1. Хеш-таблицы. Хеш. Вычисление хеша. Коллизии и разрешение их.

Хеширование – это преобразование входного массива данных определенного типа и произвольной длины в выходную битовую строку фиксированной длины.

Такие преобразования также называются хеш-функциями или функциями свертки, а их результаты называют хешем, хеш-кодом.

Хеш-таблица – это структура данных, реализующая интерфейс ассоциативного массива, то есть она позволяет хранить пары вида "ключ- значение" и выполнять три операции:

* операцию добавления новой пары;
* операцию поиска;
* операцию удаления пары по ключу.

Хеш-таблица является массивом, формируемым в определенном порядке хеш-функцией.

* Поиск любого элемента выполняется за фиксированное время (O(1))
* Добавление нового элемента выполняется за фиксированное время (O(1))
* Количество требуемой памяти пропорционально количеству возможных значений ключа

Возможна ситуация, когда мы пытаемся добавить элемент, а место занято. Эта ситуация называется коллизией

* открытое хеширование (хранение списков)
* закрытое хеширование (метод сдвига)
  + Линейное: При попытке в № i поместить значение k мы пробуем ячейку h(k, i). h(k, i) = (h’(k) + i) mod m. Функция – линейная
  + Квадратичное: h(k, i) = (h’(k) + c1i + c2i2) mod m. В отличие от линейного исследования, кластеризация (Когда элементов в массиве становится достаточно много, эффективность хеширования мала (приходится перебирать множество элементов)) слабее.

2. Графы. Представление графа. Матрица смежности. Измерение размера графа.

* Граф — это структура, представляющая собой набор объектов, в котором некоторые пары объектов в некотором смысле «связаны».
* Объекты, называемые вершинами (также называемыми узлами или точками).
* Каждая из связанных пар вершин называется ребром.

Параллельные вершины – два или более ребра, соединяющие одну и ту же пару вершин.

Петля – ребро, которое начинается и заканчивается в одной вершине.

Ориентированный, взвешенный

Количество ребер, инцидентных вершине, называется степенью вершины или **deg(v)**. Вершина инцидентна ребру, если эта вершина является одной из двух вершин, которые соединяет ребро.

Листовая вершина — это вершина с **deg(v)** = 1.

Изолированная вершина — это вершина с **deg(v)** = 0.

Лемма о рукопожатии: , где Е – количество рёбер (сумма степеней всех вершин равна количеству ребер, умноженному на 2)

Граф называется связным, если в нем нет вершины с deg = 0.

Каждый связный подграф называется компонентом.

Путь в графе — это последовательность ребер, соединяющая последовательность вершин.

Цикл — это путь, который начинается и заканчивается в одной и той же вершине.

3 обычных вида представления։

* + Список ребер
  + Матрица смежности
  + Список смежности (1 {2, 3}, 2 {1, 3, 5})

Измерение размера графа – ?

3. Поиск в графе и его применения. Обобщенный графовый поиск. Поиск в ширину и в глубину.

В ширину:

* + BFS отмечает каждый узел, посещенный только один раз
  + BFS проверить каждое ребро
  + Также ***O(|V| + |E|)*** если мы используем список соединений
  + И ***O()*** если мы используем матрицу смежности (потому что нам нужно найти каждое ребро, поэтому нужно перебрать всю матрицу).

В глубину:

* + DFS отмечает каждый узел, посещенный только один раз
  + DFS проверить каждое ребро
  + Также ***O(|V| + |E|)*** если мы используем список соединений
  + И ***O()*** если мы используем матрицу смежности (потому что нам нужно найти каждое ребро, поэтому нужно перебрать всю матрицу).

Затраты памяти при поиске в глубину кратно выше, чем при поиске в ширину

4. Поиск в глубину. Топологическая сортировка. Вычисление топологического упорядочивания.

Топологическая сортировка вершин ориентированного графа без циклов.

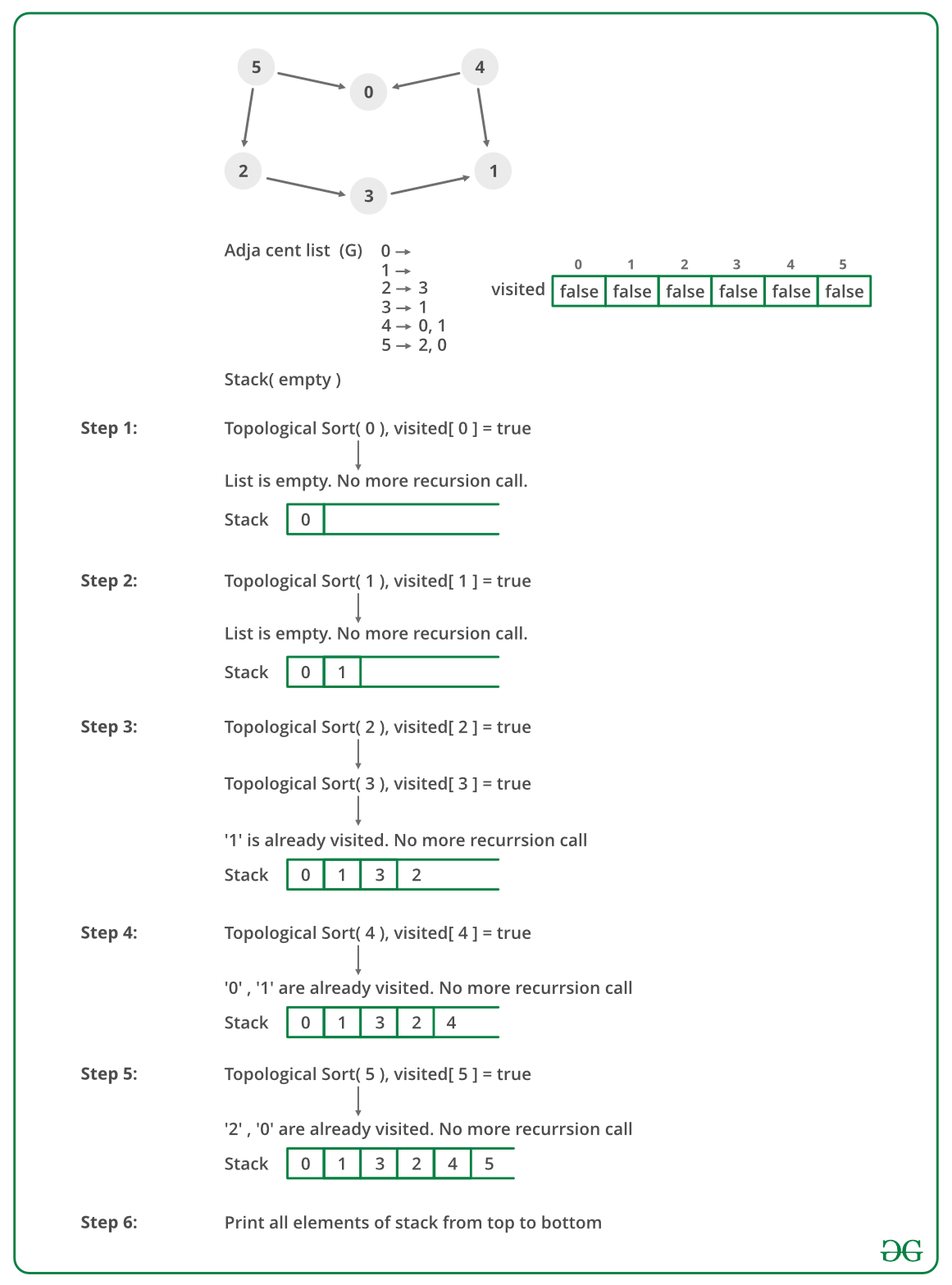
Задача: выстроить работы в последовательности, в которой никакая следующая задача не может зависеть от предыдущей (дуги направлены только вперед)

