# Исследование операций

## Приближенные алгоритмы

В.В. Лепин

Институт математики
НАН Беларуси, Минск

# Outline

- Введение и первый пример: MakeSpan problem;
- Ключевая идея при создании приближенного алгоритма: 1) построить допустимое решение, и 2) "сравнить значение критерия с его нижней границей (нижняя оценка OPT)" это не сравнение с OPT";
- Как найти нижнюю границу OPT? Комбинаторная техника, техника линейного программирования (LP-релаксация, двойственность, и др.);
- Задача о покрытии множества: продемонстрируем четыри техники: Greedy, LP+Rounding, Dual-LP+Rounding, Primal_and_dual;
- Другие техники: scaling for Knapsack, pruning for k-Center, TSP, DisjointPaths;

Процесс создания приближенного алгоритма не очень отличается от создания точного алгоритма. Он включает в себя раскрытие структуры проблемы и подбор алгоритмической техники.

Как работать со сложными задачами?

- Большинство задач, имеющих прикладное значение, являются NP-трудными.

# Как работать со сложными задачами?

- Большинство задач, имеющих прикладное значение, являются NP-трудными.
- Пока нет (а скорее вообще несуществует) эффективных алгоритмов для NP-трудных задач.

# Как работать со сложными задачами?

- Большинство задач, имеющих прикладное значение, являются NP-трудными.
- Пока нет (а скорее вообще несуществует) эффективных алгоритмов для NP-трудных задач.
- Напомним, что **эффективный алгоритм** это **детерминированный, полиномиально-временной** алгоритм, который находит **оптимальное** решение даже в **худшем случае**.

# Как работать со сложными задачами?

- Большинство задач, имеющих прикладное значение, являются NP-трудными.
- Пока нет (а скорее вообще несуществует) эффективных алгоритмов для NP-трудных задач.
- Напомним, что **эффективный алгоритм** это **детерминированный, полиномиально-временной** алгоритм, который находит **оптимальное** решение даже в **худшем случае**.
- Как работать со сложными задачами? Использовать компромисс между "качеством" и "временем решения".

# Компромисс между “качеством” и “временем решения”

1. Отказаться от ограничения – **полиномиальное время в наихудшем случае**: хотя наш алгоритм будет требовать экспоненциального времени в наихудшем случае, мы надеемся, что алгоритм будет работать быстро на практических примерах, например, к таким алгоритмам относятся: алгоритмы ветвей и границ (branch-and-bound), ветвления и отсечения (branch-and-cut), а также ветвления и оценки стоимости (branch-and-pricing). Такие алгоритмы позволяют точно решить TSP с более чем 24978 городами.

## Компромисс между “качеством” и “временем решения”

1. Отказаться от ограничения – **полиномиальное время в наихудшем случае**: хотя наш алгоритм будет требовать экспоненциального времени в наихудшем случае, мы надеемся, что алгоритм будет работать быстро на практических примерах, например, к таким алгоритмам относятся: алгоритмы ветвей и границ (branch-and-bound), ветвления и отсечения (branch-and-cut), а также ветвления и оценки стоимости (branch-and-pricing). Такие алгоритмы позволяют точно решить TSP с более чем 24978 городами.
2. Отказаться от ограничения — **точное решение**. Алгоритм будет строить **почти оптимальное решение** в надежде, что его легче найти. Например, это приближенные алгоритмы (с теоретической гарантией), эвристики и локальный поиск (без теоретической гарантии).

# Компромисс между “качеством” и “временем решения”

1. Отказаться от ограничения – **полиномиальное время в наихудшем случае**: хотя наш алгоритм будет требовать экспоненциального времени в наихудшем случае, мы надеемся, что алгоритм будет работать быстро на практических примерах, например, к таким алгоритмам относятся: алгоритмы ветвей и границ (branch-and-bound), ветвления и отсечения (branch-and-cut), а также ветвления и оценки стоимости (branch-and-pricing). Такие алгоритмы позволяют точно решить TSP с более чем 24978 городами.
2. Отказаться от ограничения — **точное решение**. Алгоритм будет строить **почти оптимальное решение** в надежде, что его легче найти. Например, это приближенные алгоритмы (с теоретической гарантией), эвристики и локальный поиск (без теоретической гарантии).
3. Отказаться от ограничения – **детерминированный алгоритм**: мы надеемся, что математическое ожидание времени работы вероятностного алгоритма будет полиномиальным.

## Стратегия приближения

# Стратегия приближения

*Хотя это может показаться парадоксом, во всей точной науке преобладает идея приближения.*

—— B. Russel

- Почему мы изучаем приближенные алгоритмы
  1. Это алгоритмы с теоретической гарантией;

*Хотя это может показаться парадоксом, во всей точной науке преобладает идея приближения.*

—— B. Russel

- Почему мы изучаем приближенные алгоритмы
  1. Это алгоритмы с теоретической гарантией;
  2. Можно использовать алгоритмическую идею – получать хорошие решения практических задач (после тонкой настройки);

# Стратегия приближения

*Хотя это может показаться парадоксом, во всей точной науке преобладает идея приближения.*

—— B. Russel

- Почему мы изучаем приближенные алгоритмы
  1. Это алгоритмы с теоретической гарантией;
  2. Можно использовать алгоритмическую идею – получать хорошие решения практических задач (после тонкой настройки);
  3. Как математически строгий способ анализа эвристики;

# Стратегия приближения

*Хотя это может показаться парадоксом, во всей точной науке преобладает идея приближения.*

—— B. Russel

- Почему мы изучаем приближенные алгоритмы
  1. Это алгоритмы с теоретической гарантией;
  2. Можно использовать алгоритмическую идею – получать хорошие решения практических задач (после тонкой настройки);
  3. Как математически строгий способ анализа эвристики;
  4. Как способ глубже понять комбинаторную сущность проблемы, чтобы раскрыть структуру проблемы;

**Definition ($\alpha$-*приближенный алгоритм*)**

Алгоритм называется $\alpha$-*приближенным алгоритмом* для минимизационной задачи, если:

**Definition ($\alpha$-*приближенный алгоритм*)**

Алгоритм называется $\alpha$-*приближенным алгоритмом* для минимизационной задачи, если:

1. Время: трудоемкость алгоритма — полиномиальная;

# Некоторые определения приближенного алгоритма

**Definition** ($\alpha$-*приближенный алгоритм*)

Алгоритм называется $\alpha$-*приближенным алгоритмом* для минимизационной задачи, если:

1. Время: трудоемкость алгоритма — полиномиальная;
2. Качество: алгоритм выводит решение $S$, значение которого находится в пределах $\alpha$ от значения оптимального решения (обозначаемого $OPT$), т.е. $Value(S) \leq \alpha OPT$.

