

Мини-задача #37 (1 балл)

Найти количество уникальных (по форме) BST. Используйте алгоритм динамического программирования.

<https://leetcode.com/problems/unique-binary-search-trees>

Мини-задача #38 (1 балла)

Помогите рыцарю выйти из подземелья!

<https://leetcode.com/problems/dungeon-game>

Алгоритмы и структуры данных

Динамическое программирование: основные принципы,
примеры

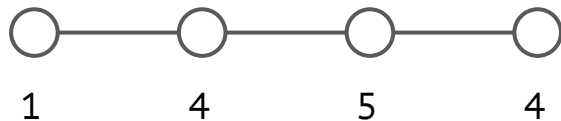


Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

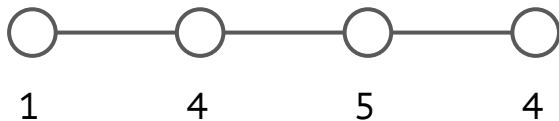
Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.



Задача

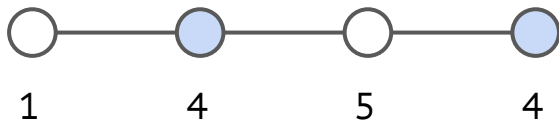
Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.



Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Задача

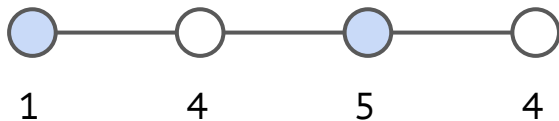
Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.



Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Задача

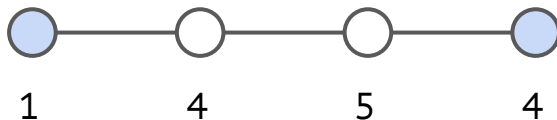
Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.



Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Задача

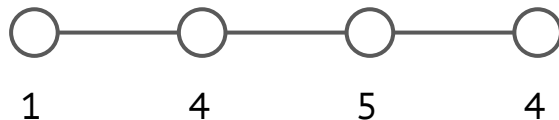
Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.



Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

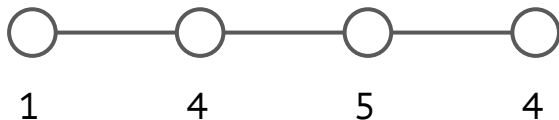


Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Варианты решения?

Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

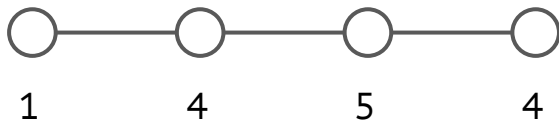


Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Варианты решения: полный перебор дает экспоненциальную сложность

Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

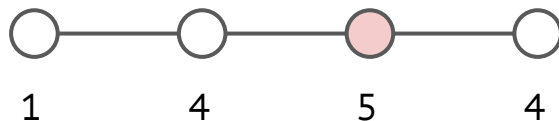


Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Варианты решения: жадный алгоритм?

Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

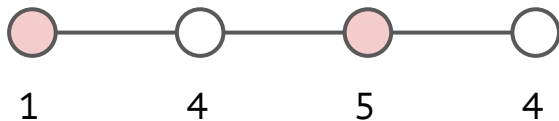


Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Варианты решения: **жадный** алгоритм?

Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

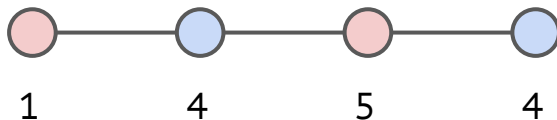


Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Варианты решения: **жадный** алгоритм?

Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

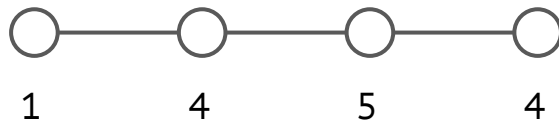


Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Варианты решения: **жадный** алгоритм? Дает неправильный ответ даже на тривиальном примере выше (6 против 8)

Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

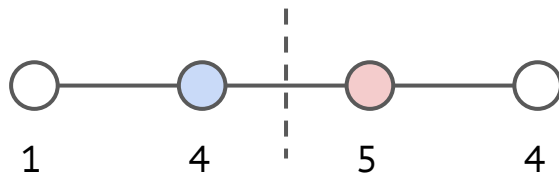


Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Варианты решения: **divide and conquer?**

Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

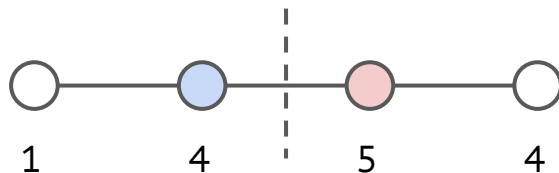


Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Варианты решения: **divide and conquer?**

Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

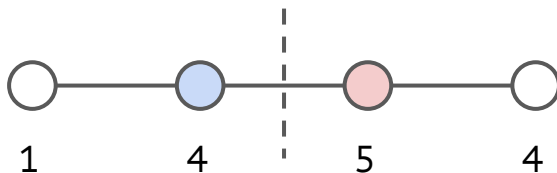


Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Варианты решения: **divide and conquer**? Абсолютно непонятно, как эффективно организовать слияние в случае конфликта.

Задача

Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

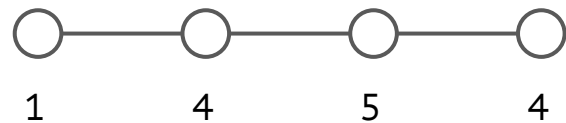


Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Варианты решения: **divide and conquer**? Абсолютно непонятно, как эффективно организовать слияние в случае конфликта.

Есть алгоритм, D&C который это делает за $O(N^2)$. Как лучше?

Оптимальная подструктура

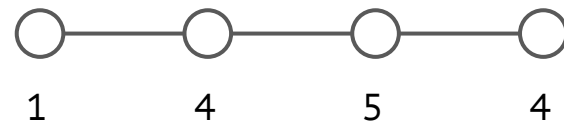


Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Мысленный эксперимент: если бы у нас было оптимальное решение, чтобы мы могли про него сказать?

Оптимальная подструктура

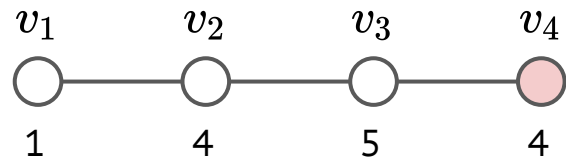


Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Мысленный эксперимент: если бы у нас было оптимальное решение, чтобы мы могли про него сказать? И как бы оно было связано с оптимальным решением для задачи **меньшего размера**?

Оптимальная подструктура



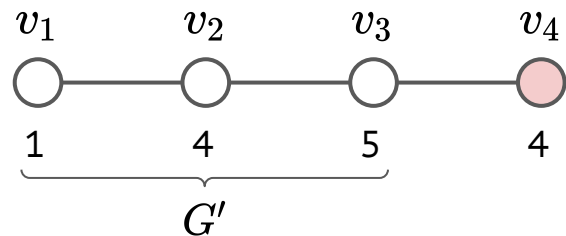
Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Пусть S - искомое подмножество. Подумаем о последней вершине в нашем пути v_n . Есть два варианта:

1) v_n не входит в S .

Оптимальная подструктура



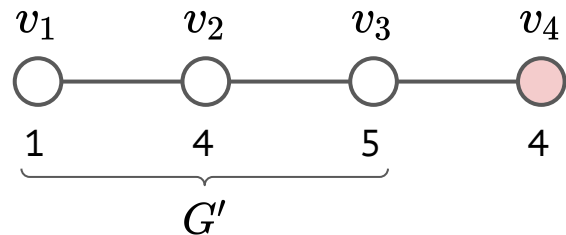
Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Пусть S - искомое подмножество. Подумаем о последней вершине в нашем пути v_n . Есть два варианта:

1) v_n не входит в S .

Оптимальная подструктура



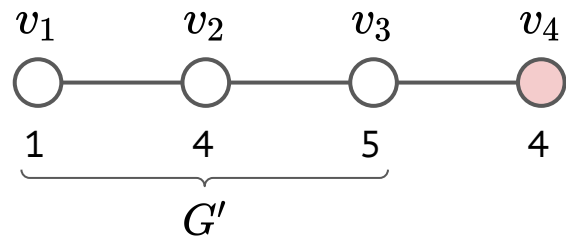
Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Пусть S - искомое подмножество. Подумаем о последней вершине в нашем пути v_n . Есть два варианта:

- 1) v_n не входит в S . Тогда, во-первых, S - множество **несмежных вершин** в G' .

Оптимальная подструктура



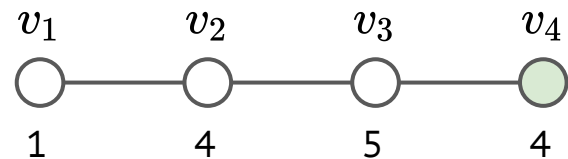
Дано: пусть есть простой путь $G = (V, E)$ в графе с весами на вершинах.

Найти: подмножество несмежных вершин (независимое множество) максимального веса.

Пусть S - искомое подмножество. Подумаем о последней вершине в нашем пути v_n . Есть два варианта:

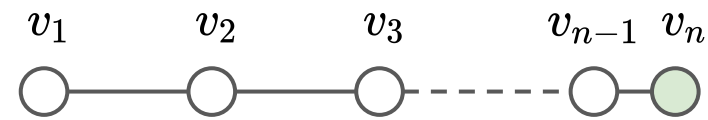
- 1) v_n не входит в S . Тогда, во-первых, S - множество **несмежных вершин** в G' . Более того, оно там является **максимальным** (иначе, почему мы не взяли его в качестве ответа?)

Оптимальная подструктура



Пусть S - искомое подмножество. Подумаем о последней вершине в нашем пути v_n . Есть два варианта:

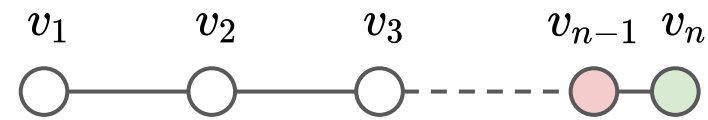
- 1) v_n не входит в S . Тогда S - **максимальное** множество **несмежных вершин** в G' .
- 2) v_n входит в S .



Оптимальная подструктура

Пусть S - искомое подмножество. Подумаем о последней вершине в нашем пути v_n . Есть два варианта:

- 1) v_n не входит в S . Тогда S - **максимальное** множество **несмежных вершин** в G' .
- 2) v_n входит в S .

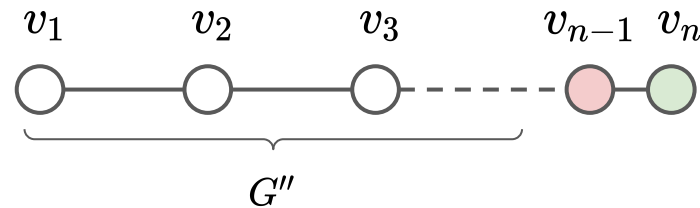


Оптимальная подструктура

Пусть S - искомое подмножество. Подумаем о последней вершине в нашем пути v_n . Есть два варианта:

- 1) v_n не входит в S . Тогда S - **максимальное** множество **несмежных вершин** в G' .
- 2) v_n входит в S . Значит v_{n-1} не входит в S .

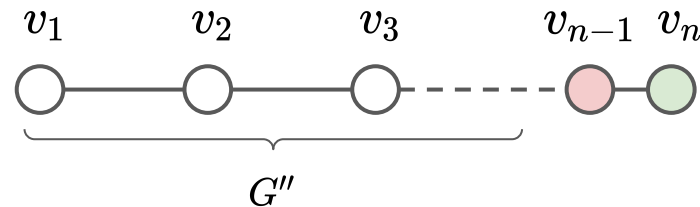
Оптимальная подструктура



Пусть S - искомое подмножество. Подумаем о последней вершине в нашем пути v_n . Есть два варианта:

- 1) v_n не входит в S . Тогда S - **максимальное** множество **несмежных вершин** в G' .
- 2) v_n входит в S . Значит v_{n-1} не входит в S .

