Структуры и алгоритмы

NP-полные задачи

Полный перебор

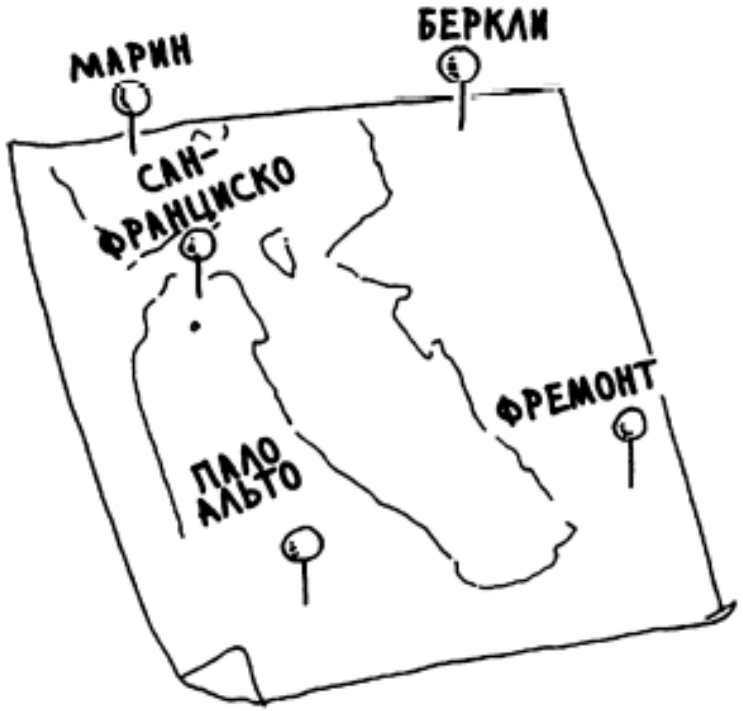
Жадные алгоритмы

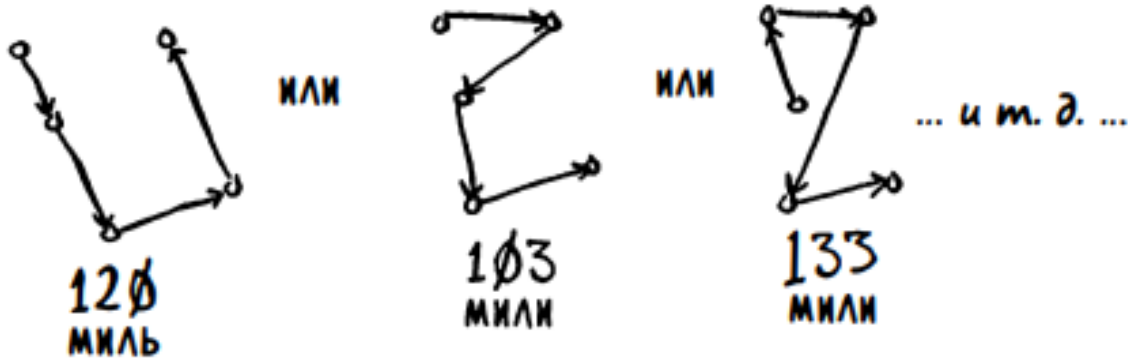
Динамическое программирование

Задача о коммивояжере

Коммивояжер хочет побывать в каждом из 5 городов так, чтобы при этом проехать минимальное общее расстояние.

Сколько маршрутов необходимо вычислить для пяти городов?



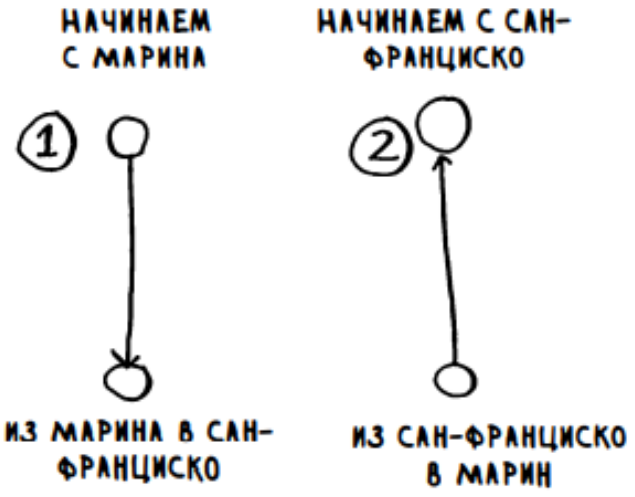


Задача о коммивояжере - шаг за шагом

Допустим, городов всего два. Выбирать приходится всего из двух маршрутов.

В задаче о коммивояжере существует ли конкретный город, с которого нужно начинать?

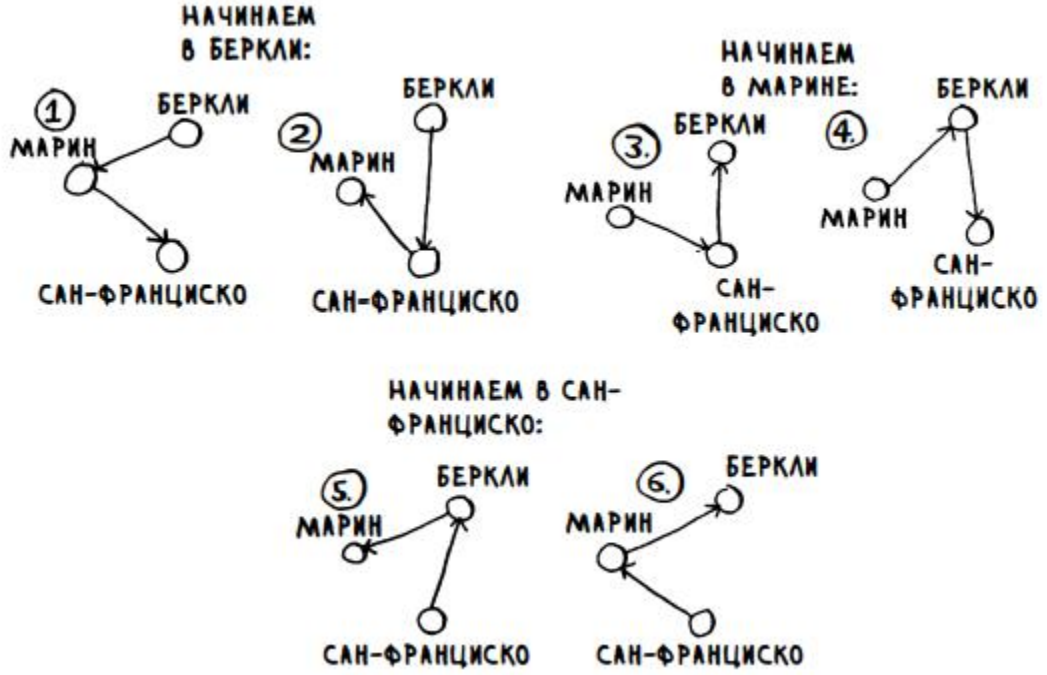
Расстояние от СФ до Марин может не совпадать с расстоянием от Марин до СФ.



Задача о коммивояжере - шаг за шагом

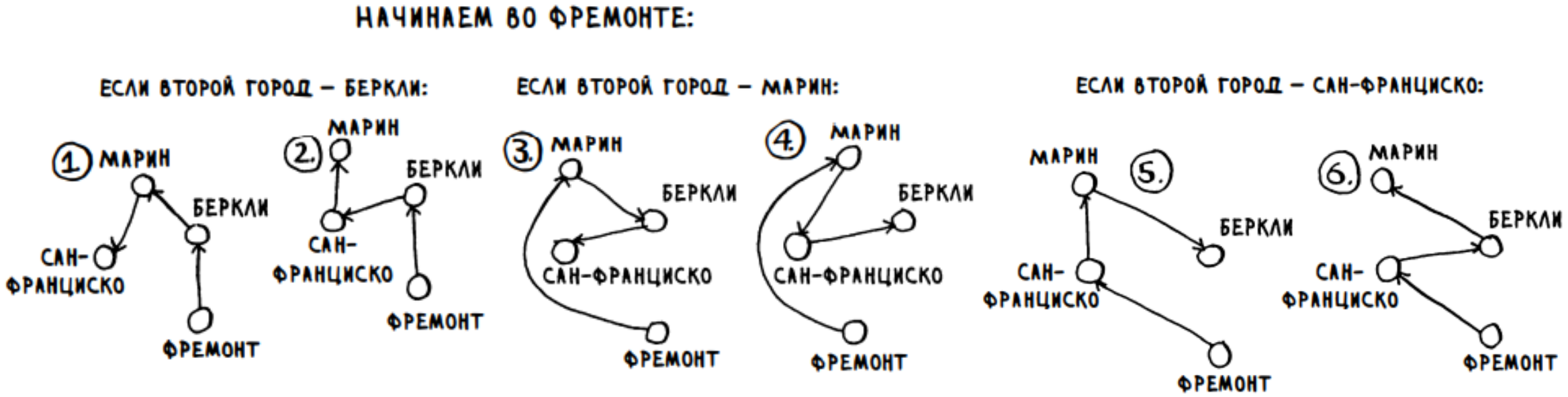
Теперь добавим к двум городам еще один. Сколько возможных маршрутов существует в этой конфигурации?

Всего шесть возможных маршрутов: по два для каждого города, с которого вы можете начать.



Задача о коммивояжере - шаг за шагом

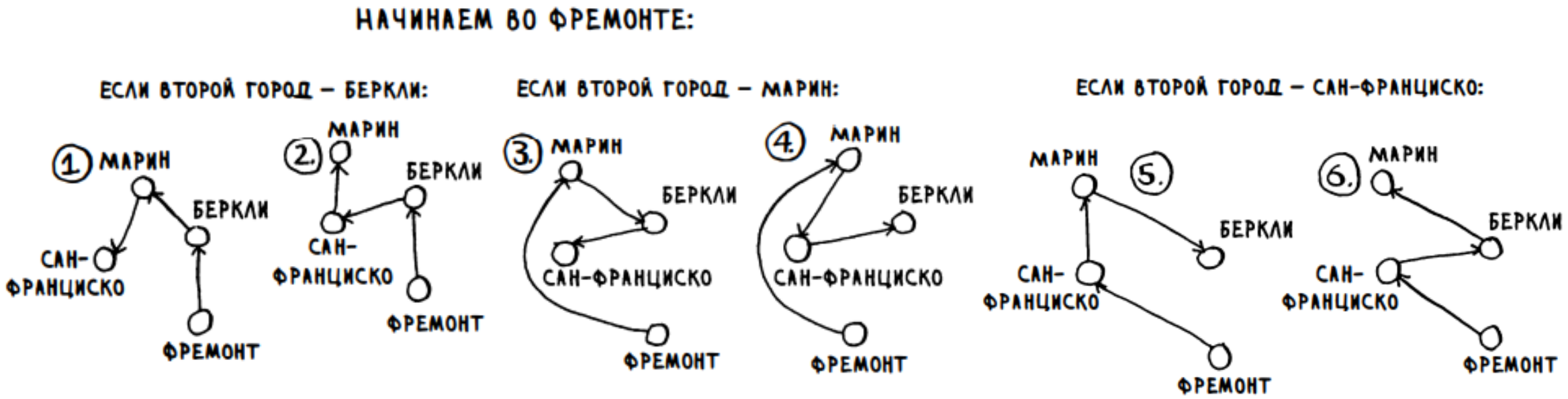
Добавим еще один город - Фремонт. Теперь допустим, что вы начали с Фремонта.