«Наивный» алгоритм нумерации вершин:

1. Находим какую-либо вершину, в которую не входят дуги, нумеруем ее.
2. Помечаем дуги, выходящие из помеченной вершины, как «не существующие».
3. Повторяем шаги (1) и (2), пока не будут занумерованы все вершины.

«Эффективный» алгоритм нумерации вершин:

1. Производим обход графа с помощью рекурсивной процедуры обхода, начиная с произвольной вершины.
2. Нумеруем каждую вершину при «прохождении ее назад» максимальным из номеров (то есть нумерация происходит в порядке убывания номеров).
3. Повторяем шаги (1) и (2), пока не останется непройденных вершин.



5. Алгоритм кратчайшего пути Дейкстры

Основной цикл выполняется максимум *n* раз, в каждом из них на нахождение минимума тратится порядка *n* операций.

На циклы поиска по соседям тратится количество операций, пропорциональное количеству рёбер *m* (поскольку каждое ребро встречается в этих циклах ровно дважды и требует константное число операций). Таким образом, общее время работы алгоритма: 

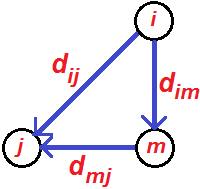
6. Алгоритм кратчайшего пути Флойда – Уоршелла

Обозначим через di,jm длину кратчайшего пути из вершины *i* в вершину *j*, который в качестве промежуточных может содержать только первые *m* вершин графа. Если в исходном графе нам известна длина каждой дуги, то мы можем сформировать матрицу D0, которая в алгоритме Флойда выступает в качестве исходной.

Вначале из этой матрицы вычисляется матрица D1. Затем по матрице D1 вычисляется матрица D2 и т. д. по формуле:

***di,jm =* min{ *di,mm-1 + dm,jm-1; di,jm-1*}**

Иными словами, можно ли короче пройти из вершины i в вершину j через вершину m.



7. Волновой алгоритм (Алгоритм Ли)

Алгоритм состоит из двух частей.

1. В первой от источника к приемнику распространяется волна.
2. Во второй выполняется обратный ход, в процессе которого из ячеек волны формируется путь.

Волна, идущая от источника к приемнику, на каждом шаге первой части алгоритма пополняется свободными ячейками, которые, во-первых, еще не принадлежат волне, и, во-вторых, являются 4-соседями ячеек, попавших в волну на предыдущем шаге.

При обратном ходе в путь включается по одной ячейке каждого шага распространения волны. При выборе из двух ячеек приоритет имеет ячейка, обеспечивающая горизонтальное продвижение.

8. Алгоритм кратчайшего пути Форда – Фалкерсона

пиздец

9. Минимальное остовное дерево. Алгоритм Прима

**Остовной связный подграф** – подграф графа G, который содержит все его вершины и каждая вершина достижима из любой другой.

**Остовное связное дерево** – подграф, включающий вершины исходного графа G, не содержащего циклы, каждая вершина которого достижима из любой другой.

Цикломатическое число **γ** показывает, сколько ребер нужно удалить из графа, чтобы в нем не осталось циклов

**γ = m – n + 1**,

m - количество ребер

n - количество вершин

Алгоритм Прима:

* Начало алгоритма: с произвольной вершины
* К текущему дереву присоединяется смежная вершина с кратчайшим ребром.
* Окончание алгоритма: либо все вершины подключены, либо невозможно подключить ни одно ребро.

10. Минимальное остовное дерево. Алгоритм Краскала

1. Удалить все ребра и получить остовной подграф с изолированными вершинами
2. Отсортировать ребра по возрастанию
3. Ребра последовательно, по возрастанию их весов, включаются в остовное дерево. Возможны случаи:

а) обе вершины включаемого ребра принадлежат одноэлементным подмножествам, тогда они объединяются в новое, связное подмножество

б) одна из вершин принадлежит связному подмножеству, другая нет, тогда включаем вторую в подмножество, которому принадлежит первая

в) обе вершины принадлежат разным связным подмножествам, тогда объединяем подмножества

г) обе вершины принадлежат одному связному подмножеству, тогда исключаем данное ребро

1. Алгоритм завершается, когда все вершины будут объединены в одно множество

11. Парадигма проектирования жадных алгоритмов. Жадный алгоритм Хаффмана

**Жадные алгоритмы**– это алгоритмы, склонные добиваться сиюминутного вознаграждения. Они делают выбор, который является *локально оптимальным*, надеясь, что в конце они придут к *глобальному оптимуму*.

**Жадность** – это простая стратегия, которая хорошо работает с некоторыми вычислительными задачами, но оказывается безуспешной с другими.