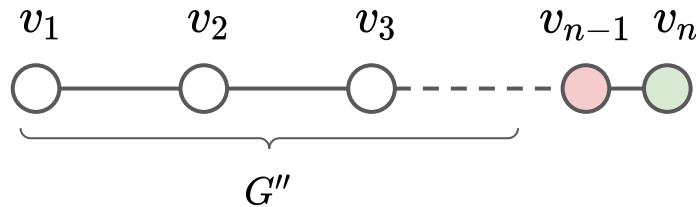
Оптимальная подструктура



Пусть S - искомое подмножество. Подумаем о последней вершине в нашем пути v_n . Есть два варианта:

- 1) v_n не входит в S . Тогда S - **максимальное** множество **несмежных вершин** в G' .
- 2) v_n входит в S . Значит v_{n-1} не входит в S . Тогда множество $S - \{v_n\}$ является **максимальным** множеством **несмежных вершин** в подграфе G'' .

Оптимальная подструктура

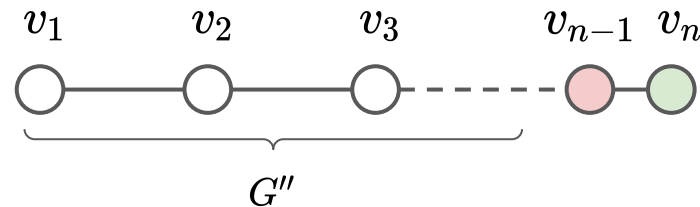


Пусть S - искомое подмножество. Подумаем о последней вершине в нашем пути v_n . Есть два варианта:

- 1) v_n не входит в S . Тогда S - **максимальное** множество **несмежных вершин** в G' .
- 2) v_n входит в S . Значит v_{n-1} не входит в S . Тогда множество $S - \{v_n\}$ является **максимальным** множеством **несмежных вершин** в подграфе G'' .

Т.е. в любом случае оптимальное решение выражается через оптимальные решения для подзадач (для G' или G'')!

Оптимальная подструктура

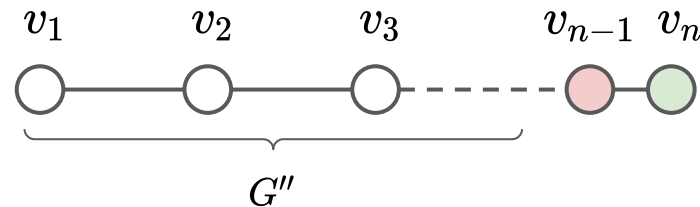


- 1) Либо S - **максимальное** множество **несмежных** вершин в G' .
- 2) Либо $S - \{v_n\}$ - **макс.** множество **несмежных** вершин в G'' .

Но понять, какой именно рекурсивный вызов нужно делать можно только в зависимости от v_n .



Оптимальная подструктура



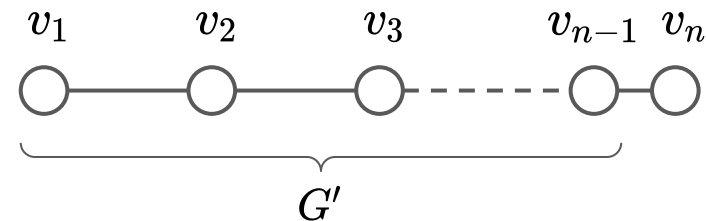
- 1) Либо S - **максимальное** множество **несмежных вершин** в G' .
- 2) Либо $S - \{v_n\}$ - **макс.** множество **несмежных вершин** в G'' .

Но понять, какой именно рекурсивный вызов нужно делать можно только в зависимости от v_n .

Почему бы не попробовать оба рекурсивных вызова?



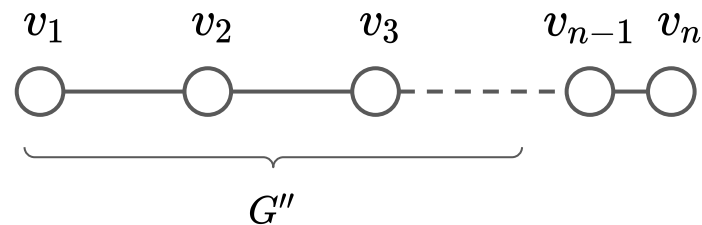
Рекурсивное решение



Алгоритм:

1. Вычисляем S_1 для G'

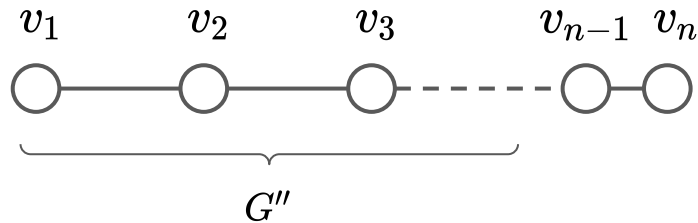
Рекурсивное решение



Алгоритм:

1. Вычисляем S_1 для G'
2. Вычисляем S_2 для G''

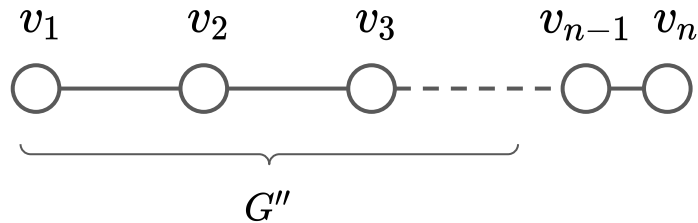
Рекурсивное решение



Алгоритм:

1. Вычисляем S_1 для G'
2. Вычисляем S_2 для G''
3. Берем в ответ v_n или нет в зависимости от того, что **больше** (ведь мы ищем **максимальное** множество)

Рекурсивное решение

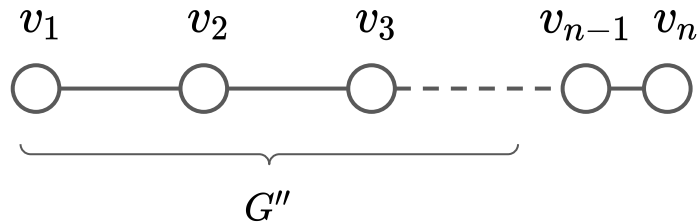


Алгоритм:

1. Вычисляем S_1 для G'
2. Вычисляем S_2 для G''
3. Берем в ответ v_n или нет в зависимости от того, что **больше** (ведь мы ищем **максимальное** множество)

Проблемы?

Рекурсивное решение



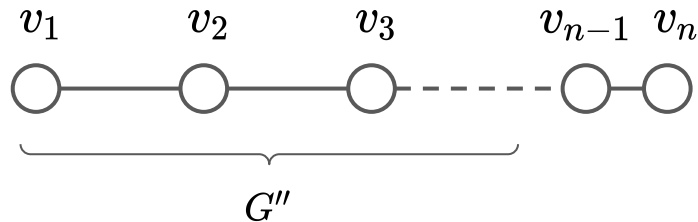
Алгоритм:

1. Вычисляем S_1 для G'
2. Вычисляем S_2 для G''
3. Берем в ответ v_n или нет в зависимости от того, что **больше** (ведь мы ищем **максимальное** множество)

Проблемы? Это же полный перебор!
Экспоненциальная сложность!



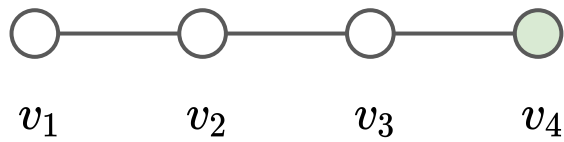
Рекурсивное решение

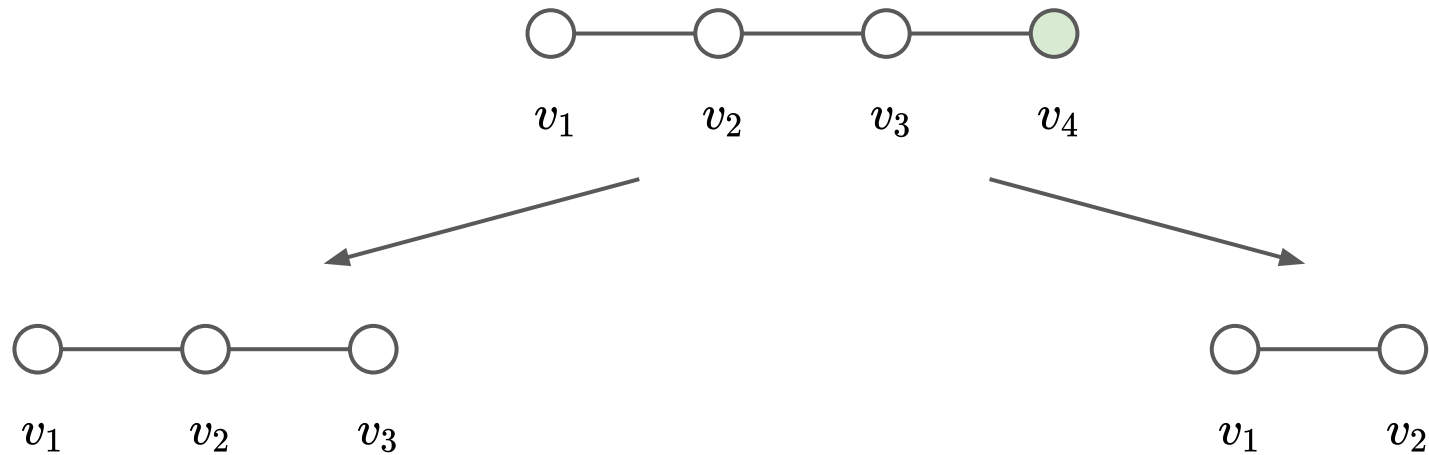


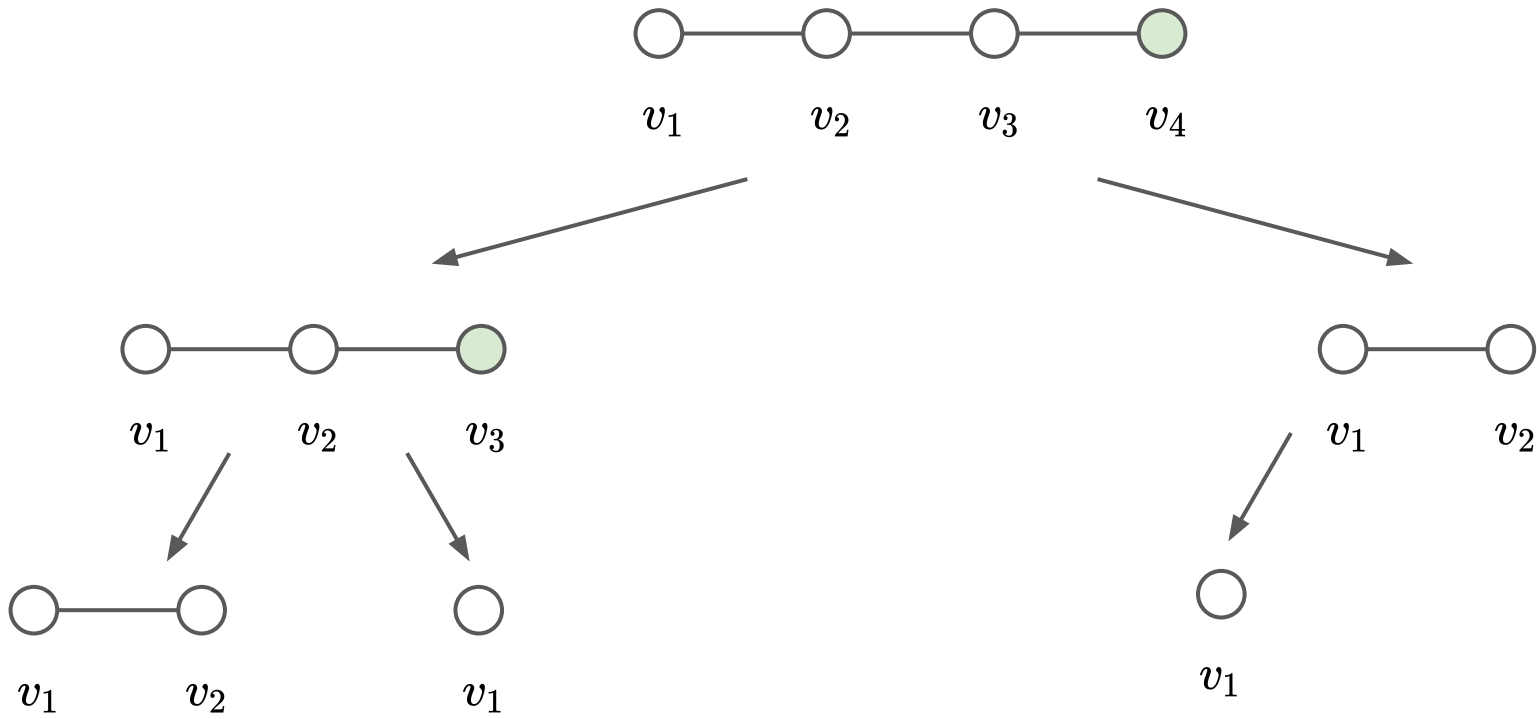
Алгоритм:

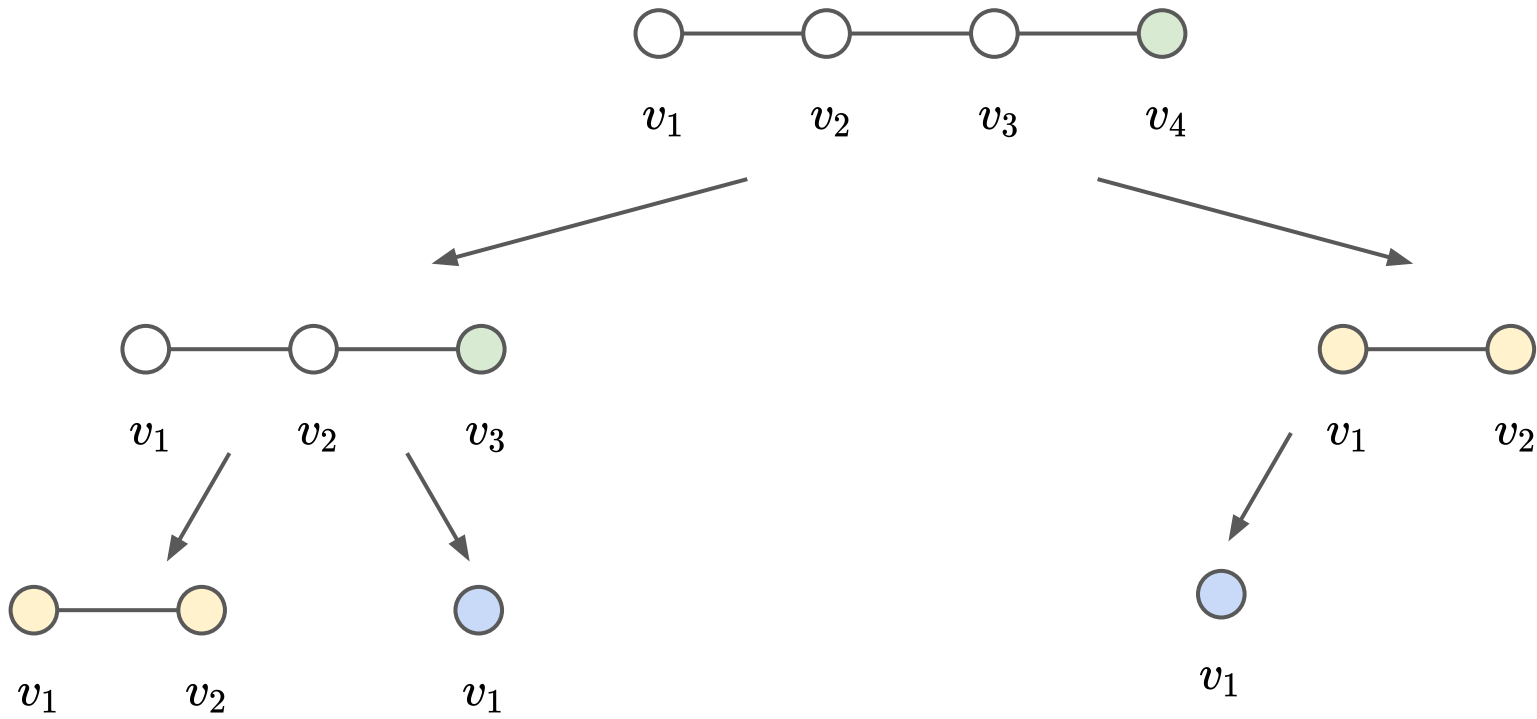
1. Вычисляем S_1 для G'
2. Вычисляем S_2 для G''
3. Берем в ответ v_n или нет в зависимости от того, что **больше** (ведь мы ищем **максимальное** множество)

Но сколько раз в этой рекурсии будут встречаться **различные** подмножества?

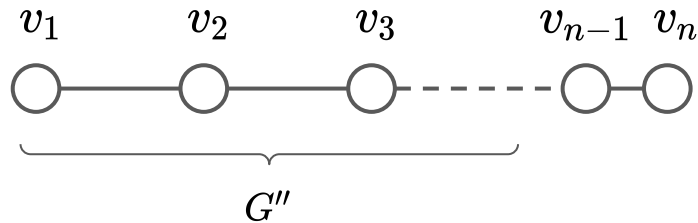








Рекурсивное решение

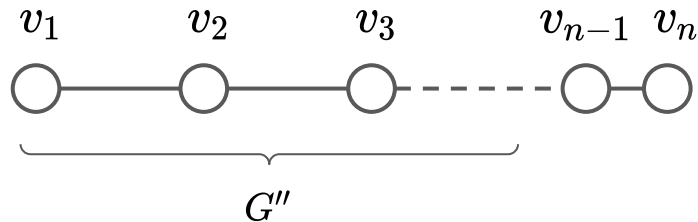


Алгоритм:

1. Вычисляем S_1 для G'
2. Вычисляем S_2 для G''
3. Берем в ответ v_n или нет в зависимости от того, что **больше** (ведь мы ищем **максимальное** множество)

Но сколько раз в этой рекурсии будут встречаться **различные** подмножества?

Рекурсивное решение

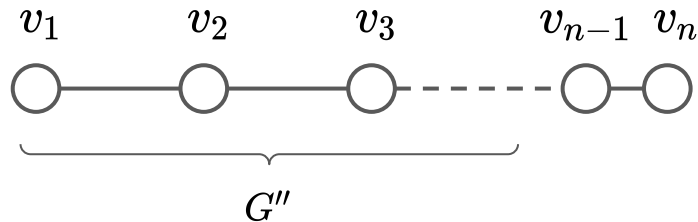


Алгоритм:

1. Вычисляем S_1 для G'
2. Вычисляем S_2 для G''
3. Берем в ответ v_n или нет в зависимости от того, что **больше** (ведь мы ищем **максимальное** множество)

Но сколько раз в этой рекурсии будут встречаться **различные** подмножества? Всего N штук!

Рекурсивное решение

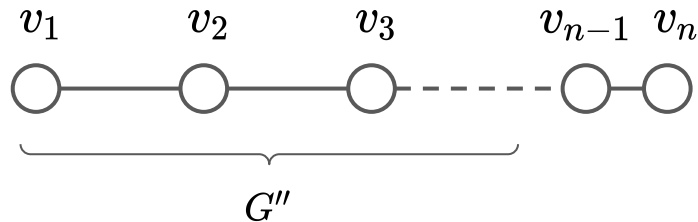


Алгоритм:

1. Вычисляем S_1 для G'
2. Вычисляем S_2 для G''
3. Берем в ответ v_n или нет в зависимости от того, что **больше** (ведь мы ищем **максимальное** множество)

Но сколько раз в этой рекурсии будут встречаться **различные** подмножества? Всего N штук! Т.к. мы всегда работаем с префиксом пути, а их количество **линейно**.

Рекурсивное решение

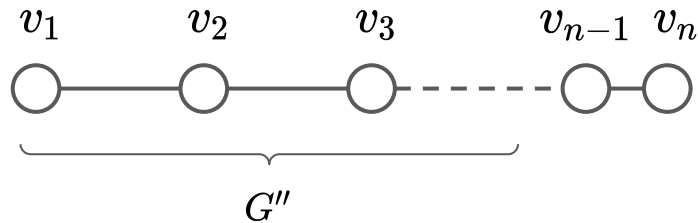


Алгоритм:

1. Вычисляем S_1 для G'
2. Вычисляем S_2 для G''
3. Берем в ответ v_n или нет в зависимости от того, что **больше** (ведь мы ищем **максимальное** множество)

Как исправить алгоритм?

Рекурсивное решение



Алгоритм:

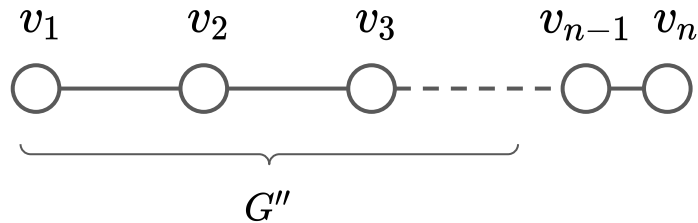
1. Вычисляем S_1 для G'
2. Вычисляем S_2 для G''
3. Берем в ответ v_n или нет в зависимости от того, что **больше** (ведь мы ищем **максимальное** множество)

Как исправить алгоритм?

Мемоизация вполне сработает.



Рекурсивное решение



Алгоритм:

1. Вычисляем S_1 для G'
2. Вычисляем S_2 для G''
3. Берем в ответ v_n или нет в зависимости от того, что **больше** (ведь мы ищем **максимальное** множество)

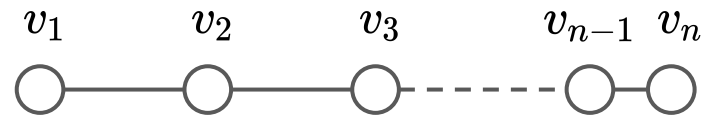
Как исправить алгоритм?

Мемоизация вполне сработает.

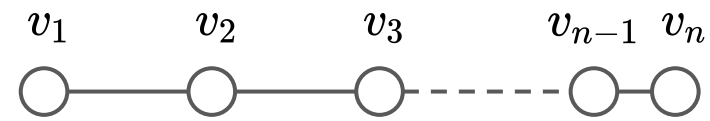
Но можно лучше!



Линейное решение



Алгоритм (для нахождения **веса** макс. независимого множества):

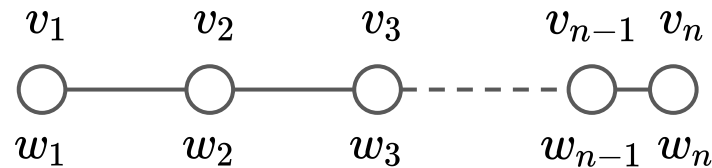


Линейное решение

Алгоритм (для нахождения **веса** макс. независимого множества):

- Заводим массив A для результата. В i -ой ячейке будет ответ для префикса размера i .

