Исследование операций

Лекция. Динамическое программирование

В.В. Лепин

Институт математики
НАН Беларуси, Минск

Темы

- Первый пример: MatrixChainMultiplication
- Элементы техники динамического программирования
- Различные способы описания подзадач
- Продвинутые варианты ДП
- Связь с техникой жадных алгоритмов: Интервальное планирование, Кратчайший путь

Динамическое программирование и его связь с принципом «разделяй и властвуй»

- Динамическое программирование обычно применяется к **оптимизационным задачам**, если:

Динамическое программирование и его связь с принципом «разделяй и властвуй»

- Динамическое программирование обычно применяется к **оптимизационным задачам**, если:
  1. Исходная проблема может быть разделена на более мелкие подзадачи, и

Динамическое программирование и его связь с принципом «разделяй и властвуй»

- Динамическое программирование обычно применяется к **оптимизационным задачам**, если:
  1. Исходная проблема может быть разделена на более мелкие подзадачи, и
  2. Рекурсия среди подзадач имеет **свойство оптимальной подструктуры**, т.е. оптимальное решение исходной задачи можно вычислить с помощью **комбинации оптимальных решений подзадач**.

Динамическое программирование и его связь с принципом «разделяй и властвуй»

- Динамическое программирование обычно применяется к **оптимизационным задачам**, если:
  1. Исходная проблема может быть разделена на более мелкие подзадачи, и
  2. Рекурсия среди подзадач имеет **свойство оптимальной подструктуры**, т.е. оптимальное решение исходной задачи можно вычислить с помощью **комбинации оптимальных решений подзадач**.
- В отличие от общей структуры «разделяй и властвуй», алгоритм динамического программирования обычно **перечисляет** все возможные стратегии деления на подзадачи.

Динамическое программирование и его связь с принципом «разделяй и властвуй»

- Динамическое программирование обычно применяется к **оптимизационным задачам**, если:
  1. Исходная проблема может быть разделена на более мелкие подзадачи, и
  2. Рекурсия среди подзадач имеет **свойство оптимальной подструктуры**, т.е. оптимальное решение исходной задачи можно вычислить с помощью **комбинации оптимальных решений подзадач**.
- В отличие от общей структуры «разделяй и властвуй», алгоритм динамического программирования обычно **перечисляет** все возможные стратегии деления на подзадачи.
- Чтобы определить значимые рекурсии, одним из ключевых шагов является определение **подходящей общей формы подзадачи**. Для этой цели полезно описать процесс решения как **многоступенчатый процесс принятия решения**.

Вернемся к технике Разделяй и властвуй

- Чтобы понять, применяется ли метод Разделяй и властвуй к данной задаче, нам нужно изучить как **вход**, так и **выход**, рассматриваемой задачи.

Вернемся к технике Разделяй и властвуй

- Чтобы понять, применяется ли метод Разделяй и властвуй к данной задаче, нам нужно изучить как **вход**, так и **выход**, рассматриваемой задачи.
  - Изучите **вход**, чтобы определить, как разбить задачу на подзадачи той же структуры, но меньшего размера: относительно легко разбить задачу на подзадачи, если входная часть связана со следующими структурами данных :

Вернемся к технике Разделяй и властвуй

- Чтобы понять, применяется ли метод Разделяй и властвуй к данной задаче, нам нужно изучить как **вход**, так и **выход**, рассматриваемой задачи.
  - Изучите **вход**, чтобы определить, как разбить задачу на подзадачи той же структуры, но меньшего размера: относительно легко разбить задачу на подзадачи, если входная часть связана со следующими структурами данных :
    - **массив**, имеющий n элементов;
    - **матрица**;
    - **множество**, имеющее n элементов;
    - **дерево**;
    - **ориентированный ациклический граф**;
    - **граф**.

Вернемся к технике Разделяй и властвуй

- Чтобы понять, применяется ли метод Разделяй и властвуй к данной задаче, нам нужно изучить как **вход**, так и **выход**, рассматриваемой задачи.
  - Изучите **вход**, чтобы определить, как разбить задачу на подзадачи той же структуры, но меньшего размера: относительно легко разбить задачу на подзадачи, если входная часть связана со следующими структурами данных :
    - **массив**, имеющий n элементов;
    - **матрица**;
    - **множество**, имеющее n элементов;
    - **дерево**;
    - **ориентированный ациклический граф**;
    - **граф**.
  - Изучите **выход** чтобы определить, как построить решение исходной задачи, используя решения подзадач.

Задача MatrixChainMultiplication: рекурсия над **последовательностью**

ВХОД:
Последовательность из n матриц $A_1, A_2, ..., A_n$; матрица A_i имеет размерность $p_{i-1} \times p_i$;

ВХОД:
Последовательность из n матриц $A_1, A_2, ..., A_n$; матрица A_i имеет размерность $p_{i-1} \times p_i$;

ВЫХОД:
Расстановка скобок в произведении $A_1 A_2 ... A_n$ таким образом, чтобы минимизировать количество скалярных умножений.

Пример

$$A_1 = \begin{bmatrix} 1 & 2 \end{bmatrix} \; A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \; A_3 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

Два решения:	$((A_1)(A_2))(A_3)$	$(A_1)((A_2)(A_3))$
#Умножений:	$1 \times 2 \times 3$	$2 \times 3 \times 4$
	$+1 \times 3 \times 4$	$+1 \times 2 \times 4$
	$= 18$	$= 32$

- Здесь мы предполагаем, что для вычисления $A_1 A_2$ требуется выполнить $1 \times 2 \times 3$ скалярных умножений.

Пример

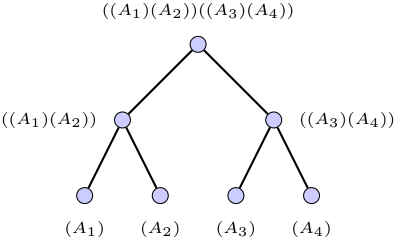
$$A_1 = \begin{bmatrix} 1 & 2 \end{bmatrix} \; A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \; A_3 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

Два решения:	$((A_1)(A_2))(A_3)$	$(A_1)((A_2)(A_3))$
#Умножений:	$1 \times 2 \times 3$	$2 \times 3 \times 4$
	$+1 \times 3 \times 4$	$+1 \times 2 \times 4$
	$= 18$	$= 32$

- Здесь мы предполагаем, что для вычисления $A_1 A_2$ требуется выполнить $1 \times 2 \times 3$ скалярных умножений.
- Задача состоит в том, чтобы определить последовательность вычислений таким образом, чтобы количество умножений было минимальным.

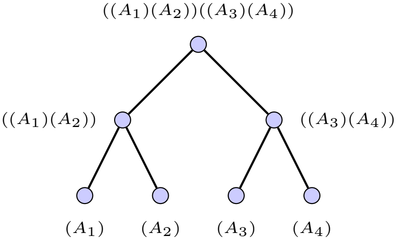
Размер пространства решения

- Интуитивно понятно, что последовательность вычислений может быть описана как двоичное дерево, где каждый узел соответствует подзадаче.



Размер пространства решения

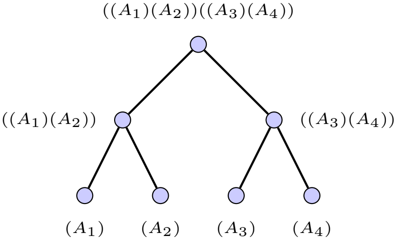
- Интуитивно понятно, что последовательность вычислений может быть описана как двоичное дерево, где каждый узел соответствует подзадаче.



- Общее количество возможных последовательностей вычисления: $\binom{2n-2}{n-1} - \binom{2n-2}{n-2}$ (число Каталана)

Размер пространства решения

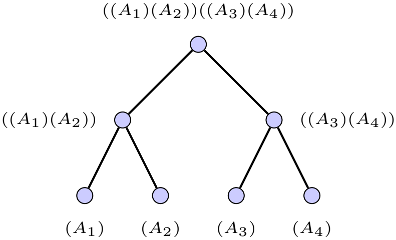
- Интуитивно понятно, что последовательность вычислений может быть описана как двоичное дерево, где каждый узел соответствует подзадаче.



- Общее количество возможных последовательностей вычисления: $\binom{2n-2}{n-1} - \binom{2n-2}{n-2}$ (число Каталана)
- Таким образом, для перечисления всех возможных последовательностей вычислений требуется экспоненциальное время.

Размер пространства решения

- Интуитивно понятно, что последовательность вычислений может быть описана как двоичное дерево, где каждый узел соответствует подзадаче.



- Общее количество возможных последовательностей вычисления: $\binom{2n-2}{n-1} - \binom{2n-2}{n-2}$ (число Каталана)
- Таким образом, для перечисления всех возможных последовательностей вычислений требуется экспоненциальное время.
- Вопрос: можем ли мы разработать эффективный алгоритм?

Алгоритм динамического программирования (S. S. Godbole, 1973)

Определение общей формы подзадачи

1. Непросто решить проблему напрямую, когда n велико. Давайте рассмотрим, возможно ли свести задачу к ряду меньших подзадач.

Определение общей формы подзадачи

1. Непросто решить проблему напрямую, когда n велико. Давайте рассмотрим, возможно ли свести задачу к ряду меньших подзадач.
2. Решение: расстановка скобок. Давайте опишем процесс решения как процесс **многошаговых решений**, где каждое решение заключается в добавлении скобок в текущую формулу.

