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

Файлы: Алгоритмы и структуры данных. Лекция 5

Хеш-табли́ца — это [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию удаления и операцию поиска пары по ключу.

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.

С точки зрения практического применения, хорошей является такая хэш-функция, которая удовлетворяет следующим условиям:

- функция должна быть простой с вычислительной точки зрения;- функция должна распределять ключи в хэш-таблице наиболее равномерно;- функция не должна отображать какую-либо связь между значениями ключей в связь между значениями адресов;- функция должна минимизировать число коллизий, то есть ситуаций, когда разным ключам соответствует одно значение хэш-функции (ключи в этом случае называются синонимами).

Наихудший случай: все ключи хэшируются в один индекс.Варианты хэш-функции - Метод деления - Метод умножения - Универсальное хэширование

Хеш-таблицы должны соответствовать следующим свойствам:

- Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение является индексом в исходном массиве.- Количество хранимых элементов массива, деленное на число возможных значений хеш-функции, называется коэффициентом заполнения хеш-таблицы (load factor, fill factor) и является важным параметром, от которого зависит среднее время выполнения операций. Оптимальное значение этого коэффициента не должно превышать 0,7.- Операции поиска, вставки и удаления должны выполняться в среднем за время O(1). Однако при такой оценке не учитываются возможные аппаратные затраты на перестройку индекса хеш-таблицы, связанную с увеличением значения размера массива и добавлением в хеш-таблицу новой пары.

Коллизии

- Мы не хотим выделять память на каждое возможное значение элемента (реально встретившихся значений обычно много меньше, чем возможных)- Значит, возможных значений h(x) меньше, чем возможных значений x и существуют такие x1, x2, что h(x1)=h(x2)- Значит, возможна ситуация, когда мы пытаемся добавить элемент, а место занято.- Эта ситуация называется коллизией

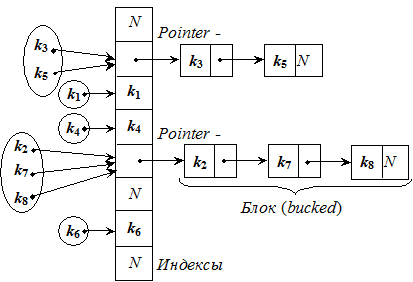
Необходимо разрешение коллизий

Правила разрешения коллизий должны определять, что делать при коллизии (куда поместить полученный элемент)Важно обеспечить, чтобы:- Правила разрешения коллизий позволяли бы разместить в контейнере любой набор значений- Правила поиска позволяли найти любой элемент, размещенный по правилам разрешения коллизий

Разрешение коллизий: открытое хеширование (хранение списков)

- Будем хранить в каждом элементе массива не значение, а список значений- Новое значение добавляем в конец списка- Поиск выполняется по списку

Разрешение коллизий хранением списков (открытое хеширование)

- В наихудшем случае время поиска O(N) – если возникнет один список- Время добавления элемента в наихудшем случае – O(N) или O(1) [если хранить адрес последнего элемента списка]Предположим, что:- Вероятности попадания элемента в любую ячейку равны- Количество ячеек M равно количеству элементов N (или хотя бы пропорционально)- Тогда средняя длина списка – 1, среднее время поиска и добавления элемента – O(1)

Разрешение коллизий: закрытое хеширование (метод сдвига)

Часто хочется упростить структуру и не хранить массив списковВ этом случае можно применить разрешение коллизий методом сдвига (хеширование с открытой адресацией, метод линейного исследования)Если мы не можем положить элемент в нужную ячейку – пытаемся положить в следующую, и так пока не найдется свободнаяПри поиске перебираем элементы, пока не встретим пустую ячейкуВстретив конец массива – переходим на первый элемент

Разрешение коллизий методом сдвига

* Метод работает, только если длина массива не меньше числа элементов
* Когда элементов в массиве становится достаточно много, эффективность хеширования мала (приходится перебирать множество элементов)
* Этот эффект называется кластеризацией (возникает кластер из занятых элементов)

Разрешение коллизий: квадратичное исследование

* При попытке в № i поместить значение k мы пробуем ячейку h( k , i )h(k , i ) = (h’(k) + c1i + c2i2) mod m
* В отличие от линейного исследования, кластеризация слабее

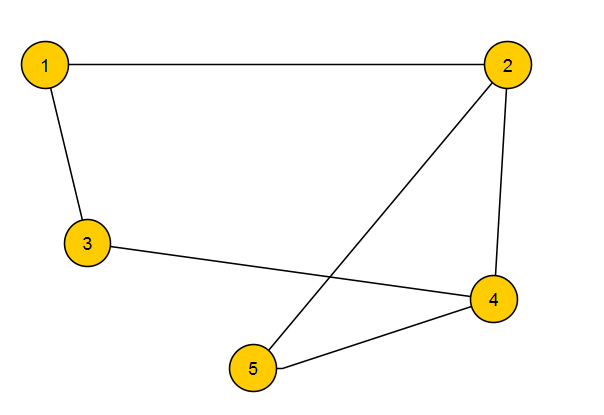
Удаление элементов

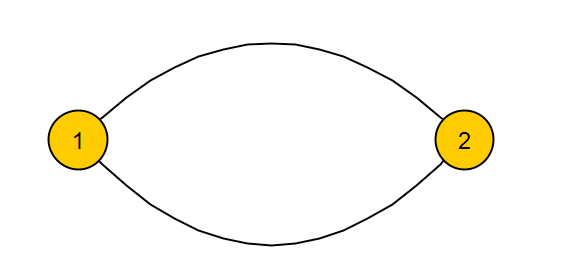
- Просто удалить элемент нельзя – нарушится поиск тех, которые были добавлены после него- Можно заменить значение на пометку Deleted

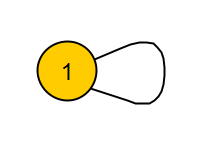
# **2. Графы. Представление графа. Матрица смежности. Измерение размера графа.**

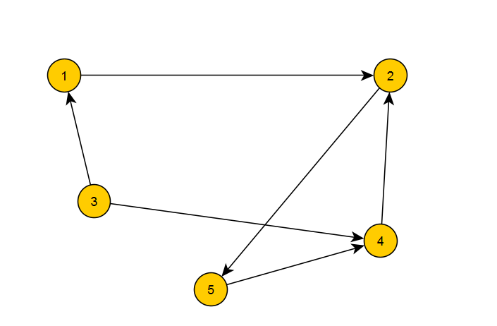
Граф — это структура, представляющая собой набор объектов, в котором некоторые пары объектов в некотором смысле «связаны».

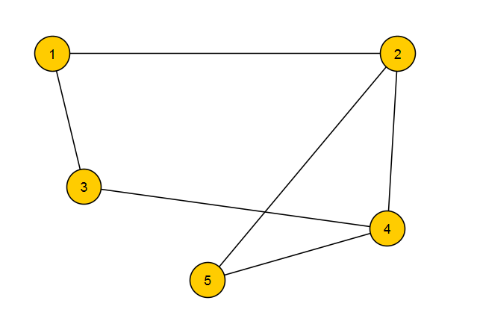
Объекты, называемые вершинами (также называемыми узлами или точками).Каждая из связанных пар вершин называется ребром.

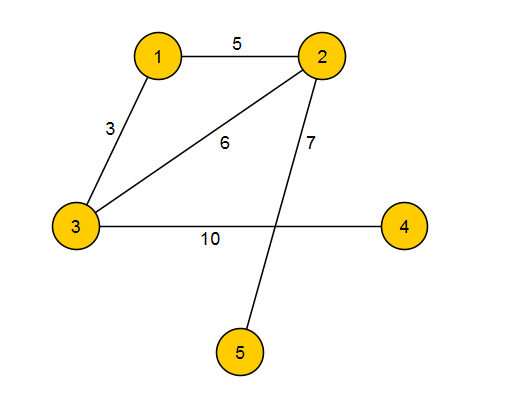
Вершины (желтые кружки) is V = {1,2,3,4,5}Ребра (черные линии) это пары вершин, которые связаны. В этом случае ребра :  
E = {(1, 2), (1, 3), (2, 4),   
 (2, 5), (3, 4), (4, 5)}Мы также определяем граф как пару V и E. Если коротко, то G = (V, E).  
Мы будем использовать это обозначение. 

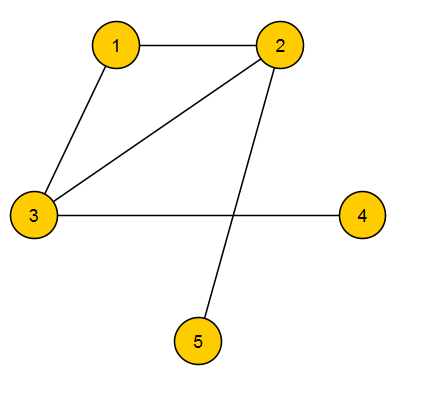
 Параллельные вершины :Два или более ребра, соединяющие одну и ту же пару вершин. В примере вершины 1 и 2 соединены с 2 ребрами.

ПетляРебро, которое начинается и заканчивается в одной вершине.

Если ребра в графе ориентированы, т. е. указывают только в одном направлении, граф называется ориентированным графом или орграфом.