В типичном случае доказательство оптимальности следует такой схеме:

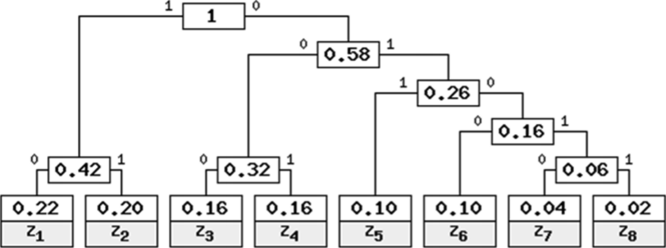
1. Доказывается, что жадный выбор на первом шаге не закрывает пути к оптимальному решению: для всякого решения есть другое, согласованное с жадным выбором и не хуже первого.
2. Показывается, что подзадача, возникающая после жадного выбора на первом шаге, аналогична исходной.
3. Рассуждение завершается по индукции.

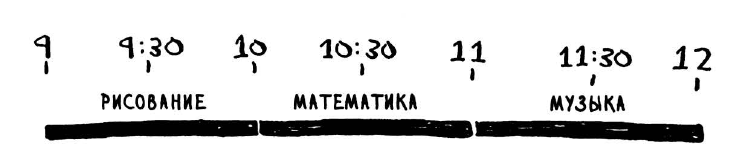
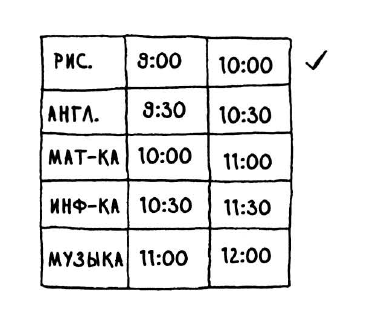
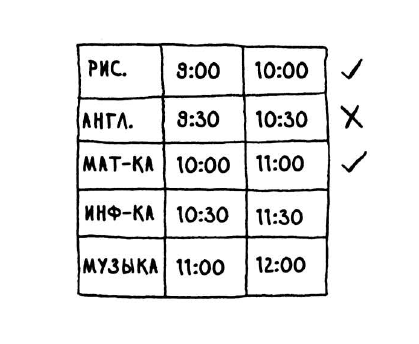
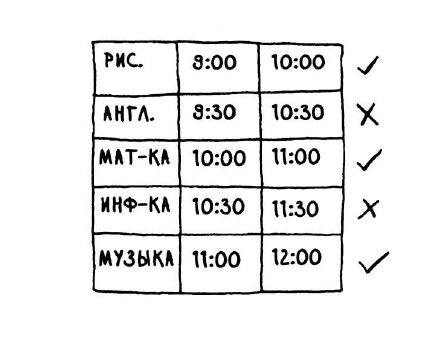
Информационной характеристикой алфавита (источника сообщений на основе этого алфавита) является *энтропия*:

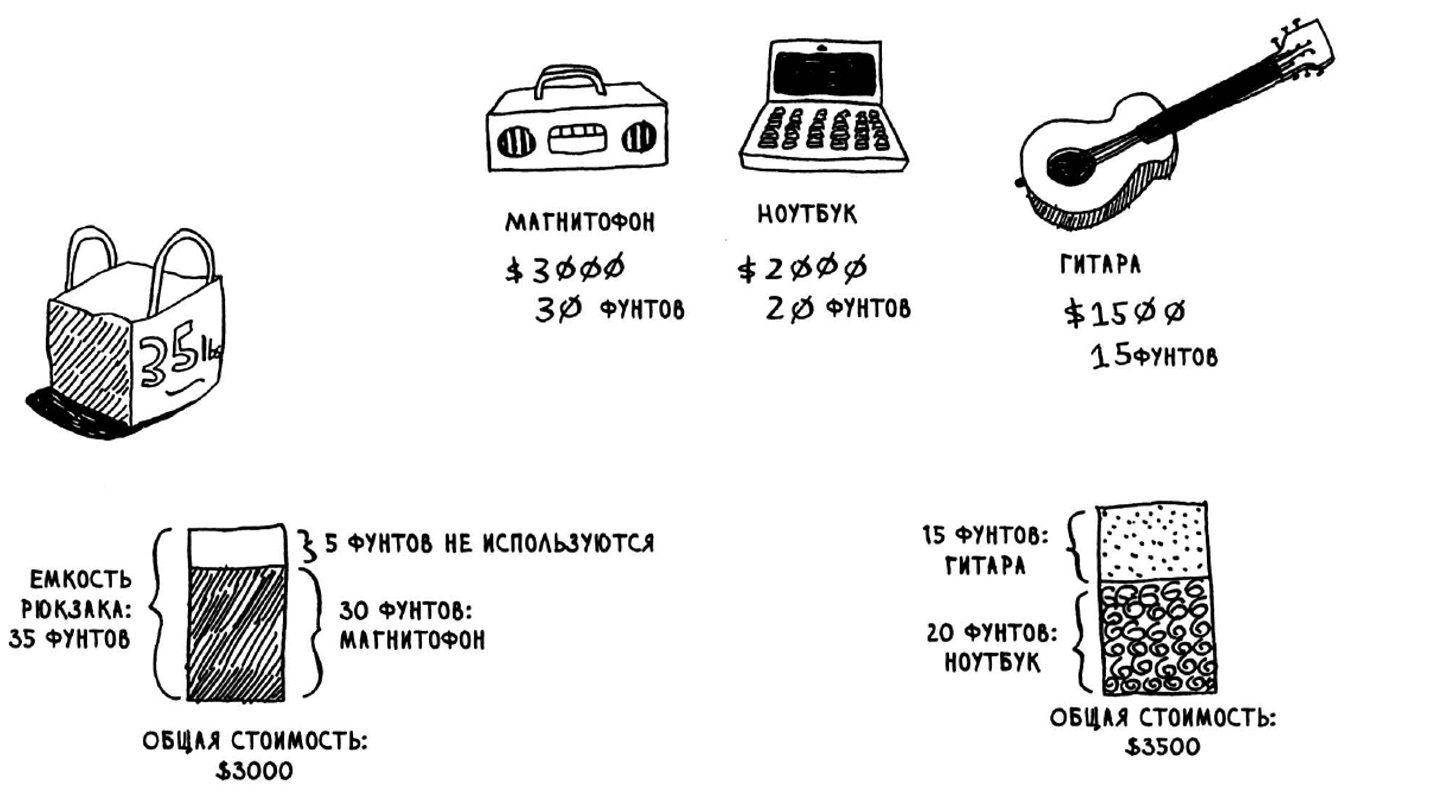


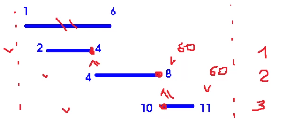
Код называется ***префиксным***, если он удовлетворяет ***условию Фано***:

