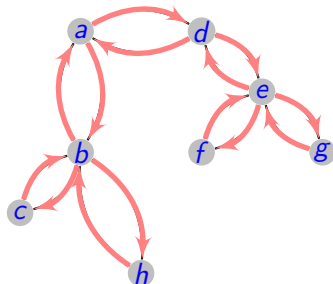


Пример работы алгоритма

Алгоритм

APPROX-TSP(G)

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

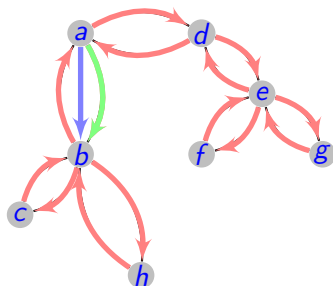


Пример работы алгоритма

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

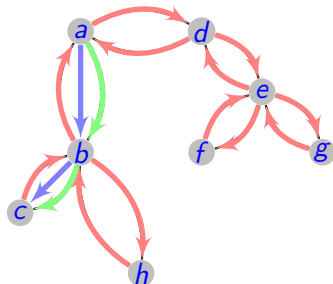


Пример работы алгоритма

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

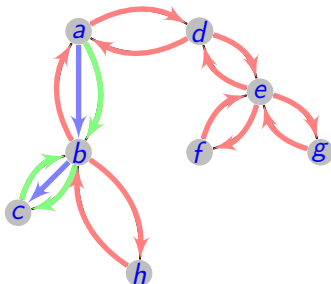


Пример работы алгоритма

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

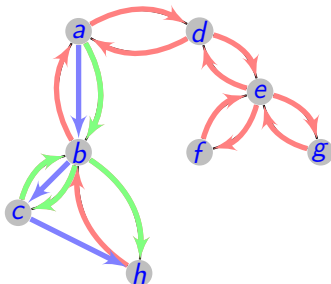


Пример работы алгоритма

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

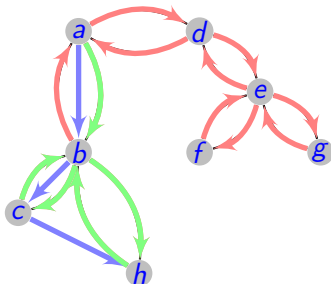


Пример работы алгоритма

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

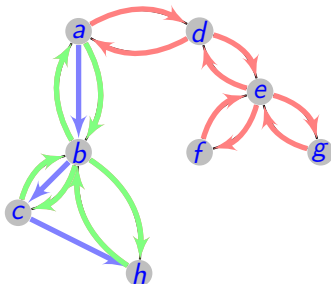


Пример работы алгоритма

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

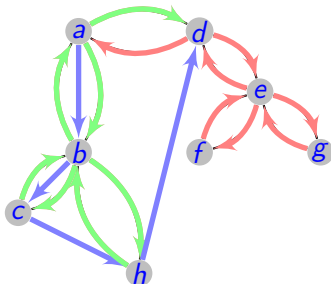


Пример работы алгоритма

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

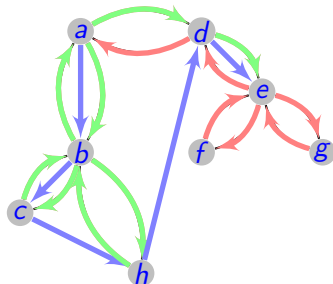


Пример работы алгоритма

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

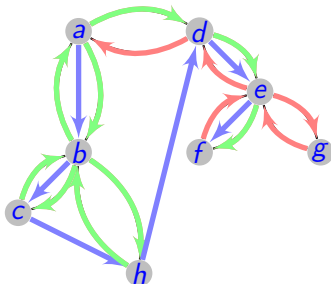


Пример работы алгоритма

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

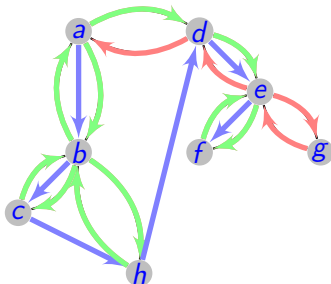


Пример работы алгоритма

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

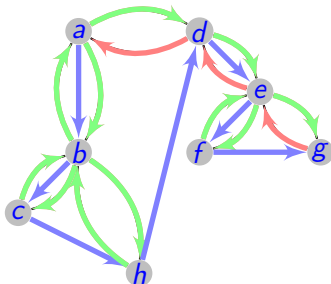


Пример работы алгоритма

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

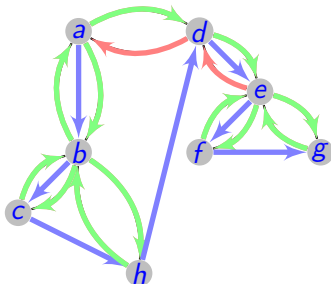


Пример работы алгоритма

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

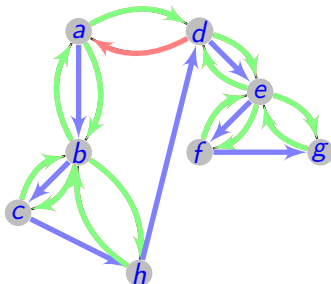


Пример работы алгоритма

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

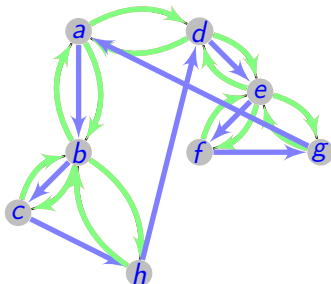


Пример работы алгоритма

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

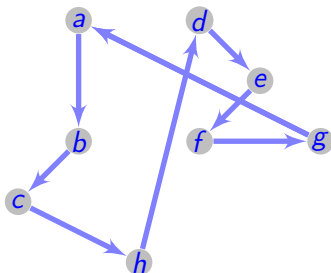