Если ребра в графе не имеют направления, граф называется неориентированным графом.

Граф, в котором каждое ребро имеет числовой «вес», называется взвешенным графом.



Терминология

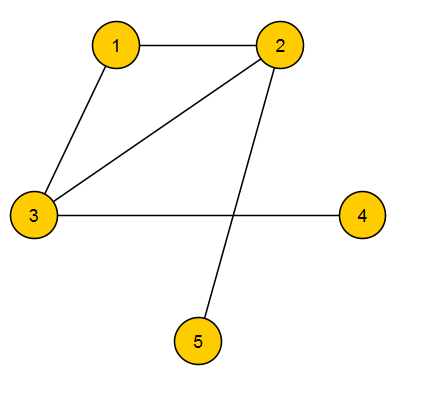
- Вершины u и v называются смежными, если u и v соединены некоторым ребром.- Количество ребер, инцидентных вершине, называется степенью вершины или deg(v). Вершина инцидентна ребру, если эта вершина является одной из двух вершин, которые соединяет ребро. Например, град(3) = 3- Листовая вершина — это вершина с deg(v) = 1, например, 5 — листовая вершина.

Изолированная вершина — это вершина с deg(v) = 0 , например, 6 — изолированная вершина.

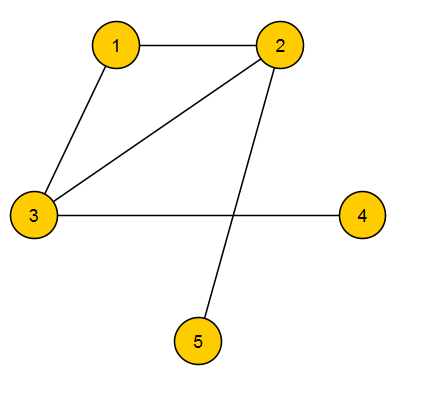
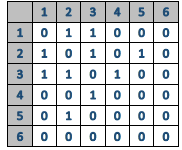
Граф называется связным, если в нем нет вершины с deg = 0.Каждый связный подграф называется компонентом.Путь в графе — это последовательность ребер, соединяющая последовательность вершин. Например, 4 → 3 → 2 → 5Цикл — это путь, который начинается и заканчивается в одной и той же вершине.Например, 3 → 2 → 1 → 3

Представление- Нужно представить график в компьютере.- 3 обычных вида представления։

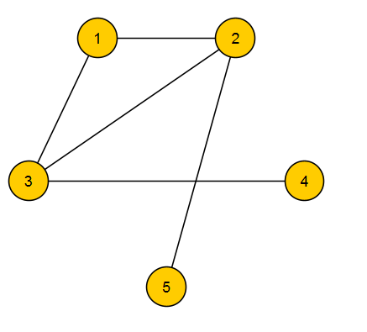
- Список ребер



{1, 2}, {2, 1}   
{1, 3}, {3, 1}   
{2, 3}, {3, 2}  
{2, 5}, {5, 2}  
{3, 4}, {4, 3}

- Матрица смежности 

- Список смежности

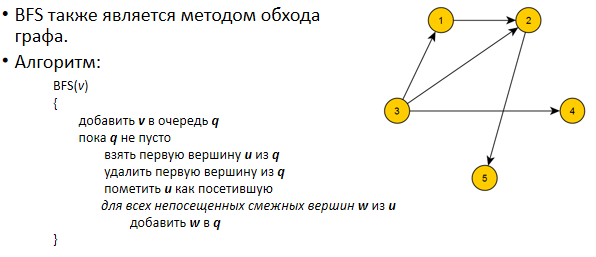
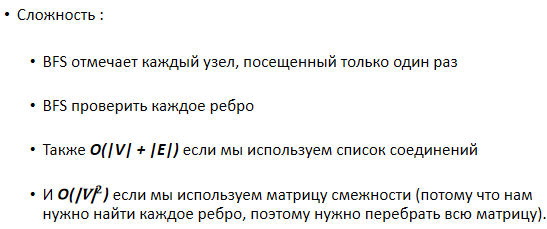
1 → {2, 3}   
2 → {1, 3, 5}  
3 → {1, 2, 4}  
4 → {3}  
5 → {2}  
6 → {}

Измерение размера графа

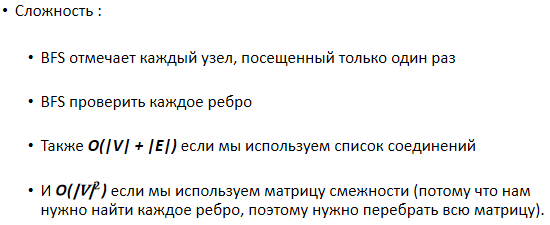
# **3. Поиск в графе и его применения. Обобщенный графовый поиск. Поиск в ширину и в глубину.**

Поиск

Breadth First Search (BFS)Поиск в ширину

Depth First Search (DFS)

Поиск в глубину

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

Посик в глубину

**https://www.youtube.com/watch?v=Tzc7Z-mOwxY&ab\_channel=%D0%A4%D0%BE%D0%BA%D1%81%D1%84%D0%BE%D1%80%D0%B4**

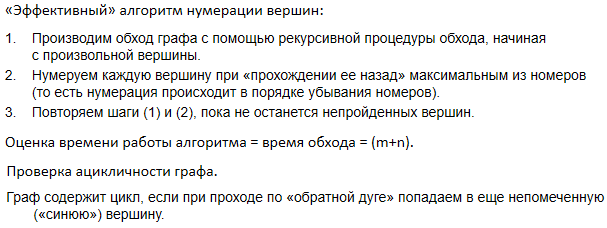
Сложность :- BFS отмечает каждый узел, посещенный только один раз- BFS проверить каждое ребро- Также O(|V| + |E|) если мы используем список соединений- И O(|V|^2) если мы используем матрицу смежности (потому что нам нужно найти каждое ребро, поэтому нужно перебрать всю матрицу).

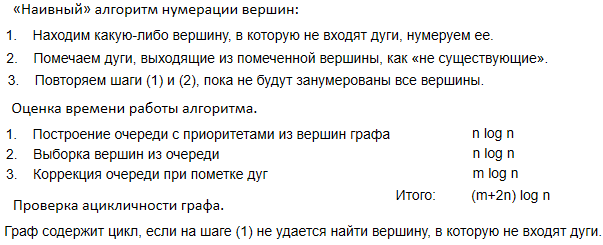
Топологическая сортировка

[**https://www.youtube.com/watch?v=spoelATn2UI&ab\_channel=%D0%94%D0%B0%D0%BD%D0%B8%D0%B8%D0%BB%D0%9F%D1%80%D0%BE%D1%85%D0%BE%D1%80%D0%BE%D0%B2**](https://www.youtube.com/watch?v=spoelATn2UI&ab_channel=%D0%94%D0%B0%D0%BD%D0%B8%D0%B8%D0%BB%D0%9F%D1%80%D0%BE%D1%85%D0%BE%D1%80%D0%BE%D0%B2)

Топологическая сортировка позволяет выстроить вершины бесконтурного ориентированного графа, в соответствии с логическими порядком и представить граф в виде уровней.

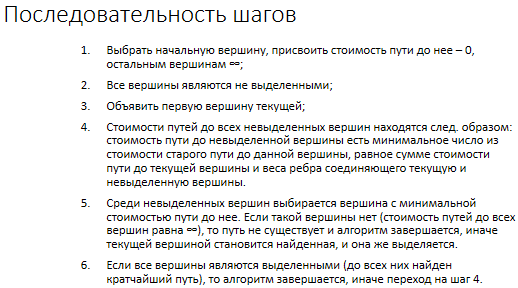
Вычисление топологического упорядочивания.

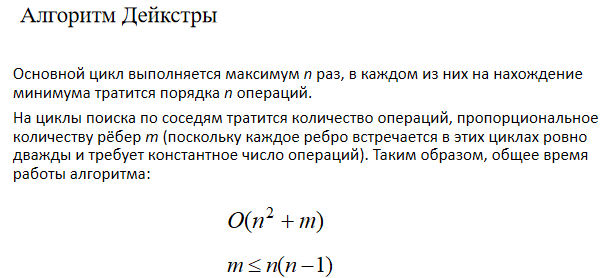
****

****

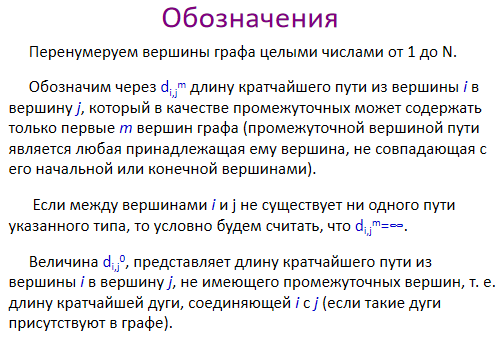
# **5. Алгоритм кратчайшего пути Дейкстры.**