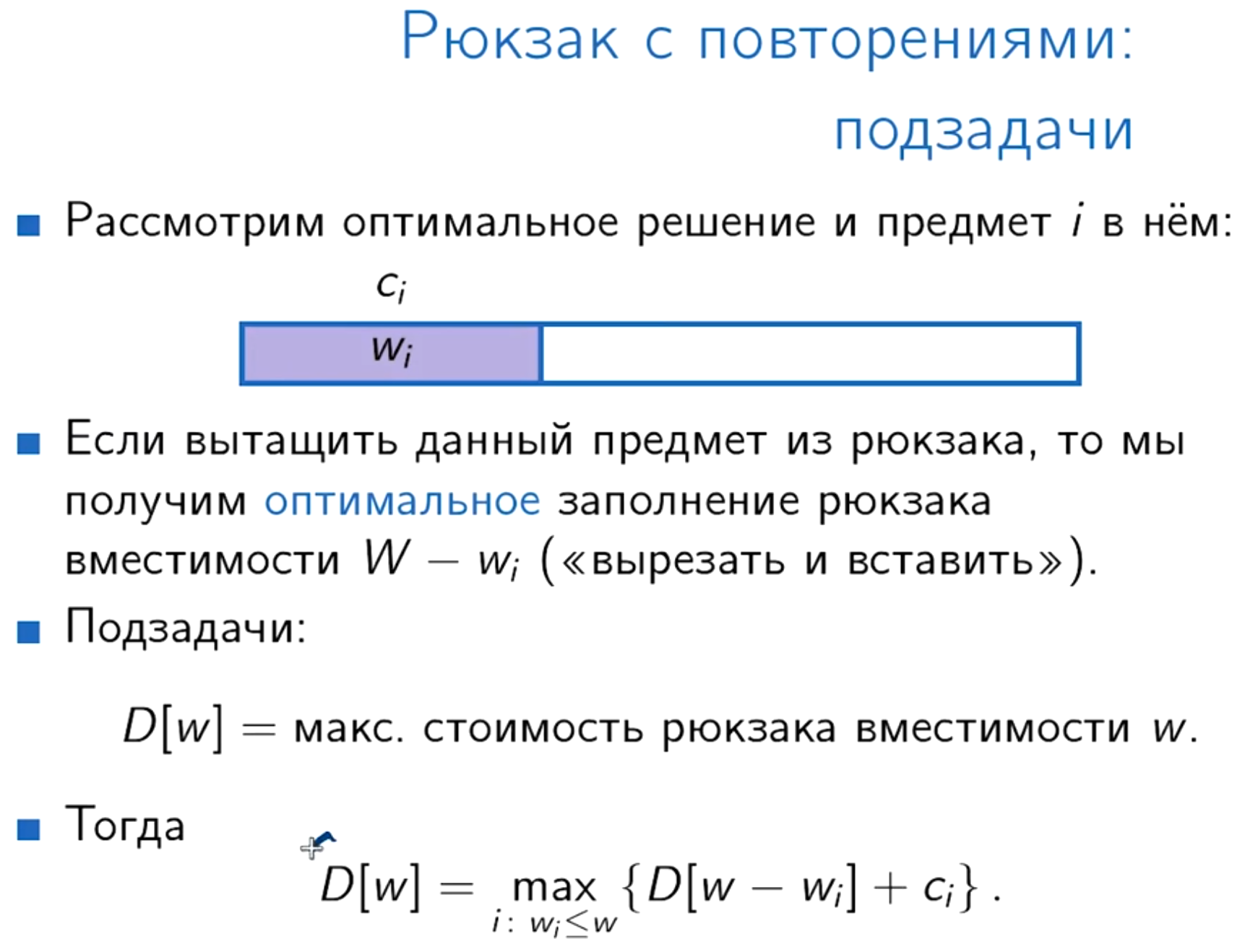
**Неравномерный код может быть однозначно декодирован, если никакой из кодов не совпадает с началом какого-либо иного, более длинного кода.**