Задача о коммивояжере - шаг за шагом

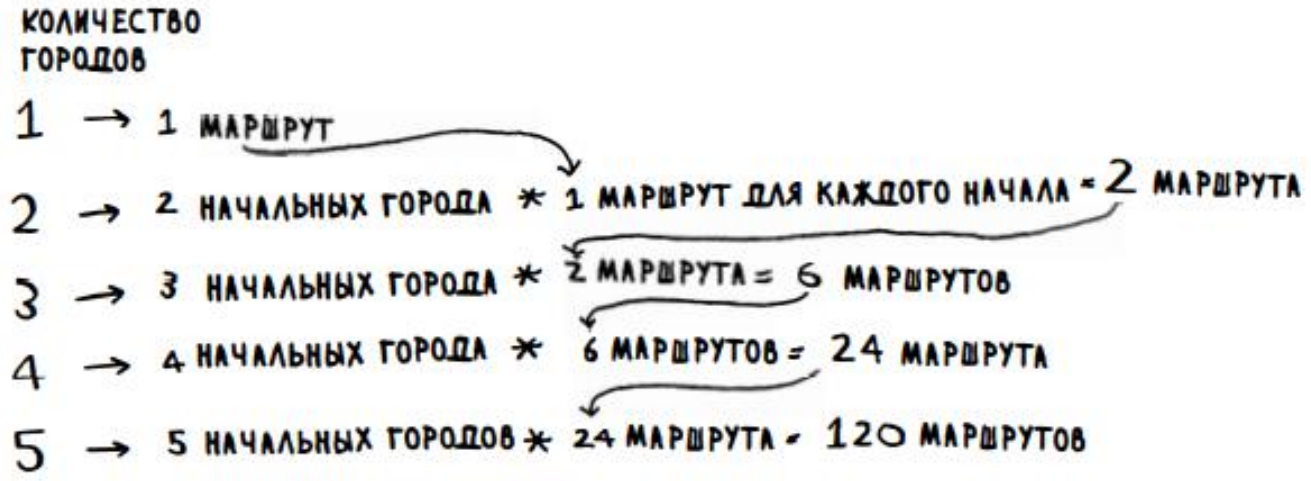
Таким образом, для каждого города есть 6 возможных вариантов.

Тогда общее количество 6 * 4 = 24 = 4!



Задача о коммивояжере - закономерность

Каждый раз, когда вы добавляете новый город, следующим образом увеличивается количество вычисляемых маршрутов.



Имеем сложность алгоритма O(n!).

ГОРОДА	ОПЕРАЦИИ
6	720
7	5040
8	40320
...	...
15	1,307,674,368,000
...	...
30	265,252,859,812,191,058,636,308,480,000,000

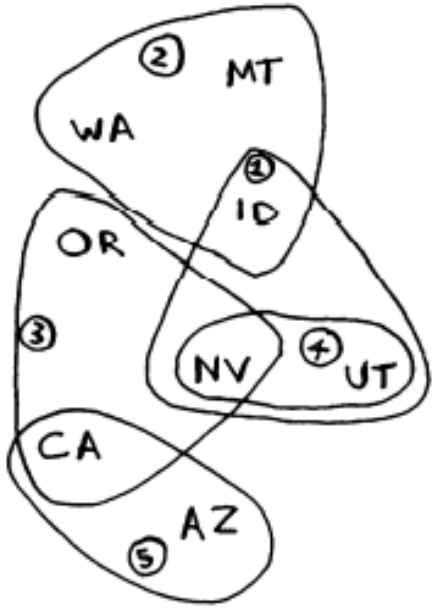
Количество операций стремительно растет

Задача о покрытии множества

Вы открываете собственную авторскую программу на радио и хотите, чтобы вас слушали во всех 50 штатах. Нужно решить, на каких радиостанциях должна транслироваться ваша передача. Каждая станция стоит денег, поэтому количество станции необходимо свести к минимуму. Имеется список станций и какие штаты они покрывают.

РАДИО-СТАНЦИЯ	ДОСТУПНА В ШТАТАХ
KONE	ID,NV,UT
KTWO	WA, ID, MT
KTHREE	OR, NV, CA
KFOUR	NV, UT
KFIVE	CA, AZ

... и т. д. ...



Задача о покрытии множества

Как найти минимальный набор станций, который бы покрывал все 50 штатов?

1. Составить список всех возможных подмножеств станций
2. Из этого списка выбирается множество с наименьшим набором станций, покрывающих все 50 штатов

Задача о покрытии множества

Сколько множеств необходимо перебрать?

1 станция: {}, {s1} – 2 варианта

Задача о покрытии множества

Сколько множеств необходимо перебрать?

1 станция: {}, {s1} – 2 варианта

2 станции: {}, {s1}, {s2}, {s1, s2} – 4 варианта

Задача о покрытии множества

Сколько множеств необходимо перебрать?

1 станция: {}, {s1} – 2 варианта

2 станции: {}, {s1}, {s2}, {s1, s2} – 4 варианта

3 станции: {}, {s1}, {s2}, {s1, s2},

{s3}, {s1, s3}, {s2, s3}, {s1, s2, s3} – 8 вариантов

Задача о покрытии множества

Сколько множеств необходимо перебрать?

1 станция: {}, {s1} – 2 варианта

2 станции: {}, {s1}, {s2}, {s1, s2} – 4 варианта

3 станции: {}, {s1}, {s2}, {s1, s2},

{s3}, {s1, s3}, {s2, s3}, {s1, s2, s3} – 8 вариантов

…

n станций: 2^n вариантов

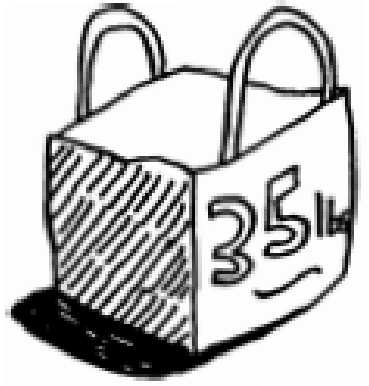
Имеем сложность алгоритма $O(2^n)$.

КОЛИЧЕСТВО СТАНЦИЙ	НЕОБХОДИМОЕ ВРЕМЯ
5	3.2 с
10	1024 с
32	13.6 года
100	4×10^{21} года

Задача о рюкзаке

Вор забрался в магазин с рюкзаком, и перед ним множество товаров, которые можно украсть. Однако емкость рюкзака не бесконечна: он выдержит не более 35 фунтов.

Требуется подобрать набор товаров максимальной стоимости, которые вместятся в рюкзак.



МАГНИТОФОН

$3ØØØ

3Ø ФУНТОВ

НОУТБУК

$2ØØØ

2Ø ФУНТОВ

ГИТАРА

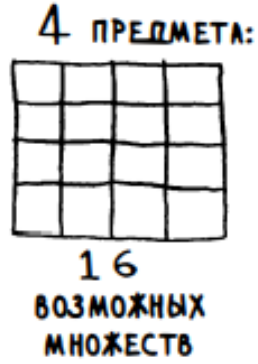
$15ØØ

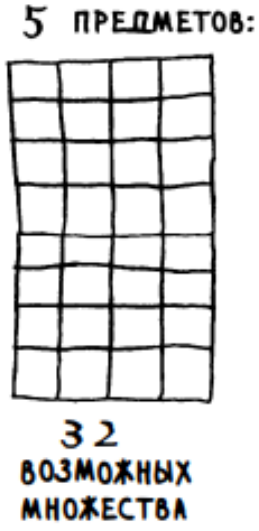
15ФУНТОВ

Задача о рюкзаке

Перебираем все возможные множества товаров и находим множество с максимальном стоимостью.

Имеем сложность алгоритма $O(2^n)$.





Классы задач P и NP

Все эти задачи объединяет то, что их невозможно решить за разумное время.

Такие задачи образуют **класс NP** – недетерминированной полиномиальной сложности. Сложность всех известных детерминированных алгоритмов, решающих эти задачи, либо экспоненциальна ($O(2^n)$), либо факториальна ($O(n!)$).

Задачи, которые мы рассматривали на прошлых парах (сортировка, хеширование, поиск кратчайшего пути и т.д.) относятся к **классу P** — классу задач полиномиальной сложности. Такие задачи называются также практически разрешимыми. Такие задачи возможно решить за время $O(n^k)$ или быстрее.

NP-полные задачи

Задача является **NP-полной**, если она входит в класс NP и к ней можно свести любую другую задачу этого класса за полиномиальное время.

Термин **NP-полная** относится к самым сложным задачам в классе NP. Эти задачи выделены тем, что если нам все-таки удастся найти полиномиальный алгоритм решения какой-либо из них, то это будет означать, что все задачи класса NP допускают полиномиальные алгоритмы решения.

На данный момент не найдено полиномиального алгоритма для решения данных задач.

NP-полные задачи

Всего было найдено более 3000 NP-полных задач.

У NP-полных задач нет известного решения, нахождение которого занимает полиномиальное время, поэтому все они считаются крайне сложными.

Какие-то задачи можно решить только в небольшом масштабе.

Далее приведем еще несколько примеров таких задач. Поскольку все эти задачи являются NP-полными, каждую из них всегда можно свести к другой.

NP-полные задачи

Задача о картинной галерее. Зная план комнат и коридоров в картинной галерее, нужно найти минимальное количество охранников, необходимых для присмотра за картинами.

Задача о клике. Имеется задача поиска максимальной клики в множестве людей (кликой называется множество людей, каждый из которых знаком со всеми остальными).

Задача раскраски карты. Вы рисуете карту мира, на которой две соседние страны не могут быть окрашены в одинаковый цвет. Требуется найти минимальное количество цветов, при котором любые две соседние страны будут окрашены в разные цвета.

NP-полные задачи

Задача планирования заданий. Имея N заданий разного объема и M идентичных машин, нужно распланировать работу таким образом, чтобы минимизировать время, за которое машины выполнят все задания.

Задача маршрутизации транспортных средств. Имеется список клиентов, находящихся в определенных местах, и парк автомобилей. Нужно найти наиболее оптимальные маршруты, позволяющие посетить всех клиентов.

и т.д.

NP-полные задачи

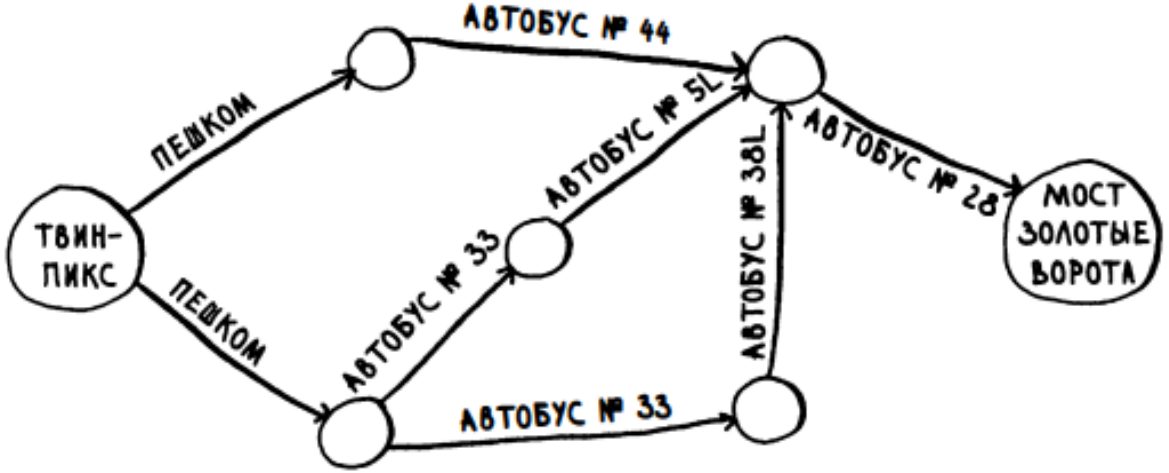
NP-полные задачи встречаются довольно часто.

Было бы полезно, если бы вы могли понять, что решаемая задача является NP-полной. В этот момент можно прекратить поиски идеального решения и, например, перейти к решению с применением приближенного алгоритма.

Однако определить, является ли ваша задача NP-полной, непросто. Часто различия между легко решаемыми и NP-полными задачами весьма незначительны.

NP-полные задачи

Вы знаете, как алгоритм вычисления кратчайшего пути из точки A в точку B.



Но если вы хотите найти кратчайший путь, соединяющий несколько точек, то задача превращается в задачу о коммивояжере, которая является NP-полной.

NP-полные задачи. Характерные признаки

- Ваш алгоритм сильно замедляется при увеличении числа данных
- Формулировка «все комбинации X» часто указывает на NP-полноту задачи
- Для нахождения решения вам приходится вычислять все возможные варианты X, потому что задачу невозможно разбить на меньшие подзадачи

NP-полные задачи. Характерные признаки

- Если в задаче встречается некоторая последовательность (например, последовательность городов, как в задаче о коммивояжере) или некоторое множество (например, множество радиостанций) и задача не имеет простого решения, она может оказаться NP-полной
- Если задачу можно переформулировать в терминах какой-то из существующих NP-полных

NP-полные задачи. Что же делать?