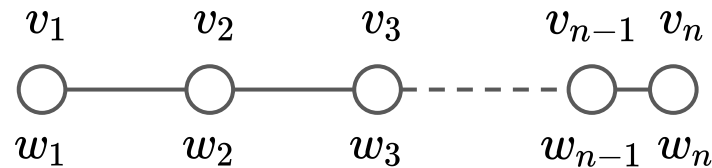
Линейное решение



Алгоритм (для нахождения **веса** макс. независимого множества):

- Заводим массив A для результата. В i -ой ячейке будет ответ для префикса размера i .
- Инициализируем: $A[0] = ?$, $A[1] = ?$

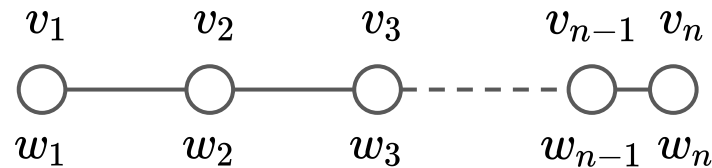
Линейное решение



Алгоритм (для нахождения **веса** макс. независимого множества):

- Заводим массив A для результата. В i -ой ячейке будет ответ для префикса размера i .
- Инициализируем: $A[0] = 0$, $A[1] = w_1$

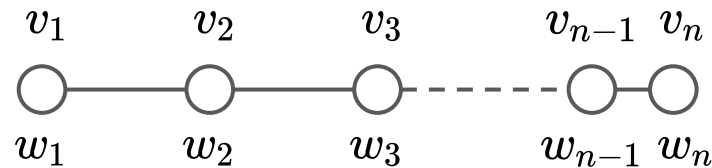
Линейное решение



Алгоритм (для нахождения **веса** макс. независимого множества):

- Заводим массив A для результата. В i -ой ячейке будет ответ для префикса размера i .
- Инициализируем: $A[0] = 0$, $A[1] = w_1$
- Идем слева направо: $A[i] = ?$

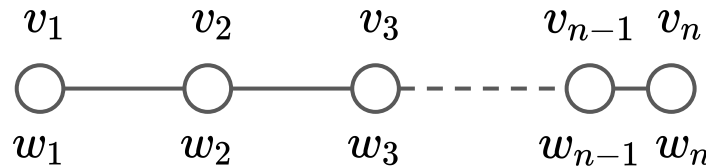
Линейное решение



Алгоритм (для нахождения **веса** макс. независимого множества):

- Заводим массив A для результата. В i -ой ячейке будет ответ для префикса размера i .
- Инициализируем: $A[0] = 0, A[1] = w_1$
- Идем слева направо: $A[i] = \max(A[i-1], A[i-2] + w_i)$

Линейное решение



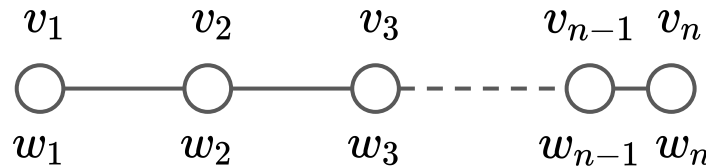
Алгоритм (для нахождения **веса** макс. независимого множества):

- Заводим массив A для результата. В i -ой ячейке будет ответ для префикса размера i .
- Инициализируем: $A[0] = 0, A[1] = w_1$
- Идем слева направо: $A[i] = \max(A[i-1], A[i-2] + w_i)$

И все! Это один проход по массиву - линейная сложность!



Линейное решение



Алгоритм (для нахождения **веса** макс. независимого множества):

- Заводим массив A для результата. В i -ой ячейке будет ответ для префикса размера i .
- Инициализируем: $A[0] = 0, A[1] = w_1$
- Идем слева направо: $A[i] = \max(A[i-1], A[i-2] + w_i)$

И все! Это один проход по массиву - линейная сложность!

Очень похожая история с вычислением чисел Фибоначчи в Лекции #1 прошлого курса.



Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма).

Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

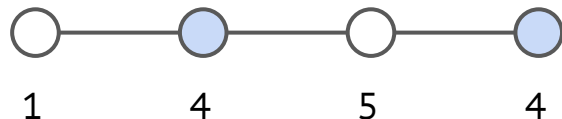
Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма). Можем быть неэффективно по памяти.

Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма). Можем быть **неэффективно** по памяти.

Ответ #2: восстановить ответ по нашему массиву A !



Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма). Можем быть **неэффективно** по памяти.

Ответ #2: восстановить ответ по нашему массиву A !



$A[0]$



$A[1]$



$A[2]$



$A[3]$



$A[4]$

Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма). Можем быть **неэффективно** по памяти.

Ответ #2: восстановить ответ по нашему массиву A !



$A[0]$ $A[1]$ $A[2]$ $A[3]$ $A[4]$

Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма). Можем быть **неэффективно** по памяти.

Ответ #2: восстановить ответ по нашему массиву A !



$\boxed{0}$ $\boxed{1}$ $\boxed{4}$ $\boxed{}$ $\boxed{}$

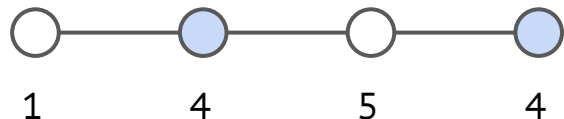
$A[0]$ $A[1]$ $A[2]$ $A[3]$ $A[4]$

Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма). Можем быть **неэффективно** по памяти.

Ответ #2: восстановить ответ по нашему массиву A !



0 1 4 6

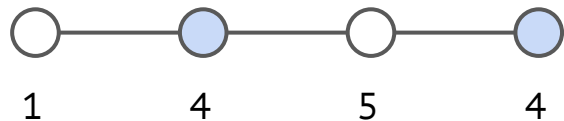
$A[0]$ $A[1]$ $A[2]$ $A[3]$ $A[4]$

Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма). Можем быть **неэффективно** по памяти.

Ответ #2: восстановить ответ по нашему массиву A !



$A[0]$ $A[1]$ $A[2]$ $A[3]$ $A[4]$

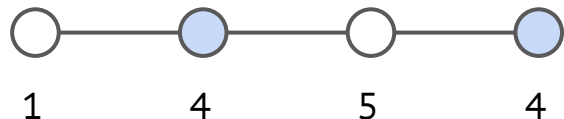
$A[0]$ $A[1]$ $A[2]$ $A[3]$ $A[4]$

Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма). Можем быть **неэффективно** по памяти.

Ответ #2: восстановить ответ по нашему массиву A !



Теперь проходим по массиву A **справа**, "вспоминая", какой выбор мы здесь сделали.

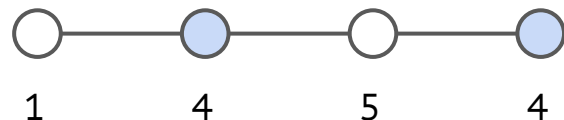
<div style="border: 1px solid black; padding: 2px 5px;">0</div>	<div style="border: 1px solid black; padding: 2px 5px;">1</div>	<div style="border: 1px solid black; padding: 2px 5px;">4</div>	<div style="border: 1px solid black; padding: 2px 5px;">6</div>	<div style="border: 1px solid black; padding: 2px 5px;">8</div>
$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$

Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма). Можем быть **неэффективно** по памяти.

Ответ #2: восстановить ответ по нашему массиву A !



Теперь проходим по массиву A **справа**, "вспоминая", какой выбор мы здесь сделали.

Если $A[i-1] \geq A[i-2] + w_i$, значит мы не брали i -ый элемент, идем дальше.

<div>0</div>	<div>1</div>	<div>4</div>	<div>6</div>	<div>8</div>
$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$

Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма). Можем быть **неэффективно** по памяти.

Ответ #2: восстановить ответ по нашему массиву A !



Теперь проходим по массиву A **справа**, "вспоминая", какой выбор мы здесь сделали.

Если $A[i-1] \geq A[i-2] + w_i$, значит мы не брали i -ый элемент, идем дальше.

Иначе, добавляем i в ответ

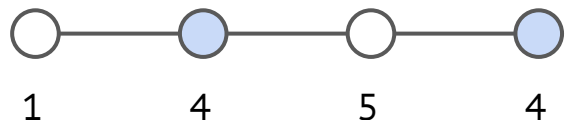
<div>0</div>	<div>1</div>	<div>4</div>	<div>6</div>	<div>8</div>
$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$

Восстановление множества

Вопрос: а что, если нужен не только вес, но и само множество?

Ответ #1: можно просто хранить максимальные подмножества префикса размера i (сохранять их на каждом шаге алгоритма). Можем быть **неэффективно** по памяти.

Ответ #2: восстановить ответ по нашему массиву A !



Теперь проходим по массиву A **справа**, "вспоминая", какой выбор мы здесь сделали.

Если $A[i-1] \geq A[i-2] + w_i$, значит мы не брали i -ый элемент, идем дальше.

Иначе, добавляем i в ответ (и $i -= 2$).

Динамическое программирование

Динамическое программирование

1. Думаем про структуру и свойства оптимального решения,

Динамическое программирование

1. Думаем про структуру и свойства оптимального решения,
2. Понимаем связь с оптимальной подструктурой.
В этот момент естественно выписать рекурсивное решение,

Динамическое программирование

1. Думаем про структуру и свойства **оптимального решения**,
2. Понимаем связь с **оптимальной подструктурой**.
В этот момент естественно выписать рекурсивное решение,
3. Если:
 - а. (различных) подзадач мало,

Еще говорят, что решение содержит **перекрывающиеся вспомогательные** задачи.

Динамическое программирование

1. Думаем про структуру и свойства оптимального решения,
2. Понимаем связь с оптимальной подструктурой.
В этот момент естественно выписать рекурсивное решение,
3. Если:
 - а. (различных) подзадач мало,
 - б. задачи большей размерности быстро решаются через задачи меньше размерности,

Динамическое программирование

1. Думаем про структуру и свойства оптимального решения,
2. Понимаем связь с оптимальной подструктурой.
В этот момент естественно выписать рекурсивное решение,
3. Если:
 - a. (различных) подзадач мало,
 - b. задачи большей размерности быстро решаются через задачи меньше размерности,
 - c. финальное решение быстро вычисляется после решения всех подзадач,

То вам подойдет динамическое программирование!

Почему динамическое программирование?

Почему динамическое программирование?

1. "Динамическое" потому, что в алгоритме мы динамически принимает решение на каждом шаге,

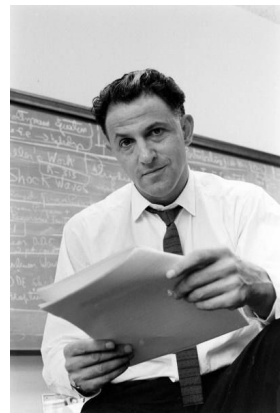
Почему динамическое программирование?

1. "Динамическое" потому, что в алгоритме мы динамически принимает решение на каждом шаге,
2. А "программирование" - это скорее замена слова "планирование", к программированию в привычном нам смысле имеет мало отношения.

Почему динамическое программирование?

1. "Динамическое" потому, что в алгоритме мы динамически принимает решение на каждом шаге,
2. А "программирование" - это скорее замена слова "планирование", к программированию в привычном нам смысле имеет мало отношения.

Использовалось Ричардом Беллманом для отвода глаз, чтобы военные не думали, что в их организации кто-то занимается математическими исследованиями.



Дискретная задача о рюкзаке



Дискретная задача о рюкзаке

Дано: дан набор из n предметов, при этом:

1. v_i - ценность i -ого предмета (неотрицательная)
2. w_i - размер i -ого предмета (положительное целое!)



Дискретная задача о рюкзаке

Дано: дан набор из n предметов, при этом:

1. v_i - ценность i -ого предмета (неотрицательная)
2. w_i - размер i -ого предмета (положительное целое!)
3. W - вместительность рюкзака (тоже целое).



Дискретная задача о рюкзаке

Дано: дан набор из n предметов, при этом:

1. v_i - ценность i -ого предмета (неотрицательная)
2. w_i - размер i -ого предмета (положительное целое!)
3. W - вместительность рюкзака (целое).