Пример работы алгоритма

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

3/2-оптимальный алгоритм

Алгоритм

APPROX-TSP-IMPROVED(G)

3/2-оптимальный алгоритм

Алгоритм

APPROX-TSP-IMPROVED(G)

- строим минимальное остовное дерево T графа G

3/2-оптимальный алгоритм

Алгоритм

APPROX-TSP-IMPROVED(G)

- строим минимальное остовное дерево T графа G
- найдем минимальное полное паросочетание всех вершин дерева T нечетной степени

3/2-оптимальный алгоритм

Алгоритм

APPROX-TSP-IMPROVED(G)

- строим минимальное остовное дерево T графа G
- найдем минимальное полное паросочетание всех вершин дерева T нечетной степени
- добавим найденные ребра в дерево T и найдем в полученном графе эйлеров цикл

3/2-оптимальный алгоритм

Алгоритм

APPROX-TSP-IMPROVED(G)

- строим минимальное остовное дерево T графа G
- найдем минимальное полное паросочетание всех вершин дерева T нечетной степени
- добавим найденные ребра в дерево T и найдем в полученном графе эйлеров цикл
- выкинем из полученного цикла все повторения вершин и вернем полученный цикл

О чем вы забыли меня спросить?

- О чем вы забыли меня спросить?

О чем вы забыли меня спросить?

- О том, как искать минимальное полное паросочетание в графе.

О чем вы забыли меня спросить?

- О том, как искать минимальное полное паросочетание в графе.
- Давайте пока просто поверим в то, что оно находится за полиномиальное время.

Анализ алгоритма

Теорема

Алгоритм APPROX-TSP-IMPROVED является $3/2$ -приближенным.

Анализ алгоритма

Теорема

Алгоритм APPROX-TSP-IMPROVED является $3/2$ -приближенным.

Доказательство

Анализ алгоритма

Теорема

Алгоритм APPROX-TSP-IMPROVED является $3/2$ -приближенным.

Доказательство

- как и в предыдущем доказательстве, вес построенного цикла не превосходит $W_T + W_P$, где W_P — вес минимального паросочетания вершин нечетной степени дерева T

Анализ алгоритма

Теорема

Алгоритм APPROX-TSP-IMPROVED является $3/2$ -приближенным.