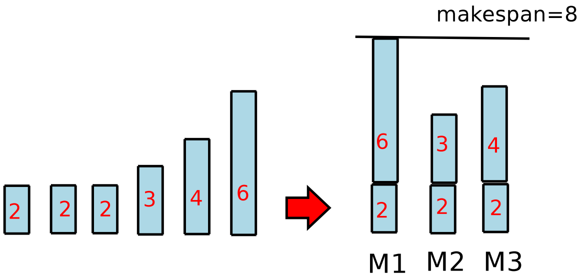
# Некоторые определения приближенного алгоритма

**Definition (PTAS)**

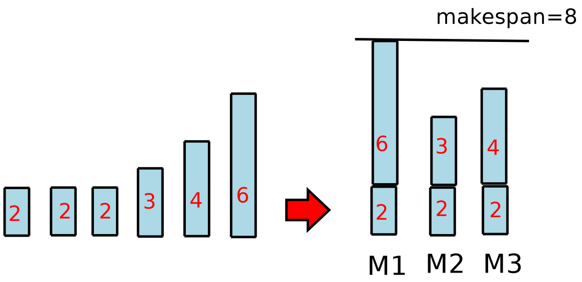
A $PTAS$ (*polynomial time approximation schema (полиномиально временной приближенной схемой)*) для минимизационной задачи называется семейство алгоритмов $\{A_\epsilon : \epsilon > 0\}$ такое, что для каждого $\epsilon$, $A_\epsilon$ является $(1+\epsilon)$-приближенным алгоритмом, работающим полиномиальное время от размера входа.

Первый пример: построение расписания для параллельно работающих машин (задача $Pm||C_{max}$)

- Практическая задача:
  - У нас есть несколько серверов для обработки множества заданий. Цель – распределить задания так, чтобы нагрузка на серверы была максимально сбалансированной.



- Практическая задача:
  - У нас есть несколько серверов для обработки множества заданий. Цель – распределить задания так, чтобы нагрузка на серверы была максимально сбалансированной.
  - Как распределить задания по серверам, так чтобы выполнить все задания как можно раньше?



# Постановка задачи

- Рассматривается задача построения расписания для системы из некоторого числа параллельно работающих машин.

# Постановка задачи

- Рассматривается задача построения расписания для системы из некоторого числа параллельно работающих машин.
- Итак, имеется множество работ и набор одинаковых машин для их выполнения. Каждая работа может быть выполнена на любой машине.

# Постановка задачи

- Рассматривается задача построения расписания для системы из некоторого числа параллельно работающих машин.
- Итак, имеется множество работ и набор одинаковых машин для их выполнения. Каждая работа может быть выполнена на любой машине.
- Под расписанием в этой модели понимается назначение каждой работы на некоторую машину в определенный момент времени.

# Постановка задачи

- Рассматривается задача построения расписания для системы из некоторого числа параллельно работающих машин.
- Итак, имеется множество работ и набор одинаковых машин для их выполнения. Каждая работа может быть выполнена на любой машине.
- Под расписанием в этой модели понимается назначение каждой работы на некоторую машину в определенный момент времени.
- Расписание называется допустимым, если:

# Постановка задачи

- Рассматривается задача построения расписания для системы из некоторого числа параллельно работающих машин.
- Итак, имеется множество работ и набор одинаковых машин для их выполнения. Каждая работа может быть выполнена на любой машине.
- Под расписанием в этой модели понимается назначение каждой работы на некоторую машину в определенный момент времени.
- Расписание называется допустимым, если:
  - каждая работа выполнена полностью,

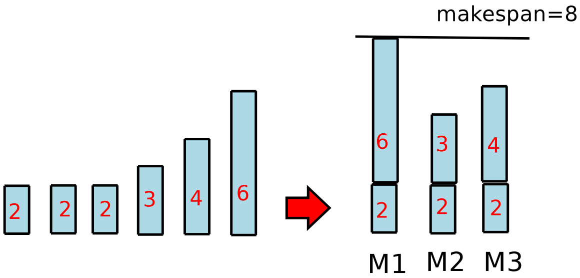
# Постановка задачи

- Рассматривается задача построения расписания для системы из некоторого числа параллельно работающих машин.
- Итак, имеется множество работ и набор одинаковых машин для их выполнения. Каждая работа может быть выполнена на любой машине.
- Под расписанием в этой модели понимается назначение каждой работы на некоторую машину в определенный момент времени.
- Расписание называется допустимым, если:
  - каждая работа выполнена полностью,
  - никакая машина не выполняет одновременно более одной работы,

# Постановка задачи

- Рассматривается задача построения расписания для системы из некоторого числа параллельно работающих машин.
- Итак, имеется множество работ и набор одинаковых машин для их выполнения. Каждая работа может быть выполнена на любой машине.
- Под расписанием в этой модели понимается назначение каждой работы на некоторую машину в определенный момент времени.
- Расписание называется допустимым, если:
  - каждая работа выполнена полностью,
  - никакая машина не выполняет одновременно более одной работы,
  - и ни одна работа не выполняется одновременно на более чем одной машине.

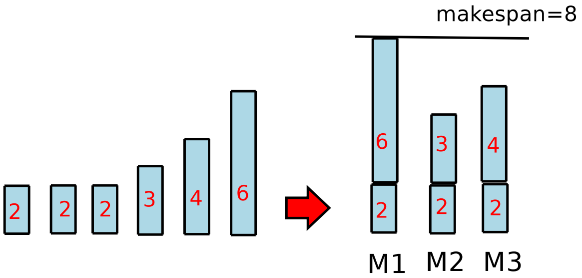
# Постановка задачи




**INPUT:**

$m$ машин $M_1, M_2, ..., M_m$, $n$ работ (каждая работа $j$ имеет время выполнения $t_j$ на любой машине);

# Постановка задачи




**INPUT:**
$m$ машин $M_1, M_2, ..., M_m$, $n$ работ (каждая работа $j$ имеет время выполнения $t_j$ на любой машине);
**OUTPUT:**
Распределение работ по машинам минимизирующее *makespan* $=C_{max}$ — время завершения последней работы,
$T = \max_i \sum_{j \in A(i)} t_j$, где $A(i)$ обозначает множество работ назначенных машине $i$.

# Сложность задачи $Pm||C_{max}$

**Theorem**

*Задача $P2||C_{max}$ является NP-трудной.*

# Сложность задачи $Pm||C_{max}$

**Theorem**

*Задача $P2||C_{max}$ является NP-трудной.*

**Доказательство.**

- Покажем, что Задача о разбиении $\leq_P P2||C_{max} \leq Y$.