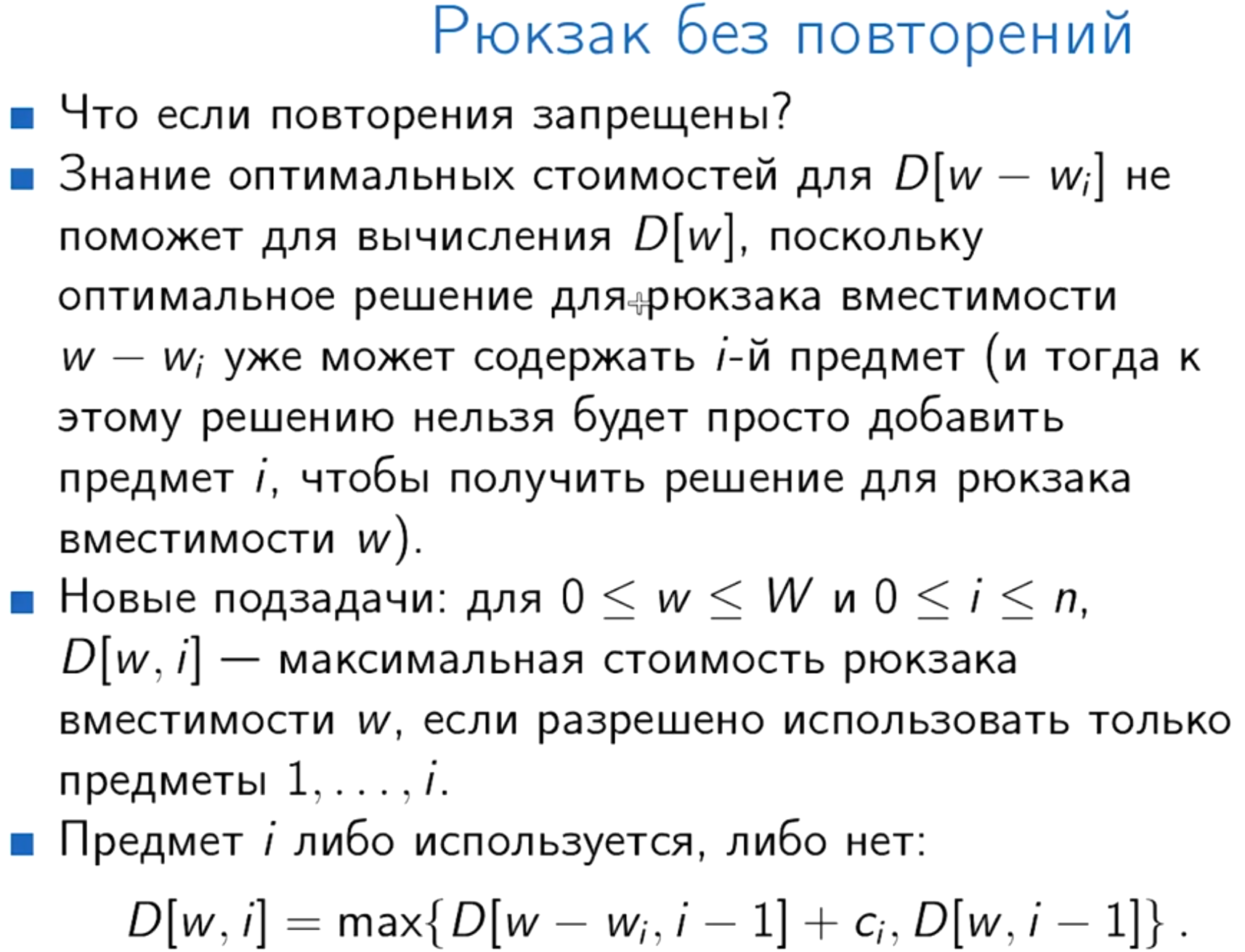


12. Парадигма проектирования жадных алгоритмов. Задача о составлении расписания. Задача о рюкзаке









13. Динамическое программирование. Числа Фибоначчи

**Динамическое программирование** – способ решения сложных задач путём разбиения их на более простые подзадачи.

Этот способ применим к задачам с оптимальной структурой, выглядящим как набор перекрывающихся подзадач, сложность которой меньше исходной.

**Оптимальная подструктура** в динамическом программировании означает, что оптимальное решение подзадач меньшего размера может быть использовано для решения исходной задачи.

Чтобы успешно решить задачу динамикой нужно:

1) Состояние динамики: параметр(ы), однозначно задающие подзадачу.

2) Значения начальных состояний.

3) Переходы между состояниями: формула пересчёта.

4) Порядок пересчёта.

**Мемоизация (запоминание, от англ. memoization (англ.) в программировании)**

сохранение результатов выполнения функций для предотвращения повторных вычислений. Это один из способов оптимизации, применяемый для увеличения скорости выполнения компьютерных программ. Перед вызовом функции проверяется, вызывалась ли функция ранее: если не вызывалась, функция вызывается и результат её выполнения сохраняется; если вызывалась, используется сохранённый результат.

• Фибоначчи

* 1. рекурсия
  2. массив (полная мемоизация)
  3. три переменные (частичная мемоизация)

• две единицы подряд

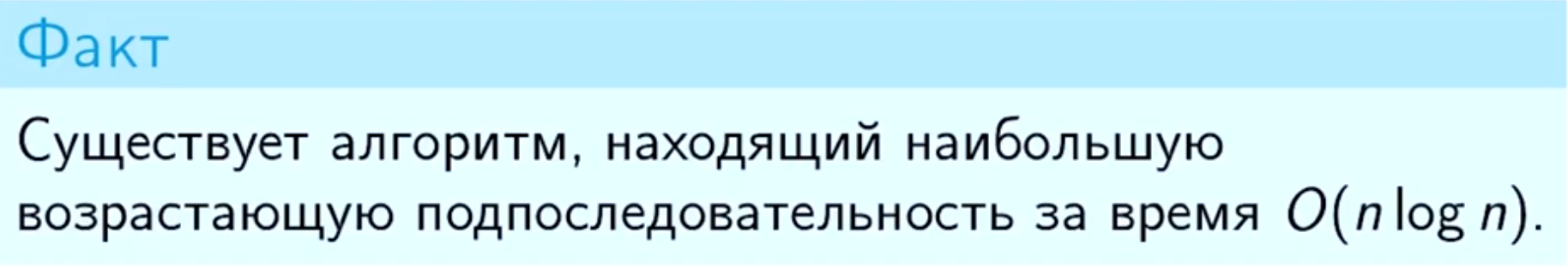
• самая длинная возрастающая подпоследовательность

• поиск пути в лабиринте

• поиск наибольшей общей подпоследовательности

• набрать точную сумму из набора чисел (2.5 способа)

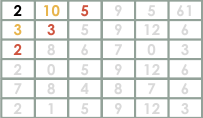
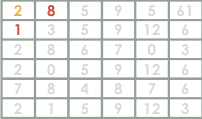
14. Динамическое программирование. Возрастающая подпоследовательность

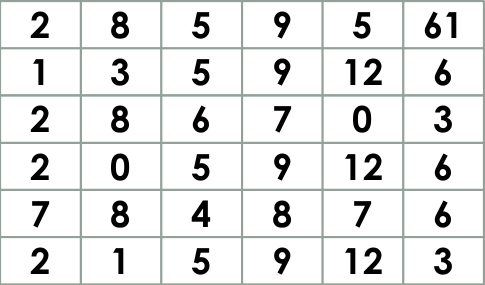
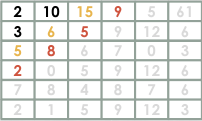
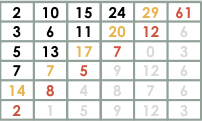
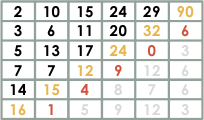
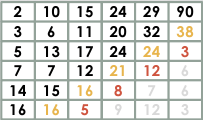
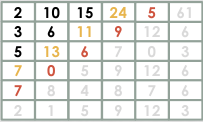


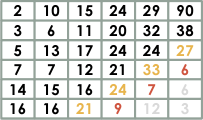
Создаём массив размера n, в который будем записывать индекс в наибольшей возрастающей подпоследовательности i-го элемента в исходном массиве по формуле D[i] = 1+max(D[j], где A[j]<A[i])

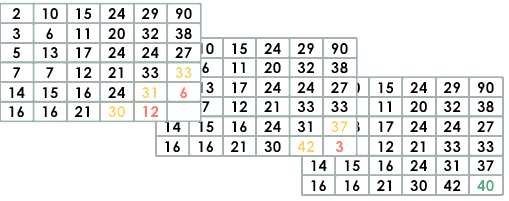
15. Динамическое программирование. Путь в лабиринте.