Доказательство

- как и в предыдущем доказательстве, вес построенного цикла не превосходит $W_T + W_P$, где W_P — вес минимального паросочетания вершин нечетной степени дерева T
- нужно показать, что $W_P \leq W_{\text{opt}}/2$

Анализ алгоритма

Теорема

Алгоритм APPROX-TSP-IMPROVED является $3/2$ -приближенным.

Доказательство

- как и в предыдущем доказательстве, вес построенного цикла не превосходит $W_T + W_P$, где W_P — вес минимального паросочетания вершин нечетной степени дерева T
- нужно показать, что $W_P \leq W_{\text{opt}}/2$
- обозначим через A множество всех вершин нечетной степени дерева T

Анализ алгоритма

Теорема

Алгоритм APPROX-TSP-IMPROVED является $3/2$ -приближенным.

Доказательство

- как и в предыдущем доказательстве, вес построенного цикла не превосходит $W_T + W_P$, где W_P — вес минимального паросочетания вершин нечетной степени дерева T
- нужно показать, что $W_P \leq W_{\text{opt}}/2$
- обозначим через A множество всех вершин нечетной степени дерева T
- рассмотрим такой гамильтонов цикл на вершинах множества A : вершины множества A в нем будут встречаться в такой последовательности, в какой они идут в оптимальном гамильтонов цикле графа G

Доказательство (продолжение)

Доказательство

Доказательство (продолжение)

Доказательство

- важно отметить, что нам не нужно строить такой цикл; нам важен лишь факт его существования

Доказательство (продолжение)

Доказательство

- важно отметить, что нам не нужно строить такой цикл; нам важен лишь факт его существования
- разбив вершины только что построенного цикла на четные и нечетные, мы получим два паросочетания

Доказательство (продолжение)

Доказательство

- важно отметить, что нам не нужно строить такой цикл; нам важен лишь факт его существования
- разбив вершины только что построенного цикла на четные и нечетные, мы получим два паросочетания
- вес хотя бы одного из них будет не более $W_{\text{opt}}/2$

Доказательство (продолжение)

Доказательство

- важно отметить, что нам не нужно строить такой цикл; нам важен лишь факт его существования
- разбив вершины только что построенного цикла на четные и нечетные, мы получим два паросочетания
- вес хотя бы одного из них будет не более $W_{\text{opt}}/2$
- значит, и вес минимального паросочетания не превосходит $W_{\text{opt}}/2$



Известные оценки

Факт

Факт

- Если $P \neq NP$, то не существует $\frac{117}{116}$ -приближенного алгоритма для задачи о коммивояжере в метрическом пространстве.

Известные оценки

Факт

- Если $P \neq NP$, то не существует $\frac{117}{116}$ -приближенного алгоритма для задачи о коммивояжере в метрическом пространстве.
- $3/2$ — лучшее известное приближение для задачи о коммивояжере в метрическом пространстве.

Факт

- Если $P \neq NP$, то не существует $\frac{117}{116}$ -приближенного алгоритма для задачи о коммивояжере в метрическом пространстве.
- $3/2$ — лучшее известное приближение для задачи о коммивояжере в метрическом пространстве.
- Для случая, когда все ребра имеют вес 1 или 2, известен $5/6$ -приближенный алгоритм.

Неприближаемость задачи о коммивояжере

Теорема

Пусть $P \neq NP$ и $\alpha(n) = c \in \mathbb{N}$. Тогда не существует $\alpha(n)$ -приближенного алгоритма для задачи о коммивояжере.

Неприближаемость задачи о коммивояжере

Теорема

Пусть $P \neq NP$ и $\alpha(n) = c \in \mathbb{N}$. Тогда не существует $\alpha(n)$ -приближенного алгоритма для задачи о коммивояжере.

Доказательство

Неприближаемость задачи о коммивояжере

Теорема

Пусть $P \neq NP$ и $\alpha(n) = c \in \mathbb{N}$. Тогда не существует $\alpha(n)$ -приближенного алгоритма для задачи о коммивояжере.

Доказательство

- предположим, что такой алгоритм все-таки существует

Неприближаемость задачи о коммивояжере

Теорема

Пусть $P \neq NP$ и $\alpha(n) = c \in \mathbb{N}$. Тогда не существует $\alpha(n)$ -приближенного алгоритма для задачи о коммивояжере.