$\square$

# Сложность задачи $Pm||C_{max}$

**Theorem**

*Задача $P2||C_{max}$ является NP-трудной.*

**Доказательство.**

- Покажем, что ЗАДАЧА О РАЗБИЕНИИ $\leq_P P2||C_{max} \leq Y$.
- Рассмотрим пример $I$ задачи о разбиении. Дано множество целых чисел $S = \{s_1, ..., s_n\}$, $\sum s_i = 2b$. Построим пример $I'$ задачи $P2||C_{max}$ следующим образом:

$\square$

# Сложность задачи $Pm||C_{max}$

**Theorem**

*Задача $P2||C_{max}$ является NP-трудной.*

**Доказательство.**

- Покажем, что Задача о разбиении $\leq_P P2||C_{max} \leq Y$.
- Рассмотрим пример $I$ задачи о разбиении. Дано множество целых чисел $S = \{s_1, \ldots, s_n\}$, $\sum s_i = 2b$. Построим пример $I'$ задачи $P2||C_{max}$ следующим образом:
  1. Каждому числу $s_i \in S$, поставим в соответствие работу $i$ с $t_i = s_i$ и $Y = b$;

**Theorem**

*Задача $P2||C_{max}$ является NP-трудной.*

**Доказательство.**

- Покажем, что Задача о разбиении $\leq_P P2||C_{max} \leq Y$.
- Рассмотрим пример $I$ задачи о разбиении. Дано множество целых чисел $S = \{s_1, ..., s_n\}$, $\sum s_i = 2b$. Построим пример $I'$ задачи $P2||C_{max}$ следующим образом:
  1. Каждому числу $s_i \in S$, поставим в соответствие работу $i$ с $t_i = s_i$ и $Y = b$;
  2. допустимое расписание существует в том и только в том случае, когда существует $S \subset A$, $\sum_{a_i \in S} a_i = b$.

□

**Theorem**

*Задача $P2||C_{max}$ является NP-трудной.*

**Доказательство.**

- Покажем, что Задача о разбиении $\leq_P P2||C_{max} \leq Y$.
- Рассмотрим пример $I$ задачи о разбиении. Дано множество целых чисел $S = \{s_1, ..., s_n\}$, $\sum s_i = 2b$. Построим пример $I'$ задачи $P2||C_{max}$ следующим образом:
  1. Каждому числу $s_i \in S$, поставим в соответствие работу $i$ с $t_i = s_i$ и $Y = b$;
  2. допустимое расписание существует в том и только в том случае, когда существует $S \subset A$, $\sum_{a_i \in S} a_i = b$.
  3. так как сумма длительностей работ равна $2b$, то загрузка обеих машин в допустимом расписании должна быть равна в точности $b$.

# Сложность задачи $Pm||C_{max}$

**Theorem**

*Задача $P2||C_{max}$ является NP-трудной.*

**Доказательство.**

- Покажем, что Задача о разбиении $\leq_P P2||C_{max} \leq Y$.
- Рассмотрим пример $I$ задачи о разбиении. Дано множество целых чисел $S = \{s_1, \ldots, s_n\}$, $\sum s_i = 2b$. Построим пример $I'$ задачи $P2||C_{max}$ следующим образом:
  1. Каждому числу $s_i \in S$, поставим в соответствие работу $i$ с $t_i = s_i$ и $Y = b$;
  2. допустимое расписание существует в том и только в том случае, когда существует $S \subset A$, $\sum_{a_i \in S} a_i = b$.
  3. так как сумма длительностей работ равна $2b$, то загрузка обеих машин в допустимом расписании должна быть равна в точности $b$.
- **Эквивалентность:** Разбиение соответствует расписанию.

$\square$

- **Ключевое замечание:** решение является разбиением: $X = (x_1, x_2, ..., x_n)$, where $x_i \in \{1, 2, ..., m\}$.

# Жадный алгоритм для задачи $Pm||C_{max}$

- **Ключевое замечание:** решение является разбиением: $X = (x_1, x_2, ..., x_n)$, where $x_i \in \{1, 2, ..., m\}$.
- Основная идея: Давайте рассматривать процесс решения как серию элементарных решений — шагов. На каждом шаге будем назначать работу машине. Рассматривая текущую работу, мы имеем $m$ вариантов.

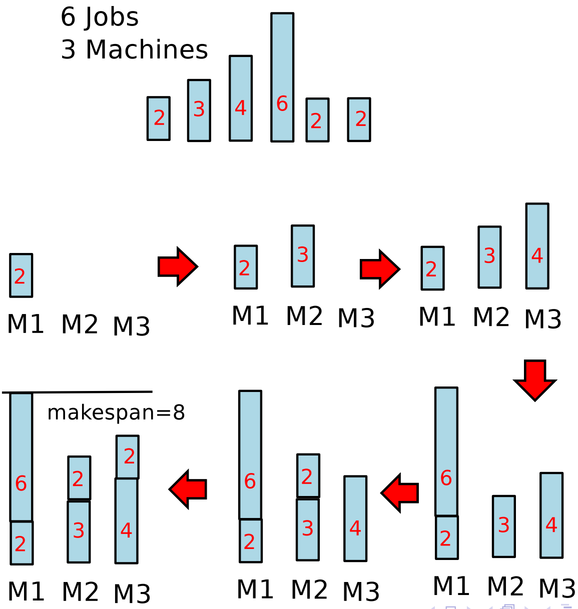
- **Ключевое замечание:** решение является разбиением: $X = (x_1, x_2, ..., x_n)$, where $x_i \in \{1, 2, ..., m\}$.
- Основная идея: Давайте рассматривать процесс решения как серию элементарных решений — шагов. На каждом шаге будем назначать работу машине. Рассматривая текущую работу, мы имеем $m$ вариантов.
- Жадная стратегия: сделать распределение наиболее сбалансированным, поэтому разумно назначить работу машине с наименьшей загрузкой.

# Жадный алгоритм

GreedyMakeSpan1

**for** $i = 1$ to $m$ **do**
    $T_i = 0$; // $T_i$ — загрузка машины $i$;
    $A_i = \texttt{NULL}$; // инициализация. Машинам не назначены работы;
**end for**
**for** $j = 1$ to $n$ **do**
    Let $k = \arg\min T_i$;
    $A_k = A_k \bigcup \{j\}$; // назначим работу $j$ машине $M_k$
    $T_k = T_k + t_j$;
**end for**

# Пример

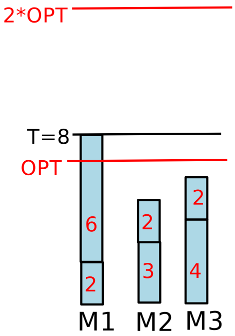


# Анализ

- Пусть $T$ — длина расписания, построенного алгоритмом $\textsc{GreedyMakeSpan1}$, и $OPT$ — длина оптимального расписания.

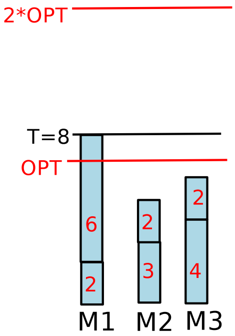
# Анализ

- Пусть $T$ — длина расписания, построенного алгоритмом GreedyMakeSpan1, и $OPT$ — длина оптимального расписания.
- Цель — оценить качество решения $T$, сравнивая $T$ с $OPT$.



# Анализ

- Пусть $T$ — длина расписания, построенного алгоритмом GreedyMakeSpan1, и $OPT$ — длина оптимального расписания.
- Цель — оценить качество решения $T$, сравнивая $T$ с $OPT$.




- Решая пример, GreedyMakeSpan1 получает $T = 8$, это не слишком плохо, поскольку $T$ не превосходит $2OPT$.