Определение общей формы подзадачи

1. Непросто решить проблему напрямую, когда n велико. Давайте рассмотрим, возможно ли свести задачу к ряду меньших подзадач.
2. Решение: расстановка скобок. Давайте опишем процесс решения как процесс **многошаговых решений**, где каждое решение заключается в добавлении скобок в текущую формулу.
3. Предположим, что в оптимальном решении O, внешние две круглые скобки добавляются на **первом шаге** так $(A_1 \ldots A_k)(A_{k+1} \ldots A_n)$.

Определение общей формы подзадачи

1. Непросто решить проблему напрямую, когда n велико. Давайте рассмотрим, возможно ли свести задачу к ряду меньших подзадач.
2. Решение: расстановка скобок. Давайте опишем процесс решения как процесс **многошаговых решений**, где каждое решение заключается в добавлении скобок в текущую формулу.
3. Предположим, что в оптимальном решении O, внешние две круглые скобки добавляются на **первом шаге** так $(A_1 \ldots A_k)(A_{k+1} \ldots A_n)$.
4. Это решение разбивает исходную задачу на две **независимых подзадачи**: вычислить расстановку скобок в $A_1 \ldots A_k$ и в $A_{k+1} \ldots A_n$.

Определение общей формы подзадачи

1. Непросто решить проблему напрямую, когда n велико. Давайте рассмотрим, возможно ли свести задачу к ряду меньших подзадач.
2. Решение: расстановка скобок. Давайте опишем процесс решения как процесс **многошаговых решений**, где каждое решение заключается в добавлении скобок в текущую формулу.
3. Предположим, что в оптимальном решении O, внешние две круглые скобки добавляются на **первом шаге** так $(A_1 \dots A_k)(A_{k+1} \dots A_n)$.
4. Это решение разбивает исходную задачу на две **независимых подзадачи**: вычислить расстановку скобок в $A_1 \dots A_k$ и в $A_{k+1} \dots A_n$.
5. Суммируя эти два случая, мы определяем общий вид подзадачи: вычислить расстановку скобок в $A_i \dots A_j$, дающую минимальное число скалярных умножений.

Структура оптимальной расстановки скобок

- Общий вид подзадачи: вычислить расстановку скобок в $A_i ... A_j$, дающую минимальное число скалярных умножений.

Структура оптимальной расстановки скобок

- Общий вид подзадачи: вычислить расстановку скобок в $A_i ... A_j$, дающую минимальное число скалярных умножений.
  Обозначим оптимальное значение решения подзадачи как $OPT(i, j)$, следовательно исходную задачу можно решить с помощью вычисления $OPT(1, n)$.

Структура оптимальной расстановки скобок

- Общий вид подзадачи: вычислить расстановку скобок в $A_i ... A_j$, дающую минимальное число скалярных умножений.
  Обозначим оптимальное значение решения подзадачи как $OPT(i, j)$, следовательно исходную задачу можно решить с помощью вычисления $OPT(1, n)$.
- Оптимальное решение исходной задачи может быть получено путем объединения оптимальных решений подзадач.
  Эта рекурсия может быть задана так
  $OPT(1, n) = OPT(1, k) + OPT(k + 1, n) + p_0 p_k p_n$

  $(A_1 ... A_k)(A_{k+1} ... A_n)$

Корректность рекуррентной формулы

- Расстановка скобок в “префексной” подпоследовательности $A_1 ... A_k$ должна быть оптимальной. Почему? Если бы существовал более экономный способ расстановки скобок, то его применение позволило бы перемножить матрицы $A_1 ... A_k$ еще эффективнее, что противоречит предположению об оптимальности первоначальной расстановки скобок. Аналогично можно прийти к выводу, что расстановка скобок в последовательности $A_{k+1} ... A_n$, также должна быть оптимальной.

Корректность рекуррентной формулы

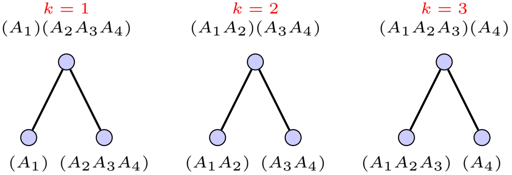
- Расстановка скобок в "префексной" подпоследовательности $A_1...A_k$ должна быть оптимальной. Почему? Если бы существовал более экономный способ расстановки скобок, то его применение позволило бы перемножить матрицы $A_1...A_k$ еще эффективнее, что противоречит предположению об оптимальности первоначальной расстановки скобок. Аналогично можно прийти к выводу, что расстановка скобок в последовательности $A_{k+1}...A_n$, также должна быть оптимальной.
- Здесь независимость между $A_1...A_k$ и $A_{k+1}...A_n$ гарантирует, что замена $OPT(1, k)$-решения на некоторое другое решение не повлияет на решение для последовательности $A_{k+1}...A_n$.

Рекурсивное решение

- Все нормально! Единственная трудность состоит в том, что мы не имеем представления о первой позиции расщепления k в оптимальном решении.

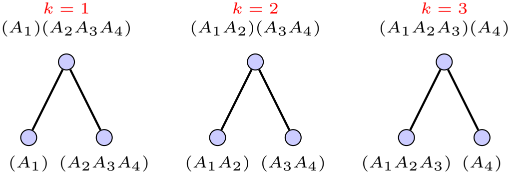
Рекурсивное решение

- Все нормально! Единственная трудность состоит в том, что мы не имеем представления о первой позиции расщепления k в оптимальном решении.
- Как преодолеть эту трудность? **Перечисление!** Мы **перечисляем все возможные варианты первого решения**, то есть для всех k, $i \leq k < j$.



Рекурсивное решение

- Все нормально! Единственная трудность состоит в том, что мы не имеем представления о первой позиции расщепления k в оптимальном решении.
- Как преодолеть эту трудность? **Перечисление!** Мы **перечисляем все возможные варианты первого решения**, то есть для всех k, $i \leq k < j$.



- Таким образом, мы имеем следующую рекурсию:

$$OPT(i,j) = \begin{cases} 0 & i = j \\ \mathbf{min}_{\mathbf{i \leq k < j}}\{OPT(i,k) + OPT(k+1,j) + p_{i-1}p_kp_j\} \end{cases}$$

Реализация рекурсии: версия 1

Версия 1: развернуть рекурсию сверху вниз

```
RECURSIVE_MATRIX_CHAIN(i, j)
if i == j then
    return 0;
end if
OPT(i, j) = +∞;
for k = i to j − 1 do
    q = RECURSIVE_MATRIX_CHAIN(i, k)
        + RECURSIVE_MATRIX_CHAIN(k + 1, j)
        +p_{i−1} p_k p_j;
    if q < OPT(i, j) then
        OPT(i, j) = q;
    end if
end for
return OPT(i, j);
```

Версия 1: развернуть рекурсию сверху вниз

$\mathrm{RECURSIVE\_MATRIX\_CHAIN}(i, j)$

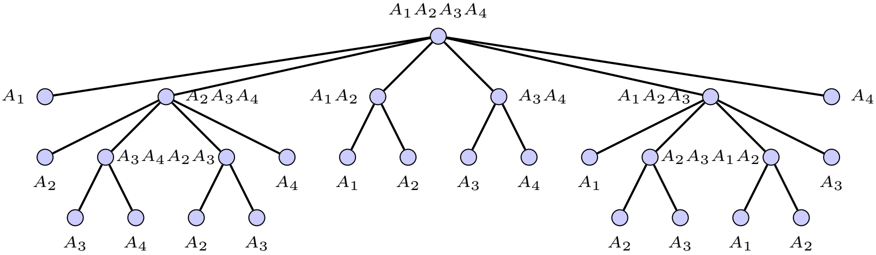
if $i == j$ **then**
    return 0;
end if
$OPT(i, j) = +\infty$;
for $k = i$ to $j - 1$ **do**
    $q = \mathrm{RECURSIVE\_MATRIX\_CHAIN}(i, k)$
        $+ \mathrm{RECURSIVE\_MATRIX\_CHAIN}(k + 1, j)$
        $+ p_{i-1} p_k p_j$;
    if $q < OPT(i, j)$ **then**
        $OPT(i, j) = q$;
    end if
end for
return $OPT(i, j)$;

- Замечание: оптимальное решение исходной задачи можно получить, вызвав $\mathrm{RECURSIVE\_MATRIX\_CHAIN}(1, n)$.

Пример

$$A_1 = \begin{bmatrix} 1 & 2 \end{bmatrix} \quad A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \quad A_3 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix} \quad A_4 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

$$1 \times 2 \qquad 2 \times 3 \qquad 3 \times 4 \qquad 4 \times 5$$



- Замечание: каждый внутренний узел дерева рекурсии представляет собой подзадачу.

Тем не менее, это не очень хорошая реализация

Theorem

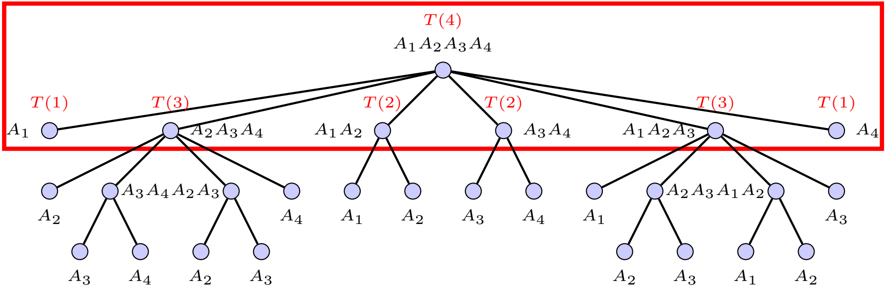
Алгоритм RECURSIVE-MATRIX-CHAIN *имеет экспоненциальную трудоемкость.*

Тем не менее, это не очень хорошая реализация

Theorem

Алгоритм RECURSIVE-MATRIX-CHAIN *имеет экспоненциальную трудоемкость.*

- Пусть $T(n)$ обозначает время, используемое для вычисления произведения n матриц. Тогда $T(n) \geq 1 + \sum_{k=1}^{n-1}(T(k) + T(n-k) + 1)$ для $n > 1$.