Доказательство

- предположим, что такой алгоритм все-таки существует
- возьмем произвольный граф и присвоим всем его ребрам вес 1

Неприближаемость задачи о коммивояжере

Теорема

Пусть $P \neq NP$ и $\alpha(n) = c \in \mathbb{N}$. Тогда не существует $\alpha(n)$ -приближенного алгоритма для задачи о коммивояжере.

Доказательство

- предположим, что такой алгоритм все-таки существует
- возьмем произвольный граф и присвоим всем его ребрам вес 1
- между любыми двумя не соединенными ребром вершинами добавим ребро веса $cn + 1$

Неприближаемость задачи о коммивояжере

Теорема

Пусть $P \neq NP$ и $\alpha(n) = c \in \mathbb{N}$. Тогда не существует $\alpha(n)$ -приближенного алгоритма для задачи о коммивояжере.

Доказательство

- предположим, что такой алгоритм все-таки существует
- возьмем произвольный граф и присвоим всем его ребрам вес 1
- между любыми двумя не соединенными ребром вершинами добавим ребро веса $cn + 1$
- заметим теперь, что если в исходном графе существует гамильтонов цикл, то в новом графе существует гамильтонов цикл веса n

Доказательство (продолжение)

Доказательство

Доказательство (продолжение)

Доказательство

- если же такого цикла в исходном графе нет, то самый дешевый цикл в новом графе имеет вес хотя бы $(nc + 1) + (n - 1) > nc$

Доказательство (продолжение)

Доказательство

- если же такого цикла в исходном графе нет, то самый дешевый цикл в новом графе имеет вес хотя бы $(nc + 1) + (n - 1) > nc$
- таким образом, с помощью $\alpha(n)$ -приближенного алгоритма для задачи о коммивояжере мы можем понять, стоимость оптимального цикла в построенном графе превосходит n или нет

Доказательство (продолжение)

Доказательство

- если же такого цикла в исходном графе нет, то самый дешевый цикл в новом графе имеет вес хотя бы $(nc + 1) + (n - 1) > nc$
- таким образом, с помощью $\alpha(n)$ -приближенного алгоритма для задачи о коммивояжере мы можем понять, стоимость оптимального цикла в построенном графе превосходит n или нет
- а это позволит нам понять (за полиномиальное время!), есть в исходном графе гамильтонов цикл или нет

Доказательство (продолжение)

Доказательство

- если же такого цикла в исходном графе нет, то самый дешевый цикл в новом графе имеет вес хотя бы $(nc + 1) + (n - 1) > nc$
- таким образом, с помощью $\alpha(n)$ -приближенного алгоритма для задачи о коммивояжере мы можем понять, стоимость оптимального цикла в построенном графе превосходит n или нет
- а это позволит нам понять (за полиномиальное время!), есть в исходном графе гамильтонов цикл или нет
- но тогда $P = NP$ □

План лекции

1 Задача о коммивояжере

2 Задача о рюкзаке

Задача о рюкзаке

Определение

Задача о рюкзаке

Определение

- Дано n предметов с весами $w_1, \dots, w_n \in \mathbb{N}$ и стоимостями $p_1, \dots, p_n \in \mathbb{N}$, а также общий вес W .

Задача о рюкзаке

Определение

- Дано n предметов с весами $w_1, \dots, w_n \in \mathbb{N}$ и стоимостями $p_1, \dots, p_n \in \mathbb{N}$, а также общий вес W .
- **Задача о рюкзаке** (knapsack problem) заключается в нахождении такого множества предметов $I \subseteq [1..n]$, что $\sum_{i \in I} w_i \leq W$ и $\sum_{i \in I} p_i$ максимально.

Задача о рюкзаке

Определение

- Дано n предметов с весами $w_1, \dots, w_n \in \mathbb{N}$ и стоимостями $p_1, \dots, p_n \in \mathbb{N}$, а также общий вес W .
- **Задача о рюкзаке** (knapsack problem) заключается в нахождении такого множества предметов $I \subseteq [1..n]$, что $\sum_{i \in I} w_i \leq W$ и $\sum_{i \in I} p_i$ максимально.
- НУО, $\max_{i \in [1..n]} w_i \leq W$.