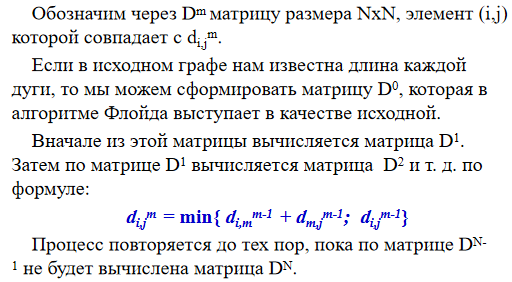
Алгоритм Дейкстры

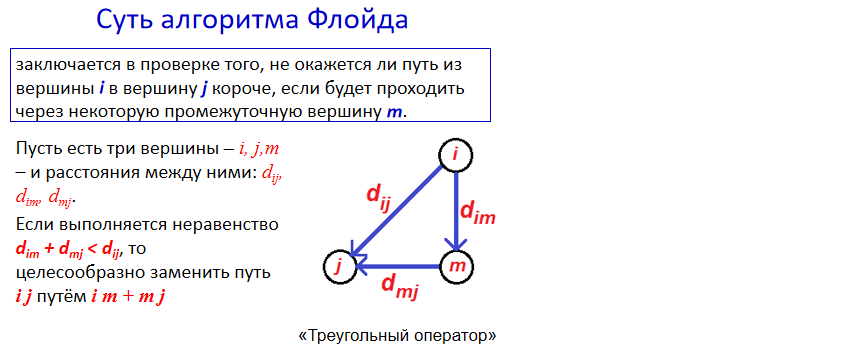


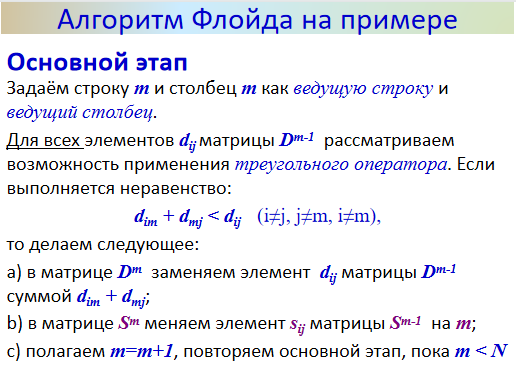


# **6. Алгоритм кратчайшего пути Флойда – Уоршелла.**

****

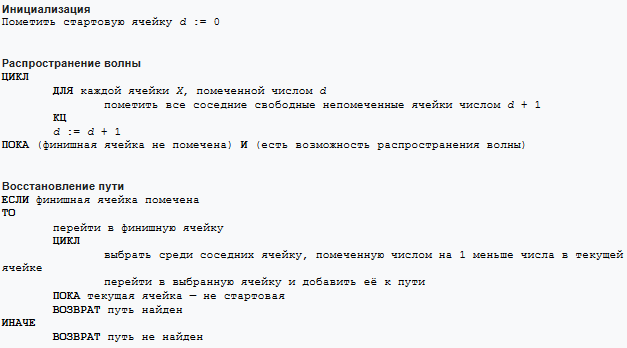
****

****

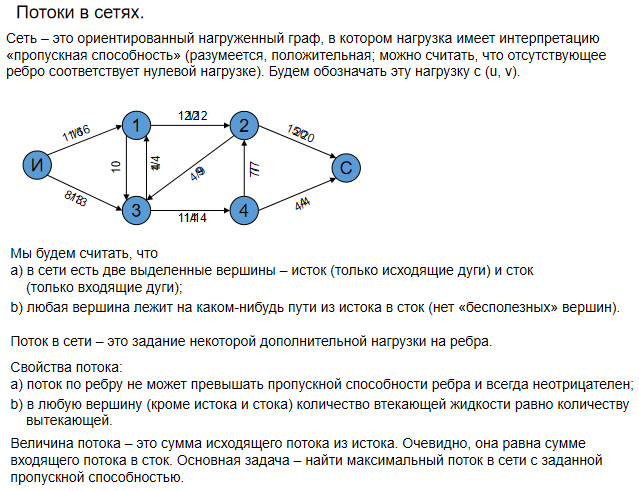
****

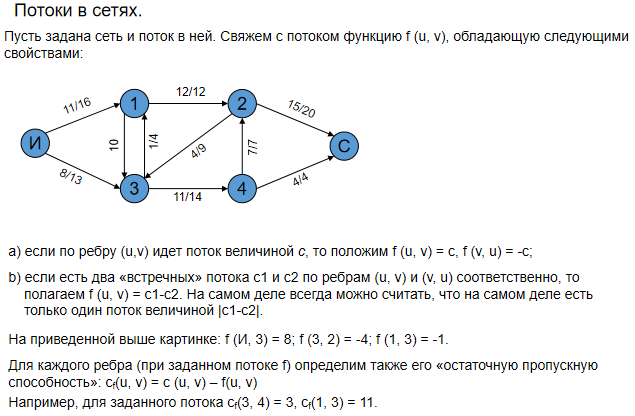
****

# **7. Волновой алгоритм (Алгоритм Ли).**

****

# **8. Алгоритм кратчайшего пути Форда – Фалкерсона.**

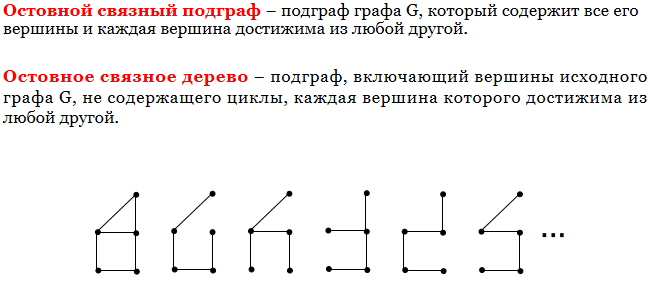
****

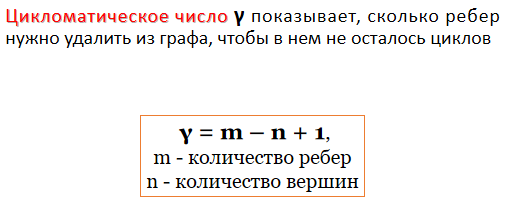
****

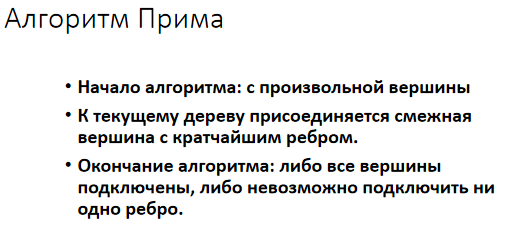


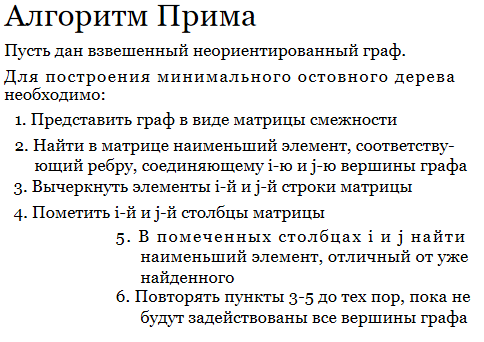
****

# **9. Минимальное остовное дерево. Алгоритм Прима.**

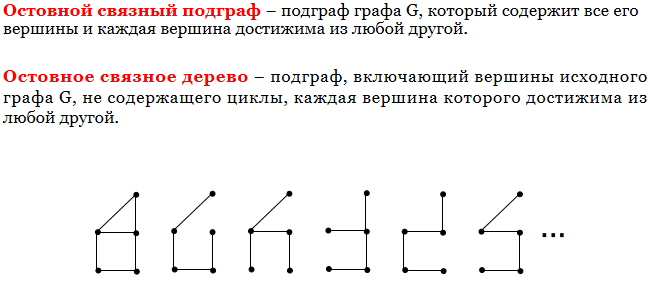
****

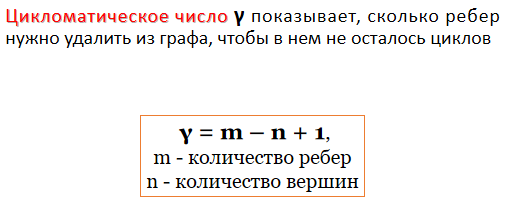
****

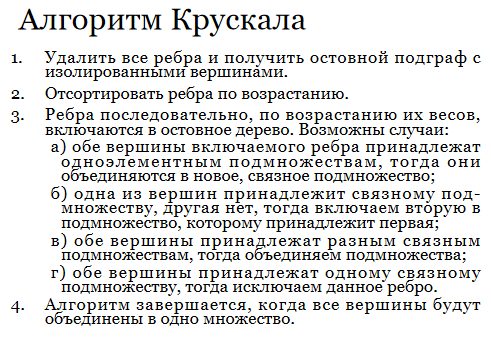
****

****

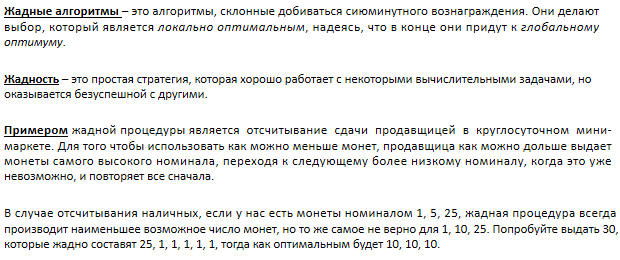
# **10. Минимальное остовное дерево. Алгоритм Краскала.**