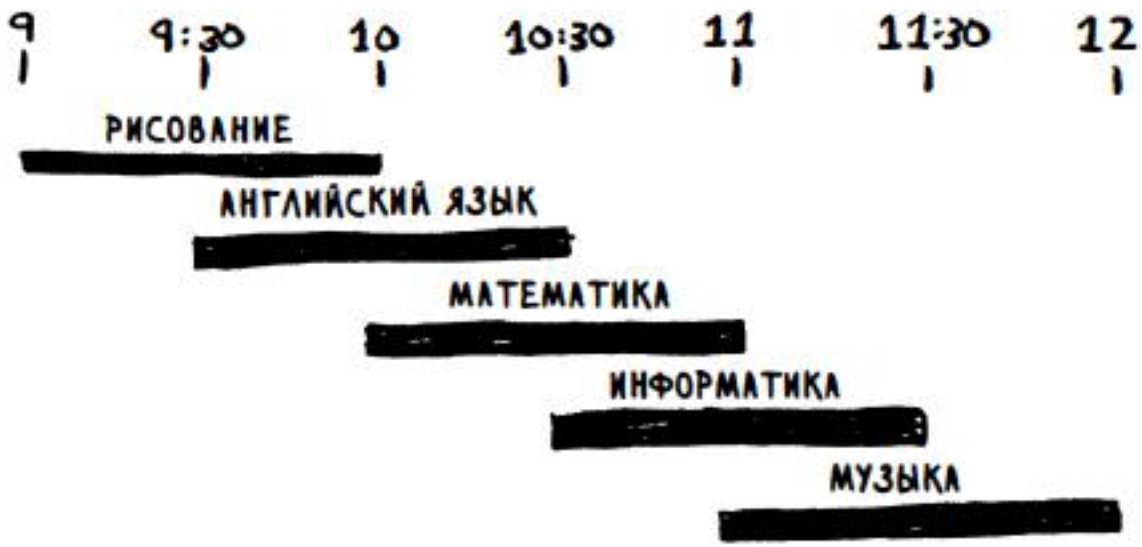
1. Перебор всех вариантов. Обычно подходит для небольших размеров входных данных (примерно до 10-20 элементов в зависимости от задачи)
2. Жадный алгоритм. Дает приближенное решение
3. Динамическое программирование. Подходит не для всех NP-полных задач

Жадные алгоритмы. Задача составления расписания

Допустим, имеется учебный класс, в котором нужно провести как можно больше уроков.

Провести в классе все уроки не получится, потому что некоторые из них перекрываются по времени.

ПРЕДМЕТ	С	ДО
РИС.	9:00	10:00
АНГЛ.	9:30	10:30
МАТ-КА	10:00	11:00
ИНФ-КА	10:30	11:30
МУЗЫКА	11:00	12:00



Жадные алгоритмы. Задача составления расписания

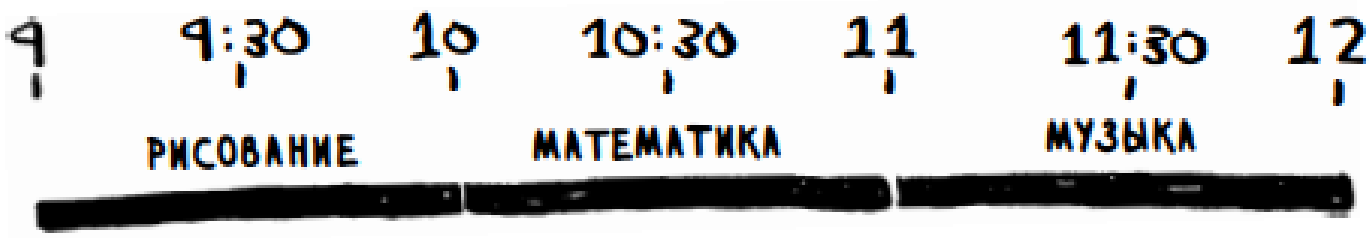
1. Выбрать урок, завершающийся раньше всех. Это первый урок, который будет проведен в классе.
2. Затем выбираются уроки, начинающиеся после завершения первого урока. Из них снова следует выбрать урок, который завершается раньше всех остальных. Он становится вторым уроком в расписании.
3. и т.д.

Жадные алгоритмы.
Задача составления расписания

РИС.	9:00	10:00	✓
АНГЛ.	9:30	10:30	
МАТ-КА	10:00	11:00	
ИНФ-КА	10:30	11:30	
МУЗЫКА	11:00	12:00	

РИС.	9:00	10:00	✓
АНГЛ.	9:30	10:30	X
МАТ-КА	10:00	11:00	✓
ИНФ-КА	10:30	11:30	
МУЗЫКА	11:00	12:00	

РИС.	9:00	10:00	✓
АНГЛ.	9:30	10:30	X
МАТ-КА	10:00	11:00	✓
ИНФ-КА	10:30	11:30	X
МУЗЫКА	11:00	12:00	✓



Жадные алгоритмы. Задача составления расписания

Жадный алгоритм прост: на каждом шаге он выбирает оптимальный вариант.

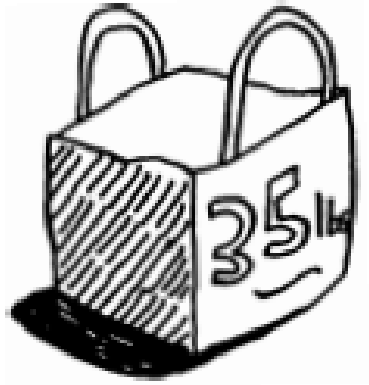
В нашем примере при выборе урока выбирается тот урок, который завершается раньше других. В технической терминологии: на каждом шаге выбирается **локально-оптимальное решение**, а в итоге вы получаете или приближаетесь к **глобально-оптимальному решению**.

В нашем примере мы нашли **оптимальное решение** задачи составления расписания.

Жадные алгоритмы. Задача о рюкзаке

Вор забрался в магазин с рюкзаком, и перед ним множество товаров, которые можно украсть. Однако емкость рюкзака не бесконечна: он выдержит не более 35 фунтов.

Требуется подобрать набор товаров максимальной стоимости, которые вместятся в рюкзак.



МАГНИТОФОН
$3ØØØ
3Ø ФУНТОВ

НОУТБУК
$2ØØØ
2Ø ФУНТОВ

ГИТАРА
$15ØØ
15ФУНТОВ

Жадные алгоритмы. Задача о рюкзаке

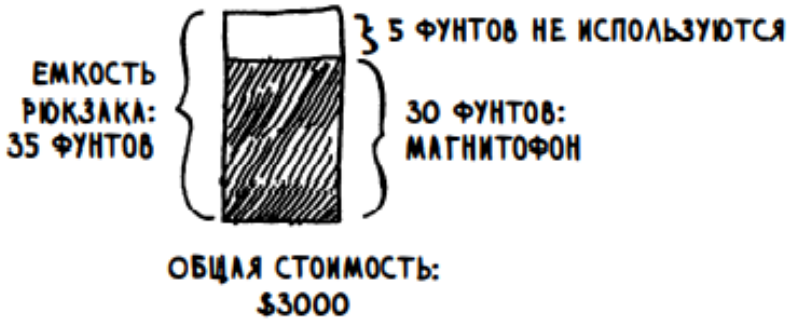
1. Выбрать самый дорогой предмет, который поместится в рюкзаке.
2. Выбрать следующий по стоимости предмет, который поместится в рюкзаке.
3. и т.д.

Однако в этом примере этот подход не даст оптимальное решение!

МАГНИТОФОН
$3000
30 ФУНТОВ

НОУТБУК
$2000
20 ФУНТОВ

ГИТАРА
$1500
15ФУНТОВ



Жадные алгоритмы. Задача о рюкзаке

В данном случае жадная стратегия не дает оптимального решения.

Однако иногда идеальное – враг хорошего.

В некоторых случаях достаточно алгоритма, способного решить задачу просто **достаточно хорошо**.

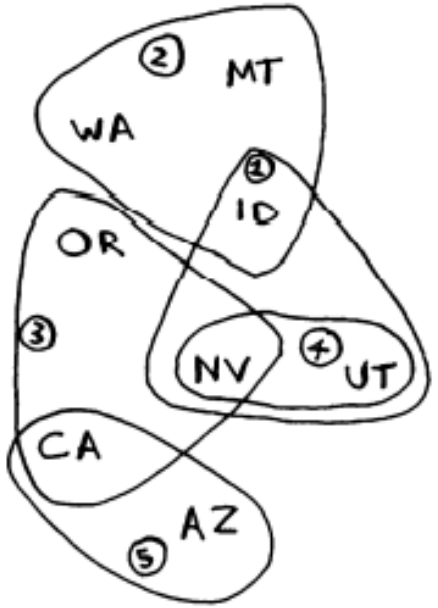
В таких областях жадные алгоритмы имеют широкое применение, потому что они просто реализуются, а полученное решение обычно близко к оптимуму.

Жадные алгоритмы. Задача о покрытии множества

Вы открываете собственную авторскую программу на радио и хотите, чтобы вас слушали во всех 50 штатах. Нужно решить, на каких радиостанциях должна транслироваться ваша передача. Каждая станция стоит денег, поэтому количество станции необходимо свести к минимуму. Имеется список станций и какие штаты они покрывают.

РАДИО-СТАНЦИЯ	ДОСТУПНА В ШТАТАХ
KONE	ID,NV,UT
KTWO	WA, ID, MT
KTHREE	OR, NV, CA
KFOUR	NV, UT
KFIVE	CA, AZ

... и т. д. ...



Жадные алгоритмы. Задача о покрытии множества

1. Выбрать станцию, покрывающую наибольшее количество штатов, еще не входящих в покрытие. Если станция будет покрывать некоторые штаты, уже входящие в покрытие, это нормально.
2. Повторять, пока остаются штаты, не входящие в покрытие.

В данном случае жадный алгоритм имеет сложность $O(n^2)$, в то время как полный перебор – $O(2^n)$.

См. пример 1

Жадные алгоритмы. Задача о покрытии множества

Так как алгоритм приближенный, то правильных решений может быть несколько.

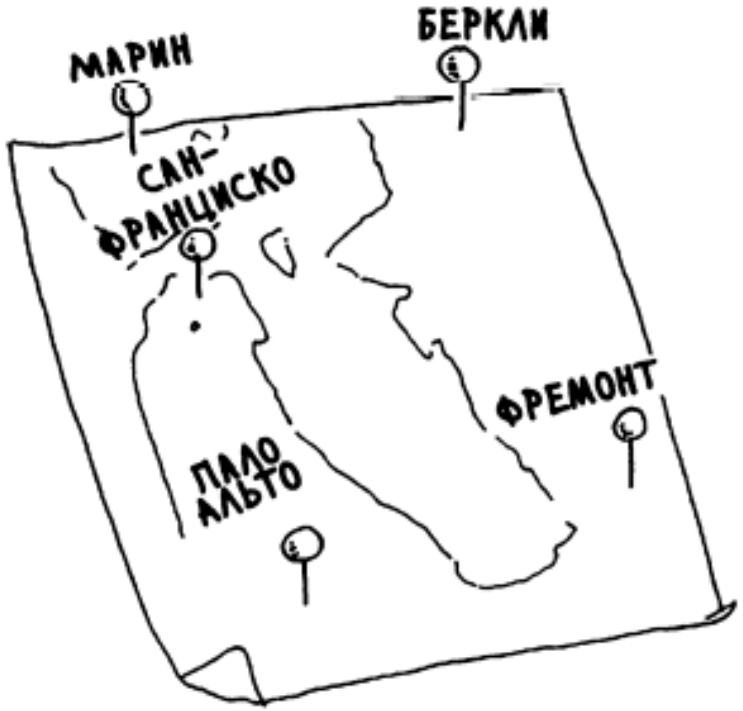
Эффективность такого алгоритма оценивается по быстроте и близости полученного решения к оптимальному.

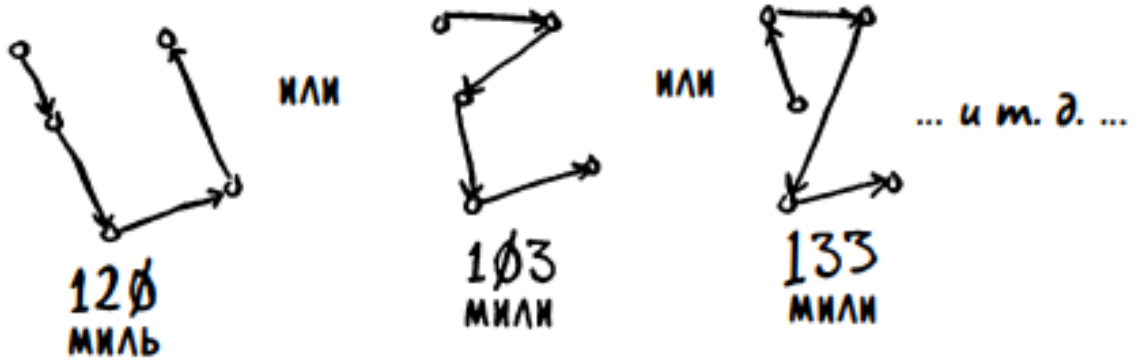
	$O(2^n)$	$O(n^2)$
КОЛИЧЕСТВО СТАНЦИЙ	ТОЧНЫЙ АЛГОРИТМ	ЖАДНЫЙ АЛГОРИТМ
5	3.2 с	2.5 с
10	102.4 с	10 с
32	13.6 года	102.4 с
100	4×10^{21} года	16.67 мин

Жадные алгоритмы. Задача о коммивояжере

Коммивояжер хочет побывать в каждом из 5 городов так, чтобы при этом проехать минимальное общее расстояние.