Задача о рюкзаке

Определение

- Дано n предметов с весами $w_1, \dots, w_n \in \mathbb{N}$ и стоимостями $p_1, \dots, p_n \in \mathbb{N}$, а также общий вес W .
- **Задача о рюкзаке** (knapsack problem) заключается в нахождении такого множества предметов $I \subseteq [1..n]$, что $\sum_{i \in I} w_i \leq W$ и $\sum_{i \in I} p_i$ максимально.
- НУО, $\max_{i \in [1..n]} w_i \leq W$.

Факт

Задача о рюкзаке

Определение

- Дано n предметов с весами $w_1, \dots, w_n \in \mathbb{N}$ и стоимостями $p_1, \dots, p_n \in \mathbb{N}$, а также общий вес W .
- **Задача о рюкзаке** (knapsack problem) заключается в нахождении такого множества предметов $I \subseteq [1..n]$, что $\sum_{i \in I} w_i \leq W$ и $\sum_{i \in I} p_i$ максимально.
- НУО, $\max_{i \in [1..n]} w_i \leq W$.

Факт

- Задача о рюкзаке является NP-трудной.

Задача о рюкзаке

Определение

- Дано n предметов с весами $w_1, \dots, w_n \in \mathbb{N}$ и стоимостями $p_1, \dots, p_n \in \mathbb{N}$, а также общий вес W .
- **Задача о рюкзаке** (knapsack problem) заключается в нахождении такого множества предметов $I \subseteq [1..n]$, что $\sum_{i \in I} w_i \leq W$ и $\sum_{i \in I} p_i$ максимально.
- НУО, $\max_{i \in [1..n]} w_i \leq W$.

Факт

- Задача о рюкзаке является NP-трудной.
- Одна из знаменитых 21-й NP-полной задачи Карпа.

Точный псевдополиномиальный алгоритм

Алгоритм

PSUEDOPOLY-KNAPSACK($\{w_i\}, \{p_i\}, W$)

Точный псевдополиномиальный алгоритм

Алгоритм

PSUEDOPOLY-KNAPSACK($\{w_i\}, \{p_i\}, W$)

- $S := \sum_{i \in [1..n]} p_i$

Точный псевдополиномиальный алгоритм

Алгоритм

PSUEDOPOLY-KNAPSACK($\{w_i\}, \{p_i\}, W$)

- $S := \sum_{i \in [1..n]} p_i$
- $P := \max_{i \in [1..n]} p_i$

Точный псевдополиномиальный алгоритм

Алгоритм

PSUEDOPOLY-KNAPSACK($\{w_i\}, \{p_i\}, W$)

- $S := \sum_{i \in [1..n]} p_i$
- $P := \max_{i \in [1..n]} p_i$
- через $w(k, p)$ обозначим минимальный объем, необходимый для того, чтобы уложить предметы с номерами, не превосходящими $k \in [1..n]$, общей стоимостью не менее $p \leq S$

Точный псевдополиномиальный алгоритм

Алгоритм

PSUEDOPOLY-KNAPSACK($\{w_i\}, \{p_i\}, W$)

- $S := \sum_{i \in [1..n]} p_i$
- $P := \max_{i \in [1..n]} p_i$
- через $w(k, p)$ обозначим минимальный объем, необходимый для того, чтобы уложить предметы с номерами, не превосходящими $k \in [1..n]$, общей стоимостью не менее $p \leq S$
- в цикле по k от 1 до n вычисляем $w(k, p)$ для каждого p от 1 до S

Точный псевдополиномиальный алгоритм

Алгоритм

PSUEDOPOLY-KNAPSACK($\{w_i\}, \{p_i\}, W$)

- $S := \sum_{i \in [1..n]} p_i$
- $P := \max_{i \in [1..n]} p_i$
- через $w(k, p)$ обозначим минимальный объем, необходимый для того, чтобы уложить предметы с номерами, не превосходящими $k \in [1..n]$, общей стоимостью не менее $p \leq S$
- в цикле по k от 1 до n вычисляем $w(k, p)$ для каждого p от 1 до S
 - ▶ $w(0, 0) = 0$; $w(0, p) = \infty, p > 0$