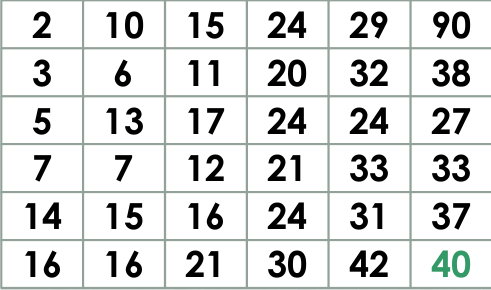
Дано прямоугольное поле размером **n**\***m** клеток. Можно совершать шаги длиной в одну клетку вправо или вниз. В каждой клетке записано некоторое натуральное число. Необходимо попасть из верхней левой клетки в правую нижнюю. Вес маршрута вычисляется как сумма чисел со всех посещенных клеток.

Необходимо найти маршрут с минимальным весом.

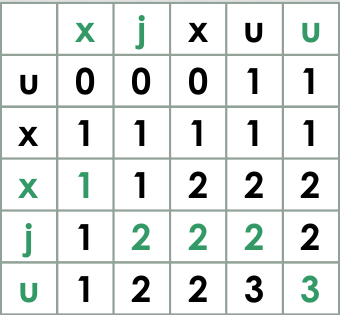
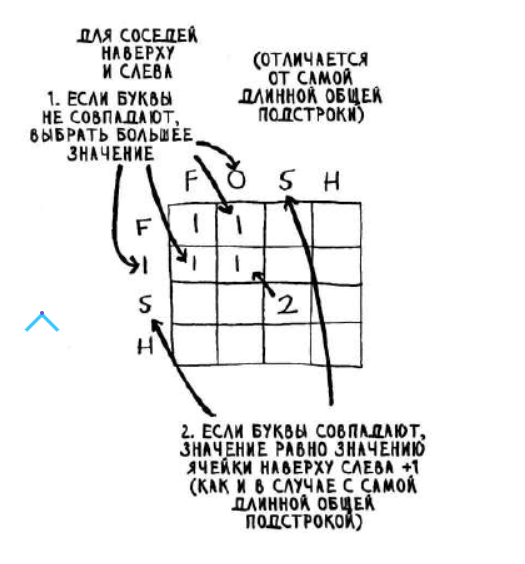




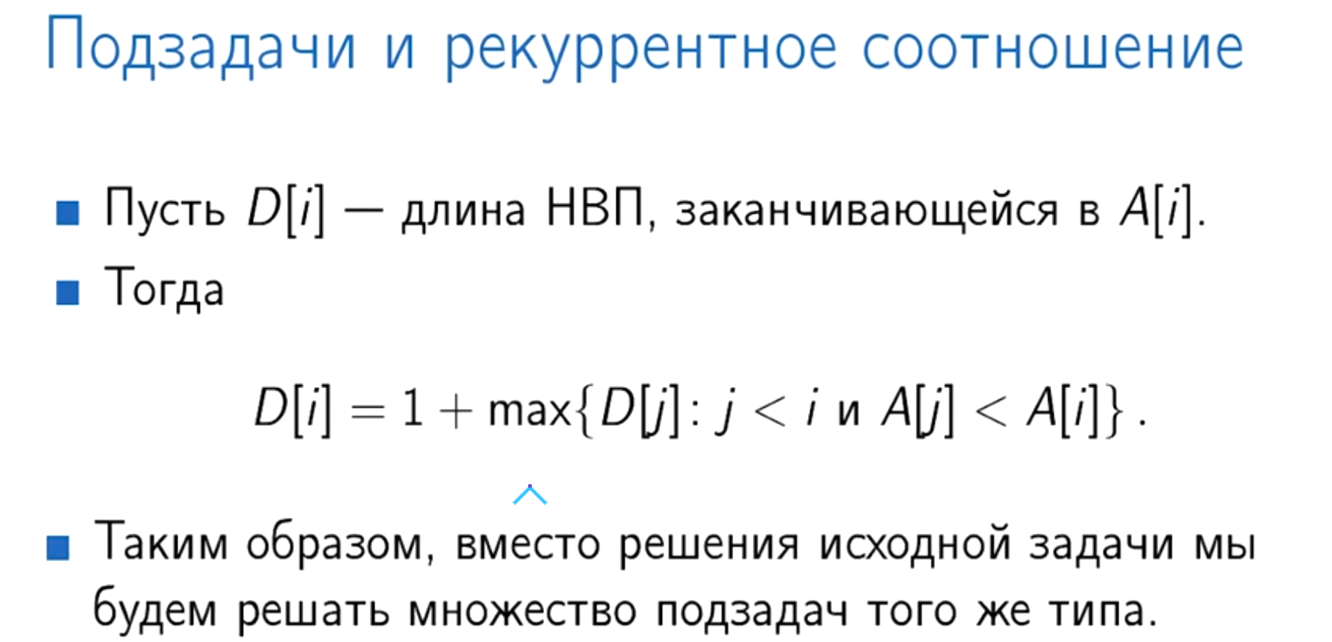




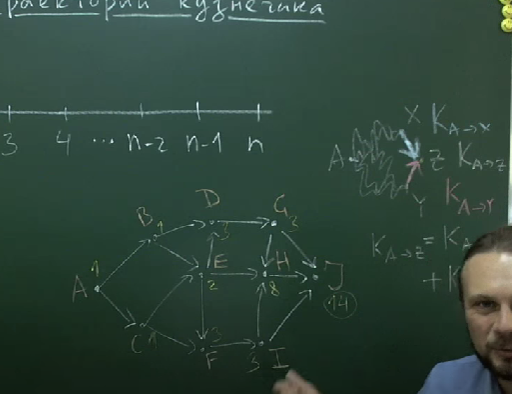
16. Динамическое программирование. Общая подпоследовательность