****

****

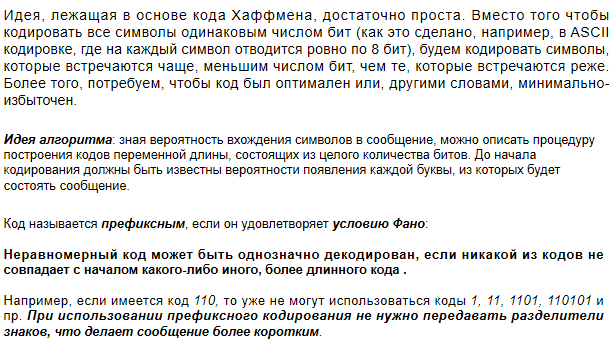
****

# **11. Парадигма проектирования жадных алгоритмов. Жадный алгоритм Хаффмана.**

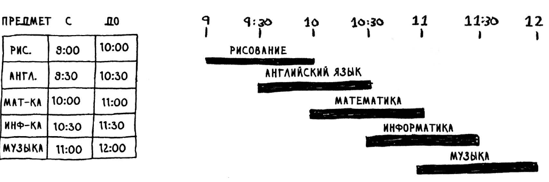
****

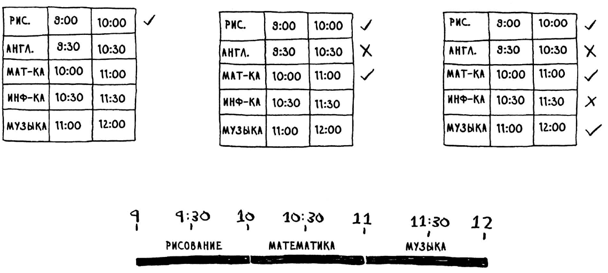
****

**алгоритм Хаффмана**

****

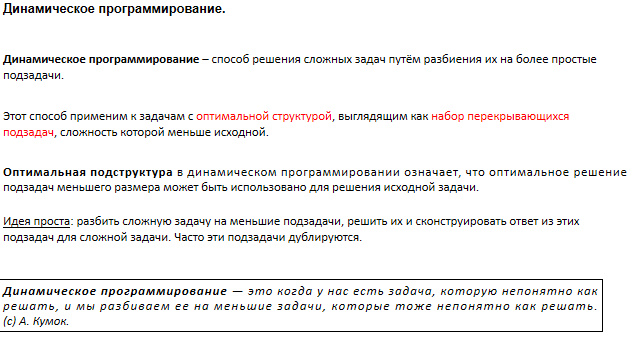
# **12. Парадигма проектирования жадных алгоритмов. Задача о составлении расписания. Задача о рюкзаке.**

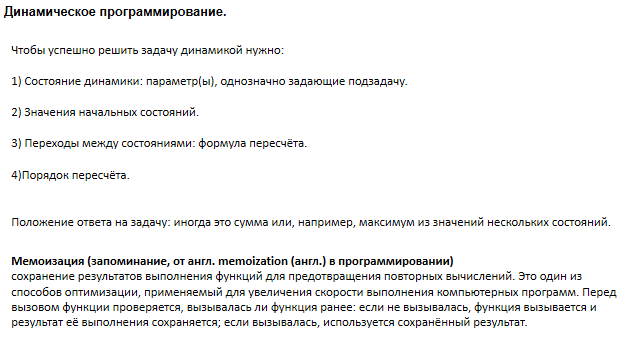
****

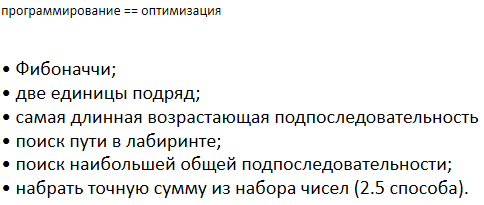
****

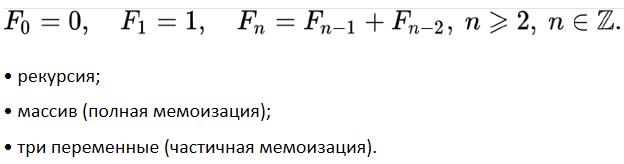
****

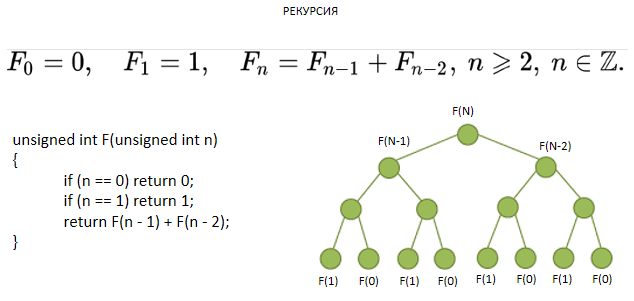
# **13. Динамическое программирование. Числа Фибоначчи.**

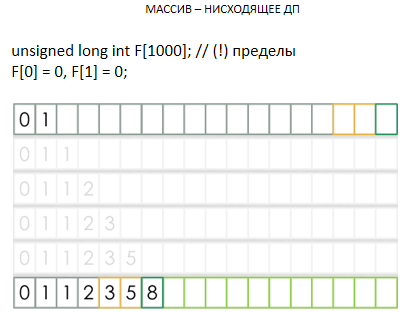
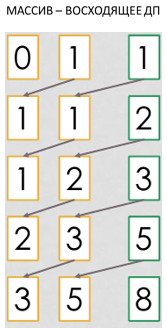
****

****

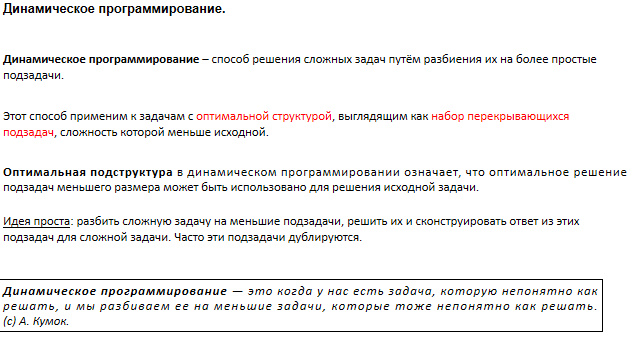
****

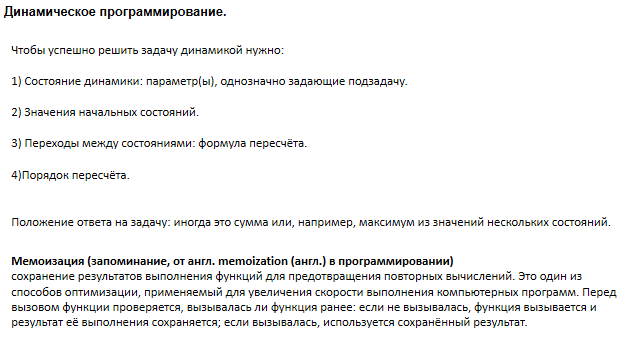
****

****

**** 

# **14. Динамическое программирование. Возрастающая подпоследовательность.**

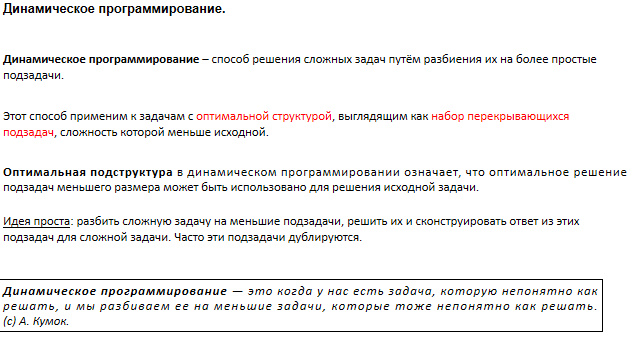
****

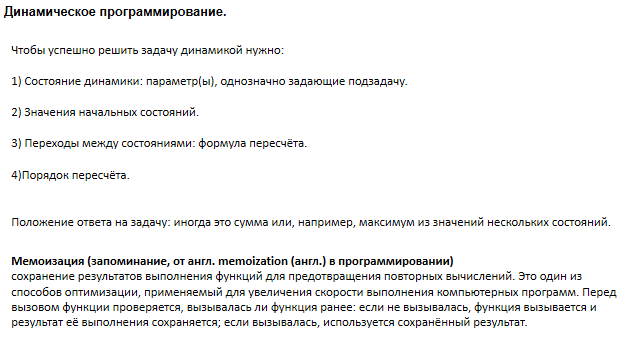
****

****

****

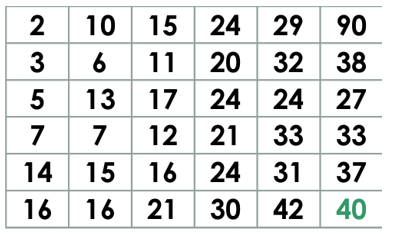
# **15. Динамическое программирование. Путь в лабиринте.**