# Использование нижней оценки на $OPT$ вместо $OPT$

- Как мы можем сравнить $T$ с $OPT$, если $OPT$ неизвестно?

# Использование нижней оценки на $OPT$ вместо $OPT$

- Как мы можем сравнить $T$ с $OPT$, если $OPT$ неизвестно?
- Заметим, что найти $OPT$ это трудная задача, однако часто намного легче получить нижнию оценку $OPT$.

# Использование нижней оценки на $OPT$ вместо $OPT$

- Как мы можем сравнить $T$ с $OPT$, если $OPT$ неизвестно?
- Заметим, что найти $OPT$ это трудная задача, однако часто намного легче получить нижнию оценку $OPT$.
- Мы сравниваем $T$ с нижней оценкой на $OPT$ вместо сравнения $T$ с $OPT$ .

# Вычисление нижней границы для $OPT$

- Рассмотрим в качестве примера задачу $Pm||C_{max}$.

# Вычисление нижней границы для $OPT$

- Рассмотрим в качестве примера задачу $Pm||C_{max}$.
- Хотя $OPT$ не известно, мы можем вычислить нижнюю границу $OPT$ следующим образом:

# Вычисление нижней границы для $OPT$

- Рассмотрим в качестве примера задачу $Pm||C_{max}$.
- Хотя $OPT$ не известно, мы можем вычислить нижнюю границу $OPT$ следующим образом:
  1. Замечание 1: $OPT \geq \frac{1}{m}\sum_j t_j = \frac{19}{3}$.

# Вычисление нижней границы для $OPT$

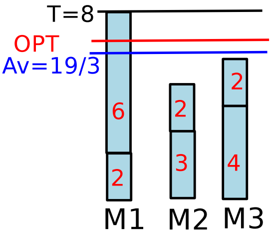
- Рассмотрим в качестве примера задачу $Pm||C_{max}$.
- Хотя $OPT$ не известно, мы можем вычислить нижнюю границу $OPT$ следующим образом:
  1. Замечание 1: $OPT \geq \frac{1}{m}\sum_j t_j = \frac{19}{3}$.
  2. Замечание 2: $OPT \geq t_j$ для любого $j$; поэтому, $OPT \geq 6$.

# Вычисление нижней границы для $OPT$

- Рассмотрим в качестве примера задачу $Pm||C_{max}$.
- Хотя $OPT$ не известно, мы можем вычислить нижнюю границу $OPT$ следующим образом:
  1. Замечание 1: $OPT \geq \frac{1}{m} \sum_j t_j = \frac{19}{3}$.
  2. Замечание 2: $OPT \geq t_j$ для любого $j$; поэтому, $OPT \geq 6$.

# Вычисление нижней границы для $OPT$

- Рассмотрим в качестве примера задачу $Pm||C_{max}$.
- Хотя $OPT$ не известно, мы можем вычислить нижнюю границу $OPT$ следующим образом:
  1. Замечание 1: $OPT \geq \frac{1}{m}\sum_j t_j = \frac{19}{3}$.
  2. Замечание 2: $OPT \geq t_j$ для любого $j$; поэтому, $OPT \geq 6$.



# Алгоритм MakeSpanAlgo1 не слижком плох

- Справедлива теорема:

**Theorem**

$T \leq 2OPT$, *т.е* GreedyMakeSpanAlgo1 *является 2-приближенным алгоритмом.*

# Алгоритм MakeSpanAlgo1 не слижком плох

- Справедлива теорема:

**Theorem**

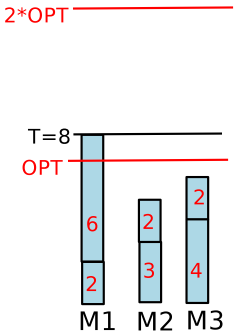
*$T \leq 2OPT$, т.е* GreedyMakeSpanAlgo1 *является 2-приближенным алгоритмом.*

# Алгоритм MAKESPANALGO1 не слижком плох

- Справедлива теорема:

**Theorem**

$T \leq 2OPT$, *т.е* GREEDYMAKESPANALGO1 *является* 2*-приближенным алгоритмом.*



## Доказательство.

- Пусть $M_i$ — машина с наибольшей загрузкой $T$;

## Доказательство.

- Пусть $M_i$ — машина с наибольшей загрузкой $T$;
- Разделим $T$ на две части: последняя работа $k$, и предыдущие работы. Т.о. $T = t_k + A$, где $A$ обозначает суммарное время выполнения предыдущих работ.

### Доказательство.

- Пусть $M_i$ — машина с наибольшей загрузкой $T$;
- Разделим $T$ на две части: последняя работа $k$, и предыдущие работы. Т.о. $T = t_k + A$, где $A$ обозначает суммарное время выполнения предыдущих работ.
- Выполняется $T \leq 2OPT$ так как

## Доказательство.

- Пусть $M_i$ — машина с наибольшей загрузкой $T$;
- Разделим $T$ на две части: последняя работа $k$, и предыдущие работы. Т.о. $T = t_k + A$, где $A$ обозначает суммарное время выполнения предыдущих работ.
- Выполняется $T \leq 2OPT$ так как
    1. $t_k \leq OPT$ (по замечанию 2)

## Доказательство.

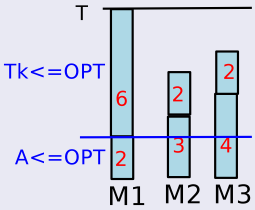
- Пусть $M_i$ — машина с наибольшей загрузкой $T$;
- Разделим $T$ на две части: последняя работа $k$, и предыдущие работы. Т.о. $T = t_k + A$, где $A$ обозначает суммарное время выполнения предыдущих работ.
- Выполняется $T \leq 2OPT$ так как
  1. $t_k \leq OPT$ (по замечанию 2)
  2. $A \leq OPT$. Почему?

## Доказательство.

- Пусть $M_i$ — машина с наибольшей загрузкой $T$;
- Разделим $T$ на две части: последняя работа $k$, и предыдущие работы. Т.о. $T = t_k + A$, где $A$ обозначает суммарное время выполнения предыдущих работ.
- Выполняется $T \leq 2OPT$ так как
  1. $t_k \leq OPT$ (по замечанию 2)
  2. $A \leq OPT$. Почему?
     - Рассмотрим состояние на момент, когда работа $k$ была назначена на машину $M_i$. В этот момент, $M_i$ имела суммарную загрузку $A$, которая является наименьшей из загрузок среди всех машин (поскольку алгоритм жадный).