Точный псевдополиномиальный алгоритм

Алгоритм

PSUEDOPOLY-KNAPSACK($\{w_i\}, \{p_i\}, W$)

- $S := \sum_{i \in [1..n]} p_i$
- $P := \max_{i \in [1..n]} p_i$
- через $w(k, p)$ обозначим минимальный объем, необходимый для того, чтобы уложить предметы с номерами, не превосходящими $k \in [1..n]$, общей стоимостью не менее $p \leq S$
- в цикле по k от 1 до n вычисляем $w(k, p)$ для каждого p от 1 до S
 - ▶ $w(0, 0) = 0; w(0, p) = \infty, p > 0$
 - ▶ $w(k + 1, p) = \min\{w(k, p), w(k, p - p_{k+1}) + w_{k+1}\}$

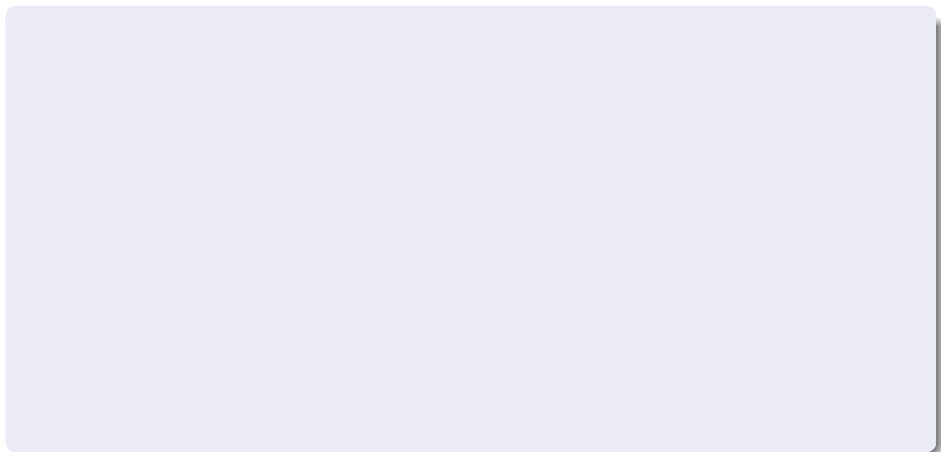
Точный псевдополиномиальный алгоритм

Алгоритм

PSUEDOPOLY-KNAPSACK($\{w_i\}, \{p_i\}, W$)

- $S := \sum_{i \in [1..n]} p_i$
- $P := \max_{i \in [1..n]} p_i$
- через $w(k, p)$ обозначим минимальный объем, необходимый для того, чтобы уложить предметы с номерами, не превосходящими $k \in [1..n]$, общей стоимостью не менее $p \leq S$
- в цикле по k от 1 до n вычисляем $w(k, p)$ для каждого p от 1 до S
 - ▶ $w(0, 0) = 0; w(0, p) = \infty, p > 0$
 - ▶ $w(k + 1, p) = \min\{w(k, p), w(k, p - p_{k+1}) + w_{k+1}\}$
- return максимальное p , для которого $w(n, p) \leq W$

Анализ времени работы



Анализ времени работы

- Ясно, что алгоритм работает не более, чем nS , то есть не более n^2P .

Анализ времени работы

- Ясно, что алгоритм работает не более, чем nS , то есть не более n^2P .
- Значит, мы могли бы решить задачу за полиномиальное время, если бы p_i были достаточно маленькими.

Анализ времени работы

- Ясно, что алгоритм работает не более, чем nS , то есть не более n^2P .
- Значит, мы могли бы решить задачу за полиномиальное время, если бы p_i были достаточно маленькими.
- Идея: выберем достаточно большое число и разделим на него все стоимости. Оптимальное решение не перестанет быть оптимальным, а время работы алгоритма уменьшится.