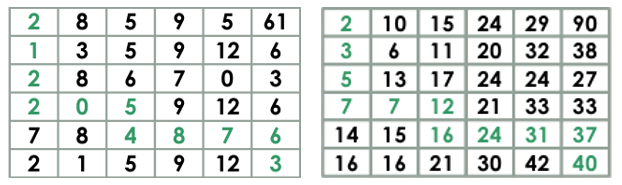
****

****

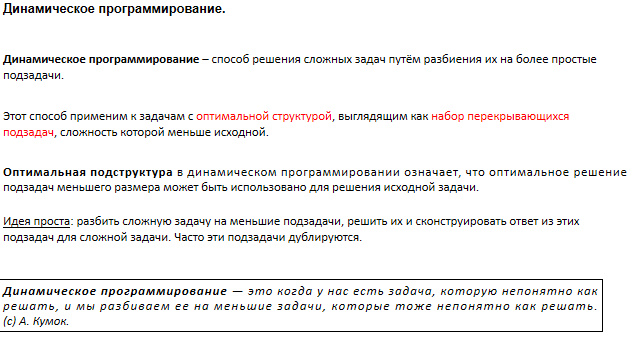
****

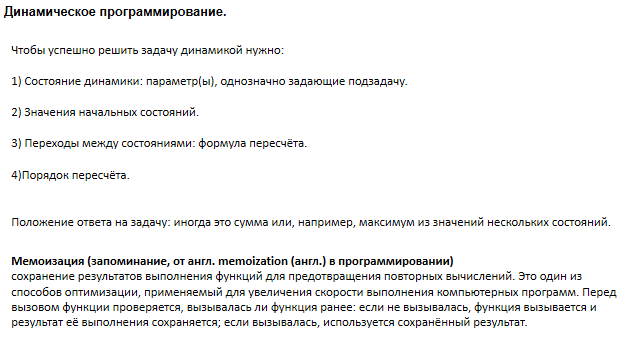
****

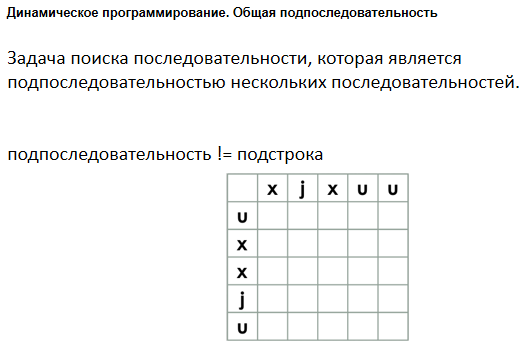
****

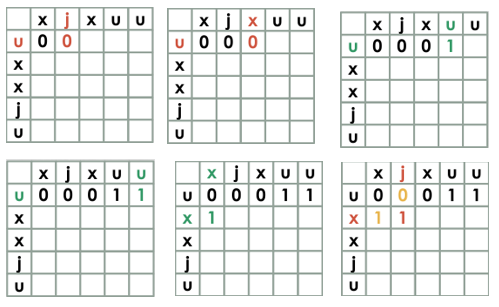
****

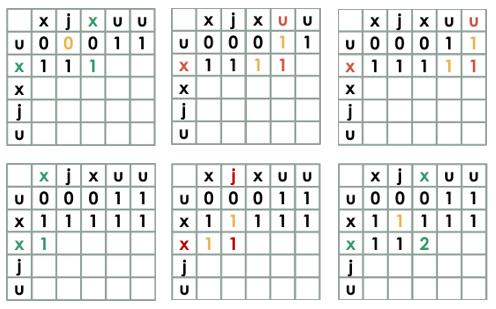
# **16. Динамическое программирование. Общая подпоследовательность.**

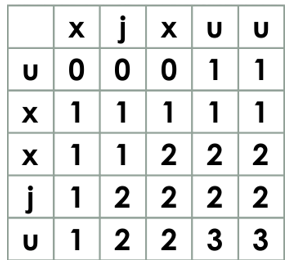
****

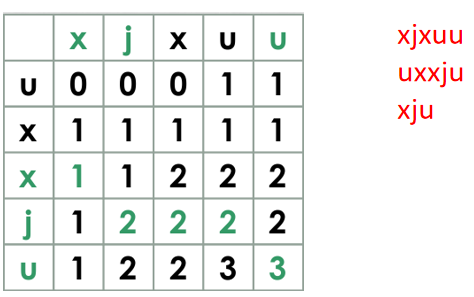
****

****

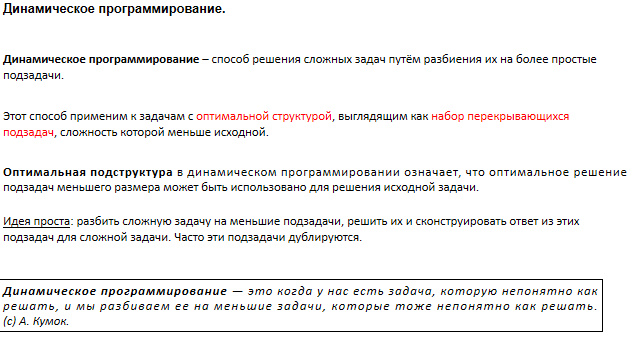
****

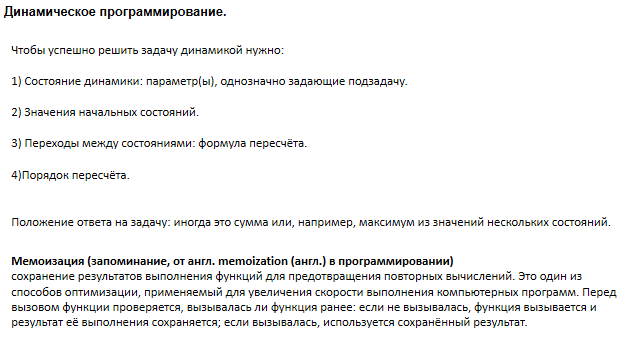
****

****

****

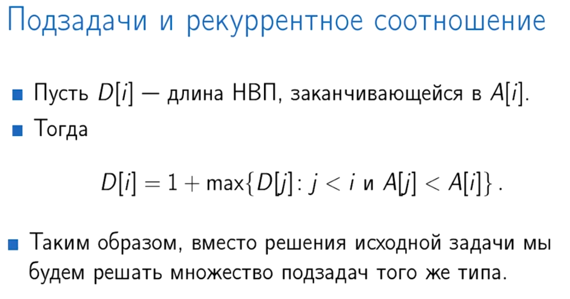
# **17. Динамическое программирование. Подзадачи и рекуррентные отношения.**

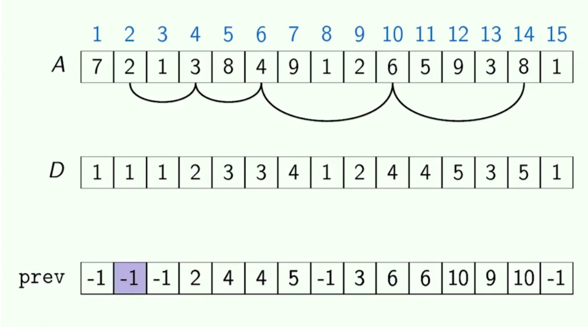
****

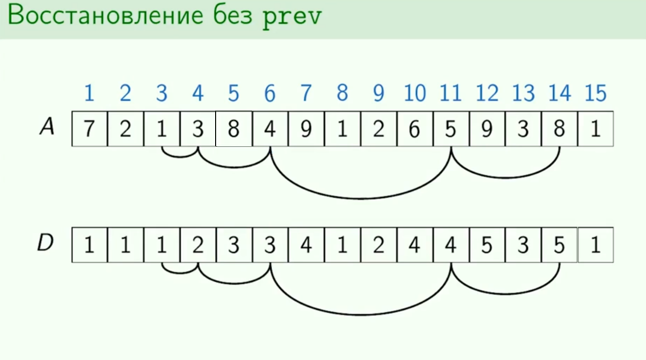
****

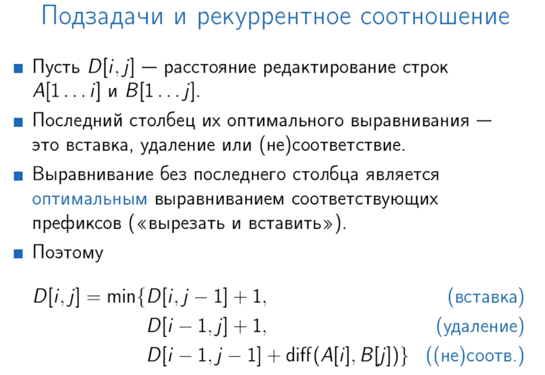
Рекурентная последовательность – последовательность заданная формулой, где каждый член выражается через один или несколько членов этой последовательности

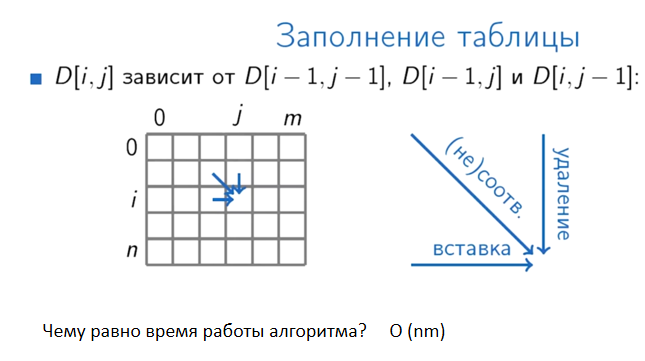
https://www.youtube.com/watch?v=BTJHGGiGcoc&ab\_channel=%D0%A4%D0%BE%D0%BA%D1%81%D1%84%D0%BE%D1%80%D0%B4