Как выглядит хороший жадный алгоритм для этой задачи?





Жадные алгоритмы. Задача о коммивояжере

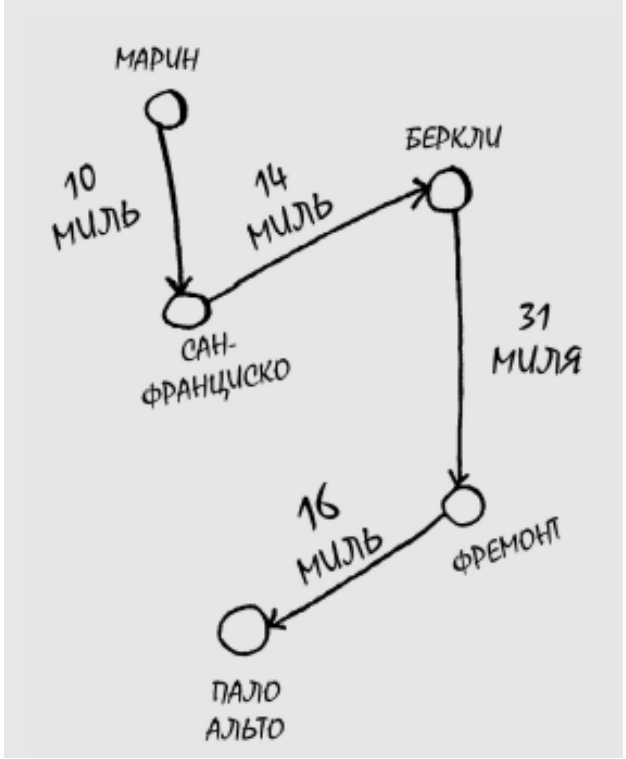
Начальный город выбирается произвольно, после чего каждый раз, когда коммивояжер выбирает следующий город, он перемещается в ближайший город из тех, что он еще не посещал.

Допустим он начинает в Марине.

Суммарное расстояние - 71 миля.

Возможно не самый короткий путь,

но он достаточно близок к нему.



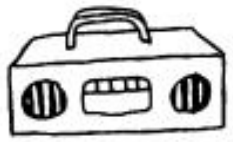
Динамическое программирование

Динамическое программирование – метод решения сложных задач, разбиваемых на подзадачи, которые решаются в первую очередь.

Динамическое программирование. Задача о рюкзаке

Вор забрался в магазин с рюкзаком, и перед ним множество товаров, которые можно украсть. Однако емкость рюкзака не бесконечна: он выдержит не более 4 фунтов.

Требуется подобрать набор товаров максимальной стоимости, которые вместятся в рюкзак.



МАГНИТОФОН
$3000
4 ФУНТА

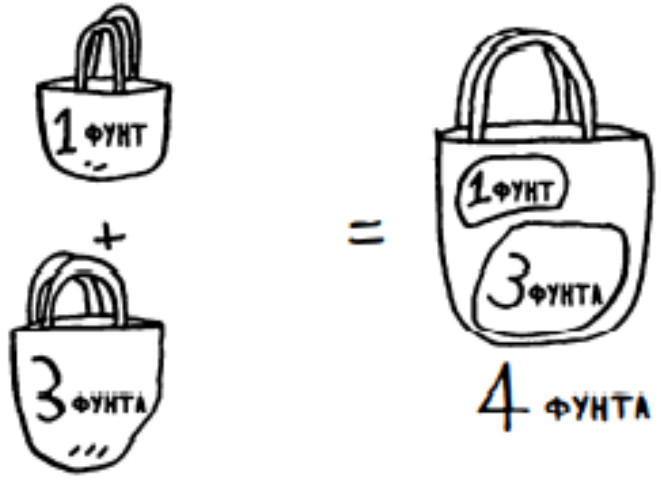
НОУТБУК
$2000
3 ФУНТА

ГИТАРА
$1500
1 ФУНТ

Динамическое программирование. Задача о рюкзаке

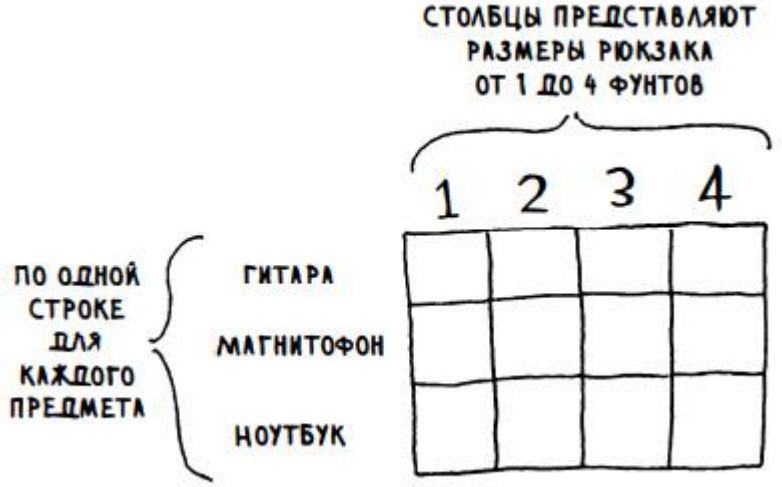
Процедура начинается с решения подзадач с постепенным переходом к решению полной задачи.

В задаче о рюкзаке начать следует с решения задачи для меньшего рюкзака (или «подрюкзака»), а потом на этой основе попытаться решить исходную задачу.



Динамическое программирование. Задача о рюкзаке

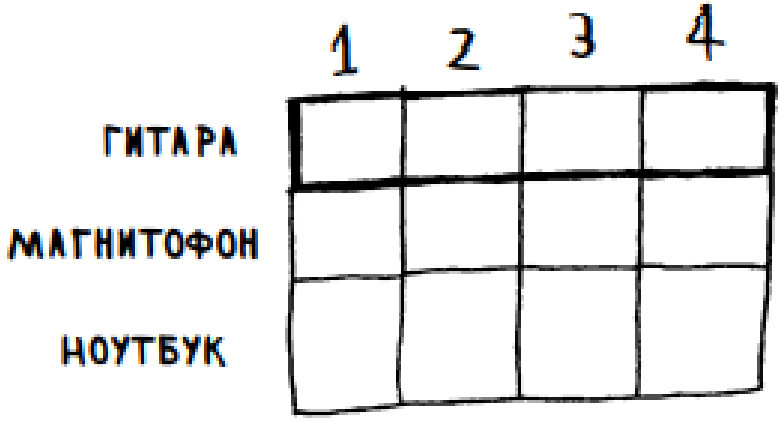
Каждый алгоритм динамического программирования начинается с таблицы. Вот как выглядит таблица для задачи о рюкзаке.



Строки таблицы представляют предметы, а столбцы - емкость рюкзака от 1 до 4 фунтов. Все эти столбцы нужны, потому что они упрощают вычисление стоимостей «подрюкзаков».

Динамическое программирование. Задача о рюкзаке

Начнем с первой строки.



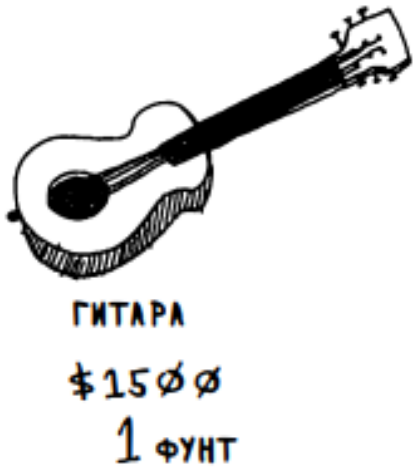
Строка снабжена пометкой «гитара»; это означает, что вы пытаетесь уложить гитару в рюкзак.

В каждой ячейке принимается простое решение: класть гитару в рюкзак или нет? Помните: мы пытаемся найти множество элементов с максимальной стоимостью.

Динамическое программирование. Задача о рюкзаке

В каждой ячейке принимается простое решение: класть гитару в рюкзак или нет?

Считаем, что два других предмета пока недоступны.

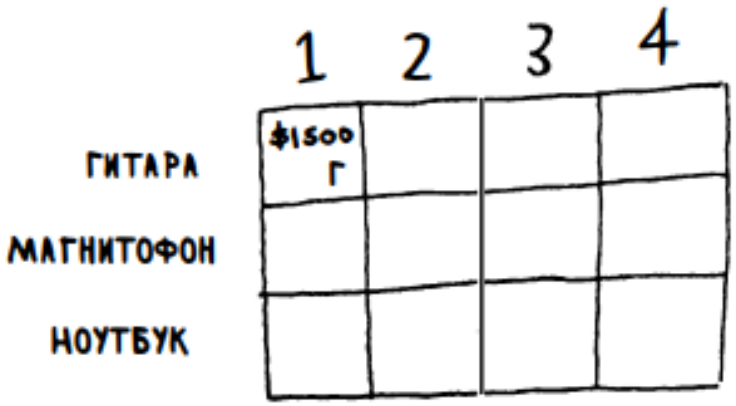


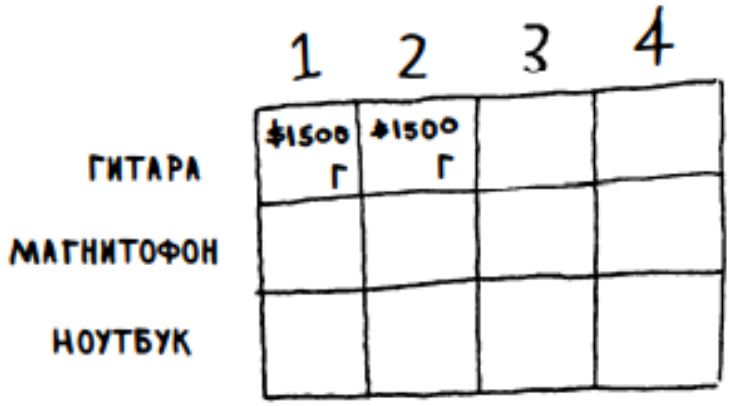
Динамическое программирование. Задача о рюкзаке