## Доказательство.

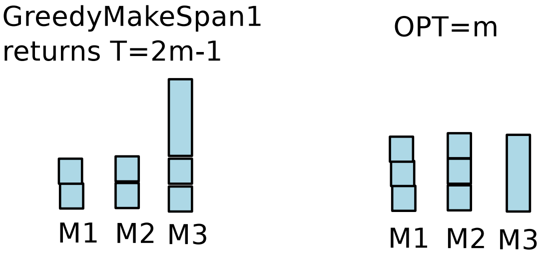
- Пусть $M_i$ — машина с наибольшей загрузкой $T$;
- Разделим $T$ на две части: последняя работа $k$, и предыдущие работы. Т.о. $T = t_k + A$, где $A$ обозначает суммарное время выполнения предыдущих работ.
- Выполняется $T \leq 2OPT$ так как
  1. $t_k \leq OPT$ (по замечанию 2)
  2. $A \leq OPT$. Почему?
     - Рассмотрим состояние на момент, когда работа $k$ была назначена на машину $M_i$. В этот момент, $M_i$ имела суммарную загрузку $A$, которая является наименьшей из загрузок среди всех машин (поскольку алгоритм жадный).
     - Формально, $A \leq \frac{1}{m}(\sum_{j=1}^{n} t_j - t_k) \leq \frac{1}{m}\sum_{j=1}^{n} t_j \leq OPT$ (по замечанию 1).



Вопрос: это корректный анализ?

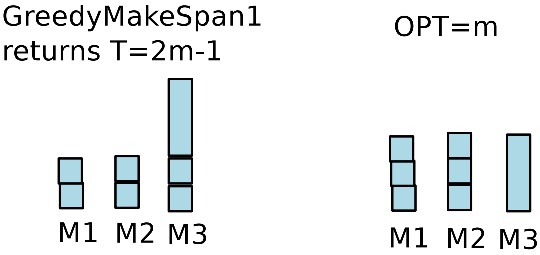
# Провера анализа алгоритма GreedyMakeSpanAlgo1

- Рассмотрим специальный пример: дано $n = m(m-1)+1$ работ с временами исполнения $t_1 = t_2 = \ldots = t_{n-1} = 1$, и $t_n = m$.



# Провера анализа алгоритма GreedyMakeSpanAlgo1

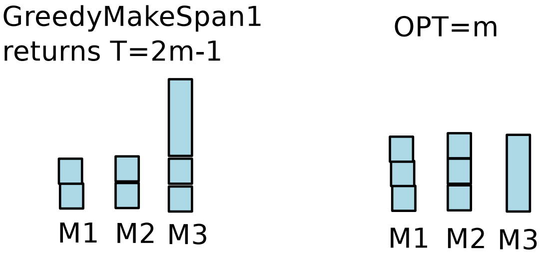
- Рассмотрим специальный пример: дано $n = m(m-1)+1$ работ с временами исполнения $t_1 = t_2 = \ldots = t_{n-1} = 1$, и $t_n = m$.




- GreedyMakeSpan1: $T = 2m-1$ (если работа $n$ назначается на последнем шаге).

# Провера анализа алгоритма GreedyMakeSpanAlgo1

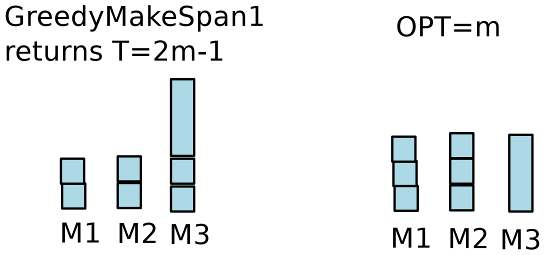
- Рассмотрим специальный пример: дано $n = m(m-1)+1$ работ с временами исполнения $t_1 = t_2 = \ldots = t_{n-1} = 1$, и $t_n = m$.




- GreedyMakeSpan1: $T = 2m - 1$ (если работа $n$ назначается на последнем шаге).
- OPT: $OPT = m$.

# Провера анализа алгоритма GreedyMakeSpanAlgo1

- Рассмотрим специальный пример: дано $n = m(m-1)+1$ работ с временами исполнения $t_1 = t_2 = \ldots = t_{n-1} = 1$, и $t_n = m$.




- GreedyMakeSpan1: $T = 2m-1$ (если работа $n$ назначается на последнем шаге).
- OPT: $OPT = m$.
- Т.о., фактор приближения: $\alpha = \frac{T}{OPT} = 2 - \frac{1}{m}$. $\alpha$ может быть произвольно близким к 2 когда $m$ возрастает.

## Другой жадный алгоритм для задачи $Pm||C_{max}$

# Идея

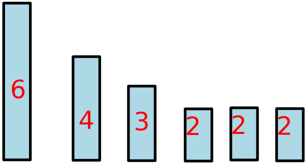
- Идея: Алгоритм GreedyMakeSpanAlgo1 дал плохой результат для приведенного примера.

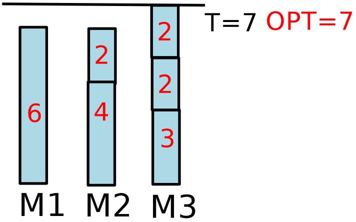
# Идея

- Идея: Алгоритм GreedyMakeSpanAlgo1 дал плохой результат для приведенного примера.
- Может не следует назначать самую продолжительную работу последней.

# Идея

- Идея: Алгоритм GreedyMakeSpanAlgo1 дал плохой результат для приведенного примера.
- Может не следует назначать самую продолжительную работу последней.
- Может применить правило — назначать продолжительные работы первыми.



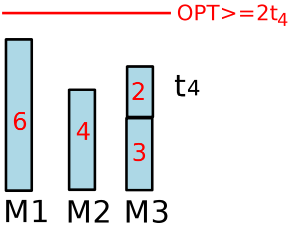
# Жадный алгоритм 2

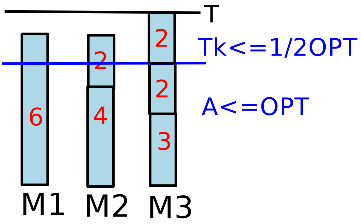
Алгоритм GreedyMakeSpan2

```
for i = 1 to m do
    T_i = 0; //инициализация;
    A_i = NULL;
end for
отсортировать работы в порядке не возрастания t_j;
//будем сначала назначать тяжелые работы
for j = 1 to m do
    Let k = argmin T_i; //назначаем работу j на машину M_k;
    A_k = A_k ⋃ {j};
    T_k = T_k + t_j;
end for
```

# Анализ

- **Замечание 3:** $OPT \geq 2t_j$ для любого $j \geq m+1$ если все работы отсортированы.
(Почему? Рассмотрим первые $m+1$ работу. По крайней мере две работы должны быть назначены на одну машину. Поэтому, $OPT \geq 2t_{m+1}$. )



# Анализ

**Theorem**

$T \leq 1.5OPT$, *т.е,* $\textsc{GreedyMakeSpanAlgo2}$ *является* $1.5$*-приближенным алгоритмом.*

# Анализ

## Theorem

$T \leq 1.5OPT$, *т.е,* GREEDYMAKESPANALGO2 *является* 1.5*-приближенным алгоритмом.*

## Доказательство.

- Пусть $M_i$ — машина с наибольшей загрузкой $T$ в решении, построенным алгоритмом GREEDYMAKESPANALGO2;

$\square$

# Анализ

**Theorem**

*$T \leq 1.5OPT$, т.е, $\textsc{GreedyMakeSpanAlgo2}$ является $1.5$-приближенным алгоритмом.*

**Доказательство.**

- Пусть $M_i$ — машина с наибольшей загрузкой $T$ в решении, построенным алгоритмом $\textsc{GreedyMakeSpanAlgo2}$;
- Разделим $T$ на две части: последняя работа $k$, и все предыдущие работы. Т.о. $T = t_k + A$, где $A$ — суммарное время выполнения предыдущих работ.