Доказательство.

- Покажем, что $T(n) \geq 2^{n-1}$, используя технику замещения.

Доказательство.

- Покажем, что $T(n) \geq 2^{n-1}$, используя технику замещения.
  - Базис: $T(1) \geq 1 = 2^{1-1}$.

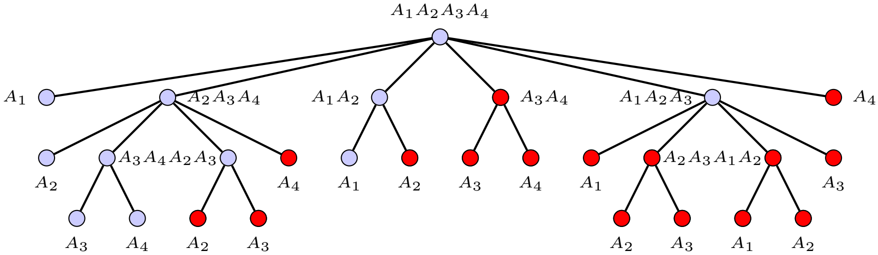
Доказательство.

- Покажем, что $T(n) \geq 2^{n-1}$, используя технику замещения.
  - Базис: $T(1) \geq 1 = 2^{1-1}$.
  - Индукция:

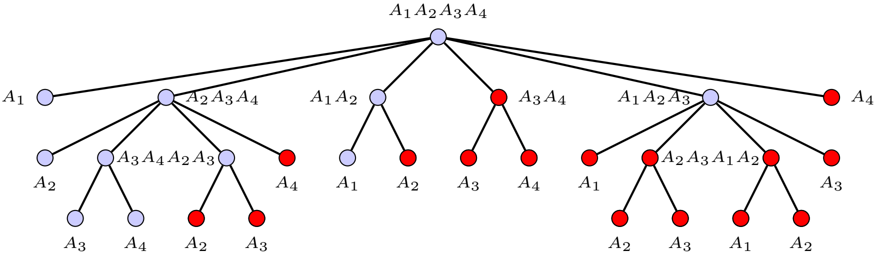
$$
\begin{aligned}
T(n) &\geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) && (1)\\
&= n + 2 \sum_{k=1}^{n-1} T(k) && (2)\\
&\geq n + 2 \sum_{k=1}^{n-1} 2^{k-1} && (3)\\
&\geq n + 2(2^{n-1} - 1) && (4)\\
&\geq n + 2^n - 2 && (5)\\
&\geq 2^{n-1} && (6)
\end{aligned}
$$

$\square$

Почему первый вариант неудачный?

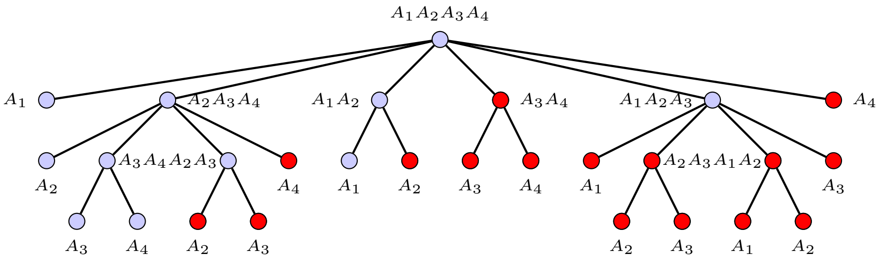


Почему первый вариант неудачный?



- Причина: есть только $O(n^2)$ подзадач. Тем не менее, некоторые подзадачи (отмечены красным) были решены неоднократно.

Почему первый вариант неудачный?



- Причина: есть только $O(n^2)$ подзадач. Тем не менее, некоторые подзадачи (отмечены красным) были решены неоднократно.
- Решение: **запомнить решения подзадач** используя массив $OPT[1..n; 1..n]$ для дальнейшего поиска. Вычисление числа Фибоначчи является хорошим примером техники «запоминания».

Реализация рекурсии: вариант 2

Техника “запоминания"

MEMORIZE_MATRIX_CHAIN(i, j)

if $OPT[i, j] \neq NULL$ **then**
    return $OPT(i, j)$;
end if
if $i == j$ **then**
    $OPT[i, j] = 0$;
else
    for $k = i$ to $j - 1$ **do**
        $q =$ MEMORIZE_MATRIX_CHAIN(i, k)
            $+$MEMORIZE_MATRIX_CHAIN$(k + 1, j)$
            $+p_{i-1}p_k p_j$;
        if $q < OPT[i, j]$ **then**
            $OPT[i, j] = q$;
        end if
    end for
end if
return $OPT[i, j]$;

Техника “запоминания"

- Исходную проблему можно решить, вызвав $\mathrm{MEMORIZE\_MATRIX\_CHAIN}\ (1, n)$ со всеми $OPT[i, j]$, инициализированными как NULL.

Техника “запоминания"

- Исходную проблему можно решить, вызвав $\mathrm{MEMORIZE\_MATRIX\_CHAIN}\ (1, n)$ со всеми $OPT[i, j]$, инициализированными как NULL.
- Трудоемкость: $O(n^3)$ (Расчет каждой записи $OPT[i, j]$ требует $O(n)$ рекурсивных вызовов в строке 8.)

Техника "запоминания"

- Исходную проблему можно решить, вызвав $\mathrm{MEMORIZE\_MATRIX\_CHAIN}\ (1, n)$ со всеми $OPT[i, j]$, инициализированными как NULL.
- Трудоемкость: $O(n^3)$ (Расчет каждой записи $OPT[i, j]$ требует $O(n)$ рекурсивных вызовов в строке 8.)
- Обратите внимание, что существуют экспоненциальное число способов расстановки скобок. Алгоритм ДП находит оптимальное решение всего за время $O(n^3)$, поскольку он избегает перечисления избыточных вариантов.

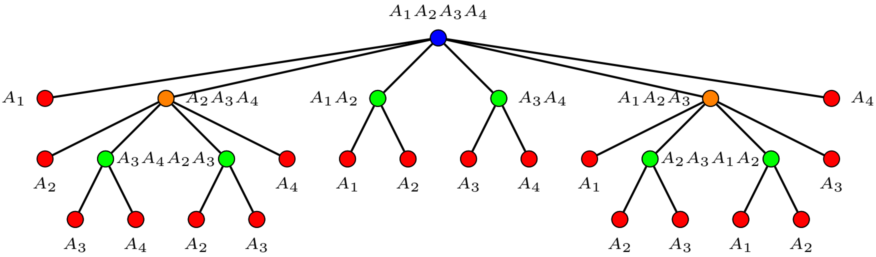
Реализация рекурсии: вариант 3

Вариант 3: Ускоренная реализация: развертывание рекурсии снизу вверх

MATRIX_CHAIN_MULTIPLICATION$(p_0, p_1, ..., p_n)$

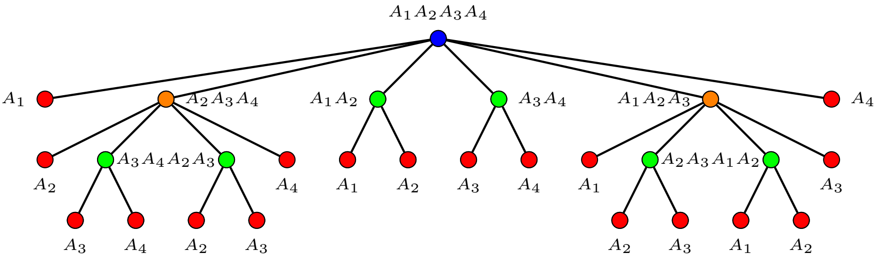
for $i = 1$ to n **do**
    $OPT(i,i) = 0;$
end for
for $l = 2$ to n **do**
    for $i = 1$ to $n - l + 1$ **do**
        $j = i + l - 1;$
        $OPT(i,j) = +\infty;$
        for $k = i$ to $j - 1$ **do**
            $q = OPT(i,k) + OPT(k+1,j) + p_{i-1}p_kp_j;$
            if $q < OPT(i,j)$ **then**
                $OPT(i,j) = q;$
                $S(i,j) = k;$
            end if
        end for
    end for
end for
return $OPT(1,n);$

Дерево рекурсии: интуитивно понятный вид вычисления снизу вверх



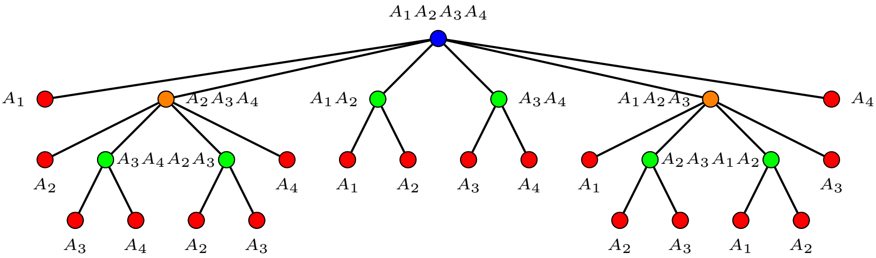
- Решение подзадач снизу вверх, т.е.