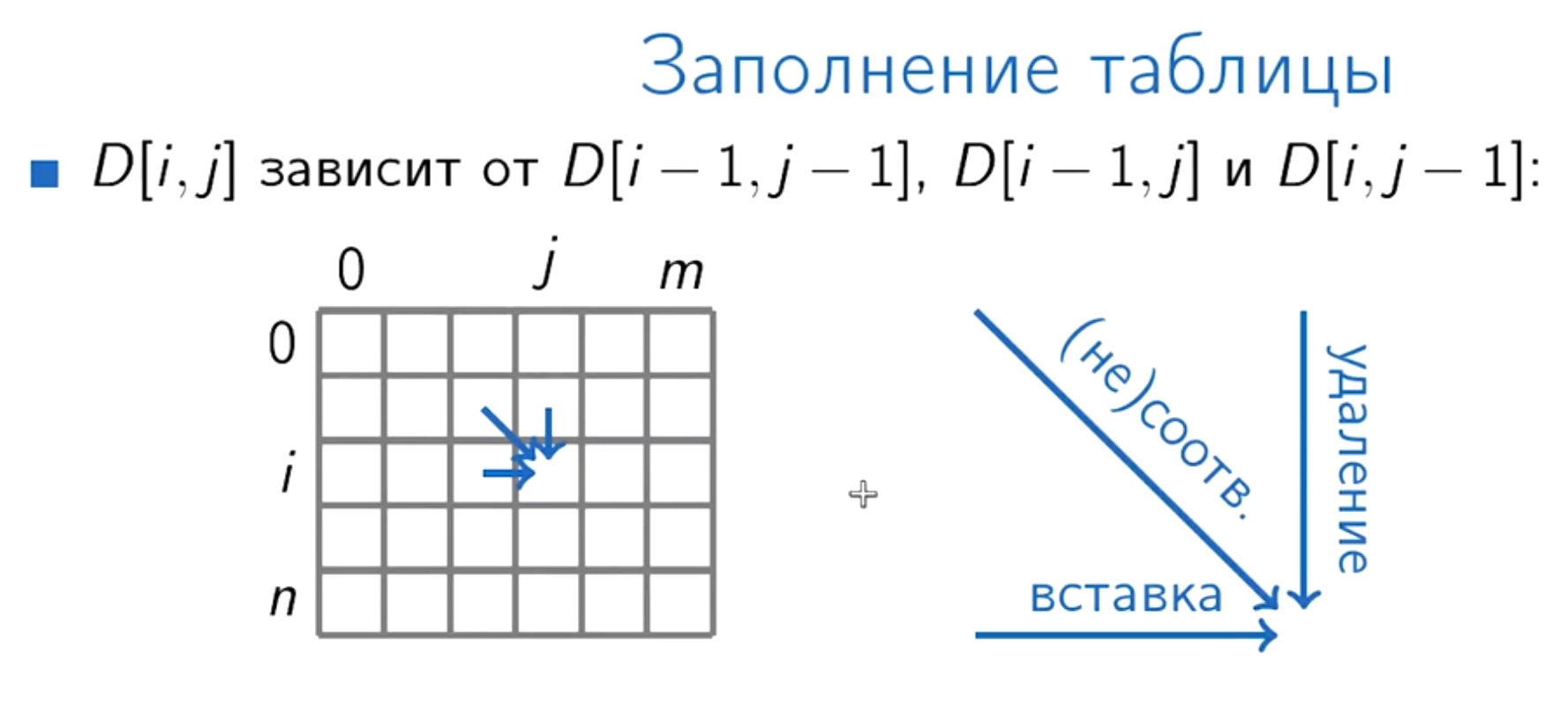
17. Динамическое программирование. Подзадачи и рекуррентные отношения