Найти: $S \in \{1, 2, \dots, n\}$ - подмножество предметов, такое что значение $\sum_{i \in S} v_i$ максимально, при этом $\sum_{i \in S} w_i \leq W$



Динамическое программирование

1. Думаем про структуру и свойства **оптимального решения**,
 2. Понимаем связь с **оптимальной подструктурой**.
В этот момент естественно записать рекурсивное решение,
3. Если:
- a. (различных) подзадач мало,
 - b. задачи большей размерности быстро решаются через задачи меньше размерности,
 - c. финальное решение быстро вычисляется после решения всех подзадач,

То вам подойдет **динамическое программирование**!

Дискретная задача о рюкзаке

Пусть есть оптимальное решение S , максимизирующее $\sum_{i \in S} v_i$.

Порассуждаем о том, как к нему относится "последний" предмет v_n .

Дискретная задача о рюкзаке

Пусть есть оптимальное решение S , максимизирующее $\sum_{i \in S} v_i$.

Порассуждаем о том, как к нему относится "последний" предмет v_n .

Случай #1: предмет n не входит в S .

Дискретная задача о рюкзаке

Пусть есть оптимальное решение S , максимизирующее $\sum_{i \in S} v_i$.

Порассуждаем о том, как к нему относится "последний" предмет v_n .

Случай #1: предмет n не входит в S . Тогда S - решение для задачи о рюкзаке с набором из оставшихся предметов $1, 2, \dots, n-1$ (с той же вместительностью рюкзака W)

Дискретная задача о рюкзаке

Пусть есть оптимальное решение S , максимизирующее $\sum_{i \in S} v_i$.

Порассуждаем о том, как к нему относится "последний" предмет v_n .

Случай #1: предмет n не входит в S . Тогда S - решение для задачи о рюкзаке с набором из оставшихся предметов $1, 2, \dots, n-1$ (с той же вместительностью рюкзака W)

(иначе, если есть решение получше, взяли бы его и для задачи из n предметов)

Дискретная задача о рюкзаке

Пусть есть оптимальное решение S , максимизирующее $\sum_{i \in S} v_i$.

Порассуждаем о том, как к нему относится "последний" предмет v_n .

Случай #1: предмет n не входит в S . Тогда S - решение для задачи о рюкзаке с набором из оставшихся предметов $1, 2, \dots, n-1$ (с той же вместительностью рюкзака W)

Случай #2: предмет n входит в S . Что можете сказать про решение для задачи меньшей размерности?

Дискретная задача о рюкзаке

Пусть есть оптимальное решение S , максимизирующее $\sum_{i \in S} v_i$.

Порассуждаем о том, как к нему относится "последний" предмет v_n .

Случай #1: предмет n не входит в S . Тогда S - решение для задачи о рюкзаке с набором из оставшихся предметов $1, 2, \dots, n-1$ (с той же вместительностью рюкзака W)

Случай #2: предмет n входит в S . Что можете сказать про решение для задачи меньшей размерности?

$S - \{n\}$ - это решение задачи о рюкзаке...

Дискретная задача о рюкзаке

Пусть есть оптимальное решение S , максимизирующее $\sum_{i \in S} v_i$.

Порассуждаем о том, как к нему относится "последний" предмет v_n .

Случай #1: предмет n не входит в S . Тогда S - решение для задачи о рюкзаке с набором из оставшихся предметов $1, 2, \dots, n-1$ (с той же вместительностью рюкзака W)

Случай #2: предмет n входит в S . Что можете сказать про решение для задачи меньшей размерности?

$S - \{n\}$ - это решение задачи о рюкзаке из первых $1, 2, \dots, n-1$ предметов...

Дискретная задача о рюкзаке

Пусть есть оптимальное решение S , максимизирующее $\sum_{i \in S} v_i$.

Порассуждаем о том, как к нему относится "последний" предмет v_n .

Случай #1: предмет n не входит в S . Тогда S - решение для задачи о рюкзаке с набором из оставшихся предметов $1, 2, \dots, n-1$ (с той же вместительностью рюкзака W)

Случай #2: предмет n входит в S . Что можете сказать про решение для задачи меньшей размерности?

$S - \{n\}$ - это решение задачи о рюкзаке из первых $1, 2, \dots, n-1$ предметов при ограничении вместительности рюкзака $W - \{w_n\}$!

Дискретная задача о рюкзаке

Пусть есть оптимальное решение S , максимизирующее $\sum_{i \in S} v_i$.

Порассуждаем о том, как к нему относится "последний" предмет v_n .

Случай #1: предмет n не входит в S . Тогда S - решение для задачи о рюкзаке с набором из оставшихся предметов $1, 2, \dots, n-1$ (с той же вместительностью рюкзака W)

Случай #2: предмет n входит в S . Что можете сказать про решение для задачи меньшей размерности?

$S - \{n\}$ - это решение задачи о рюкзаке из первых $1, 2, \dots, n-1$ предметов при ограничении вместительности рюкзака $W - \{w_n\}$!

Док-во: если есть решение лучше - возьмите его и добавьте n .

Рекуррентное соотношение

Обозначим $V_{i,x}$ оптимальное решение задачи о рюкзаке, такое что:

1. Используем только $1, 2, \dots, i$ предметов,
2. Размерность рюкзака ограничена x .

Рекуррентное соотношение

Обозначим $V_{i,x}$ оптимальное решение задачи о рюкзаке, такое что:

1. Используем только $1, 2, \dots, i$ предметов,
2. Размерность рюкзака ограничена x .

Тогда:

$$V_{i,x} = \max \begin{cases} V_{i-1,x} \\ v_i + V_{i-1,x-w_i} \end{cases}$$

Рекуррентное соотношение

Обозначим $V_{i,x}$ оптимальное решение задачи о рюкзаке, такое что:

1. Используем только $1, 2, \dots, i$ предметов,
2. Размерность рюкзака ограничена x .

Тогда:

$$V_{i,x} = \max \begin{cases} V_{i-1,x} \\ v_i + V_{i-1,x-w_i} \end{cases}$$

Но есть важное исключение! Если $w_i > x$, то $V_{i,x} = V_{i-1,x}$ (i -ый предмет просто не влезает)

Рассматриваемые подзадачи

Какие есть подзадачи?

Рассматриваемые подзадачи

Какие есть подзадачи?

1. Подзадачи с меньшим количеством используемых **предметов**.
Полная аналогия с независимыми множествами: рассматриваются **префиксы** множества $\{1, 2, \dots, i\}$

Рассматриваемые подзадачи

Какие есть подзадачи?

1. Подзадачи с меньшим количеством используемых **предметов**.
Полная аналогия с независимыми множествами: рассматриваются **префиксы** множества $\{1, 2, \dots, i\}$
2. Но кроме того, мы уменьшаем вместимость рюкзака в подзадачах! Подзадачи с какими размерами могут встретиться?

Дискретная задача о рюкзаке

Дано: дан набор из n предметов, при этом:

1. v_i - ценность i -ого предмета (неотрицательная)
2. w_i - размер i -ого предмета (положительное **целое** !)
3. W - вместительность рюкзака (тоже **целое**).



Рассматриваемые подзадачи

Какие есть подзадачи?

1. Подзадачи с меньшим количеством используемых **предметов**.
Полная аналогия с независимыми множествами: рассматриваются **префиксы** множества $\{1, 2, \dots, i\}$
2. Но кроме того, мы уменьшаем вместимость рюкзака в подзадачах! Подзадачи с какими размерами могут встретиться?

Т.к. и W и все w_i **целые**, то размерности рюкзака в подзадачах (в самом худшем случае) будут $\{0, 1, 2, \dots, W\}$.

Рассматриваемые подзадачи

Какие есть подзадачи?

1. Подзадачи с меньшим количеством используемых **предметов**.
Полная аналогия с независимыми множествами: рассматриваются **префиксы** множества $\{1, 2, \dots, i\}$
2. Но кроме того, мы уменьшаем вместимость рюкзака в подзадачах! Подзадачи с какими размерами могут встретиться?

Т.к. и W и все w_i **целые**, то размерности рюкзака в подзадачах (в самом худшем случае) будут $\{0, 1, 2, \dots, W\}$.

В обоих смыслах подзадач у нас мало - **линейное** количество.

Решение через восходящий анализ

Раз у нас две шкалы подзадач, то теперь нам нужен двумерный массив A , в котором будем записывать результат.

Решение через восходящий анализ

Раз у нас две шкалы подзадач, то теперь нам нужен двумерный массив A , в котором будем записывать результат.

Инициализация: $A[0][x] = 0$ для всех $x \in \{0, 1, 2, \dots, W\}$

Решение через восходящий анализ

Раз у нас две шкалы подзадач, то теперь нам нужен двумерный массив A , в котором будем записывать результат.

Инициализация: $A[0][x] = 0$ для всех $x \in \{0, 1, 2, \dots, W\}$

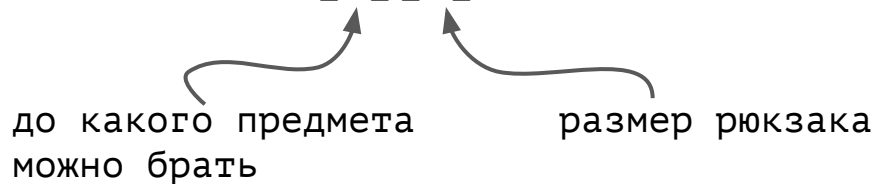
до какого предмета
можно брать

размер рюкзака

Решение через восходящий анализ

Раз у нас две шкалы подзадач, то теперь нам нужен двумерный массив A , в котором будем записывать результат.

Инициализация: $A[0][x] = 0$ для всех $x \in \{0, 1, 2, \dots, W\}$



до какого предмета
можно брать

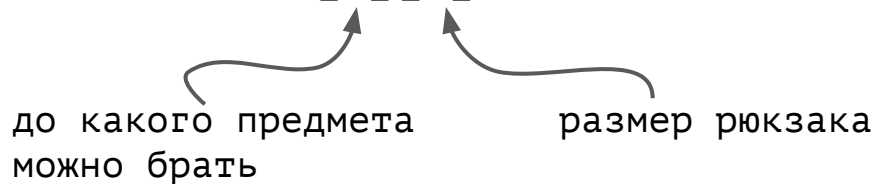
размер рюкзака

Заполнение массива: **for** $i = 1, 2, 3, \dots, n$:
 for $x = 0, 1, 2, \dots, W$:

Решение через восходящий анализ

Раз у нас две шкалы подзадач, то теперь нам нужен двумерный массив A , в котором будем записывать результат.

Инициализация: $A[0][x] = 0$ для всех $x \in \{0, 1, 2, \dots, W\}$



до какого предмета
можно брать

размер рюкзака

Заполнение массива:

```
for i = 1, 2, 3, ..., n:  
  for x = 0, 1, 2, ..., W:  
     $A[i][x] = \max(A[i - 1][x], A[i - 1][x - w_i] + v_i)$ 
```

Решение через восходящий анализ

Раз у нас две шкалы подзадач, то теперь нам нужен двумерный массив A , в котором будем записывать результат.

Инициализация: $A[0][x] = 0$ для всех $x \in \{0, 1, 2, \dots, W\}$

до какого предмета
можно брать

размер рюкзака

Заполнение массива:

for $i = 1, 2, 3, \dots, n$:

for $x = 0, 1, 2, \dots, W$:

$A[i][x] = \max(A[i - 1][x], A[i - 1][x - w_i] + v_i)$

(кроме случая, когда $w_i > x$,
там просто берем $A[i-1][x]$)

Решение через восходящий анализ

Сложность алгоритма?

Всего-то $O(n*W)$!



Решение через восходящий анализ

Сложность алгоритма?

Всего-то $O(n*W)$!