□

## Анализ

**Theorem**

$T \leq 1.5OPT$, *т.е,* GreedyMakeSpanAlgo2 *является* 1.5-*приближенным алгоритмом.*

**Доказательство.**

- Пусть $M_i$ — машина с наибольшей загрузкой $T$ в решении, построенным алгоритмом GreedyMakeSpanAlgo2;
- Разделим $T$ на две части: последняя работа $k$, и все предыдущие работы. Т.о. $T = t_k + A$, где $A$ — суммарное время выполнения предыдущих работ.
- $T \leq 1.5OPT$ поскольку:

□

# Анализ

**Theorem**

$T \leq 1.5OPT$, *т.е,* GREEDYMAKESPANALGO2 *является* 1.5*-приближенным алгоритмом.*

**Доказательство.**

- Пусть $M_i$ — машина с наибольшей загрузкой $T$ в решении, построенным алгоритмом GREEDYMAKESPANALGO2;
- Разделим $T$ на две части: последняя работа $k$, и все предыдущие работы. Т.о. $T = t_k + A$, где $A$ — суммарное время выполнения предыдущих работ.
- $T \leq 1.5OPT$ поскольку:
  1. $t_k \leq \frac{1}{2}OPT$ (по замечанию 3)

□

**Theorem**

$T \leq 1.5OPT$, *т.е,* GreedyMakeSpanAlgo2 *является* $1.5$-*приближенным алгоритмом.*

**Доказательство.**

- Пусть $M_i$ — машина с наибольшей загрузкой $T$ в решении, построенным алгоритмом GreedyMakeSpanAlgo2;
- Разделим $T$ на две части: последняя работа $k$, и все предыдущие работы. Т.о. $T = t_k + A$, где $A$ — суммарное время выполнения предыдущих работ.
- $T \leq 1.5OPT$ поскольку:
  1. $t_k \leq \frac{1}{2}OPT$ (по замечанию 3)
  2. $A \leq OPT$ (теже аргумены, что и в предыдущей теореме)

□

## Ключевые элементы построения приближенного алгоритма

# Дилемма

- Мы сталкиваемся с дилеммой:

# Дилемма

- Мы сталкиваемся с дилеммой:
  1. Чтобы установить гарантию аппроксимации, мы должны сравнить стоимость решения со стоимостью $OPT$ оптимального решения.

# Дилемма

- Мы сталкиваемся с дилеммой:
  1. Чтобы установить гарантию аппроксимации, мы должны сравнить стоимость решения со стоимостью $OPT$ оптимального решения.
  2. Однако, задача вычисления оптимального решения и его стоимости $OPT$ является NP-трудной.

# Дилемма

- Мы сталкиваемся с дилеммой:
  1. Чтобы установить гарантию аппроксимации, мы должны сравнить стоимость решения со стоимостью $OPT$ оптимального решения.
  2. Однако, задача вычисления оптимального решения и его стоимости $OPT$ является NP-трудной.
- Вопрос: Как связать конструкцию алгоритма с оценкой на $OPT$?

# Дилемма

- Мы сталкиваемся с дилеммой:
  1. Чтобы установить гарантию аппроксимации, мы должны сравнить стоимость решения со стоимостью $OPT$ оптимального решения.
  2. Однако, задача вычисления оптимального решения и его стоимости $OPT$ является NP-трудной.
- Вопрос: Как связать конструкцию алгоритма с оценкой на $OPT$?
- Ответ на этот вопрос может являться ключевым шагом в дизайне приближенного алгоритма.

# Ключевые элементы построения приближенного алгоритма

- **Стратегия:** Найти нижнюю границу для $OPT$ и **сравнение решения с нижней границей** вместо **сравнения напрямую с $OPT$!**

# Ключевые элементы построения приближенного алгоритма

- **Стратегия:** Найти нижнюю границу для $OPT$ и **сравнение решения с нижней границей** вместо **сравнения напрямую с $OPT$!**
- **Шаг 1:** Хотя задача является NP-трудной, может существовать полиномиально-временной алгоритм вычисления **“нижней границы”** для $OPT$.

# Ключевые элементы построения приближенного алгоритма

- **Стратегия:** Найти нижнюю границу для $OPT$ и **сравнение решения с нижней границей** вместо **сравнения напрямую с $OPT$!**
- **Шаг 1:** Хотя задача является NP-трудной, может существовать полиномиально-временной алгоритм вычисления **“нижней границы”** для $OPT$.
- **Шаг 2:** Мы должны разработать метод получения приемлемого решения и сравнить это решение с нижней границей ОПТ.

# Вычисление нижней границы

- Как найти нижнюю границу для $OPT$?

# Вычисление нижней границы

- Как найти нижнюю границу для $OPT$?
  1. Комбинаторный подход: есть способ вычисления оценки; тогда можно разработать жадный или ДП алгоритм;

# Вычисление нижней границы

- Как найти нижнюю границу для $OPT$?
  1. Комбинаторный подход: есть способ вычисления оценки; тогда можно разработать жадный или ДП алгоритм;
  2. методы, основанные на линейном программировании: LP-релаксация, двойственность, и т.п.

Другой пример: Задача о покрытии множества (Set Cover)

- Практические задачи:

- Практические задачи:
  - Антивирусный пакет идентифицирует вирусы на основе набора «ключевых слов», а ключевое слово соответствует нескольким вирусам. Вопрос в том, как выбрать небольшое «репрезентативное» множество ключевых слов для обнаружения всех вирусов.

- Практические задачи:
  - Антивирусный пакет идентифицирует вирусы на основе набора «ключевых слов», а ключевое слово соответствует нескольким вирусам. Вопрос в том, как выбрать небольшое «репрезентативное» множество ключевых слов для обнаружения всех вирусов.
  - Создать комитет, в котором будет как можно меньше людей, чтобы охватить все необходимые навыки

**Задача о покрытии:**

**INPUT:** множество из $n$ элементов $U = \{1, 2, ..., n\}$, и $m$ подмножеств $U$, обозначаемых как $S_1, S_2, ..., S_m$. Каждое подмножество $S_i$ имеет вес $w_i$.