Дерево рекурсии: интуитивно понятный вид вычисления снизу вверх



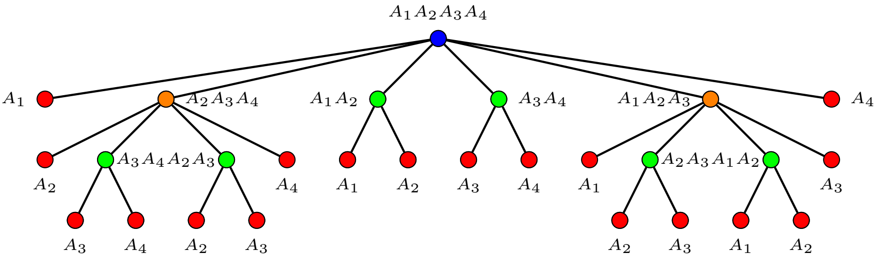
- Решение подзадач снизу вверх, т.е.
    1. Решение подзадач, отмеченных красным в первую очередь;

Дерево рекурсии: интуитивно понятный вид вычисления снизу вверх



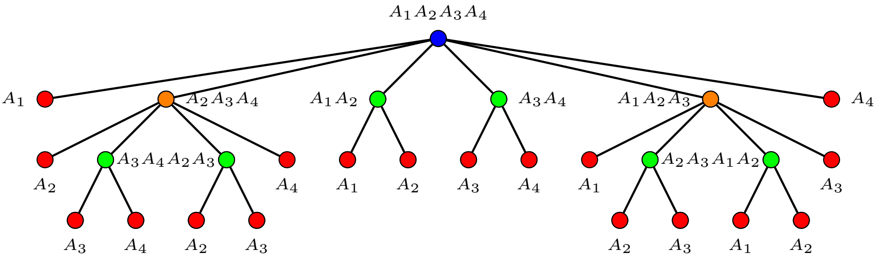
- Решение подзадач снизу вверх, т.е.
  1. Решение подзадач, отмеченных красным в первую очередь;
  2. Затем решение подзадач, отмеченных зеленым;

Дерево рекурсии: интуитивно понятный вид вычисления снизу вверх



- Решение подзадач снизу вверх, т.е.
  1. Решение подзадач, отмеченных красным в первую очередь;
  2. Затем решение подзадач, отмеченных зеленым;
  3. Затем решение подзадач, отмеченных оранжевым;

Дерево рекурсии: интуитивно понятный вид вычисления снизу вверх



- Решение подзадач снизу вверх, т.е.
    1. Решение подзадач, отмеченных красным в первую очередь;
    2. Затем решение подзадач, отмеченных зеленым;
    3. Затем решение подзадач, отмеченных оранжевым;
    4. Наконец мы можем решить исходную задачу, отмеченную синим цветом.

Шаг 1 алгоритма снизу вверх

OPT

1	2	3	4	
0	6			1
	0	24		2
		0	60	3
			0	4

SPLITTER

1	2	3	4	
	1			1
		2		2
			3	3
				4

Шаг 1:

$OPT[1,2] = p_0 \times p_1 \times p_2 = 1 \times 2 \times 3 = 6;$

$OPT[2,3] = p_1 \times p_2 \times p_3 = 2 \times 3 \times 4 = 24;$

$OPT[3,4] = p_2 \times p_3 \times p_4 = 3 \times 4 \times 5 = 60;$

Шаг 2 алгоритма снизу вверх

OPT

	1	2	3	4
1	0	6	18	
2		0	24	64
3			0	60
4				0

SPLITTER

	1	2	3	4
1		1	2	
2			2	3
3				3
4				

Шаг 2:

$$OPT[1,3] = \min \begin{cases} OPT[1,2] + OPT[3,3] + p_0 \times p_2 \times p_3 (= 18) \\ OPT[1,1] + OPT[2,3] + p_0 \times p_1 \times p_3 (= 32) \end{cases}$$

Т.о., $SPLITTER[1,3] = 2$.

$$OPT[2,4] = \min \begin{cases} OPT[2,2] + OPT[3,4] + p_1 \times p_2 \times p_4 (= 90) \\ OPT[2,3] + OPT[4,4] + p_1 \times p_3 \times p_4 (= 64) \end{cases}$$

Т.о, $SPLITTER[2,4] = 3$.

Шаг 3 алгоритма снизу вверх

OPT

1	2	3	4	
0	6	18	38	1
	0	24	64	2
		0	60	3
			0	4

SPLITTER

1	2	3	4	
	1	2	3	1
		2	3	2
			3	3
				4

Шаг 3:

$$OPT[1,4] = \min \begin{cases} OPT[1,1] + OPT[2,4] + p_0 \times p_1 \times p_4 (= 74) \\ OPT[1,2] + OPT[3,4] + p_0 \times p_2 \times p_4 (= 81) \\ OPT[1,3] + OPT[4,4] + p_0 \times p_3 \times p_4 (= 38) \end{cases}$$

Т.о., $SPLITTER[1,4] = 3$.

Вопрос: Мы вычислили оптимальную **стоимость**, но как получить оптимальную **расстановку скобок**?

Последний шаг: построение оптимального решения с помощью «обратного хода»

- Идея: **обратного хода!** Начиная с $OPT[1, n]$, мы отслеживаем источник $OPT[1, n]$, то есть какой вариант разделения мы выбираем на каждом этапе принятия решения.

Последний шаг: построение оптимального решения с помощью «обратного хода»

- Идея: **обратного хода!** Начиная с $OPT[1, n]$, мы отслеживаем источник $OPT[1, n]$, то есть какой вариант разделения мы выбираем на каждом этапе принятия решения.
- В частности, используется вспомогательный массив $S[1..n, 1..n]$.

Последний шаг: построение оптимального решения с помощью «обратного хода»

- Идея: **обратного хода!** Начиная с $OPT[1, n]$, мы отслеживаем источник $OPT[1, n]$, то есть какой вариант разделения мы выбираем на каждом этапе принятия решения.
- В частности, используется вспомогательный массив $S[1..n, 1..n]$.
  - В переменную $S[i, j]$ записывается оптимальное решение, то есть значение k, такое, что оптимальные скобки $A_i ... A_j$ вставляются между $A_k A_{k+1}$.

Последний шаг: построение оптимального решения с помощью «обратного хода»

- Идея: **обратного хода!** Начиная с $OPT[1,n]$, мы отслеживаем источник $OPT[1,n]$, то есть какой вариант разделения мы выбираем на каждом этапе принятия решения.
- В частности, используется вспомогательный массив $S[1..n, 1..n]$.
  - В переменную $S[i,j]$ записывается оптимальное решение, то есть значение k, такое, что оптимальные скобки $A_i ... A_j$ вставляются между $A_k A_{k+1}$.
  - Таким образом, оптимальным решением исходной задачи $A_{1..n}$ является $A_{1..S[1,n]} A_{S[1,n]+1..n}$.

Последний шаг: построение оптимального решения с помощью «обратного хода»

- Идея: **обратного хода!** Начиная с $OPT[1,n]$, мы отслеживаем источник $OPT[1,n]$, то есть какой вариант разделения мы выбираем на каждом этапе принятия решения.
- В частности, используется вспомогательный массив $S[1..n, 1..n]$.
  - В переменную $S[i,j]$ записывается оптимальное решение, то есть значение k, такое, что оптимальные скобки $A_i \ldots A_j$ вставляются между $A_k A_{k+1}$.
  - Таким образом, оптимальным решением исходной задачи $A_{1..n}$ является $A_{1..S[1,n]} A_{S[1,n]+1..n}$.
- Отметим, что: Оптимальный вариант не может быть определен до момента решения всех подзадач.

Обратный ход: шаг 1

OPT

	1	2	3	4
1	0	6	18	38
2		0	24	64
3			0	60
4				0

SPLITTER

	1	2	3	4
1		1	2	3
2			2	3
3				3
4				

Шаг 1: $(\ A_1 A_2 A_3\ )\ (\ A_4\ )$

Обратный ход: шаг 2

OPT

	1	2	3	4
1	0	6	18	38
2		0	24	64
3			0	60
4				0

SPLITTER

	1	2	3	4
1		1	2	3
2			2	3
3				3
4				

Шаг 1: $(\ A_1A_2A_3\ )\ (\ A_4\ )$

Шаг 2: $(\ (\ A_1A_2\ )\ (\ A_3\ )\ )\ (\ A_4\ )$

Обратный ход: шаг 3

OPT

	1	2	3	4	
	0	6	18	38	1
		0	24	64	2
			0	60	3
				0	4

SPLITTER

	1	2	3	4	
		1	2	3	1
			2	3	2
				3	3
					4

Шаг 1: $(\ A_1 A_2 A_3\ )\ (\ A_4\ )$

Шаг 2: $(\ (\ A_1 A_2\ )\ (\ A_3\ )\ )\ (\ A_4\ )$

Шаг 3: $(\ (\ (\ A_1\ )\ (\ A_2\ )\ )\ (\ A_3\ )\ )\ (\ A_4\ )$

Резюме: элементы динамического программирования

1. Обычно нелегко решить большую проблему напрямую. Давайте рассмотрим, можно ли разложить проблему на более мелкие подзадачи.