Замечание: мы опять нашли только суммарный вес вещей из оптимального решения, но и само решение можно восстановить из таблички за один проход (аналогично **независимым множествам**).

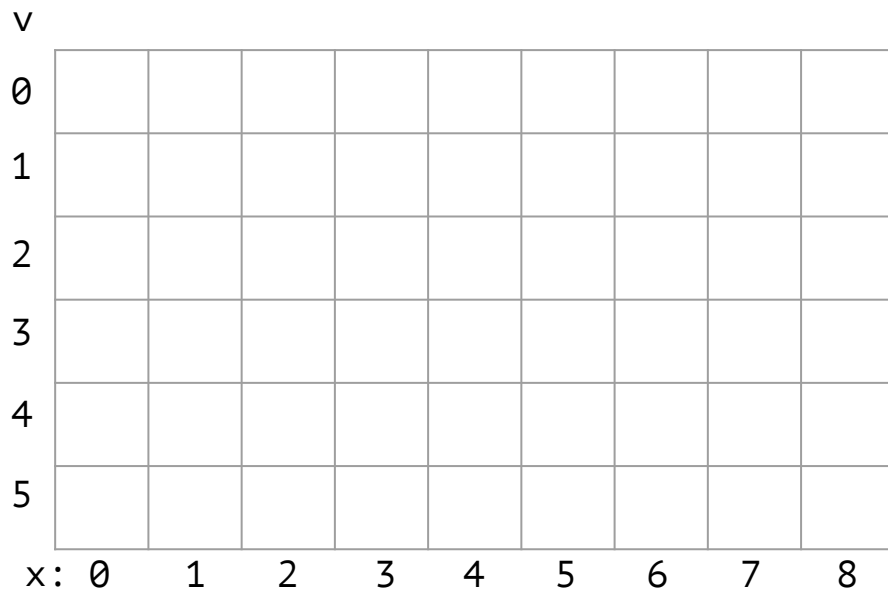


Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$




Решение через восходящий анализ

Раз у нас две шкалы подзадач, то теперь нам нужен двумерный массив A , в котором будем записывать результат.

Инициализация: $A[0][x] = 0$ для всех $x \in \{0, 1, 2, \dots, W\}$

до какого предмета
можно брать

размер рюкзака



Заполнение массива:

```
for i = 1, 2, 3, ..., n:
    for x = 0, 1, 2, ..., W:
        A[i][x] = max(A[i - 1][x], A[i - 1][x - wi] + vi)
```

(кроме случая, когда $w_i > x$,
там просто берем $A[i-1][x]$)

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1									
2									
3									
4									
5									
x:	0	1	2	3	4	5	6	7	8

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1									
2									
3									
4									
5									
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0								
2	0								
3	0								
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:  
  for x = 0, 1, 2, ..., W:  
    A[i][x] =  
      max(A[i - 1][x],  
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0					
2	0								
3	0								
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:  
  for x = 0, 1, 2, ..., W:  
    A[i][x] =  
      max(A[i - 1][x],  
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3				
2	0								
3	0								
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0								
3	0								
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1					
3	0								
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```


Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3				
3	0								
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:  
  for x = 0, 1, 2, ..., W:  
    A[i][x] =  
      max(A[i - 1][x],  
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3			
3	0								
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3			
3	0								
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3	4		
3	0								
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3	4	4	4
3	0								
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3	4	4	4
3	0	0	1	1	3				
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3	4	4	4
3	0	0	1	1	3	5	5	6	6
4	0								
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:  
  for x = 0, 1, 2, ..., W:  
    A[i][x] =  
      max(A[i - 1][x],  
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3	4	4	4
3	0	0	1	1	3	5	5	6	6
4	0	0	1	4	4	5	5	7	9
5	0								
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```


Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3	4	4	4
3	0	0	1	1	3	5	5	6	6
4	0	0	1	4	4	5	5	7	9
5	0	0	1	4	4	5	6	7	9
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3	4	4	4
3	0	0	1	1	3	5	5	6	6
4	0	0	1	4	4	5	5	7	9
5	0	0	1	4	4	5	6	7	9
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```



Это максимум, сколько
вы соберете, а какие
были предметы?

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v

0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3	4	4	4
3	0	0	1	1	3	5	5	6	6
4	0	0	1	4	4	5	5	7	9
5	0	0	1	4	4	5	6	7	9

x: 0 1 2 3 4 5 6 7 8



```
for i = 1, 2, 3, ..., n:  
  for x = 0, 1, 2, ..., W:  
    A[i][x] =  
      max(A[i - 1][x],  
          A[i - 1][x - wi] + vi)
```



Это максимум, сколько
вы соберете, а какие
были предметы?

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

✓
✗

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3	4	4	4
3	0	0	1	1	3	5	5	6	6
4	0	0	1	4	4	5	5	7	9
5	0	0	1	4	4	5	6	7	9
x:	0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:  
  for x = 0, 1, 2, ..., W:  
    A[i][x] =  
      max(A[i - 1][x],  
          A[i - 1][x - wi] + vi)
```

← Это максимум, сколько вы соберете, а какие были предметы?

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v									
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3	3	3
2	0	0	1	1	3	3	4	4	4
3	0	0	1	1	3	5	5	6	6
4	0	0	1	4	4	5	5	7	9
5	0	0	1	4	4	5	6	7	9
x:	0	1	2	3	4	5	6	7	8



```
for i = 1, 2, 3, ..., n:  
  for x = 0, 1, 2, ..., W:  
    A[i][x] =  
      max(A[i - 1][x],  
          A[i - 1][x - wi] + vi)
```



Это максимум, сколько
вы соберете, а какие
были предметы?

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v		0	1	2	3	4	5	6	7	8
0		0	0	0	0	0	0	0	0	0
1	✗	0	0	0	0	3	3	3	3	3
2	✗	0	0	1	1	3	3	4	4	4
3	✓	0	0	1	1	3	5	5	6	6
4	✓	0	0	1	4	4	5	5	7	9
5	✗	0	0	1	4	4	5	6	7	9
x:		0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```

← Это максимум, сколько вы соберете, а какие были предметы?

Пример

Пусть $W = 8$, $\{v_1 = 3, w_1 = 4\}$, $\{v_2 = 1, w_2 = 2\}$, $\{v_3 = 5, w_3 = 5\}$, $\{v_4 = 4, w_4 = 3\}$, $\{v_5 = 2, w_5 = 3\}$

v		0	1	2	3	4	5	6	7	8
0		0	0	0	0	0	0	0	0	0
1	✗	0	0	0	0	3	3	3	3	3
2	✗	0	0	1	1	3	3	4	4	4
3	✓	0	0	1	1	3	5	5	6	6
4	✓	0	0	1	4	4	5	5	7	9
5	✗	0	0	1	4	4	5	6	7	9
x:		0	1	2	3	4	5	6	7	8