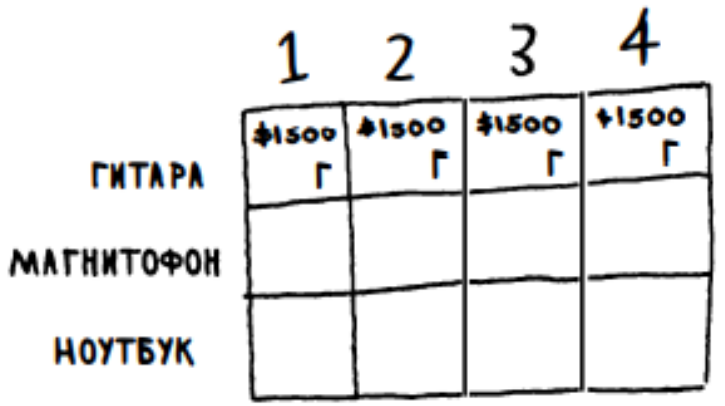
ГИТАРА

$1500

1 ФУНТ



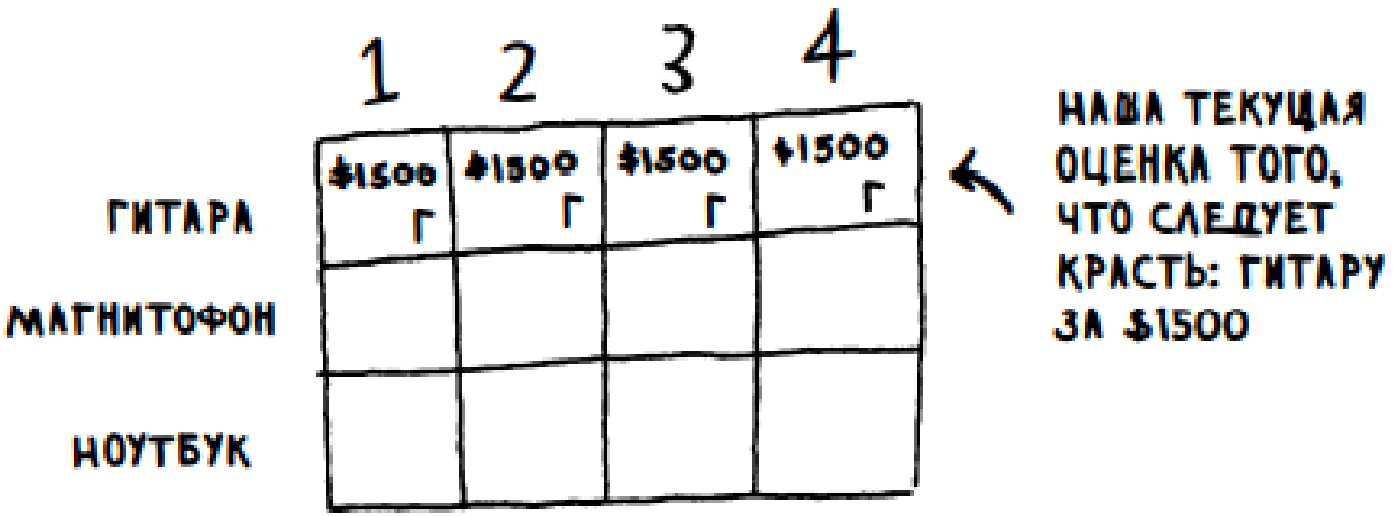




Динамическое программирование. Задача о рюкзаке

Помните, что мы стремимся обеспечить максимальную стоимость предметов в рюкзаке.

Эта строка представляет текущую лучшую оценку максимума. И так, на данный момент из этой строки следует, что для рюкзака с емкостью 4 фунта максимальная стоимость предметов составит $ 1500.

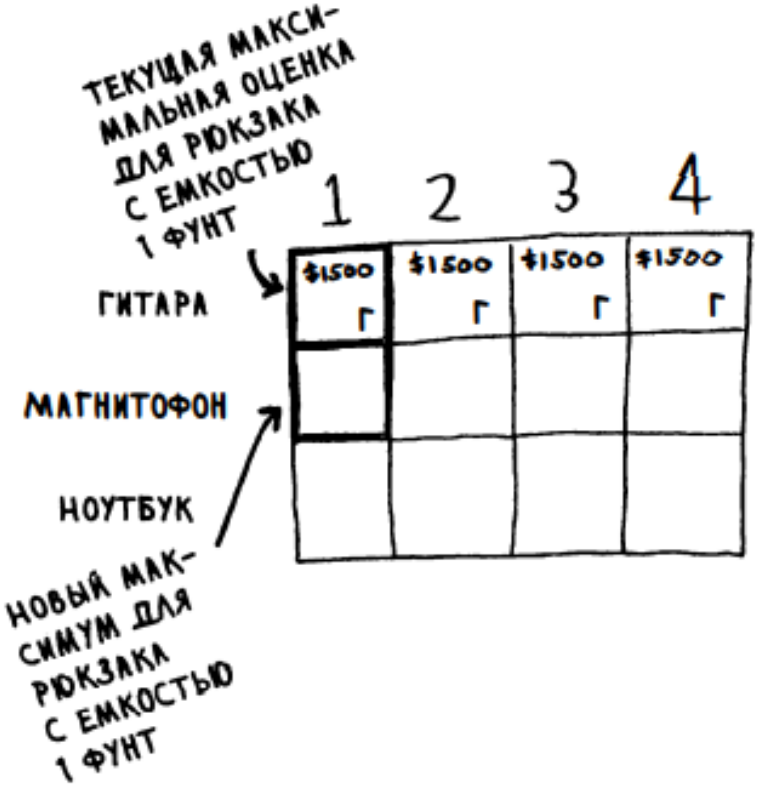


Динамическое программирование. Задача о рюкзаке

Займемся следующей строкой, которая относится к магнитофону.

Теперь, когда вы перешли ко второй строке, появляется выбор между магнитофоном и гитарой.

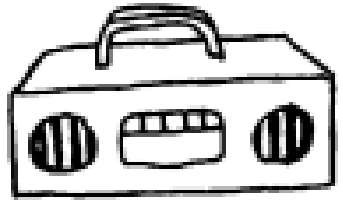
В каждой строке можно взять предмет этой строки или предметы, находящиеся в верхних строках. Таким образом, сейчас нельзя выбрать ноутбук, но можно выбрать магнитофон и/или гитару.



Динамическое программирование. Задача о рюкзаке

Начнем с первой ячейки (рюкзак с емкостью 1 фунт).

Так как магнитофон не помещается в рюкзак, максимальная оценка для 1-фунтового рюкзака остается равной $ 1 500.

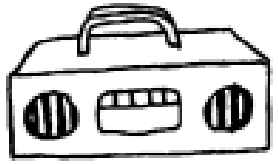


МАГНИТОФОН

$ 3000

4 ФУНТА

Динамическое программирование. Задача о рюкзаке



МАГНИТОФОН

$3000

4 ФУНТА

	1	2	3	4
ГИТАРА	$1500 Г	$1500 Г	$1500 Г	$1500 Г
МАГНИТОФОН	$1500 Г			
НОУТБУК				

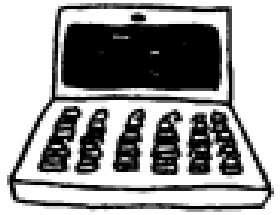
	1	2	3	4
ГИТАРА	$1500 Г	$1500 Г	$1500 Г	$1500 Г
МАГНИТОФОН	$1500 Г	$1500 Г	$1500 Г	
НОУТБУК				

	1	2	3	4
ГИТАРА	$1500 Г	$1500 Г	$1500 Г	$1500 Г
МАГНИТОФОН	$1500 Г	$1500 Г	$1500 Г	$3000 М
НОУТБУК				

Динамическое программирование. Задача о рюкзаке

Теперь проделаем то же для ноутбука!

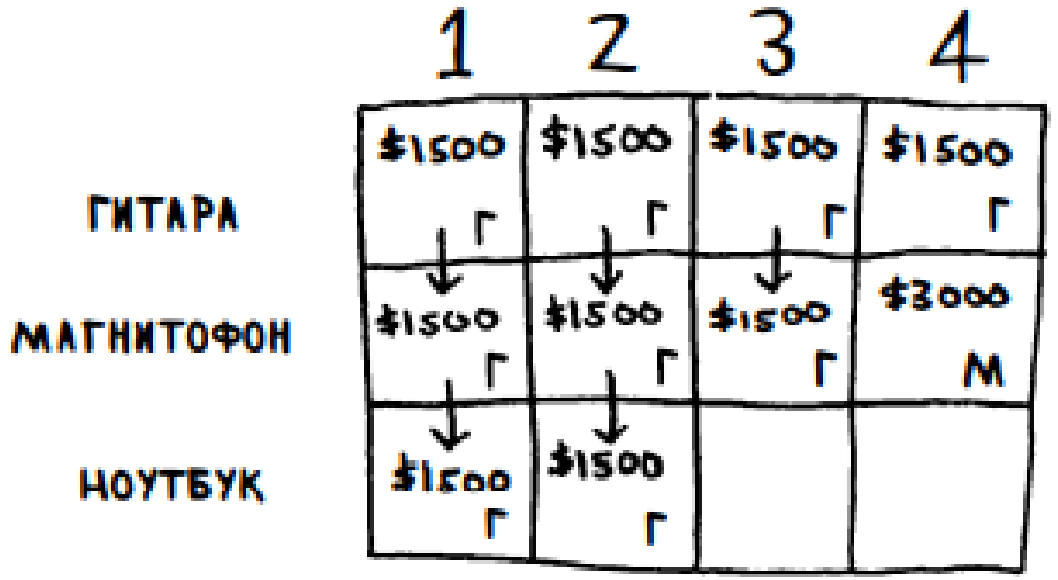
Ноутбук весит 3 фунта, поэтому он не поместится в рюкзак с емкостью 1 или 2 фунта. Оценка для первых двух ячеек остается на уровне $ 1500.



НОУТБУК

$ 2000

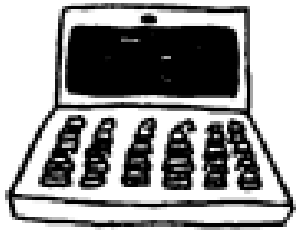
3 ФУНТА



Динамическое программирование. Задача о рюкзаке

Для 3 фунтов старая оценка составляла $ 1 500.

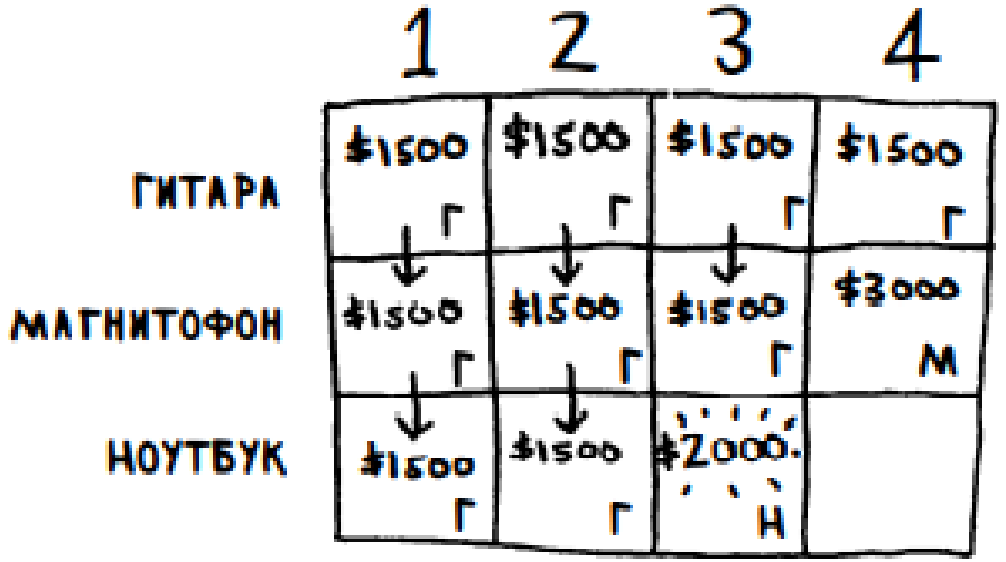
Но теперь вы можете выбрать ноутбук, который стоит $2000. Следовательно, новая максимальная оценка равна $2000!



НОУТБУК

$ 2 Ø Ø Ø

3 ФУНТА

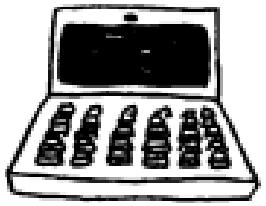


Динамическое программирование. Задача о рюкзаке

При 4 фунтах ситуация становится по-настоящему интересной.

В настоящее время оценка составляет $3000. В рюкзак можно положить ноутбук, но он стоит всего $2000.

При этом ноутбук весит всего 3 фунта, так что 1 фунт еще свободен! На это место можно еще что-нибудь положить.