Резюме: элементы динамического программирования

1. Обычно нелегко решить большую проблему напрямую. Давайте рассмотрим, можно ли разложить проблему на более мелкие подзадачи.
   Как определить подзадачи?

Резюме: элементы динамического программирования

1. Обычно нелегко решить большую проблему напрямую. Давайте рассмотрим, можно ли разложить проблему на более мелкие подзадачи.
   Как определить подзадачи?
   - Сначала опишем процесс решения как многоступенчатый процесс **решений**.

Резюме: элементы динамического программирования

1. Обычно нелегко решить большую проблему напрямую. Давайте рассмотрим, можно ли разложить проблему на более мелкие подзадачи.
   Как определить подзадачи?
   - Сначала опишем процесс решения как многоступенчатый процесс **решений**.
   - Нужно рассмотреть несколько примеров подзадач. Рассматриваем **первый / последний шаги решения (в некотором порядке)** в оптимальном решении. **первый / последний шаг решения** может иметь несколько вариантов. Перечисляем все возможные варианты шагов решения и исследуем сгенерированные подзадачи.

Резюме: элементы динамического программирования

1. Обычно нелегко решить большую проблему напрямую. Давайте рассмотрим, можно ли разложить проблему на более мелкие подзадачи.
   Как определить подзадачи?
   - Сначала опишем процесс решения как многоступенчатый процесс **решений**.
   - Нужно рассмотреть несколько примеров подзадач. Рассматриваем **первый / последний шаги решения (в некотором порядке)** в оптимальном решении. **первый / последний шаг решения** может иметь несколько вариантов. Перечисляем все возможные варианты шагов решения и исследуем сгенерированные подзадачи.
   - Затем мы определили общую форму подзадач, проанализировав все возможные формы подзадач.

Резюме: элементы динамического программирования

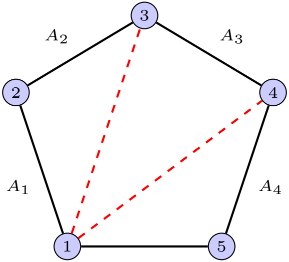
1. Обычно нелегко решить большую проблему напрямую. Давайте рассмотрим, можно ли разложить проблему на более мелкие подзадачи.
   Как определить подзадачи?
   - Сначала опишем процесс решения как многоступенчатый процесс **решений**.
   - Нужно рассмотреть несколько примеров подзадач. Рассматриваем **первый / последний шаги решения (в некотором порядке)** в оптимальном решении. **первый / последний шаг решения** может иметь несколько вариантов. Перечисляем все возможные варианты шагов решения и исследуем сгенерированные подзадачи.
   - Затем мы определили общую форму подзадач, проанализировав все возможные формы подзадач.
2. Покажите, что рекурсия среди подзадач может быть задана как **структура оптимальных решений**, то есть оптимальное решение задачи содержит в себе оптимальные решения подзадач.

Резюме: элементы динамического программирования

1. Обычно нелегко решить большую проблему напрямую. Давайте рассмотрим, можно ли разложить проблему на более мелкие подзадачи.
   Как определить подзадачи?
   - Сначала опишем процесс решения как многоступенчатый процесс **решений**.
   - Нужно рассмотреть несколько примеров подзадач. Рассматриваем **первый / последний шаги решения (в некотором порядке)** в оптимальном решении. **первый / последний шаг решения** может иметь несколько вариантов. Перечисляем все возможные варианты шагов решения и исследуем сгенерированные подзадачи.
   - Затем мы определили общую форму подзадач, проанализировав все возможные формы подзадач.
2. Покажите, что рекурсия среди подзадач может быть задана как **структура оптимальных решений**, то есть оптимальное решение задачи содержит в себе оптимальные решения подзадач.
3. Программирование: если рекурсивный алгоритм решает одну и ту же подзадачу снова и снова, можно использовать запоминание, чтобы избежать повторения решения одних и тех же подзадач.

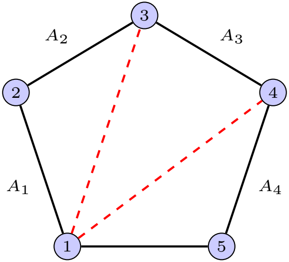
Вопрос: $O(n^3)$ — это нижняя граница?

$O(n \log n)$-алгоритм (Hu, Shing 1981)



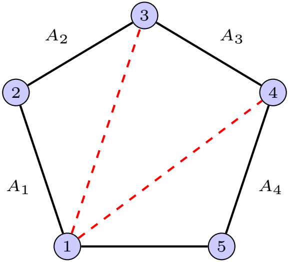
- Существует биекция между расстановкой скобок и разбиением выпуклого многоугольника на непересекающиеся треугольники.

$O(n \log n)$-алгоритм (Hu, Shing 1981)



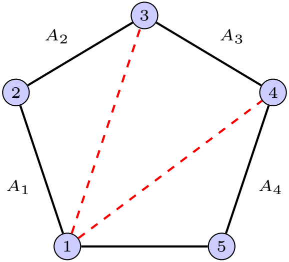
- Существует биекция между расстановкой скобок и разбиением выпуклого многоугольника на непересекающиеся треугольники.
  - Каждому узлу приписывается вес w_i, а треугольнику — произведение весов его узлов.

$O(n \log n)$-алгоритм (Hu, Shing 1981)



- Существует биекция между расстановкой скобок и разбиением выпуклого многоугольника на непересекающиеся треугольники.
  - Каждому узлу приписывается вес w_i, а треугольнику — произведение весов его узлов.
  - Разбиение (красные пунктирные линии) имеет сумму весов 38. Фактически это соответствует расстановке скобок $((((A_1\ )\ (\ A_2\ )\ (\ A_3\ ))\ (\ A_4\ )$.

$O(n \log n)$-алгоритм (Hu, Shing 1981)

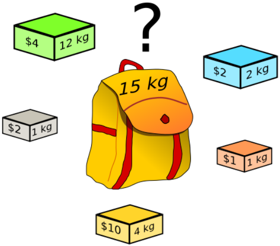


- Существует биекция между расстановкой скобок и разбиением выпуклого многоугольника на непересекающиеся треугольники.
  - Каждому узлу приписывается вес w_i, а треугольнику — произведение весов его узлов.
  - Разбиение (красные пунктирные линии) имеет сумму весов 38. Фактически это соответствует расстановке скобок $((((A_1\ )\ (\ A_2\ )\ (\ A_3\ ))\ (\ A_4\ )$.
- Оптимальное разбиение можно найти за время $O(n \log n)$.

Задача 0/1-рюкзак: рекурсия над **множествами**

Задача о рюкзаке

- Рассмотрим набор предметов, где каждый предмет имеет вес и значение. Цель состоит в том, чтобы выбрать подмножество элементов, чтобы общий вес был меньше заданного предела, а общее значение было как можно большим.



Формализованное определение:

ВХОД: Набор предметов $S = \{1, 2, ..., n\}$. Предмет i имеет вест w_i и значение v_i. Общий весовой предел W;
ВЫХОД: Подмножество предметов с максимальной общей стоимостью и общим весом ниже W.

0/1-рюкзак

Формализованное определение:

ВХОД: Набор предметов $S = \{1, 2, ..., n\}$. Предмет i имеет вест w_i и значение v_i. Общий весовой предел W;
ВЫХОД: Подмножество предметов с максимальной общей стоимостью и общим весом ниже W.

- Здесь, "0/1" означает, что мы должны выбрать предмет (1) или отказаться от него (0), и мы не можем выбрать часть предмета.

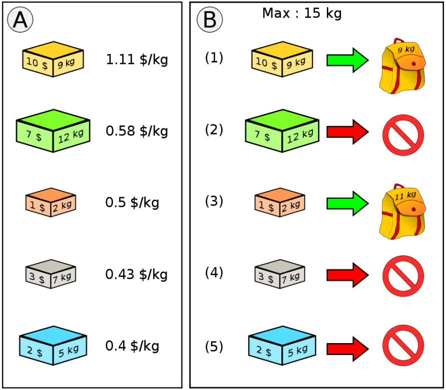
0/1-рюкзак

Формализованное определение:

ВХОД: Набор предметов $S = \{1, 2, ..., n\}$. Предмет i имеет вест w_i и значение v_i. Общий весовой предел W;
ВЫХОД: Подмножество предметов с максимальной общей стоимостью и общим весом ниже W.

- Здесь, "$0/1$" означает, что мы должны выбрать предмет (1) или отказаться от него (0), и мы не можем выбрать часть предмета.
- Напротив, задача Дробный рюкзак позволяет выбрать дробный элемент, скажем, 0.5.

0/1-рюкзак: интуитивный алгоритм



- Интуитивно понятный метод: сначала выбирайте «дорогие» предметы.
- Но это не оптимальное решение.

Определение общей формы подзадач

- Непросто решить проблему с n элементами напрямую. Давайте рассмотрим, возможно ли свести к меньшим подзадачам.

Определение общей формы подзадач

- Непросто решить проблему с n элементами напрямую. Давайте рассмотрим, возможно ли свести к меньшим подзадачам.
- Решение: подмножество предметов. Давайте опишем процесс решения как **многошаговый** процесс. На i-ом шаге принятия решения мы решаем, должен ли быть выбран элемент i.