```
for i = 1, 2, 3, ..., n:
  for x = 0, 1, 2, ..., W:
    A[i][x] =
      max(A[i - 1][x],
          A[i - 1][x - wi] + vi)
```

← Это максимум, сколько вы соберете, а какие были предметы?

Дискретная задача о рюкзаке

Дано: дан набор из n предметов, при этом:

1. v_i - ценность i -ого предмета (неотрицательная)
2. w_i - размер i -ого предмета (положительное целое!)
3. W - вместительность рюкзака (целое).

Найти: $S \in \{1, 2, \dots, n\}$ - подмножество предметов, такое что значение $\sum_{i \in S} v_i$ максимально, при этом $\sum_{i \in S} w_i \leq W$

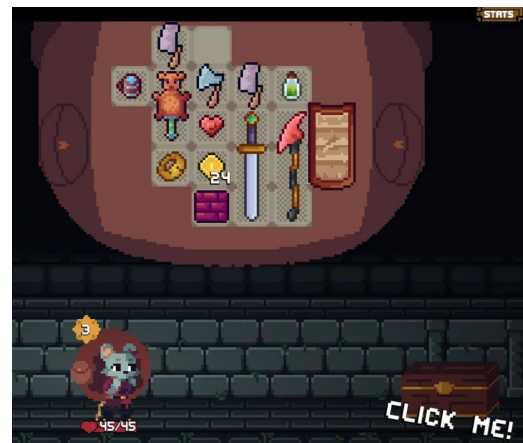


Непрерывная задача о рюкзаке

Дано: дан набор из n предметов, при этом:

1. v_i - ценность i -ого предмета (неотрицательная)
2. w_i - размер i -ого предмета (положительное ~~целое!~~)
3. W - вместительность рюкзака (~~целое~~).

Найти: $S \in \{1, 2, \dots, n\}$ - подмножество предметов, такое что значение $\sum_{i \in S} v_i$ максимально, при этом $\sum_{i \in S} w_i \leq W$



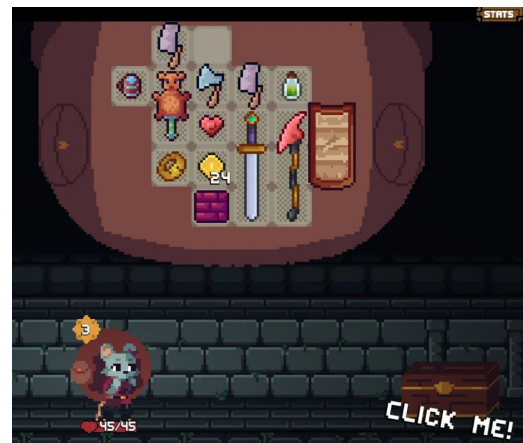
Непрерывная задача о рюкзаке

Дано: дан набор из n предметов, при этом:

1. v_i - ценность i -ого предмета (неотрицательная)
2. w_i - размер i -ого предмета (положительное ~~целое!~~)
3. W - вместительность рюкзака (~~целое~~).

И теперь можно брать **часть** предмета, не обязательно целиком.

Найти нужно такие пропорции каждого из предметов, чтобы максимизировать прибыль.



Непрерывная задача о рюкзаке

В непрерывной задаче о рюкзаке динамика не срабатывает, ведь нет четкого перехода на подзадачу.

Непрерывная задача о рюкзаке

В непрерывной задаче о рюкзаке динамика не сработает, ведь нет четкого перехода на подзадачу.

Сработает **жадный алгоритм**: сортируем все задачи по удельной стоимости (т.е. v_i/w_i), сначала заполняем рюкзак самым дорогим товаром, когда он кончится – берем дешевле и т.д.

Непрерывная задача о рюкзаке

В непрерывной задаче о рюкзаке динамика не работает, ведь нет четкого перехода на подзадачу.

Сработает **жадный алгоритм**: сортируем все задачи по удельной стоимости (т.е. v_i/w_i), сначала заполняем рюкзак самым дорогим товаром, когда он кончится – берем дешевле и т.д.

Упражнение #1: доказать корректность жадного алгоритма.

Непрерывная задача о рюкзаке

В непрерывной задаче о рюкзаке динамика не работает, ведь нет четкого перехода на подзадачу.

Сработает **жадный алгоритм**: сортируем все задачи по удельной стоимости (т.е. v_i/w_i), сначала заполняем рюкзак самым дорогим товаром, когда он кончится – берем дешевле и т.д.

Упражнение #1: доказать корректность жадного алгоритма.

Упражнение #2: доказать, что такой алгоритм не работает с дискретной задачей о рюкзаке.

Задача о порядке перемножения матриц

Пусть есть набор матриц **совместных** размеров: $A_1, A_2, A_3, \dots, A_n$

Задача о порядке перемножения матриц

Пусть есть набор матриц **совместных** размеров: $A_1, A_2, A_3, \dots, A_n$

При вычислении их **произведения** $A_1 A_2 A_3 \dots A_n$ можем поставить скобки в разных местах: $(A_1(A_2(A_3 \dots A_n) \dots)))$

$((A_1 A_2)(A_3 \dots A_n) \dots))$

$((((A_1 A_2) A_3) \dots A_n) \dots)$

.....

Задача о порядке перемножения матриц

Пусть есть набор матриц **совместных** размеров: $A_1, A_2, A_3, \dots, A_n$

При вычислении их **произведения** $A_1 A_2 A_3 \dots A_n$ можем поставить скобки в разных местах: $(A_1(A_2(A_3 \dots A_n) \dots)))$

$((A_1 A_2)(A_3 \dots A_n) \dots))$

$((((A_1 A_2) A_3) \dots A_n) \dots)$

.....

На что влияет расстановка скобок?

Задача о порядке перемножения матриц

Пусть есть набор матриц **совместных** размеров: $A_1, A_2, A_3, \dots, A_n$

При вычислении их **произведения** $A_1 A_2 A_3 \dots A_n$ можем поставить скобки в разных местах: $(A_1(A_2(A_3 \dots A_n) \dots)))$

$((A_1 A_2)(A_3 \dots A_n) \dots))$

$((((A_1 A_2) A_3) \dots A_n) \dots)$

.....

На что влияет расстановка скобок?

Точно не на корректность, операция ассоциативна.

Задача о порядке перемножения матриц

Пусть есть набор матриц **совместных** размеров: $A_1, A_2, A_3, \dots, A_n$

При вычислении их **произведения** $A_1 A_2 A_3 \dots A_n$ можем поставить скобки в разных местах: $(A_1(A_2(A_3 \dots A_n) \dots)))$

$((A_1 A_2)(A_3 \dots A_n) \dots))$

$((((A_1 A_2) A_3) \dots A_n) \dots)$

.....

На что влияет расстановка скобок?

Точно не на корректность, операция ассоциативна.

На время работы!

Задача о порядке перемножения матриц

Пусть есть A - матрица размера $p \times q$, а B - матрица размера $q \times r$.
 $A * B = C$ - матрица размера $p \times r$.

Количество скалярных операций при таком перемножении будет $p * q * r$

Задача о порядке перемножения матриц

Пусть есть A - матрица размера $p \times q$, а B - матрица размера $q \times r$.
 $A * B = C$ - матрица размера $p \times r$.

Количество скалярных операций при таком перемножении будет $p * q * r$

Пример: A_1 - матрица размера 10×100 ,
 A_2 - матрица размера 100×5 ,
 A_3 - матрица размера 5×50 .

Вычисляем $(A_1 A_2) A_3$ за $(10 * 100 * 5) + (10 * 5 * 50) = 7500$ операций

Задача о порядке перемножения матриц

Пусть есть A - матрица размера $p \times q$, а B - матрица размера $q \times r$.
 $A * B = C$ - матрица размера $p \times r$.

Количество скалярных операций при таком перемножении будет $p * q * r$

Пример: A_1 - матрица размера 10×100 ,
 A_2 - матрица размера 100×5 ,
 A_3 - матрица размера 5×50 .

Вычисляем $(A_1 A_2) A_3$ за $(10 * 100 * 5) + (10 * 5 * 50) = 7500$ операций,
 $A_1 (A_2 A_3)$ за $(100 * 5 * 50) + (10 * 100 * 50) = 75000$ операций.

Задача о порядке перемножения матриц

Пусть есть A - матрица размера $p \times q$, а B - матрица размера $q \times r$.
 $A * B = C$ - матрица размера $p \times r$.

Количество скалярных операций при таком перемножении будет $p * q * r$

Пример: A_1 - матрица размера 10×100 ,
 A_2 - матрица размера 100×5 ,
 A_3 - матрица размера 5×50 .

Вычисляем $(A_1 A_2) A_3$ за $(10 * 100 * 5) + (10 * 5 * 50) = 7500$ операций,
 $A_1 (A_2 A_3)$ за $(100 * 5 * 50) + (10 * 100 * 50) = 75000$ операций.

Т.е. большая разница, в каком порядке перемножать матрицы!

Задача о порядке перемножения матриц

Пусть есть набор матриц **совместных** размеров: $A_1, A_2, A_3, \dots, A_n$

При вычислении их **произведения** $A_1 A_2 A_3 \dots A_n$ можем поставить скобки в разных местах: $(A_1(A_2(A_3 \dots A_n) \dots)))$

$((A_1 A_2)(A_3 \dots A_n) \dots))$

$((((A_1 A_2) A_3) \dots A_n) \dots)$

.....

Задача: выбрать расстановку скобок, минимизирующую общее количество скалярных операций при перемножении.

Задача о порядке перемножения матриц

Пусть есть набор матриц **совместных** размеров: $A_1, A_2, A_3, \dots, A_n$

При вычислении их **произведения** $A_1 A_2 A_3 \dots A_n$ можем поставить скобки в разных местах: $(A_1(A_2(A_3 \dots A_n) \dots)))$

$((A_1 A_2)(A_3 \dots A_n) \dots))$

$((((A_1 A_2) A_3) \dots A_n) \dots)$

.....

Задача: выбрать расстановку скобок, минимизирующую общее количество скалярных операций при перемножении.

Если решать простым перебором, получим **экспоненциальную** сложность (прямолинейно доказывается по индукции).

Оптимальная подструктура

Обозначим произведение $A_i A_{i+1} \dots A_j$, как $A_{i..j}$ (здесь $i \leq j$).

Оптимальная подструктура

Обозначим произведение $A_i A_{i+1} \dots A_j$, как $A_{i..j}$ (здесь $i \leq j$).

Если задача не является **тривиальной**, т.е. $i < j$, то любая расстановка скобок разобьет $A_{i..j}$ на перемножение двух матриц: A_k и A_{k+1} , где $i \leq k < j$.

Оптимальная подструктура

Обозначим произведение $A_i A_{i+1} \dots A_j$, как $A_{i..j}$ (здесь $i \leq j$).

Если задача не является **тривиальной**, т.е. $i < j$, то любая расстановка скобок разобьет $A_{i..j}$ на перемножение двух матриц: A_k и A_{k+1} , где $i \leq k < j$.

Т.е. сначала будет вычислено произведение $A_{i..k}$, затем $A_{k+1..j}$, после чего получившиеся матрицы перемножаются.

Оптимальная подструктура

Обозначим произведение $A_i A_{i+1} \dots A_j$, как $A_{i..j}$ (здесь $i \leq j$).

Если задача не является **тривиальной**, т.е. $i < j$, то любая расстановка скобок разобьет $A_{i..j}$ на перемножение двух матриц: A_k и A_{k+1} , где $i \leq k < j$.

Т.е. сначала будет вычислено произведение $A_{i..k}$, затем $A_{k+1..j}$, после чего получившиеся матрицы перемножаются.

Суммарное количество скалярных операций будет суммой операций при вычислении $A_{i..k}$ и $A_{k+1..j}$ + скалярные операции из их перемножения.

Оптимальная подструктура

Пусть теперь есть **оптимальная** расстановка скобок в $A_i A_{i+1} \dots A_j$,
приводящая к перемножению матриц $A_{i..k}$ и $A_{k+1..j}$

Оптимальная подструктура

Пусть теперь есть **оптимальная** расстановка скобок в $A_i A_{i+1} \dots A_j$, приводящая к перемножению матриц $A_{i..k}$ и $A_{k+1..j}$

Тогда в последовательности $A_i * A_{i+1} * \dots A_k$ скобки тоже расставлены **оптимально**.

Оптимальная подструктура

Пусть теперь есть **оптимальная** расстановка скобок в $A_i A_{i+1} \dots A_j$, приводящая к перемножению матриц $A_{i..k}$ и $A_{k+1..j}$

Тогда в последовательности $A_i * A_{i+1} * \dots A_k$ скобки тоже расставлены **оптимально** (если бы была более оптимальная расстановка, ее бы и взяли в общем оптимальном решении).

Оптимальная подструктура

Пусть теперь есть **оптимальная** расстановка скобок в $A_i A_{i+1} \dots A_j$, приводящая к перемножению матриц $A_{i..k}$ и $A_{k+1..j}$

Тогда в последовательности $A_i * A_{i+1} * \dots A_k$ скобки тоже расставлены **оптимально** (если бы была более оптимальная расстановка, ее бы и взяли в общем оптимальном решении).

Аналогичное верно и для $A_{k+1} * A_{k+2} * \dots A_j$.

Оптимальная подструктура

Пусть теперь есть **оптимальная** расстановка скобок в $A_i A_{i+1} \dots A_j$, приводящая к перемножению матриц $A_{i..k}$ и $A_{k+1..j}$

Тогда в последовательности $A_i * A_{i+1} * \dots A_k$ скобки тоже расставлены **оптимально** (если бы была более оптимальная расстановка, ее бы и взяли в общем оптимальном решении).

Аналогичное верно и для $A_{k+1} * A_{k+2} * \dots A_j$.

Из этого факта получим **рекуррентное соотношение**: нужно найти наиболее эффективное разбиение, финальным ответом будет сумма скалярных величин из префикса и суффикса (и плюс скалярные операции из их **перемножения**).

Оптимальная подструктура

Обозначим $m[i, j]$ - минимальное количество скалярных операций при перемножении последовательности $A_i A_{i+1} \dots A_j$, где $1 \leq i \leq j \leq n$.

Ответом на всю задачу будет $m[1, n]$.

Оптимальная подструктура

Обозначим $m[i, j]$ - минимальное количество скалярных операций при перемножении последовательности $A_i A_{i+1} \dots A_j$, где $1 \leq i \leq j \leq n$.

Ответом на всю задачу будет $m[1, n]$.

Случай $i = j$ - тривиален. Имеем одну матрицу \Rightarrow перемножать ничего не нужно $\Rightarrow m[i, i] = 0$.

Оптимальная подструктура

Обозначим $m[i, j]$ - минимальное количество скалярных операций при перемножении последовательности $A_i A_{i+1} \dots A_j$, где $1 \leq i \leq j \leq n$.

Ответом на всю задачу будет $m[1, n]$.

Случай $i = j$ - тривиален. Имеем одну матрицу \Rightarrow перемножать ничего не нужно $\Rightarrow m[i, i] = 0$.

Обозначим размерности каждой матрицы A_i через $p_{i-1} \times p_i$.

Оптимальная подструктура

Обозначим $m[i, j]$ - минимальное количество скалярных операций при перемножении последовательности $A_i A_{i+1} \dots A_j$, где $1 \leq i \leq j \leq n$.

Ответом на всю задачу будет $m[1, n]$.

Случай $i = j$ - тривиален. Имеем одну матрицу \Rightarrow перемножать ничего не нужно $\Rightarrow m[i, i] = 0$.

Обозначим размерности каждой матрицы A_i через $p_{i-1} \times p_i$.

Если знаем структуру оптимального разбиения (т.е. знаем k), то имеем: $m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$.

Оптимальная подструктура

Обозначим $m[i, j]$ - минимальное количество скалярных операций при перемножении последовательности $A_i A_{i+1} \dots A_j$, где $1 \leq i \leq j \leq n$.

Ответом на всю задачу будет $m[1, n]$.

Случай $i = j$ - тривиален. Имеем одну матрицу \Rightarrow перемножать ничего не нужно $\Rightarrow m[i, i] = 0$.

Обозначим размерности каждой матрицы A_i через $p_{i-1} \times p_i$.

Если знаем структуру оптимального разбиения (т.е. знаем k), то имеем: $m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$.

Но мы не знаем! Поэтому, давайте перебирать все варианты.

Оптимальная подструктура

Тогда получаем:
$$m[i, j] = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & i < j \end{cases}$$

Оптимальная подструктура

Тогда получаем:
$$m[i, j] = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & i < j \end{cases}$$

Если будем вычислять в лоб, то снова получим экспоненциальную сложность. Поэтому опять нужен восходящий анализ.

Восходящий анализ

Тогда получаем:
$$m[i, j] = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & i < j \end{cases}$$

Если будем вычислять в лоб, то снова получим экспоненциальную сложность. Поэтому опять нужен восходящий анализ.

Что важно: можем найти $m[i, j]$ зная только m **от меньших промежутков**. Поэтому снова заполняем таблицу.

Восходящий анализ

$$\text{Тогда получаем: } m[i, j] = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & i < j \end{cases}$$

Если будем вычислять в лоб, то снова получим экспоненциальную сложность. Поэтому опять нужен восходящий анализ.

Что важно: можем найти $m[i, j]$ зная только m **от меньших промежутков**. Поэтому снова заполняем таблицу.

Дополнительно заведем таблицу s , куда будем записывать индекс k , при котором получилась оптимальная расстановка.

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]:  
    for i in [1, n - l + 1]:  
        j = i + l - 1  
        m[i, j] =  $\infty$ 
```