НОУТБУК

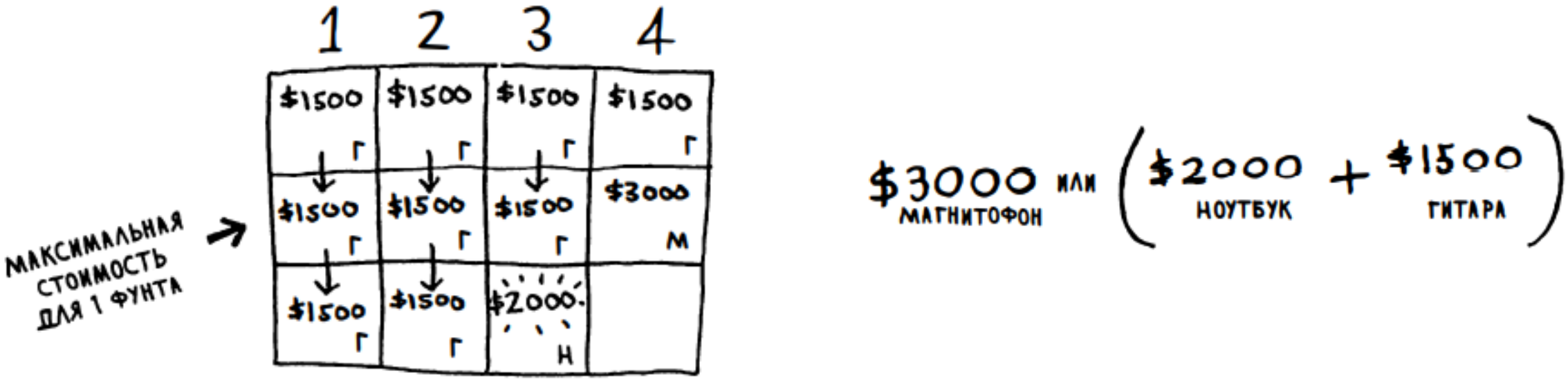
$2000

3 ФУНТА

$3000 МАГНИТОФОН ИЛИ ($2000 НОУТБУК + ??? СВОБОДНОЕ МЕСТО НА 1 ФУНТ)

Динамическое программирование. Задача о рюкзаке

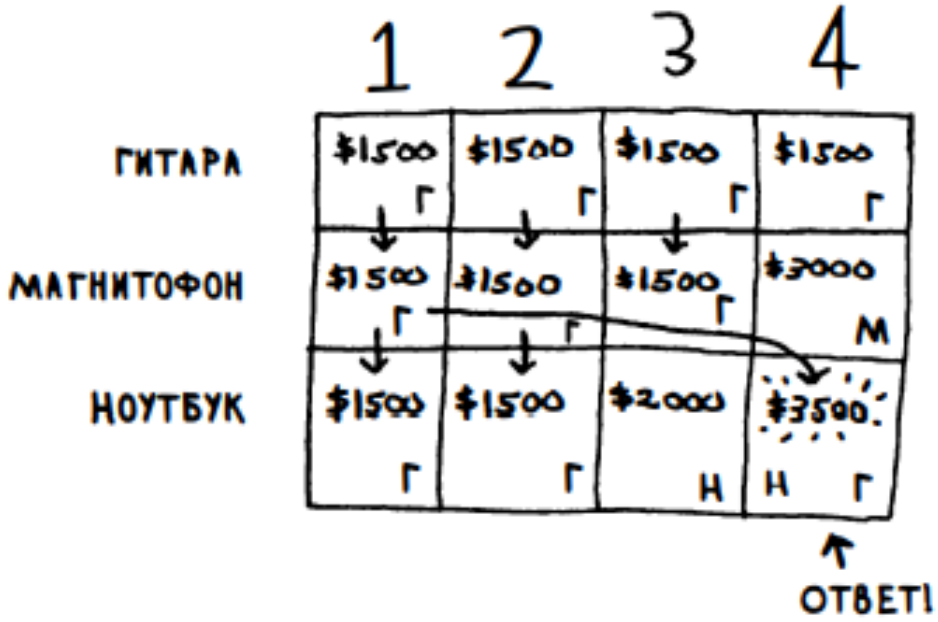
Какую максимальную стоимость можно разместить в 1 фунте?



Динамическое программирование. Задача о рюкзаке

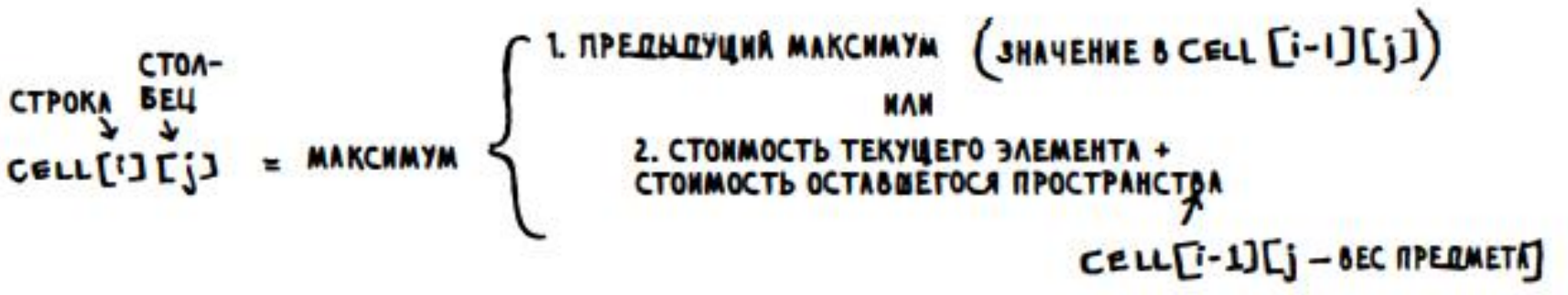
Если в рюкзаке остается свободное место, вы можете использовать ответы на эти подзадачи для определения того, чем заполнить это пространство.

Вместо магнитофона лучше взять ноутбук + гитару за $3500.



Динамическое программирование. Задача о рюкзаке

Стоимость каждой ячейки вычисляется по постоянной формуле, которая выглядит так:



Мы объединили решения двух подзадач для решения еще одной, большей задачи.

Динамическое программирование. Задача о рюкзаке

Что произойдет при добавлении элемента?

IPHONE
$2000
1 фунт

	1	2	3	4
ГИТАРА	$1500 Г	$1500 Г	$1500 Г	$1500 Г
МАГНИТОФОН	$1500 Г	$1500 Г	$1500 Г	$3000 М
НОУТБУК	$1500 Г	$1500 Г	$2000 Н	$3500 Н Г

	1	2	3	4
ГИТАРА	$1500 Г	$1500 Г	$1500 Г	$1500 Г
МАГНИТОФОН	$1500 Г	$1500 Г	$1500 Г	$3000 М
НОУТБУК	$1500 Г	$1500 Г	$2000 Н	$3500 Н Г
IPHONE				

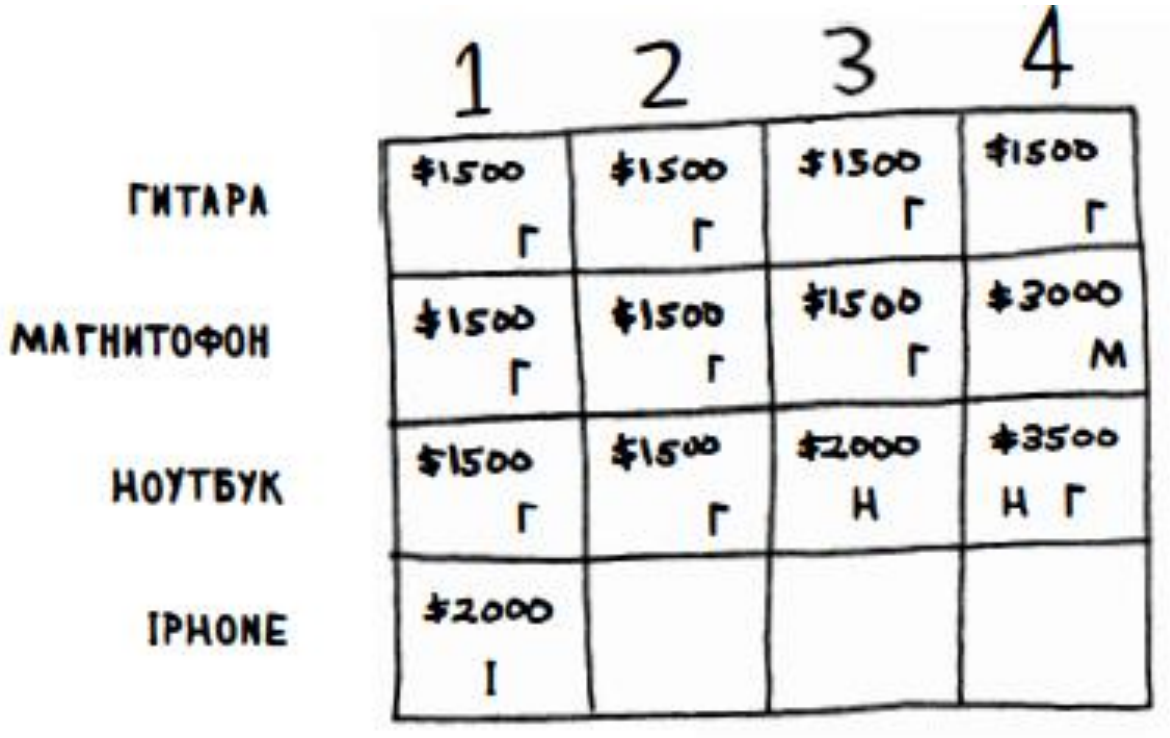
НОВЫЙ ОТВЕТ

Динамическое программирование. Задача о рюкзаке

Начнем с первой ячейки.

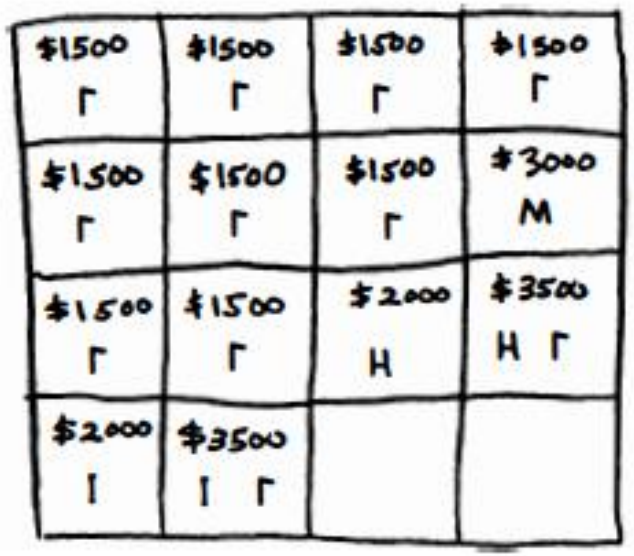
iPhone сам по себе помещается в рюкзак с емкостью 1 фунт.

Старый максимум был равен $ 1 500, но iPhone стоит $2000. Значит, берем iPhone.



Динамическое программирование. Задача о рюкзаке

В следующей ячейке можно разместить iPhone и гитару.



Для ячейки 3 ничего лучшего, чем снова взять iPhone вместе с гитарой нет.

Динамическое программирование. Задача о рюкзаке

А вот в последней ячейке ситуация становится более интересной.

Текущий максимум равен $3500. Вы снова можете взять iPhone, и у вас еще останется свободное место на 3 фунта.

Динамическое программирование. Задача о рюкзаке

Может ли значение в столбце уменьшиться?

Нет. При каждой итерации сохраняется текущая оценка максимума. Эта оценка ни при каких условиях не может быть меньше предыдущей!

Что произойдет при изменении порядка строк?

Ответ не изменится. Он не зависит от порядка строк.

Можно ли заполнять таблицу по столбцам, а не по строкам?

В данной задаче это ни на что не влияет, но в других задачах возможны изменения.

Динамическое программирование. Задача о рюкзаке

Что произойдет при добавлении меньшего по весу элемента?

Допустим, вы можете выбрать ожерелье, которое весит 0,5 фунта и стоит $ 1 000.