Определение общей формы подзадач

- Непросто решить проблему с n элементами напрямую. Давайте рассмотрим, возможно ли свести к меньшим подзадачам.
- Решение: подмножество предметов. Давайте опишем процесс решения как **многошаговый** процесс. На i-ом шаге принятия решения мы решаем, должен ли быть выбран элемент i.
- Давайте рассмотрим **первое решение**, т.е. содержит ли оптимальное решение элемент n или нет (здесь мы рассматриваем элементы от последнего до первого). Это решение имеет два варианта:

Определение общей формы подзадач

- Непросто решить проблему с n элементами напрямую. Давайте рассмотрим, возможно ли свести к меньшим подзадачам.
- Решение: подмножество предметов. Давайте опишем процесс решения как **многошаговый** процесс. На i-ом шаге принятия решения мы решаем, должен ли быть выбран элемент i.
- Давайте рассмотрим **первое решение**, т.е. содержит ли оптимальное решение элемент n или нет (здесь мы рассматриваем элементы от последнего до первого). Это решение имеет два варианта:
  1. Берем: Тогда достаточно оптимально выбрать предметы из $\{1, 2, ..., n-1\}$ с ограничением веса рюкзака $W - w_n$.

Определение общей формы подзадач

- Непросто решить проблему с n элементами напрямую. Давайте рассмотрим, возможно ли свести к меньшим подзадачам.
- Решение: подмножество предметов. Давайте опишем процесс решения как **многошаговый** процесс. На i-ом шаге принятия решения мы решаем, должен ли быть выбран элемент i.
- Давайте рассмотрим **первое решение**, т.е. содержит ли оптимальное решение элемент n или нет (здесь мы рассматриваем элементы от последнего до первого). Это решение имеет два варианта:
  1. Берем: Тогда достаточно оптимально выбрать предметы из $\{1, 2, ..., n-1\}$ с ограничением веса рюкзака $W - w_n$.
  2. Не берем: В этом случае, мы должны выбрать оптимально элементы из $\{1, 2, ..., n-1\}$ с ограничением веса рюкзака W.

Определение общей формы подзадач

- Непросто решить проблему с n элементами напрямую. Давайте рассмотрим, возможно ли свести к меньшим подзадачам.
- Решение: подмножество предметов. Давайте опишем процесс решения как **многошаговый** процесс. На i-ом шаге принятия решения мы решаем, должен ли быть выбран элемент i.
- Давайте рассмотрим **первое решение**, т.е. содержит ли оптимальное решение элемент n или нет (здесь мы рассматриваем элементы от последнего до первого). Это решение имеет два варианта:
  1. Берем: Тогда достаточно оптимально выбрать предметы из $\{1, 2, ..., n-1\}$ с ограничением веса рюкзака $W - w_n$.
  2. Не берем: В этом случае, мы должны выбрать оптимально элементы из $\{1, 2, ..., n-1\}$ с ограничением веса рюкзака W.
- В обоих случаях исходная задача сводится к меньшим подзадачам.

- Суммируя эти два случая, мы можем установить общий вид подзадач: выбрать элементы из $\{1, 2, ..., i\}$, так чтобы их общий вес не превышал w, а суммарная ценность была наибольшей. Обозначим оптимальное значение решения как $OPT(\{1, 2, ..., i\}, w)$.

Принцип оптимальности

- Суммируя эти два случая, мы можем установить общий вид подзадач: выбрать элементы из $\{1, 2, ..., i\}$, так чтобы их общий вес не превышал w, а суммарная ценность была наибольшей. Обозначим оптимальное значение решения как $OPT(\{1, 2, ..., i\}, w)$.
- Тогда мы можем доказать принцип оптимальности:

$$OPT(\{1, 2, ..., n\}, W) = \max \begin{cases} OPT(\{1, 2, ..., n-1\}, W) \\ OPT(\{1, 2, ..., n-1\}, W - w_n) + v_n \end{cases}$$

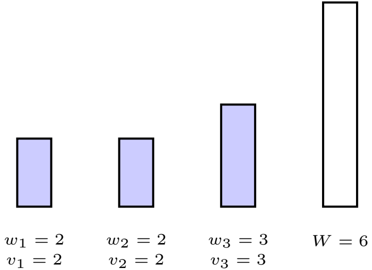
Алгоритм

$\textsc{Knapsack}(n, W)$

for $w = 1$ to W **do**
    $OPT[0, w] = 0;$
end for
for $i = 1$ to n **do**
    for $w = 1$ to W **do**
        $OPT[i, w] = \max\{OPT[i-1, w], v_i + OPT[i-1, w-w_i]\};$
    end for
end for
return $OPT[n, W];$

- Здесь $OPT[i, w]$ представляет $OPT(\{1, 2, ..., i\}, w)$ для упрощения обозначений.

Пример: Шаг 1



В начале все $OPT[0, w] = 0$

$w =$	0	1	2	3	4	5	6
$i = 3$							
$i = 2$							
$i = 1$							
$i = 0$	0	0	0	0	0	0	0

Шаг 2

$$
\begin{aligned}
OPT[1,2] = \max\{ & \\
& OPT[0,2](=0), \\
& OPT[0,0] + V_1(=0+2)\} \\
&= 2
\end{aligned}
$$

$w =$	0	1	2	3	4	5	6
$i = 3$							
$i = 2$							
$i = 1$	0	0	2	2	2	2	2
$i = 0$	0	0	0	0	0	0	0

Шаг 3

$$
\begin{aligned}
OPT[2,4] = \max\{ & \\
& OPT[1,4](=2), \\
& OPT[1,2] + V_2(=2+2)\} \\
& = 4
\end{aligned}
$$

$w =$	0	1	2	3	4	5	6
$i = 3$							
$i = 2$	0	0	2	2	4	4	4
$i = 1$	0	0	2	2	2	2	2
$i = 0$	0	0	0	0	0	0	0

Шаг 4

$$
\begin{aligned}
OPT[3,3] = \max\{ & \\
& OPT[2,3](=2), \\
& OPT[2,0] + V_3(=0+3)\} \\
& = 3
\end{aligned}
$$

$w =$	0	1	2	3	4	5	6
$i=3$	0	0	2	3	4	5	5
$i=2$	0	0	2	2	4	4	4
$i=1$	0	0	2	2	2	2	2
$i=0$	0	0	0	0	0	0	0

Обратный ход

$$
\begin{aligned}
OPT[3,6] = \max\{ & \\
& OPT[2,6](=4), \\
& OPT[2,3] + V_3(=2+3)\} \\
& \quad = 5
\end{aligned}
$$

Решение: Выбрать элемент 3

$$
\begin{aligned}
OPT[2,3] = \max\{ & \\
& OPT[1,3](=2), \\
& OPT[1,1] + V_2(=0+2)\} \\
& \quad = 2
\end{aligned}
$$

Решение: Выбрать элемент 2

$w =$	0	1	2	3	4	5	6
$i = 3$	0	0	2	3	4	5	5
$i = 2$	0	0	2	2	4	4	4
$i = 1$	0	0	2	2	2	2	2
$i = 0$	0	0	0	0	0	0	0

Анализ трудоемкости

- Трудоемкость: $O(nW)$. (Подсказка: для каждой записи в матрице требуется только одно сравнение; у нас есть $O(nW)$ записей в матрице.)

Анализ трудоемкости

- Трудоемкость: $O(nW)$. (Подсказка: для каждой записи в матрице требуется только одно сравнение; у нас есть $O(nW)$ записей в матрице.)
- Отметим, что:
  1. Этот алгоритм неэффективен, когда W велико, скажем, $W = 1$ миллиону.

Анализ трудоемкости

- Трудоемкость: $O(nW)$. (Подсказка: для каждой записи в матрице требуется только одно сравнение; у нас есть $O(nW)$ записей в матрице.)
- Отметим, что:
  1. Этот алгоритм неэффективен, когда W велико, скажем, $W = 1$ миллиону.
  2. Напомним, что у полиномиального алгоритм время выполнения в худшем случае должно быть полиномом от $mW = m2^{\log W} = m2^{\text{длина входа}}$. Экспоненциальный!

Анализ трудоемкости

- Трудоемкость: $O(nW)$. (Подсказка: для каждой записи в матрице требуется только одно сравнение; у нас есть $O(nW)$ записей в матрице.)
- Отметим, что:
  1. Этот алгоритм неэффективен, когда W велико, скажем, $W = 1$ миллиону.
  2. Напомним, что у полиномиального алгоритм время выполнения в худшем случае должно быть полиномом от $mW = m2^{\log W} = m2^{\text{длина входа}}$. Экспоненциальный!
  3. Это алгоритм с псевдополиномиальной трудоемкостью: полином от **величины** W а не от **длины двоичной записи** W, равной $(\log W)$.

Анализ трудоемкости

- Трудоемкость: $O(nW)$. (Подсказка: для каждой записи в матрице требуется только одно сравнение; у нас есть $O(nW)$ записей в матрице.)
- Отметим, что:
  1. Этот алгоритм неэффективен, когда W велико, скажем, $W = 1$ миллиону.
  2. Напомним, что у полиномиального алгоритм время выполнения в худшем случае должно быть полиномом от $mW = m2^{\log W} = m2^{\text{длина входа}}$. Экспоненциальный!
  3. Это алгоритм с псевдополиномиальной трудоемкостью: полином от **величины** W а не от **длины двоичной записи** W, равной $(\log W)$.
  4. Мы вернемся к этому алгоритму при разработке приближенного алгоритма.