**OUTPUT:** найти набор подмножеств, имеющее наименьший суммарный вес, такое, что все элементы $U$ покрываются.

# Задача о покрытии: постановка

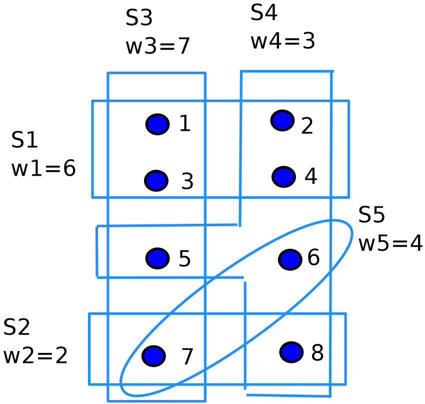
**Задача о покрытии:**

**INPUT:** множество из $n$ элементов $U = \{1, 2, ..., n\}$, и $m$ подмножеств $U$, обозначаемых как $S_1, S_2, ..., S_m$. Каждое подмножество $S_i$ имеет вес $w_i$.

**OUTPUT:** найти набор подмножеств, имеющее наименьший суммарный вес, такое, что все элементы $U$ покрываются.

Отметим, что Задача о покрытии играет важную роль в разработке приближенных алгоритмов. Кроме того она возникает в точном алгоритме для Задачи о паросочетании.

# Пример




- Вопрос: как выбрать несколько подмножеств, с наименьшим суммарным весом, так чтобы все элементы покрывались?

- **Замечание:** решение является набором подмножество. Рассмотрим процесс решения как серию решений. На каждом этапе принятия решения мы решаем выбрать подмножество или отказаться от него.

- **Замечание:** решение является набором подмножество. Рассмотрим процесс решения как серию решений. На каждом этапе принятия решения мы решаем выбрать подмножество или отказаться от него.
- **Жадная стратегия:** мы должны рассмотреть два аспекта подмножества:

# Жадный алгоритм

- **Замечание:** решение является набором подмножество. Рассмотрим процесс решения как серию решений. На каждом этапе принятия решения мы решаем выбрать подмножество или отказаться от него.
- **Жадная стратегия:** мы должны рассмотреть два аспекта подмножества:
  1. Малый вес — хорошо;

# Жадный алгоритм

- **Замечание:** решение является набором подмножество. Рассмотрим процесс решения как серию решений. На каждом этапе принятия решения мы решаем выбрать подмножество или отказаться от него.
- **Жадная стратегия:** мы должны рассмотреть два аспекта подмножества:
  1. Малый вес — хорошо;
  2. покрывает много элементов — хорошо.

# Жадный алгоритм

- **Замечание:** решение является набором подмножество. Рассмотрим процесс решения как серию решений. На каждом этапе принятия решения мы решаем выбрать подмножество или отказаться от него.
- **Жадная стратегия:** мы должны рассмотреть два аспекта подмножества:
    1. Малый вес — хорошо;
    2. покрывает много элементов — хорошо.

# Жадный алгоритм

- **Замечание:** решение является набором подмножество. Рассмотрим процесс решения как серию решений. На каждом этапе принятия решения мы решаем выбрать подмножество или отказаться от него.
- **Жадная стратегия:** мы должны рассмотреть два аспекта подмножества:
  1. Малый вес — хорошо;
  2. покрывает много элементов — хорошо.

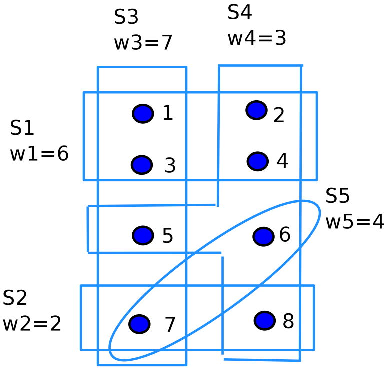
  Таким образом, разумно выбрать подмножество на основе **"отношения веса к числу элементов, которые оно покрывает"**.

# Жадный алгоритм

Greedy-Set-Cover

$I = \texttt{NULL}$; $//I$ — подмножество индексов, выбранных подмножеств;
$R = U$; $//$ $R$ — оставшиеся не покрытые элементы;
**while** $R \neq \texttt{NULL}$ **do**
    $j = \arg\min_i \frac{w_i}{|S_i \cap R|}$;
    Let $p_j = \frac{w_j}{|S_j \cap R|}$;
    **for all** элемента $e \in R \cap S_j$ **do**
        Положить $price(e) = p_j$;
    **end for**
    $I = I \cup \{j\}$;
    $R = R - S_j$;
**end while**

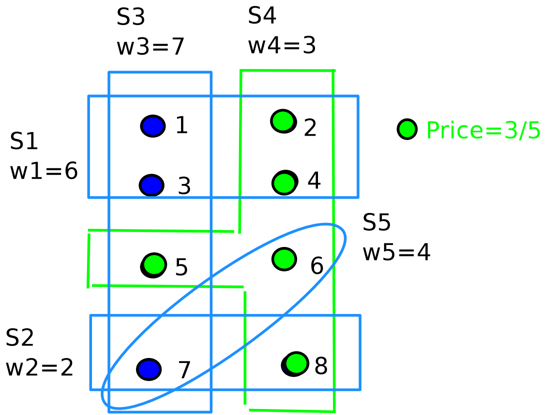
# Пример: Шаг 1




Шаг 1:

- $R = U = \{1, 2, 3, 4, 5, 6, 7, 8\}$;
- $p_1 = \frac{w_1}{|S_1 \cap R|} = \frac{6}{4}$; $p_2 = \frac{w_2}{|S_2 \cap R|} = \frac{2}{2}$; $p_3 = \frac{w_3}{|S_3 \cap R|} = \frac{7}{4}$; $p_4 = \frac{w_4}{|S_4 \cap R|} = \frac{3}{5}$; $p_5 = \frac{w_5}{|S_5 \cap R|} = \frac{4}{2}$;
- Выбираем $S_4$ поскольку $p_4$ является наименьшим.

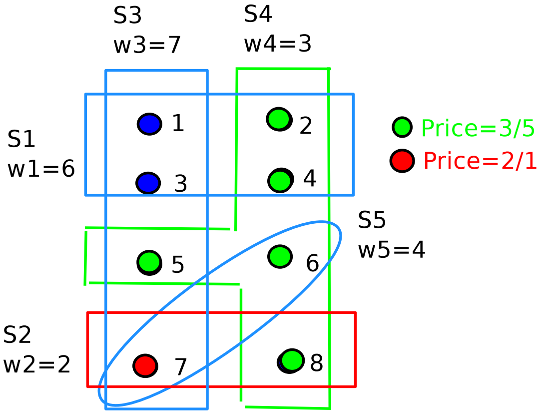
# Пример: Шаг 2




Шаг 2:

- $R = \{1, 3, 7\}$;
- $p_1 = \frac{w_1}{|S_1 \cap R|} = \frac{6}{2}$; $p_2 = \frac{w_2}{|S_2 \cap R|} = \frac{2}{1}$; $p_3 = \frac{w_3}{|S_3 \cap R|} = \frac{7}{3}$;
  $p_5 = \frac{w_5}{|S_5 \cap R|} = \frac{4}{1}$;
- Выбираем $S_2$ поскольку $p_2$ является наименьшим.