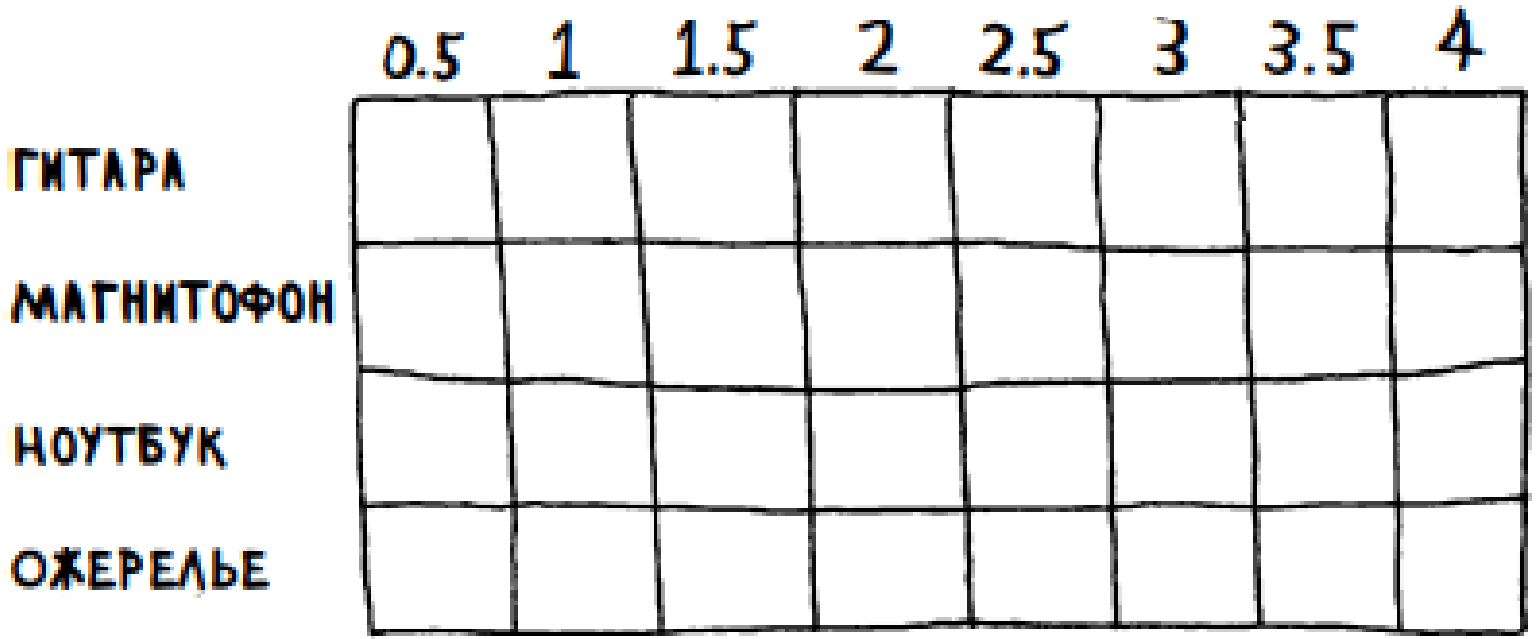
Пока структура таблицы предполагает, что все веса являются целыми числами. Теперь вы решаете взять ожерелье. Остается еще 3,5 фунта. Какую максимальную стоимость можно разместить в объеме 3,5 фунта?

Неизвестно! Вы вычисляли стоимость только для рюкзаков с емкостью 1, 2, 3 и 4 фунта. Теперь придется определять стоимость для рюкзака на 3,5 фунта.

Динамическое программирование. Задача о рюкзаке

Что произойдет при добавлении меньшего по весу элемента?

Из-за ожерелья приходится повысить точность представления весов, поэтому таблица должна измениться.



Динамическое программирование. Задача о рюкзаке

Возможно ли, что при лучшем решении в рюкзаке остается пустое место?

Да. Представьте, что вы можете также положить в рюкзак бриллиант.

Бриллиант очень крупный: он весит 3,5 фунта и стоит 1 миллион долларов - намного больше, чем любые другие предметы.

Безусловно, нужно брать именно его! Но в рюкзаке остается еще пустое место на 0,5 фунта, и в нем ничего не поместится.

Динамическое программирование. Задача о рюкзаке

Можно ли взять часть предмета?

Допустим, вы наполняете рюкзак в продуктовом магазине. Вы можете украсть мешки с чечевицей и рисом. Если весь мешок не помещается, его можно открыть и отсыпать столько, сколько унесете.

В этом случае вы уже не действуете по принципу «все или ничего» – можно взять только часть предмета. Как решить такую задачу методом динамического программирования?

Никак. В решении, полученном методом динамического программирования, вы либо берете предмет, либо не берете. Алгоритм не предусматривает возможность взять половину предмета.

Динамическое программирование. Задача о рюкзаке

Можно ли взять часть предмета?

Однако проблему можно решить с помощью жадного алгоритма!

Сначала вы берете самый ценный предмет - настолько большую его часть, насколько возможно.

Когда самый ценный предмет будет исчерпан, вы берете максимально возможную часть следующего по ценности предмета и т. д.

Допустим, вы можете выбирать из следующих товаров.

Динамическое программирование. Задача о рюкзаке

Можно ли взять часть предмета?

Допустим, вы можете выбирать из следующих товаров.

Фунт киноа стоит дороже, чем фунт любого другого товара. А раз так - набираем столько киноа, сколько сможем унести! И если вам удастся набить им свой рюкзак, то это и будет лучшее из возможных решений.

Если киноа кончится, а в рюкзаке еще остается свободное место, возьмите следующий по ценности товар и т. д.

Динамическое программирование

- Динамическое программирование применяется для оптимизации какой-либо характеристики при заданных ограничениях. В задаче о рюкзаке требуется максимизировать стоимость отобранных предметов с ограничениями по емкости рюкзака.
- Динамическое программирование работает только в ситуациях, в которых задача может быть разбита на автономные подзадачи, не зависящие друг от друга.

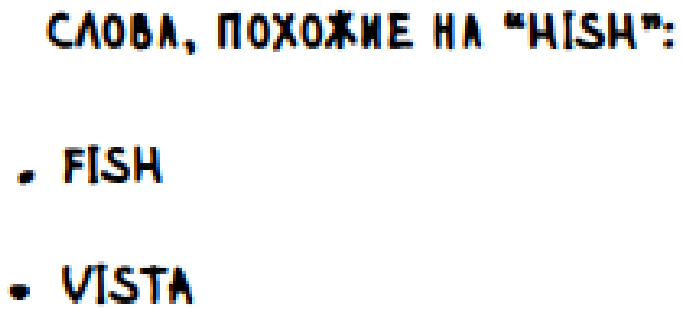
Динамическое программирование

- в каждом решении из области динамического программирования строится таблица
- значения ячеек таблицы обычно соответствуют оптимизируемой характеристике. Для задачи о рюкзаке значения представляли общую стоимость товаров
- каждая ячейка представляет подзадачу, поэтому вы должны подумать о том, как разбить задачу на подзадачи. Это поможет вам определиться с осями.

Динамическое программирование. Самая длинная общая подстрока

Рассмотрим еще один пример. Допустим, вы открыли сайт dictionary.com.

Пользователь вводит слово, а сайт возвращает определение. Но если пользователь ввел несуществующее слово, нужно предположить, какое слово имелось в виду. Алекс ищет определение «fish», но он случайно ввел «hish». Такого слова в словаре нет, но зато у вас есть список похожих слов.



Какое слово он хотел ввести на самом деле?

Динамическое программирование. Самая длинная общая подстрока

Как должна выглядеть таблица для этой задачи? Вы должны ответить на следующие вопросы:

1. Какие значения должны содержаться в ячейках?
2. Как разбить эту задачу на подзадачи?
3. Каков смысл осей таблицы?

В динамическом программировании вы пытаетесь максимизировать некоторую характеристику. В данном случае ищется самая длинная подстрока, общая в двух словах.

Какую общую подстроку содержат hish и fish? А как насчет hish и vista? Именно это требуется вычислить.

Динамическое программирование. Самая длинная общая подстрока

Как говорилось ранее, значения в ячейках обычно представляют ту характеристику, которую вы пытаетесь оптимизировать.

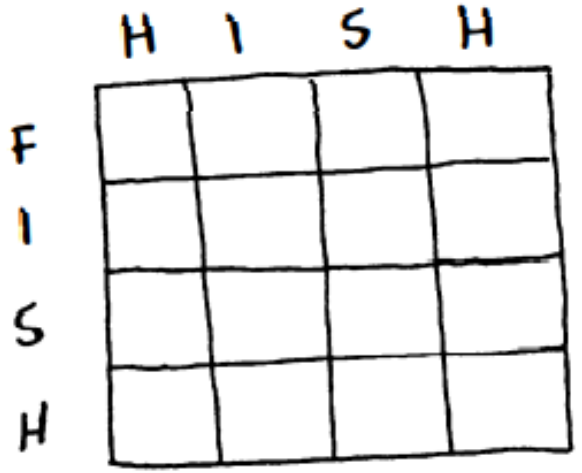
Вероятно, в данном случае этой характеристикой будет число: длина самой длинной подстроки, общей для двух строк.

Динамическое программирование. Самая длинная общая подстрока

Как разделить эту задачу на подзадачи?

Например, можно заняться сравнением подстрок. Вместо того чтобы сравнивать hish и fish, можно сначала сравнить his и fis.

Каждая ячейка будет содержать длину самой длинной подстроки, общей для двух подстрок. Такое решение также подсказывает, что строками и столбцами таблицы, вероятно, будут два слова.

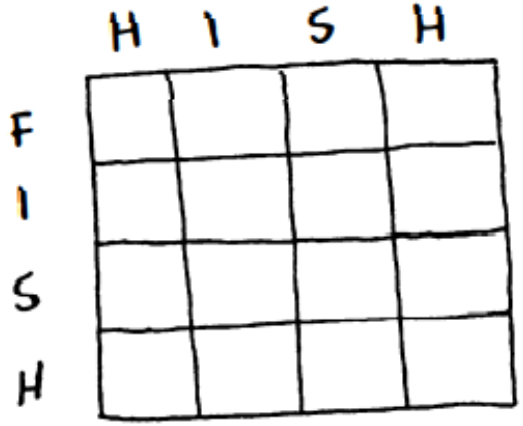


Динамическое программирование. Самая длинная общая подстрока

По какой формуле заполняются ячейки таблицы?

Мы можем немного упростить свою задачу, потому что уже знаем решение - у hish и fish имеется общая подстрока длины 3: ish.

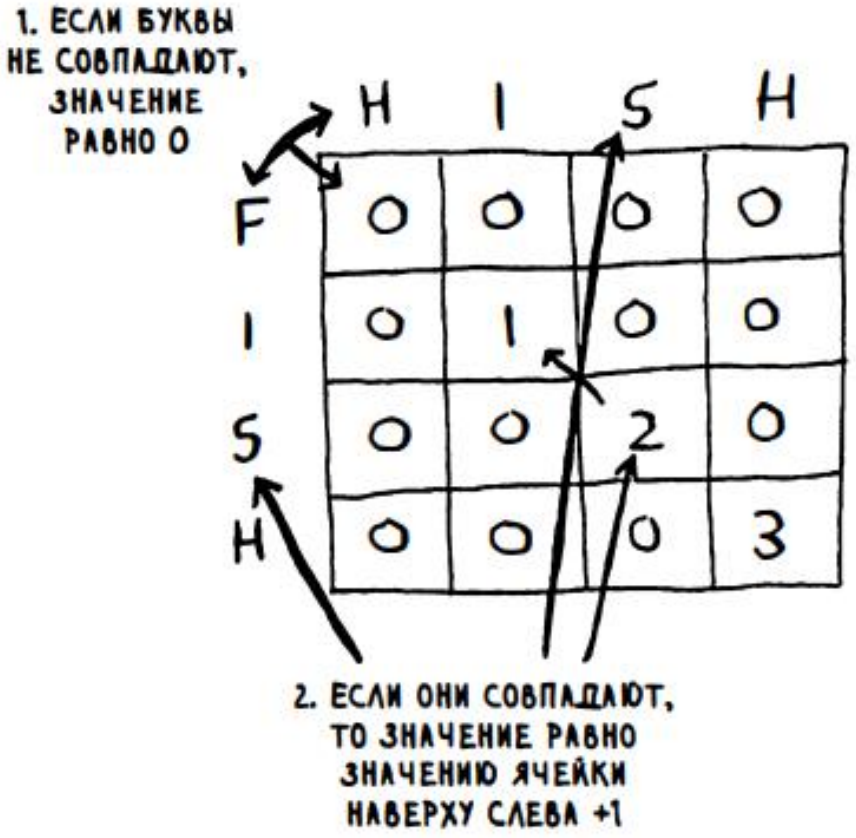
Обычно простого способа вычислить формулу для ячейки не существует. Вам придется экспериментировать и искать работоспособное решение. Иногда алгоритм предоставляет не точный рецепт, а основу, на которую вы наращиваете свою идею.



Динамическое программирование. Самая длинная общая подстрока

Каждая ячейка содержит значение подзадачи.

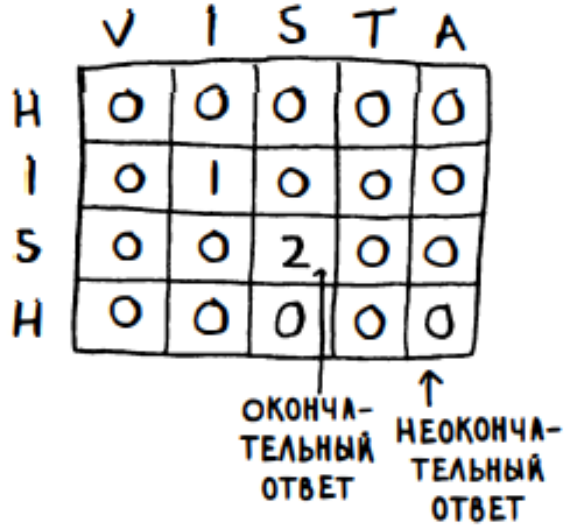
Итоговая версия таблицы и формула:




```
if word_a[i] == word_b[j]:      <····· Буквы совпадают
  cell[i][j] = cell[i-1][j-1] + 1
else:
  cell[i][j] = 0                <····· Буквы не совпадают
```

Динамическое программирование. Самая длинная общая подстрока

Аналогичная таблица для строк hish и vista:



Важный момент: в этой задаче окончательное решение далеко не всегда находится в последней ячейке!