← длина подпоследовательности

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]:  
    for i in [1, n - l + 1]:  
        j = i + l - 1  
        m[i, j] =  $\infty$   
  
        for k in [i, j - 1]:  
            q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$ 
```

← длина подпоследовательности

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]: ←————— длина подпоследовательности  
    for i in [1, n - l + 1]:  
        j = i + l - 1  
        m[i, j] =  $\infty$ 
```

```
        for k in [i, j - 1]:  
            q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$ 
```

```
            if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```

Пример

$$A_1 \quad 30 \times 35$$

$$A_2 \quad 35 \times 15$$

$$A_3 \quad 15 \times 5$$

$$A_4 \quad 5 \times 10$$

$$A_5 \quad 10 \times 20$$

$$A_6 \quad 20 \times 25$$

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m						
	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]:  
    for i in [1, n - l + 1]:  
        j = i + l - 1  
        m[i, j] =  $\infty$ 
```

← длина подпоследовательности

```
    for k in [i, j - 1]:  
        q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$ 
```

```
    if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m						
	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```


```
for l in [2, n]: ←————— длина подпоследовательности  
    for i in [1, n - l + 1]:  
        j = i + l - 1  
        m[i, j] =  $\infty$ 
```

```
        for k in [i, j - 1]:  
            q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$ 
```

```
            if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]:  l = 2  
    for i in [1, n - l + 1]:  
        j = i + l - 1  
        m[i, j] =  $\infty$ 
```

```
        for k in [i, j - 1]:  
            q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$ 
```

```
            if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]: ← l = 2  
    for i in [1, n - l + 1]: i in [1, 5]  
        j = i + l - 1       j = i + 1  
        m[i, j] =  $\infty$ 
```

```
    for k in [i, j - 1]:  
        q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$ 
```

```
    if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m	1	2	3	4	5	6
	0					
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]: ← l = 2  
    for i in [1, n - l + 1]: i in [1, 5]  
        j = i + l - 1        j = i + 1  
        m[i, j] =  $\infty$ 
```

```
for k in [i, j - 1]: ← k = i  
    q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$ 
```

```
if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```


Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]: ← l = 2  
    for i in [1, n - l + 1]: i in [1, 5]  
        j = i + l - 1        j = i + 1  
        m[i, j] = ∞
```

```
for k in [i, j - 1]: ← k = i  
    q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$  q = m[i, i] +  
                                     m[j, j] + ...  
    if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]: ← l = 2  
    for i in [1, n - l + 1]: i in [1, 5]  
        j = i + l - 1        j = i + 1  
        m[i, j] = ∞
```

```
for k in [i, j - 1]: ← k = i  
    q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$     q =  $p_{i-1} * p_i * p_{i+1}$ 
```

```
if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```

Пример

$$A_1 \quad 30 \times 35$$

$$A_2 \quad 35 \times 15$$

$$A_3 \quad 15 \times 5$$

$$A_4 \quad 5 \times 10$$

$$A_5 \quad 10 \times 20$$

$$A_6 \quad 20 \times 25$$

m	1	2	3	4	5	6
1	0	15 750				
2		0				
3			0			
4				0		
5					0	
6						0

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m		1	2	3	4	5	6
1	0	15 750					
2		0	2625				
3			0				
4				0			
5					0		
6							0

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m	1	2	3	4	5	6
1	0	15 750				
2		0	2625			
3			0	750		
4				0		
5					0	
6						0

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m		1	2	3	4	5	6
	1	0	15 750				
	2		0	2625			
	3			0	750		
	4				0	1000	
	5					0	500
	6						0

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]: ← l = 2  
    for i in [1, n - l + 1]: i in [1, 5]  
        j = i + l - 1        j = i + 1  
        m[i, j] = ∞
```

```
for k in [i, j - 1]: ← k = i  
    q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$     q =  $p_{i-1} * p_i * p_{i+1}$ 
```

```
if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]: ← l = 3  
    for i in [1, n - l + 1]: i in [1, 4]  
        j = i + l - 1        j = i + 2  
        m[i, j] =  $\infty$ 
```

```
    for k in [i, j - 1]:  
        q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$ 
```

```
    if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```


Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m		1	2	3	4	5	6
1	0	15 750					
2		0	2625				
3			0	750			
4				0	1000		
5					0	500	
6							0

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]: ← l = 3  
    for i in [1, n - l + 1]: i in [1, 4]  
        j = i + l - 1        j = i + 2  
        m[i, j] =  $\infty$ 
```

```
    for k in [i, j - 1]:  
        q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$ 
```

```
    if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]: ← l = 3  
    for i in [1, n - l + 1]: i in [1, 4]  
        j = i + l - 1       j = i + 2  
        m[i, j] = ∞
```

```
    for k in [i, j - 1]: ← k in [i, i + 1]  
        q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$ 
```

```
    if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m		1	2	3	4	5	6
1	0	15 750					
2		0	2625				
3			0	750			
4				0	1000		
5					0	500	
6							0

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m		1	2	3	4	5	6
1	0	15 750					
2		0	2625				
3			0	750			
4				0	1000		
5					0	500	
6							0

$$A_1(A_2A_3) \Rightarrow m[1, 1] + m[2, 3] + (30 \times 35) \times 5 = 0 + 2625 + 5250 = 7875$$

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

		m					
		1	2	3	4	5	6
1	0	15 750					
2		0	2625				
3			0	750			
4				0	1000		
5					0	500	
6							0

$$A_1(A_2A_3) \Rightarrow m[1, 1] + m[2, 3] + (30 \times 35) \times 5 = 0 + 2625 + 5250 = 7875$$

$$(A_1A_2)A_3 \Rightarrow m[1, 2] + m[3, 3] + (30 \times 15) \times 5 = 15750 + 0 + 2250 = 18000$$

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

		m					
		1	2	3	4	5	6
1		0	15 750	7875			
2			0	2625			
3				0	750		
4					0	1000	
5						0	500
6							0

$$A_1(A_2A_3) \Rightarrow m[1, 1] + m[2, 3] + (30 \times 35) \times 5 = 0 + 2625 + 5250 = 7875$$

$$(A_1A_2)A_3 \Rightarrow m[1, 2] + m[3, 3] + (30 \times 15) \times 5 = 15750 + 0 + 2250 = 18000$$



Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

		1	2	3	4	5	6
1		0	15 750	7875			
2			0	2625			
3				0	750		
4					0	1000	
5						0	500
6							0

Т.е. чтобы найти значение в ячейке на текущей активной диагонали, перебираем варианты из треугольника под ним

$$A_1(A_2A_3) \Rightarrow m[1,1] + m[2,3] + (30 \times 35) \times 5 = 0 + 2625 + 5250 = 7875$$

$$(A_1A_2)A_3 \Rightarrow m[1,2] + m[3,3] + (30 \times 15) \times 5 = 15750 + 0 + 2250 = 18000$$



Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m	1	2	3	4	5	6
1	0	15 750	7875			
2		0	2625	4375		
3			0	750		
4				0	1000	
5					0	500
6						0

Т.е. чтобы найти значение в ячейке на текущей активной диагонали, перебираем варианты из треугольника под ним

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m	1	2	3	4	5	6
1	0	15 750	7875			
2		0	2625	4375		
3			0	750	2500	
4				0	1000	
5					0	500
6						0

Т.е. чтобы найти значение в ячейке на текущей активной диагонали, перебираем варианты из треугольника под ним

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m	1	2	3	4	5	6
1	0	15 750	7875			
2		0	2625	4375		
3			0	750	2500	
4				0	1000	3500
5					0	500
6						0

Т.е. чтобы найти значение в ячейке на текущей активной диагонали, перебираем варианты из треугольника под ним

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m	1	2	3	4	5	6
1	0	15 750	7875	9375		
2		0	2625	4375		
3			0	750	2500	
4				0	1000	3500
5					0	500
6						0

Т.е. чтобы найти значение в ячейке на текущей активной диагонали, перебираем варианты из треугольника под ним

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m	1	2	3	4	5	6
1	0	15 750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	500
6						0

Т.е. чтобы найти значение в ячейке на текущей активной диагонали, перебираем варианты из треугольника под ним

В конце концов заполняем всю верхнетреугольную матрицу

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m		1	2	3	4	5	6
1		0	15 750	7875	9375	11875	15125
2			0	2625	4375	7125	10500
3				0	750	2500	5375
4					0	1000	3500
5						0	500
6							0

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m		1	2	3	4	5	6
1		0	15 750	7875	9375	11875	15125
2			0	2625	4375	7125	10500
3				0	750	2500	5375
4					0	1000	3500
5						0	500
6							0

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m		1	2	3	4	5	6
1		0	15 750	7875	9375	11875	15125
2			0	2625	4375	7125	10500
3				0	750	2500	5375
4					0	1000	3500
5						0	500
6							0

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m		1	2	3	4	5	6
1		0	15 750	7875	9375	11875	15125
2			0	2625	4375	7125	10500
3				0	750	2500	5375
4					0	1000	3500
5						0	500
6							0

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m		1	2	3	4	5	6
1		0	15 750	7875	9375	11875	15125
2			0	2625	4375	7125	10500
3				0	750	2500	5375
4					0	1000	3500
5						0	500
6							0

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

		m					
		1	2	3	4	5	6
1		0	15 750	7875	9375	11875	15125
2			0	2625	4375	7125	10500
3				0	750	2500	5375
4					0	1000	3500
5						0	500
6							0

Аналогично прошлым задачам заполняется матрица S , в которой фиксируется выбор, какое k выбирали.

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

		m					
		1	2	3	4	5	6
1		0	15 750	7875	9375	11875	15125
2			0	2625	4375	7125	10500
3				0	750	2500	5375
4					0	1000	3500
5						0	500
6							0

Аналогично прошлым задачам заполняется матрица S , в которой фиксируется выбор, какое k выбирали.

Сложность алгоритма?

Восходящий анализ

```
for i in [1, n]:  
    m[i, i] = 0
```

```
for l in [2, n]: ←————— длина подпоследовательности  
    for i in [1, n - l + 1]:  
        j = i + l - 1  
        m[i, j] =  $\infty$ 
```

```
        for k in [i, j - 1]:  
            q = m[i, k] + m[k + 1, j] +  $p_{i-1}p_kp_j$ 
```

```
            if q < m[i, j]: { m[i, j] = q, s[i, j] = k }
```

Пример

A_1 30 × 35

A_2 35 × 15

A_3 15 × 5

A_4 5 × 10

A_5 10 × 20

A_6 20 × 25

m	1	2	3	4	5	6
1	0	15 750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	500
6						0

Аналогично прошлым задачам заполняется матрица S , в которой фиксируется выбор, какое k выбирали.

Сложность алгоритма?

$O(n^3)$



Мини-задача #37 (1 балл)

Найти количество уникальных (по форме) BST. Используйте алгоритм динамического программирования.

<https://leetcode.com/problems/unique-binary-search-trees>

Мини-задача #38 (1 балла)

Помогите рыцарю выйти из подземелья!

<https://leetcode.com/problems/dungeon-game>

Takeaways

- Новый инструмент решения задач: **динамическое программирование!**
- Ищите оптимальную подструктуру.
- Выписывайте рекуррентное соотношение.
- Убеждайтесь, что подзадач мало, после чего используйте **восходящий анализ**.