Почему рассматриваются предметы от последнего до первого?

- Давайте рассмотрим два способа выбора предметов:

Почему рассматриваются предметы от последнего до первого?

- Давайте рассмотрим два способа выбора предметов:
  1. Если мы рассмотрим произвольный элемент i, то подзадача превращается в "выбрать элементы суммарно как можно более дорогие из **подмножества** s с пределом веса w". У нас есть следующая рекурсия:

$$OPT(\{1,2,...,n\},W) = \max \begin{cases} OPT(\{1,2,...,n\}-\{i\},W) \\ OPT(\{1,2,...,n\}-\{i\},W-w_i)+v_i \end{cases}$$

Почему рассматриваются предметы от последнего до первого?

- Давайте рассмотрим два способа выбора предметов:
  1. Если мы рассмотрим произвольный элемент i, то подзадача превращается в "выбрать элементы суммарно как можно более дорогие из **подмножества** s с пределом веса w". У нас есть следующая рекурсия:

$$OPT(\{1,2,...,n\},W) = \max \begin{cases} OPT(\{1,2,...,n\} - \{i\}, W) \\ OPT(\{1,2,...,n\} - \{i\}, W - w_i) + v_i \end{cases}$$

  2. Напротив, если мы рассмотрим предметы от последнего до первого, то подзадача может быть постановлена как "выбрать элементы суммарно как можно более дорогие из $\{1,2,...,i\}$ с пределом веса w" и мы имеем следующую рекурсию:

Почему рассматриваются предметы от последнего до первого?

- Давайте рассмотрим два способа выбора предметов:
  1. Если мы рассмотрим произвольный элемент i, то подзадача превращается в “выбрать элементы суммарно как можно более дорогие из **подмножества** s с пределом веса w”. У нас есть следующая рекурсия:

$$OPT(\{1,2,...,n\},W) = \max\begin{cases} OPT(\{1,2,...,n\}-\{i\},W) \\ OPT(\{1,2,...,n\}-\{i\},W-w_i)+v_i \end{cases}$$

  2. Напротив, если мы рассмотрим предметы от последнего до первого, то подзадача может быть постановлена как “выбрать элементы суммарно как можно более дорогие из $\{1,2,...,i\}$ с пределом веса w” и мы имеем следующую рекурсию:

$$OPT(\{1,2,...,n\},W) = \max\begin{cases} OPT(\{1,2,...,n-1\},W) \\ OPT(\{1,2,...,n-1\},W-w_n)+v_n \end{cases}$$

Почему рассматриваются предметы от последнего до первого?

- Давайте рассмотрим два способа выбора предметов:
  1. Если мы рассмотрим произвольный элемент i, то подзадача превращается в "выбрать элементы суммарно как можно более дорогие из **подмножества** s с пределом веса w". У нас есть следующая рекурсия:

$$OPT(\{1,2,...,n\},W) = \max \begin{cases} OPT(\{1,2,...,n\} - \{i\}, W) \\ OPT(\{1,2,...,n\} - \{i\}, W - w_i) + v_i \end{cases}$$

  2. Напротив, если мы рассмотрим предметы от последнего до первого, то подзадача может быть постановлена как "выбрать элементы суммарно как можно более дорогие из $\{1,2,...,i\}$ с пределом веса w" и мы имеем следующую рекурсию:

$$OPT(\{1,2,...,n\},W) = \max \begin{cases} OPT(\{1,2,...,n-1\}, W) \\ OPT(\{1,2,...,n-1\}, W - w_n) + v_n \end{cases}$$

- Фактически, первый вариант приводит к экспоненциальному числу подзадач. Напротив, второй вариант — $O(nW)$.

Вершинное покрытие: рекурсия над деревьями

Задача Вершинное покрытие

Формализованное определение:

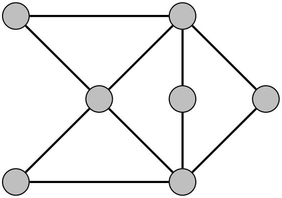
ВХОД: $G = (V, E)$ — граф
ВЫХОД: наименьшее подмножество вершин $S \subseteq V$, такое, что у каждого ребра хотя бы одна из его концевых вершин находится в S

Задача Вершинное покрытие

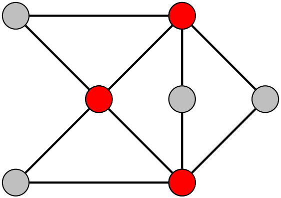
Формализованное определение:

ВХОД: $G = (V, E)$ — граф
ВЫХОД: наименьшее подмножество вершин $S \subseteq V$, такое, что у каждого ребра хотя бы одна из его концевых вершин находится в S

- Например, сколько вершин необходимо, чтобы покрыть все ребра в следующем графе?

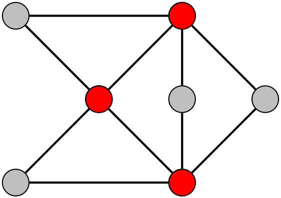


Вершинное покрытие



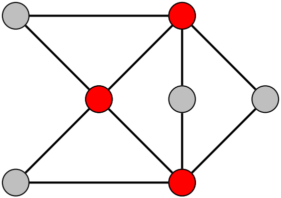
- Вершины, раскрашенные красным образуют вершинное покрытие.

Вершинное покрытие



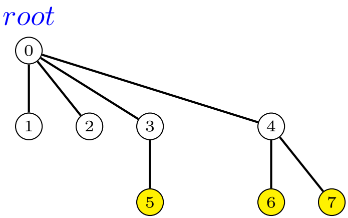
- Вершины, раскрашенные красным образуют вершинное покрытие.
- Задача Вершинное покрытие является NP-трудной.

Вершинное покрытие



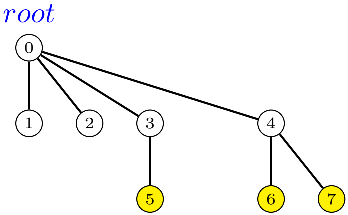
- Вершины, раскрашенные красным образуют вершинное покрытие.
- Задача Вершинное покрытие является NP-трудной.
- Тем не менее, легко найти минимальное покрытие вершин для **деревьев**.

Принцип оптимальности



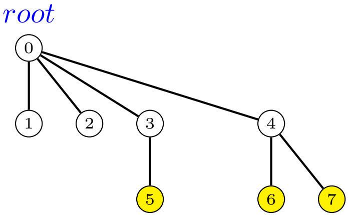
- Дано корневое дерево с n узлами. Давайте рассмотрим, можно ли свести задачу к меньшим подзадачам.

Принцип оптимальности



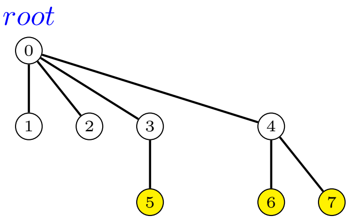
- Дано корневое дерево с n узлами. Давайте рассмотрим, можно ли свести задачу к меньшим подзадачам.
- Решение: **пошаговый выбор вершин** в подмножество. На каждом шаге мы решаем, следует ли брать вершину.

Принцип оптимальности



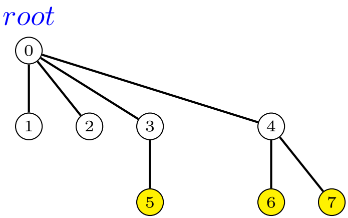
- Дано корневое дерево с n узлами. Давайте рассмотрим, можно ли свести задачу к меньшим подзадачам.
- Решение: **пошаговый выбор вершин** в подмножество. На каждом шаге мы решаем, следует ли брать вершину.
- **Первый** шаг построения решения — содержит ли оптимальное решение **корневую** вершину или нет. Два варианта:

Принцип оптимальности



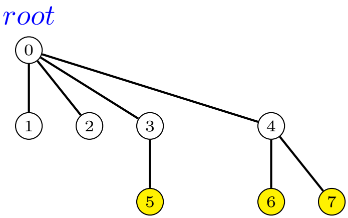
- Дано корневое дерево с n узлами. Давайте рассмотрим, можно ли свести задачу к меньшим подзадачам.
- Решение: **пошаговый выбор вершин** в подмножество. На каждом шаге мы решаем, следует ли брать вершину.
- **Первый** шаг построения решения — содержит ли оптимальное решение **корневую** вершину или нет. Два варианта:
  1. Взять: далее достаточно решать задачу для свисающих поддеревьев;

Принцип оптимальности

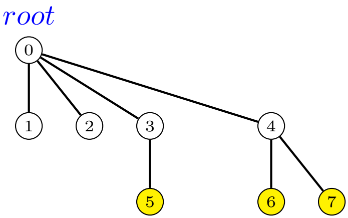


- Дано корневое дерево с n узлами. Давайте рассмотрим, можно ли свести задачу к меньшим подзадачам.
- Решение: **пошаговый выбор вершин** в подмножество. На каждом шаге мы решаем, следует ли брать вершину.
- **Первый** шаг построения решения — содержит ли оптимальное решение **корневую** вершину или нет. Два варианта:
  1. Взять: далее достаточно решать задачу для свисающих поддеревьев;
  2. Не брать: тогда мы должны брать всех сыновей этого узла, а затем решать задачу для всех поддеревьев, подвешенных к ним.