Динамическое программирование. Самая длинная общая подстрока

Вернемся к исходному вопросу: какая строка ближе к hish?

У строк hish и fish есть общая подстрока длиной в три буквы.

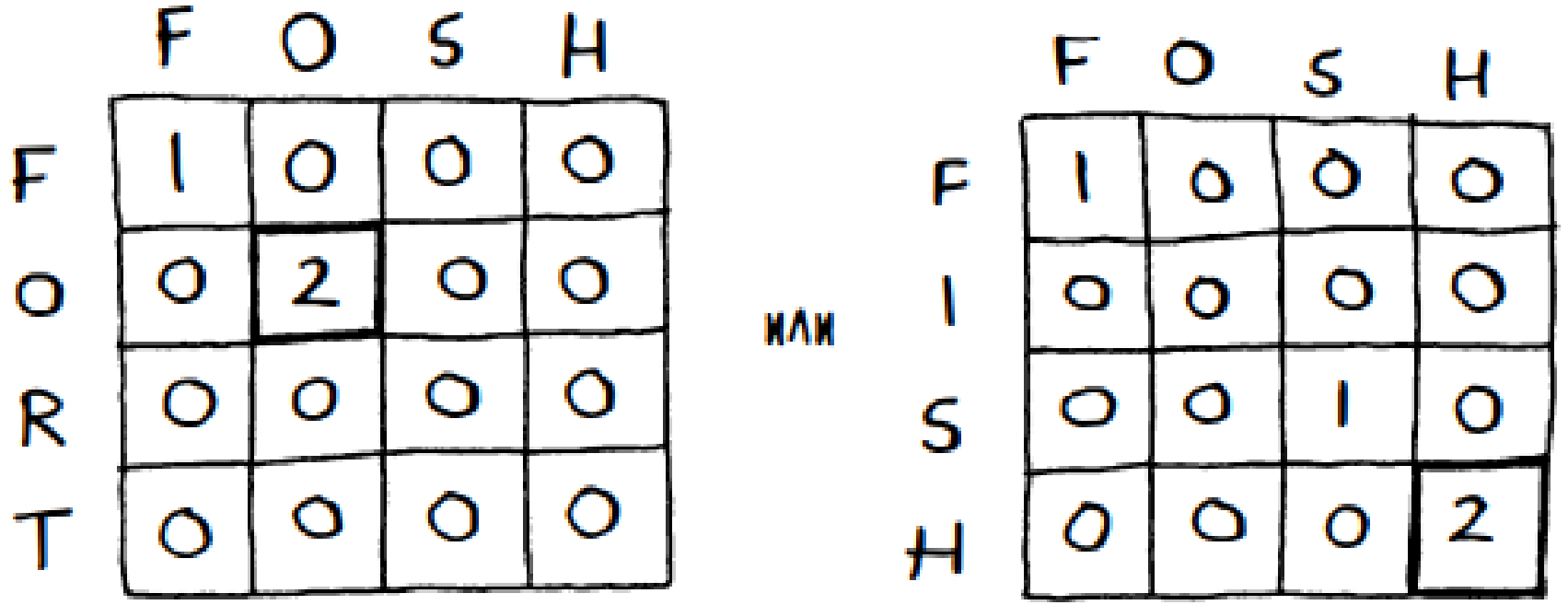
У hish и vista есть общая подстрока из двух букв.

Скорее всего, пользователь хотел ввести строку fish.

Динамическое программирование. Самая длинная общая подпоследовательность

Предположим, пользователь ввел строку fosh.

Какое слово он имел в виду: fish или fort?



Динамическое программирование. Самая длинная общая подпоследовательность

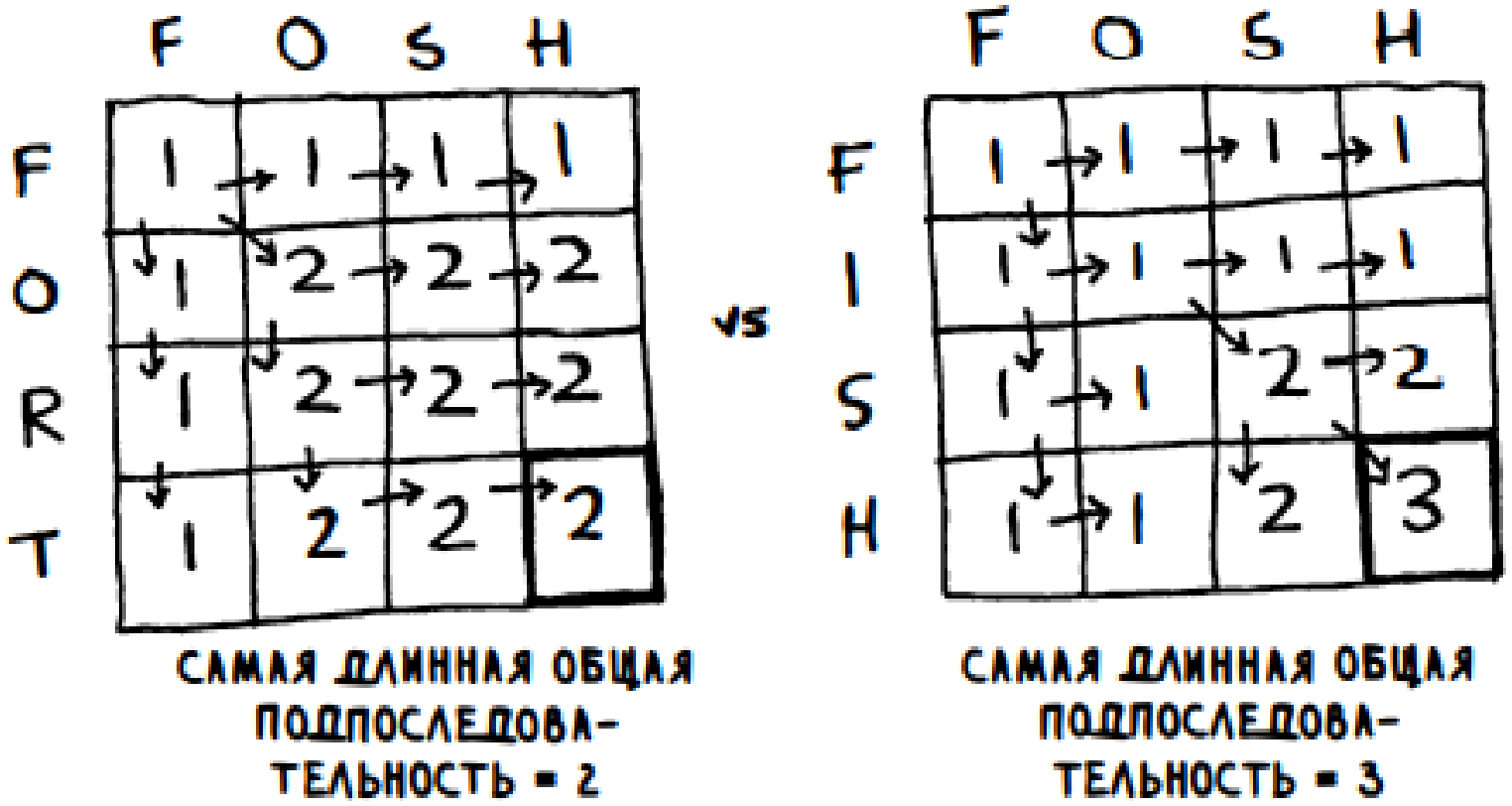
Длина подстрок одинакова: две буквы! Но fosh при этом ближе к fish:

Мы сравниваем самую длинную общую подстроку, а на самом деле нужно сравнивать самую длинную общую подпоследовательностъ: количество букв в последовательности, общих для двух слов.

Как вычислить самую длинную общую подпоследовательность?

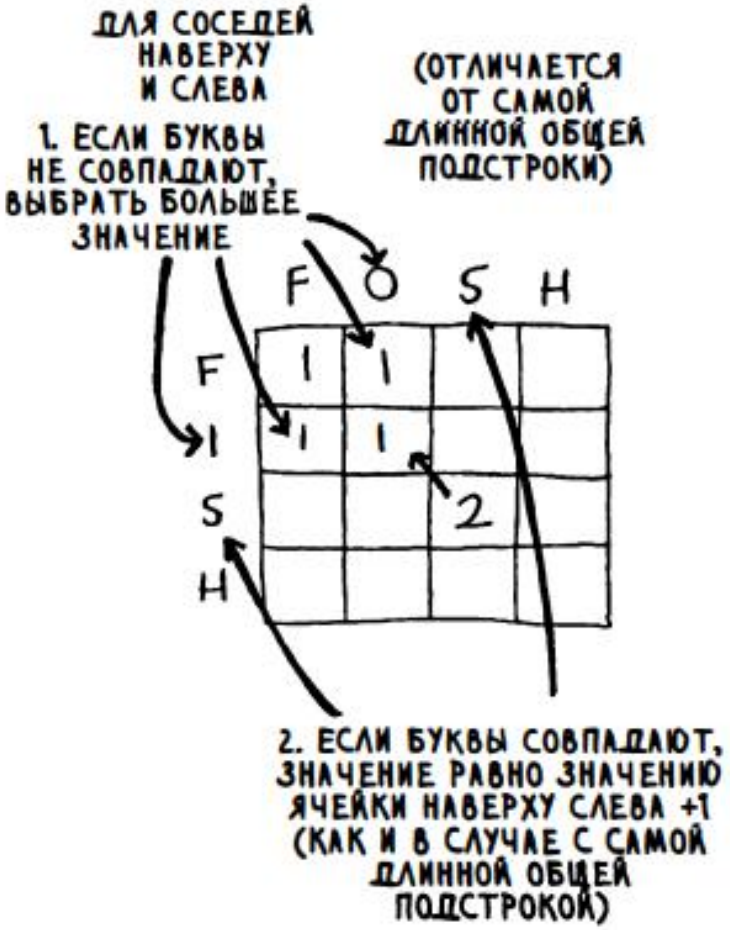
Динамическое программирование. Самая длинная общая подпоследовательность

Таблицы решения:



Динамическое программирование. Самая длинная общая подпоследовательность

Формула для заполнения каждой ячейки:




```
if word_a[i] == word_b[j]:      <-------  Буквы совпадают
  cell[i][j] = cell[i-1][j-1] + 1
else:      <-------------------------------------------  Буквы не совпадают
  cell[i][j] = m a x(cell[i-1][j], cell[i][j-1])
```

Динамическое программирование. Примеры применения

- Биологи используют самую длинную общую подпоследовательность для выявления сходства в цепях ДНК. По этой метрике можно судить о сходстве двух видов животных, двух заболеваний и т. д.
- Вы когда-нибудь пользовались ключом **diff** (например, в команде **git diff**)? Этот ключ выводит информацию о различиях между двумя файлами, а для этого он использует динамическое программирование

Динамическое программирование. Примеры применения

- Мы также упоминали о сходстве строк. Расстояние Левенштейна оценивает, насколько похожи две строки, а для его вычисления применяется динамическое программирование. Расстояние Левенштейна используется в самых разных областях, от проверки орфографии до выявления отправки пользователем данных, защищенных авторским правом.
- Вы когда-нибудь работали в приложении, поддерживающем перенос слов, например Microsoft Word? Как определить, где следует расставить переносы, чтобы длина строки оставалась более или менее постоянной? Динамическое программирование!

Динамическое программирование. Итоги

- Динамическое программирование применяется при оптимизации некоторой характеристики.
- Динамическое программирование работает только в ситуациях, в которых задача может быть разбита на автономные подзадачи.
- Обычно решение задачи из области динамического программирования можно представить в виде таблицы.
- Каждая ячейка такой таблицы представляет подзадачу, а ее значение соответствует оптимизируемой характеристике.
- Не существует единой формулы для вычисления решений методом динамического программирования.