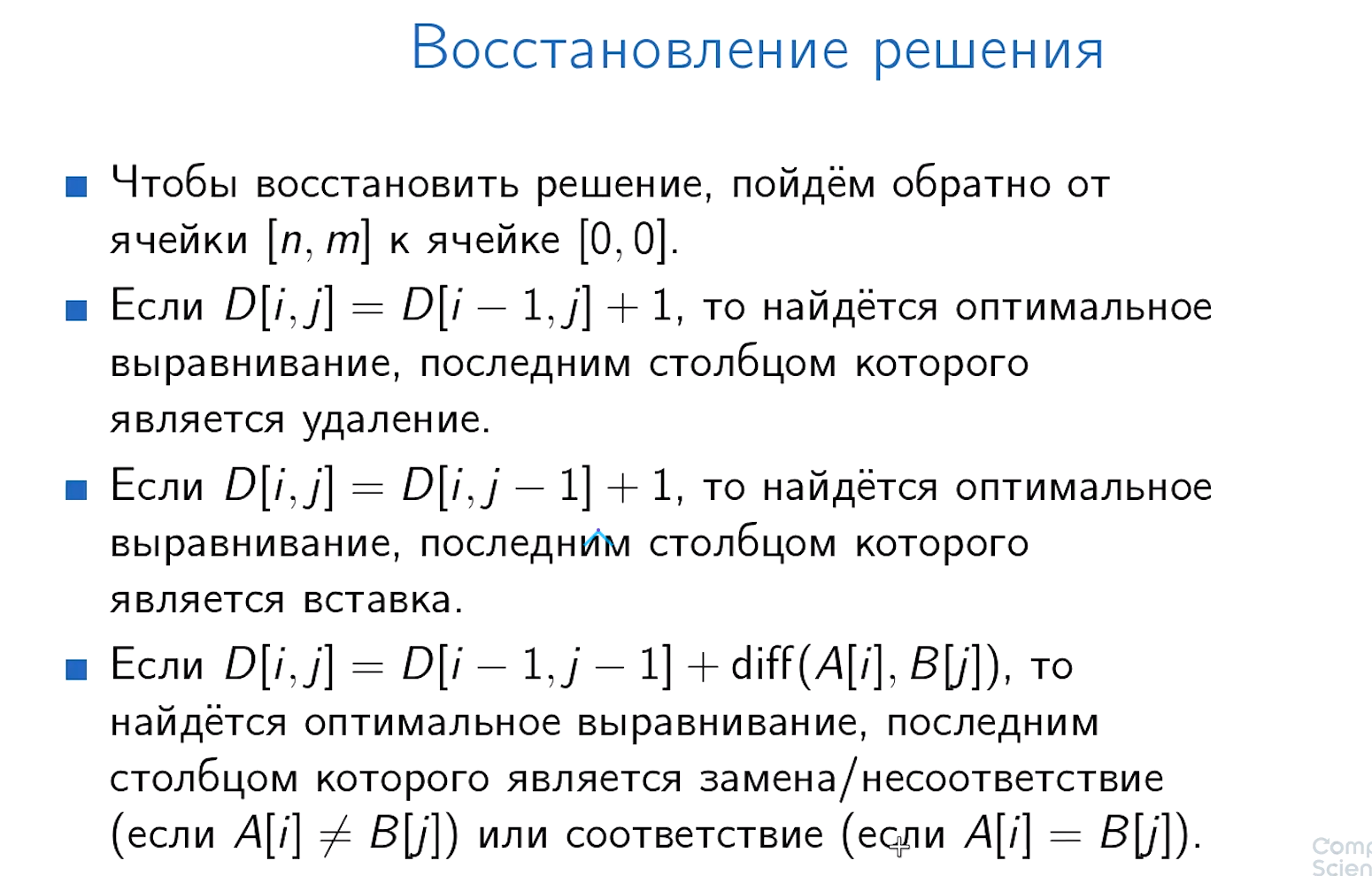


18. Динамическое программирование. Задача о лягушке



19. Динамическое программирование. Расстояние редактирования. Взвешенное расстояние редактирования



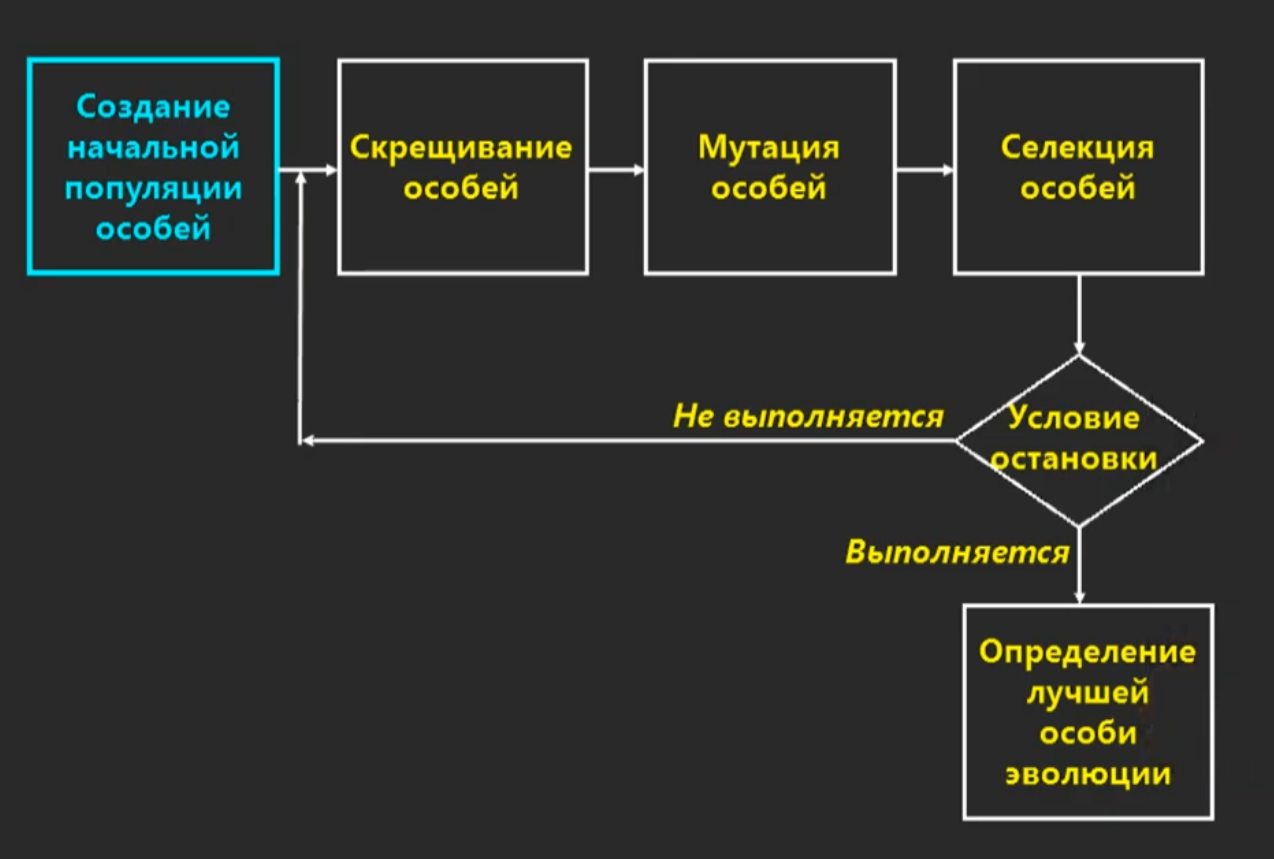


20. Динамическое программирование. Задача о рюкзаке. Задача о рюкзаке с повторением и без повторения

21. Динамическое программирование. Перемножение матриц

пиздец

22. Недетерминированные полиномиальные задачи (NP-задачи). Задача коммивояжера. Генетический алгоритм



23. Недетерминированные полиномиальные задачи (NP-задачи). Задача коммивояжера. Метод ветвей и границ (алгоритм Литтла)

24. Недетерминированные полиномиальные задачи (NP-задачи). Задача коммивояжера. Муравьиный алгоритм

24. Алгоритмы с возвратом. Задача о ходе коня. Задача о ферзях

25. Алгоритм k-ближайших соседей

26. Асимптотические обозначения. Математическое определение. Обозначение Омега-большое, Тета-большое и о-малое

27. Сортировки данных. Классы алгоритмов сортировки. Оценка алгоритмов сортировки. Обменная сортировка. Пузырьковая сортировка. Быстрая сортировка. Сортировка вставками

28. Динамические структуры данных. Вектор. Связанный список. Двухсвязный список. Стек. Очередь. Словарь. Множество

29. Поиск. Последовательный поиск. Бинарный поиск. Интерполяционный поиск. Поиск по деревьям

30. Рекурсия. Виды рекурсивных функций. Перебор с помощью рекурсии. Решение задач с помощью