Пример

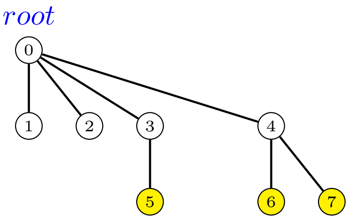


Пример



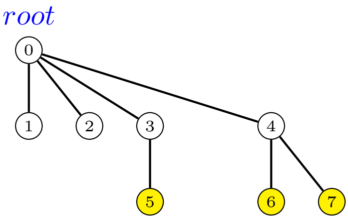
- В обоих случаях исходная проблема сводится к меньшим подзадачам.

Пример



- В обоих случаях исходная проблема сводится к меньшим подзадачам.
- Общая форма подзадач: найти наименьшее вершинное покрытие на дереве с корнем в узле v. Обозначим оптимальное решение как $OPT(v)$.

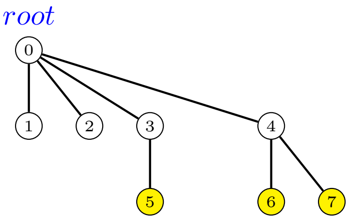
Пример



- В обоих случаях исходная проблема сводится к меньшим подзадачам.
- Общая форма подзадач: найти наименьшее вершинное покрытие на дереве с корнем в узле v. Обозначим оптимальное решение как $OPT(v)$.
- Таким образом, мы имеем следующую рекурсию:

$$OPT(root) = \min \begin{cases} 1 + \sum_c OPT(c) & c \text{ – сын} \\ \#\text{сыновей} + \sum_g OPT(g) & g \text{ – внук} \end{cases}$$

Пример



- В обоих случаях исходная проблема сводится к меньшим подзадачам.
- Общая форма подзадач: найти наименьшее вершинное покрытие на дереве с корнем в узле v. Обозначим оптимальное решение как $OPT(v)$.
- Таким образом, мы имеем следующую рекурсию:

$$OPT(root) = \min \begin{cases} 1 + \sum_c OPT(c) & c \text{ – сын} \\ \#\text{сыновей} + \sum_g OPT(g) & g \text{ – внук} \end{cases}$$

- Трудоемкость: $O(n)$ (Причина: каждый узел будет рассмотрен дважды.)

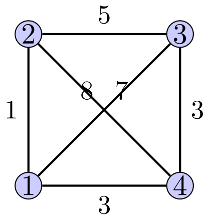
Алгоритм Bellman-Held-Karp для задачи коммивояжера: рекурсия над **графами**

Задача коммивояжера (Travelling Salesman Problem)

ВХОД: список из n городов (обозначим как V), и расстояния между городами d_{ij} $(1 \leq i, j \leq n)$;
ВЫХОД: кратчайший тур, который посещает каждый город ровно один раз и возвращается в исходный город

Задача коммивояжера (Travelling Salesman Problem)

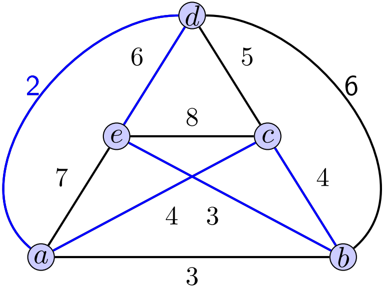
ВХОД: список из n городов (обозначим как V), и расстояния между городами d_{ij} $(1 \leq i, j \leq n)$;
ВЫХОД: кратчайший тур, который посещает каждый город ровно один раз и возвращается в исходный город

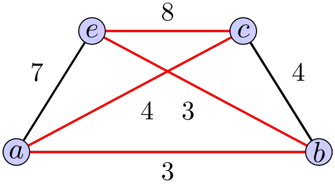


#Обходов: 6

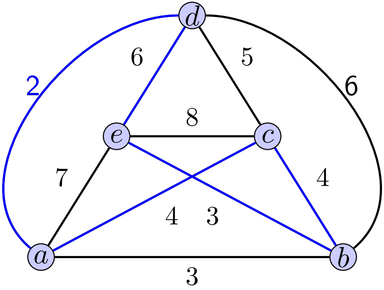
- Тур 1: $1 \to 2 \to 3 \to 4 \to 1$ (12)
- Тур 2: $1 \to 2 \to 4 \to 3 \to 1$ (21)
- Тур 3: $1 \to 3 \to 2 \to 4 \to 1$ (23)
- ....

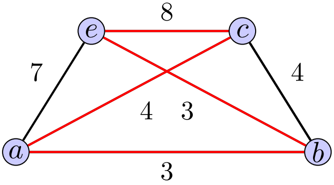
Декомпозиция исходной задачи на подзадачи





Декомпозиция исходной задачи на подзадачи





- Обратите внимание, что нелегко получить оптимальное решение исходной задачи (например, обход синего цвета) посредством оптимального решения подзадачи (например, обход красного цвета).

Рассмотрим близкую по постановке задачу

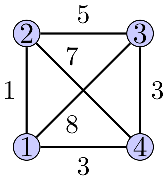
- Давайте рассмотрим тесно связанную задачу: вычислить $M(s, S, e)$, длину цепи, начинающейся в городе s, проходящей через каждый город из S ровно один раз и заканчивающейся в городе e.

Рассмотрим близкую по постановке задачу

- Давайте рассмотрим тесно связанную задачу: вычислить $M(s, S, e)$, длину цепи, начинающейся в городе s, проходящей через каждый город из S ровно один раз и заканчивающейся в городе e.
- Задачу TSP легко решить, когда для всех подмножеств вычислено $M(s, S, e)$, где $S \subseteq V$ и $e \in V$.

Рассмотрим близкую по постановке задачу

- Давайте рассмотрим тесно связанную задачу: вычислить $M(s, S, e)$, длину цепи, начинающейся в городе s, проходящей через каждый город из S ровно один раз и заканчивающейся в городе e.
- Задачу TSP легко решить, когда для всех подмножеств вычислено $M(s, S, e)$, где $S \subseteq V$ и $e \in V$.



- Например, самый короткий тур может быть рассчитан так:

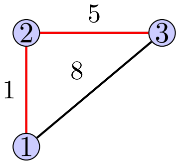
$$\min\{ \begin{aligned} & d_{2,1} + M(1, \{3, 4\}, 2), \\ & d_{3,1} + M(1, \{2, 4\}, 3), \\ & d_{4,1} + M(1, \{2, 3\}, 4)\} \end{aligned}$$

Рассмотрим самый маленькую задачу, вычисления $M(s, S, e)$

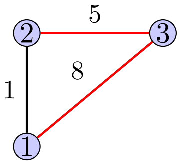
- Тривиально вычислить $M(s, S, e)$, когда S состоит только из одного города.

Рассмотрим самый маленькую задачу, вычисления $M(s, S, e)$

- Тривиально вычислить $M(s, S, e)$, когда S состоит только из одного города.



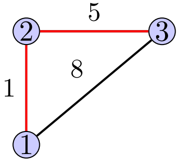
$M(1, \{2\}, 3) = d_{12} + d_{23}$



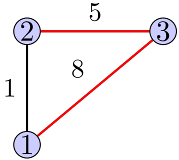
$M(1, \{3\}, 2) = d_{13} + d_{32}$

Рассмотрим самый маленькую задачу, вычисления $M(s, S, e)$

- Тривиально вычислить $M(s, S, e)$, когда S состоит только из одного города.



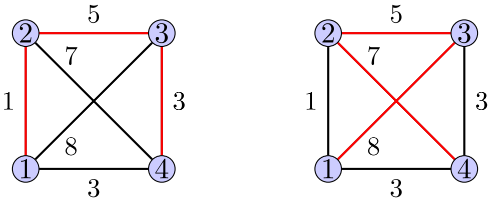
$M(1, \{2\}, 3) = d_{12} + d_{23}$



$M(1, \{3\}, 2) = d_{13} + d_{32}$

- Но как решить задачу большего размера, скажем, $M(1, \{2, 3\}, 4)$?

Декомпозиция большой задачи на меньшие задачи



- $M(1, \{2, 3\}, 4) = \min\{d_{34} + M(1, \{2\}, 3), d_{24} + M(1, \{3\}, 2)\}$

Алгоритм Bellman-Held-Karp [1962]

function $\mathrm{TSP}(D)$

return $\min_{e \in V, e \neq s} M(s, V - \{e\}, e) + d_{es}$;

function $\mathrm{M}(s, S, e)$

if $S = \{v\}$ **then**

  $\mathrm{M}(s, S, e) = d_{sv} + d_{ve}$;

  return $\mathrm{M}(s, S, e)$;

end if

return $\min_{i \in S,\ i \neq e} \mathrm{M}(s, S - \{i\}, i) + d_{ei}$;

Пример

	{b}	{c}	{d}	{e}	{b, c}	{b, d}	{b, e}	{c, d}	{c, e}	{d, e}	{b, c ,d}	{b, c, e}	{b, d, e}	{d, c, e}
b	-	8	8	10	-	-	-	11	15	11	-	-	-	18
c	7	-	7	15	-	12	14	-	-	16	-	-	15	-
d	9	9	-	13	12	-	12	-	18	-	-	17	-	-
e	6	12	8	-	11	11	-	15	-	-	14	-	-	-

- Сложность по памяти: $\sum_{k=2}^{n-1} k\binom{n-1}{k} + n - 1 = (n-1)2^{n-2}$
- Трудоемкость: $\sum_{k=2}^{n-1} k(k-1)\binom{n-1}{k} + n - 1 = O(2^n n^2)$.

Задачи о кратчайших путях: рекурсия над **графами**