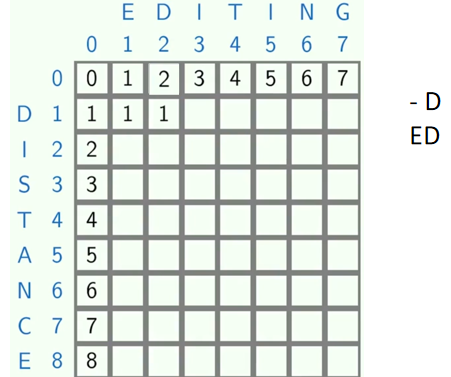
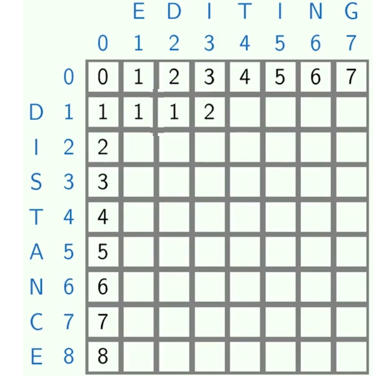
****

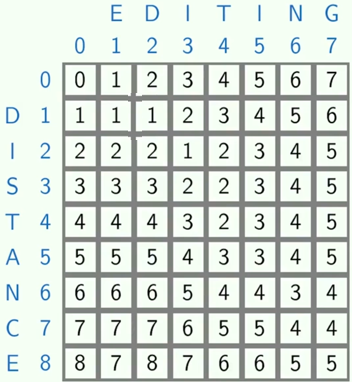
****

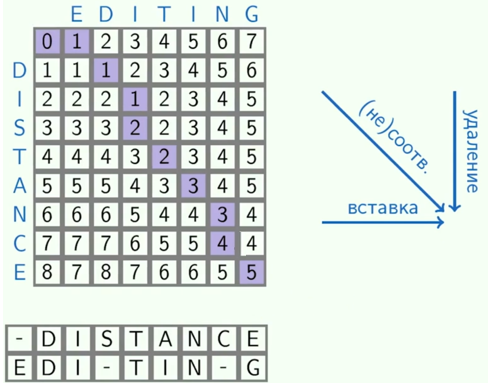
****

****

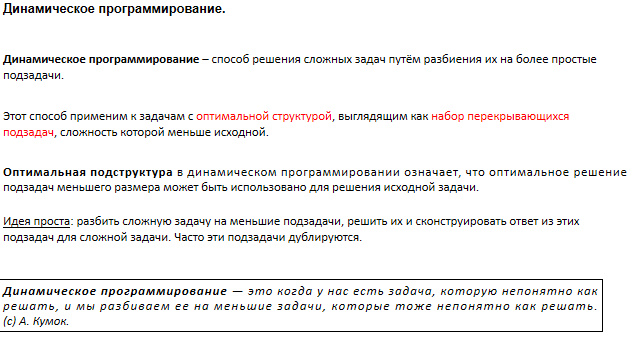
****

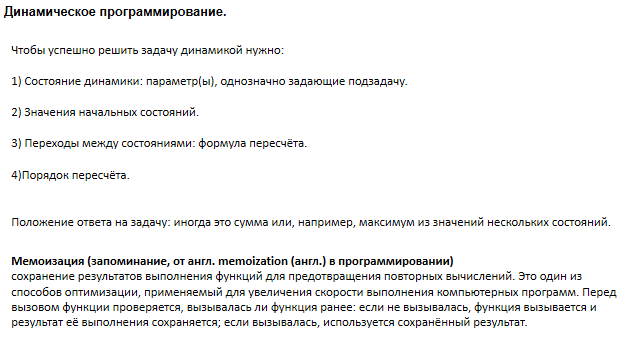
** **

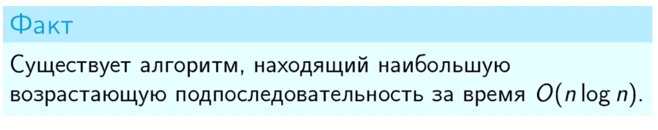
** **

****

# **18. Динамическое программирование. Задача о лягушке.**

****

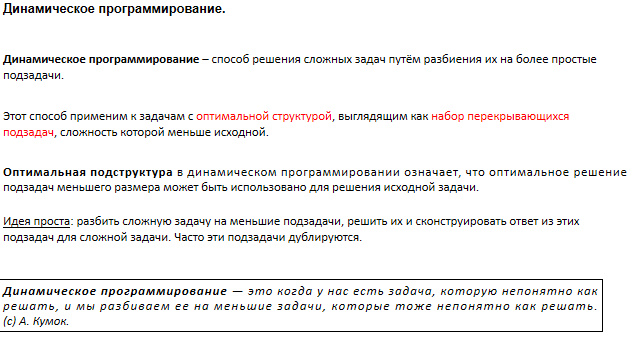
****

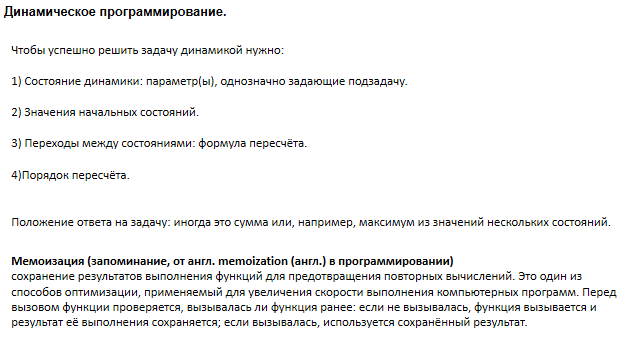


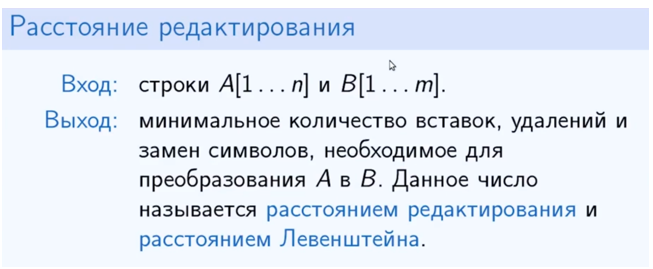
https://www.youtube.com/watch?v=1h\_3VYrpHFc&ab\_channel=%D0%A4%D0%BE%D0%BA%D1%81%D1%84%D0%BE%D1%80%D0%B4

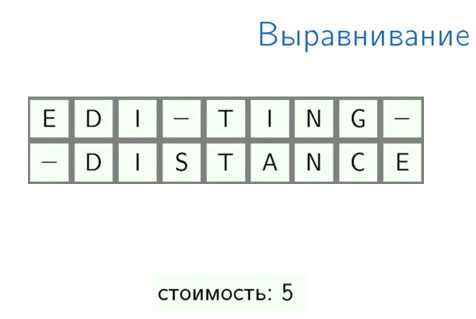


# **19. Динамическое программирование. Расстояние редактирования. Взвешенное расстояние редактирования.**

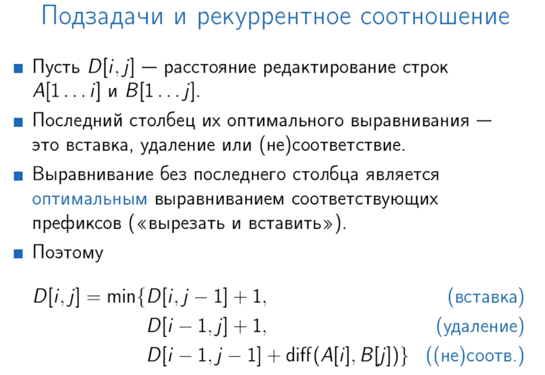
****

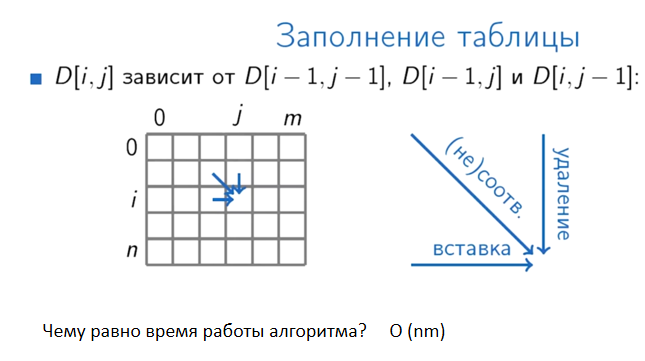
****

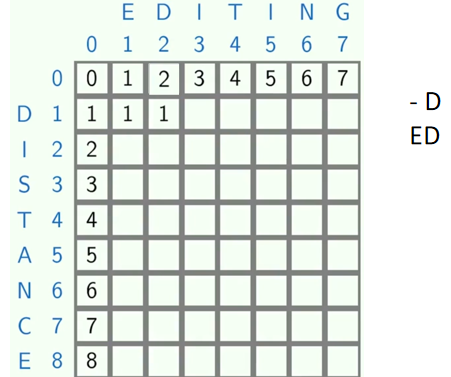
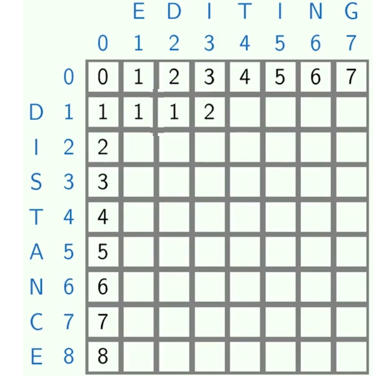