Анализ времени работы

- Ясно, что алгоритм работает не более, чем nS , то есть не более n^2P .
- Значит, мы могли бы решить задачу за полиномиальное время, если бы p_i были достаточно маленькими.
- Идея: выберем достаточно большое число и разделим на него все стоимости. Оптимальное решение не перестанет быть оптимальным, а время работы алгоритма уменьшится.
- В чем же тогда подвох?

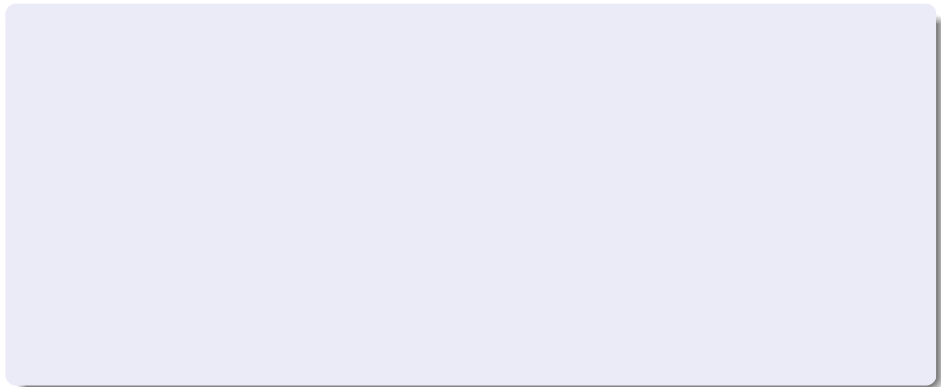
Анализ времени работы

- Ясно, что алгоритм работает не более, чем nS , то есть не более n^2P .
- Значит, мы могли бы решить задачу за полиномиальное время, если бы p_i были достаточно маленькими.
- Идея: выберем достаточно большое число и разделим на него все стоимости. Оптимальное решение не перестанет быть оптимальным, а время работы алгоритма уменьшится.
- В чем же тогда подвох?
- Стоимости могут перестать быть целыми после деления.

Анализ времени работы

- Ясно, что алгоритм работает не более, чем nS , то есть не более n^2P .
- Значит, мы могли бы решить задачу за полиномиальное время, если бы p_i были достаточно маленькими.
- Идея: выберем достаточно большое число и разделим на него все стоимости. Оптимальное решение не перестанет быть оптимальным, а время работы алгоритма уменьшится.
- В чем же тогда подвох?
- Стоимости могут перестать быть целыми после деления.
- Округлим их и будем надеяться, что потеряли мы от этого не очень много.

Полностью полиномиальная приближенная схема



Полностью полиномиальная приближенная схема

- итак, пусть задан ϵ , определяющий, с какой точностью мы хотим найти ответ

Полностью полиномиальная приближенная схема

- итак, пусть задан ϵ , определяющий, с какой точностью мы хотим найти ответ
- пусть $K_\epsilon = \frac{P}{(1+1/\epsilon)n}$

Полностью полиномиальная приближенная схема

- итак, пусть задан ϵ , определяющий, с какой точностью мы хотим найти ответ
- пусть $K_\epsilon = \frac{P}{(1+1/\epsilon)n}$
- поделим все p_i на K_ϵ и округлим: $p'_i = \lceil p_i / K_\epsilon \rceil$

Полностью полиномиальная приближенная схема

- итак, пусть задан ϵ , определяющий, с какой точностью мы хотим найти ответ
- пусть $K_\epsilon = \frac{P}{(1+1/\epsilon)n}$
- поделим все p_i на K_ϵ и округлим: $p'_i = \lceil p_i / K_\epsilon \rceil$
- запустим PSEUDOPOLY-КНАПСАК на полученном наборе стоимостей