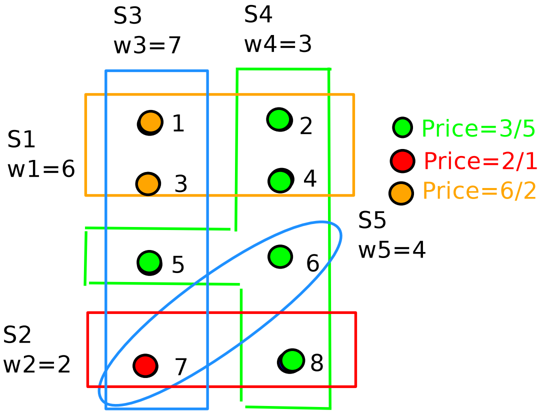
# Пример: Шаг 3




Шаг 3:

- $R = \{1, 3\}$;
- $p_1 = \frac{w_1}{|S_1 \cap R|} = \frac{6}{2}$; $p_3 = \frac{w_3}{|S_3 \cap R|} = \frac{7}{2}$;
- Выбираем $S_1$ поскольку $p_1$ является наименьшим.

# Пример: Шаг 4




Шаг 4:

- $R = \{\}$. Выполнено!
- Решение: $I = \{S_1, S_2, S_4\}$ с суммой весов: 11.
- Оптимальное решение: $\{S_3, S_4\}$ с суммой весов: 10.

- Докажем теорему:

**Theorem**

*Алгоритм* GREEDY-SET-COVER *является* $H(f)$*-приближенным, где* $f = \max_i |S_i|$.

- Докажем теорему:

**Theorem**

*Алгоритм* GREEDY-SET-COVER *является* $H(f)$*-приближенным, где* $f = \max_i |S_i|$.

- Пример: алгоритм возвращает подмножество $I = \{S_1, S_2, S_4\}$ с суммарным весом 11. Мы гарантируем, что $W = 11 \leq H(5)OPT = H(5)10$.

**Доказательство.**

Пусть $S^*$ — оптимальное решение, и $S$ — решение, выдаваемое алгоритмом Greedy-Set-Cover. Мы имеем:

$$\begin{aligned}
\sum_{S_i \in S} w_i &= \sum_{e \in U} price(e) && \text{(см. строки 7-9)} && (1)\\
&\leq \sum_{S_j \in S^*} \sum_{e \in S_j} price(e) && \text{(поскольку S* покрывает U)} && (2)\\
&\leq \sum_{S_j \in S^*} H(|S_j|) w_j && \text{(по лемме)} && (3)\\
&\leq \sum_{S_j \in S^*} H(f) w_j && (d \text{ является наибольшим}) && (4)\\
&= H(f) OPT && && (5)
\end{aligned}$$

□

# Нижняя граница

**Lemma**

*Для каждого подмножества* $S_i$, $\sum_{e \in S_i} price(e) \leq H(|S_i|)w_i$.

**Lemma**

*Для каждого подмножества* $S_i$, $\sum_{e \in S_i} price(e) \leq H(|S_i|) w_i$.

- Например, рассмотрим $S_1$. Мы имеем:

$$
\begin{aligned}
\Sigma_{e \in S_1} price(e) &= \left(\frac{w_4}{5} + \frac{w_4}{5}\right) + \left(\frac{w_1}{2} + \frac{w_1}{2}\right) && (6) \\
&\leq \left(\frac{w_1}{4} + \frac{w_1}{4}\right) + \left(\frac{w_1}{2} + \frac{w_1}{2}\right) && (7) \\
&\leq \left(\frac{w_1}{4} + \frac{w_1}{3}\right) + \left(\frac{w_1}{2} + \frac{w_1}{1}\right) && (8) \\
&= w_1 H(4)
\end{aligned}
$$

# Нижняя граница

**Lemma**

*Для каждого подмножества* $S_i$, $\sum_{e\in S_i} price(e) \leq H(|S_i|)w_i$.

- Например, рассмотрим $S_1$. Мы имеем:

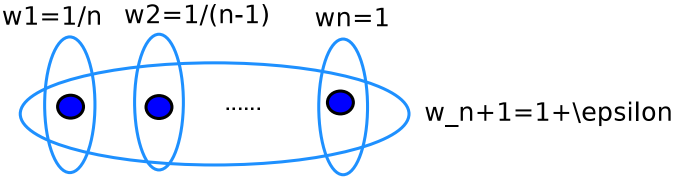
$$\begin{aligned}
\Sigma_{e\in S_1} price(e) &= \left(\frac{w_4}{5}+\frac{w_4}{5}\right)+\left(\frac{w_1}{2}+\frac{w_1}{2}\right) && (6)\\
&\leq \left(\frac{w_1}{4}+\frac{w_1}{4}\right)+\left(\frac{w_1}{2}+\frac{w_1}{2}\right) && (7)\\
&\leq \left(\frac{w_1}{4}+\frac{w_1}{3}\right)+\left(\frac{w_1}{2}+\frac{w_1}{1}\right) && (8)\\
&= w_1 H(4)
\end{aligned}$$

- Здесь, по критерию выбора на Шаге 2: $p_4 = \frac{w_4}{5}$ меньше чем $p_1 = \frac{w_1}{4}$, и на шаге 3: $p_1 = \frac{w_1}{2}$ меньше, чем $p_3 = \frac{w_3}{2}$.

Вопрос: на сколько точен анализ?

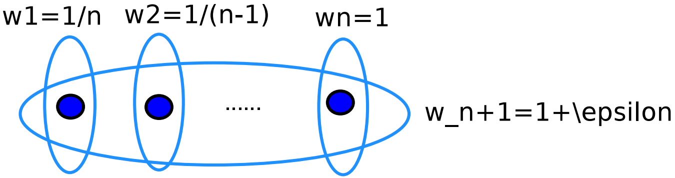
# Точный пример

- Дано: $n+1$ подмножеств с весами $\frac{1}{n}, \frac{1}{n-1}, \ldots, \frac{1}{2}, 1$ и $1+\epsilon$.



# Точный пример

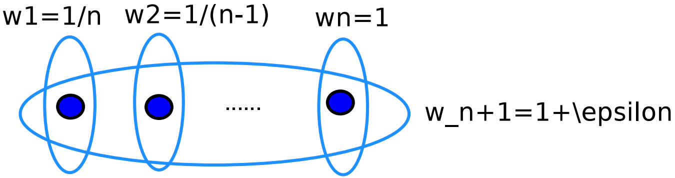
- Дано: $n+1$ подмножеств с весами $\frac{1}{n}, \frac{1}{n-1}, \ldots, \frac{1}{2}, 1$ и $1+\epsilon$.




- Оптимальное решение: одно подмножество с весом $1+\epsilon$;

# Точный пример

- Дано: $n+1$ подмножеств с весами $\frac{1}{n}, \frac{1}{n-1}, \ldots, \frac{1}{2}, 1$ и $1+\epsilon$.




- Оптимальное решение: одно подмножество с весом $1+\epsilon$;
- Алгоритм Greedy-Set-Cover дает решение, состоящее из $n$ подмножеств с суммарным весом $\frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{2} + 1 = H(n)$.