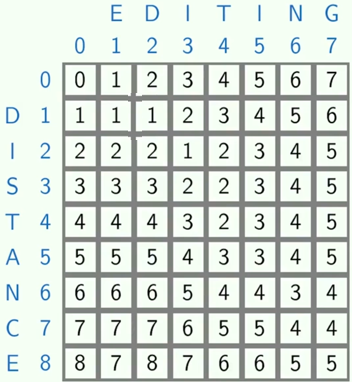


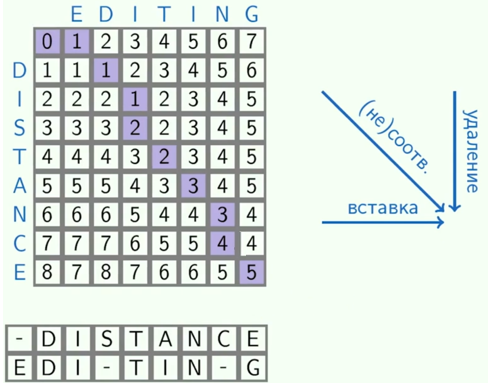


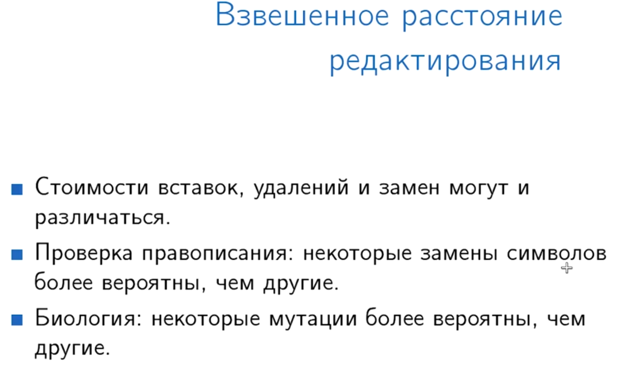
****

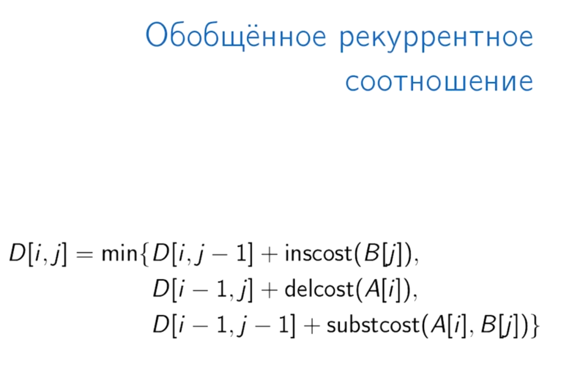
****

** **

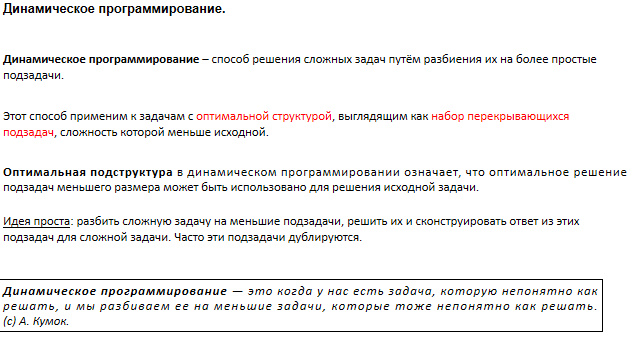
** **

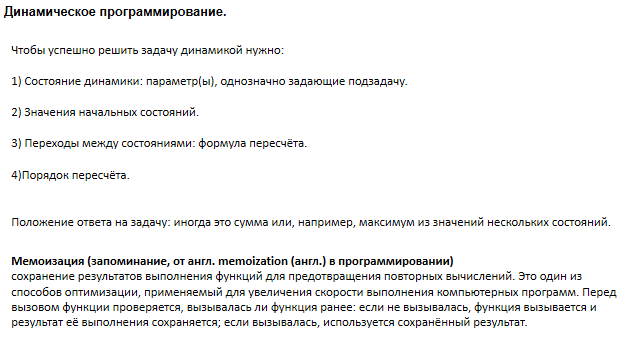
****

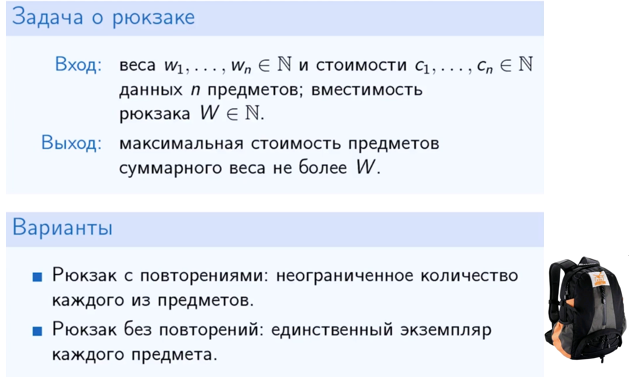


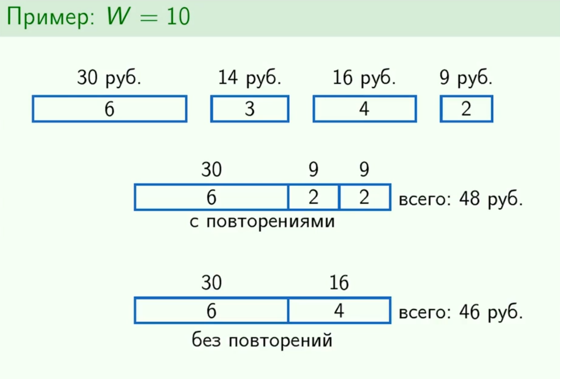


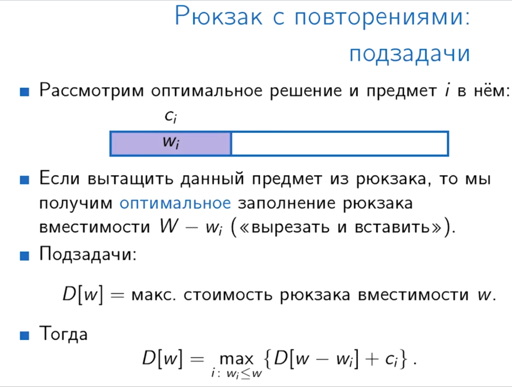
# **20. Динамическое программирование. Задача о рюкзаке. Задача о рюкзаке с повторением и без повторения.**