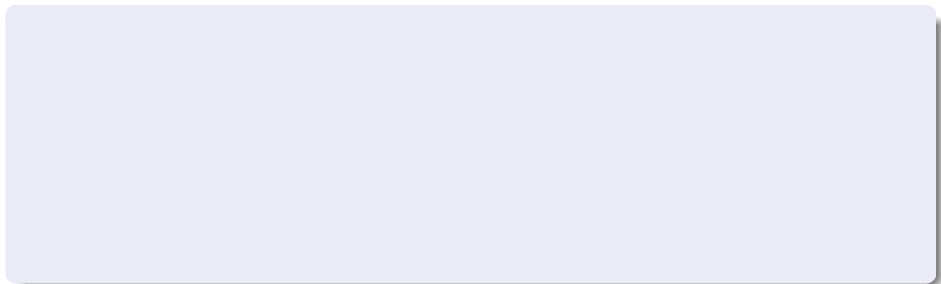
Полностью полиномиальная приближенная схема

- итак, пусть задан ϵ , определяющий, с какой точностью мы хотим найти ответ
- пусть $K_\epsilon = \frac{P}{(1+1/\epsilon)n}$
- поделим все p_i на K_ϵ и округлим: $p'_i = \lceil p_i / K_\epsilon \rceil$
- запустим PSEUDOPOLY-КНАПСАК на полученном наборе стоимостей
- время работы алгоритма не превосходит $n^2 \max p'_i \leq n^3 P(1 + 1/\epsilon) / P$

Полностью полиномиальная приближенная схема

- итак, пусть задан ϵ , определяющий, с какой точностью мы хотим найти ответ
- пусть $K_\epsilon = \frac{P}{(1+1/\epsilon)n}$
- поделим все p_i на K_ϵ и округлим: $p'_i = \lceil p_i / K_\epsilon \rceil$
- запустим PSUEDOPOLY-KNAPSACK на полученном наборе стоимостей
- время работы алгоритма не превосходит $n^2 \max p'_i \leq n^3 P(1 + 1/\epsilon) / P$
- осталось доказать, что полученный алгоритм дает достаточно хорошее приближение

Оценка приближения



Оценка приближения

- пусть A_ϵ — общая стоимость набора предметов, который вернул алгоритм

Оценка приближения

- пусть A_ϵ — общая стоимость набора предметов, который вернул алгоритм
- заметим, что $A_\epsilon \geq A_{\text{opt}} - K_\epsilon n$: псевдополиномиальный алгоритм находит точное решение, поэтому отклонение могло появиться только при округлении; при округлении могло потеряться не более 1 на каждом предмете, эта потеря домножается на K_ϵ при обратном переходе от p'_i к p_i

Оценка приближения (продолжение)

Оценка приближения (продолжение)

- $A_{\text{opt}} \geq P$, так как в рюкзак можно просто засунуть самый дорогой предмет

Оценка приближения (продолжение)

- $A_{\text{opt}} \geq P$, так как в рюкзак можно просто засунуть самый дорогой предмет
- тогда

$$\frac{A_{\epsilon}}{A_{\text{opt}}} \geq \frac{A_{\text{opt}} - K_{\epsilon}}{A_{\text{opt}}} =$$
$$1 - \frac{Pn}{A_{\text{opt}}n(1 + 1/\epsilon)} \geq 1 - \frac{1}{1 + 1/\epsilon} = \frac{1}{\epsilon + 1}$$

Оценка приближения (продолжение)

- $A_{\text{opt}} \geq P$, так как в рюкзак можно просто засунуть самый дорогой предмет
- тогда

$$\frac{A_{\epsilon}}{A_{\text{opt}}} \geq \frac{A_{\text{opt}} - K_{\epsilon}}{A_{\text{opt}}} =$$
$$1 - \frac{Pn}{A_{\text{opt}}n(1 + 1/\epsilon)} \geq 1 - \frac{1}{1 + 1/\epsilon} = \frac{1}{\epsilon + 1}$$

- следовательно, $A_{\epsilon} \geq (1 - \epsilon)A_{\text{opt}}$

Что мы узнали за сегодня?

Что мы узнали за сегодня?

Что мы узнали за сегодня?

Что мы узнали за сегодня?

- $3/2$ -приближенный алгоритм для задачи о коммивояжере в метрическом пространстве.

Что мы узнали за сегодня?

Что мы узнали за сегодня?

- $3/2$ -приближенный алгоритм для задачи о коммивояжере в метрическом пространстве.
- Полностью полиномиальную приближенную схему для задачи о рюкзаке.

Спасибо за внимание!