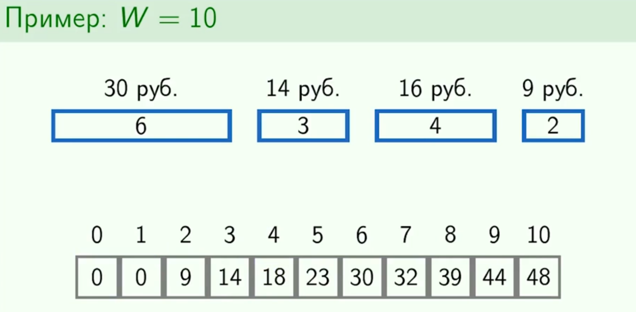
****

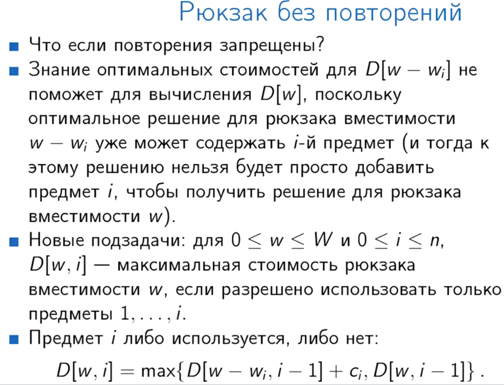
****

****

****

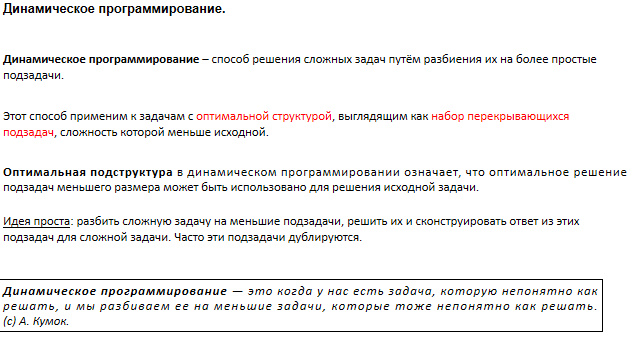
****

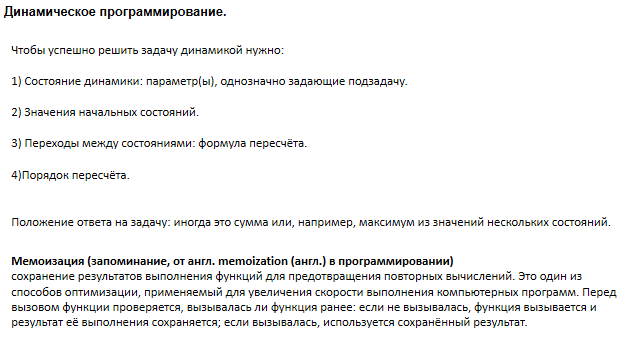
****

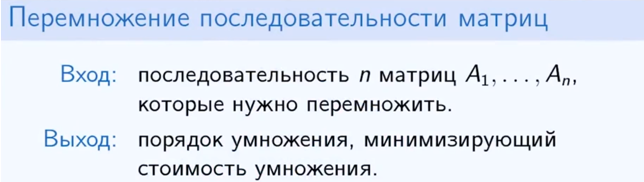
****

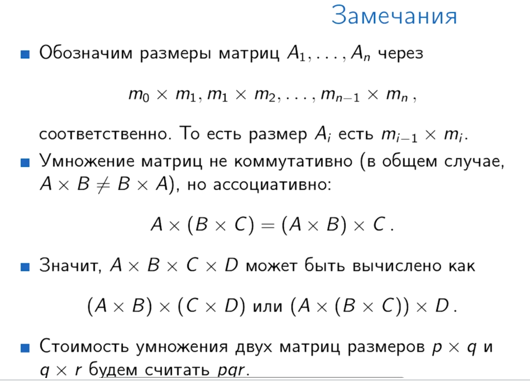
****

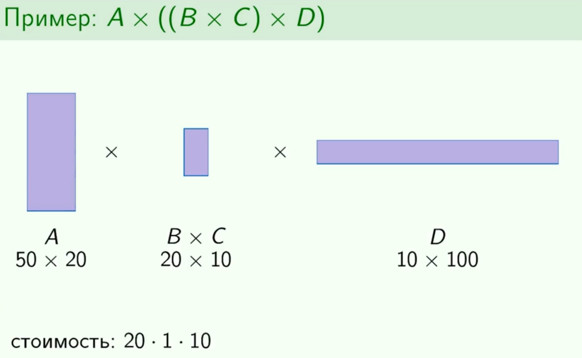
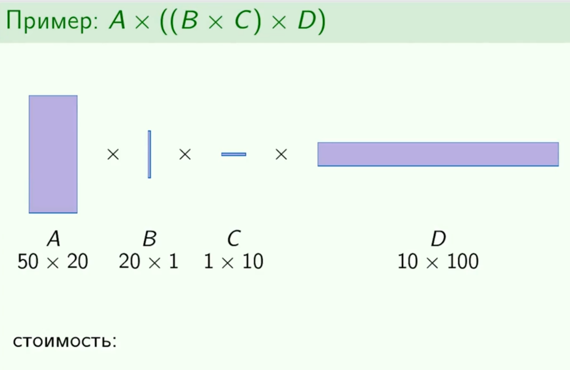
# **21. Динамическое программирование. Перемножение матриц.**

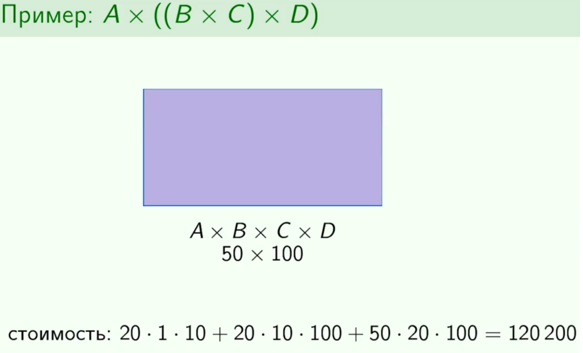
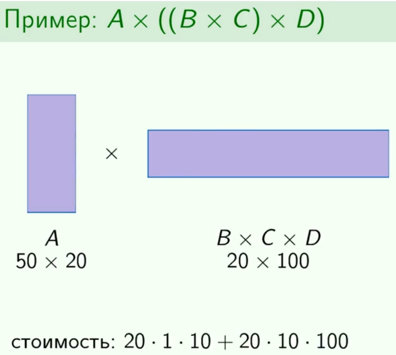
****

****

****

****

****

****

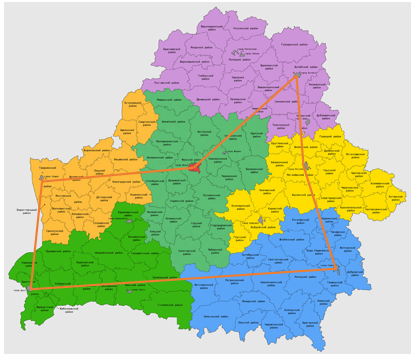
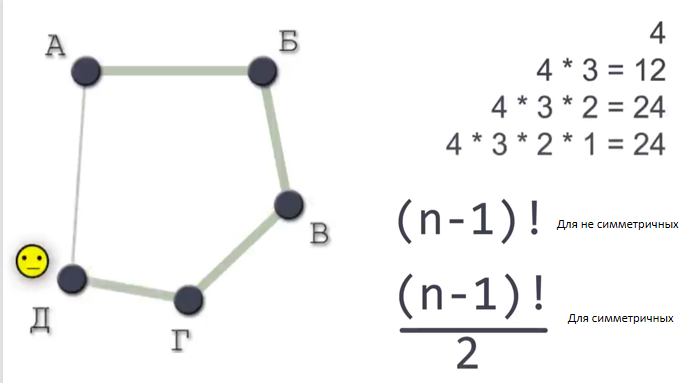
# **22. Недетерминированные полиномиальные задачи (NP-задачи). Задача коммивояжера. Генетический алгоритм.**

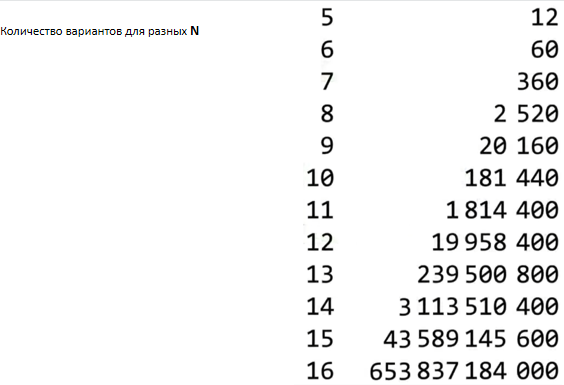
<https://www.youtube.com/watch?v=aB_6ZsLjzzc&ab_channel=QWERTY>

NP задачи это

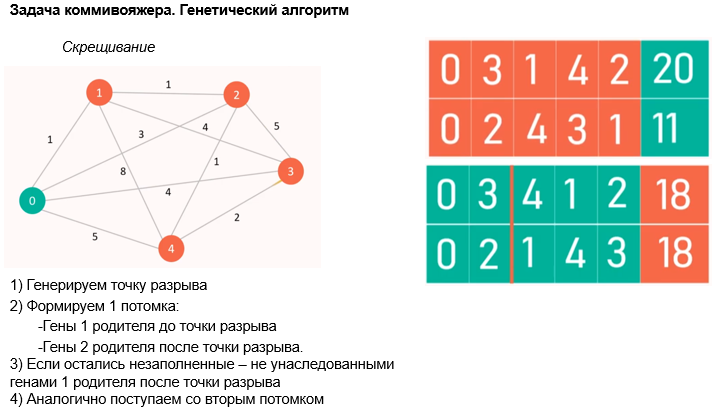
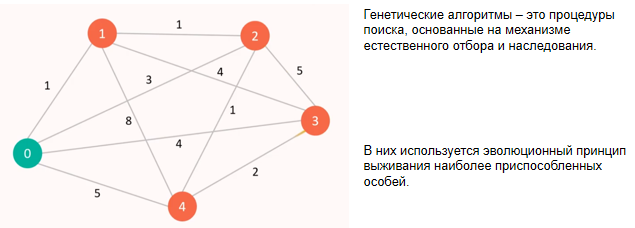
P задачи это

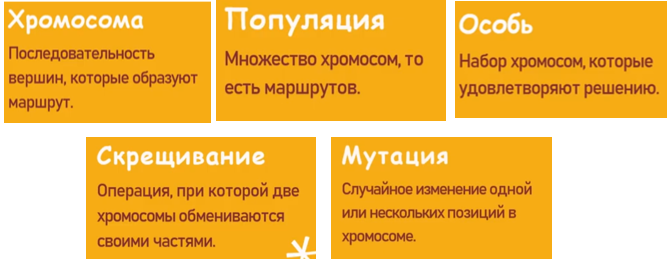
Задача коммивояжера

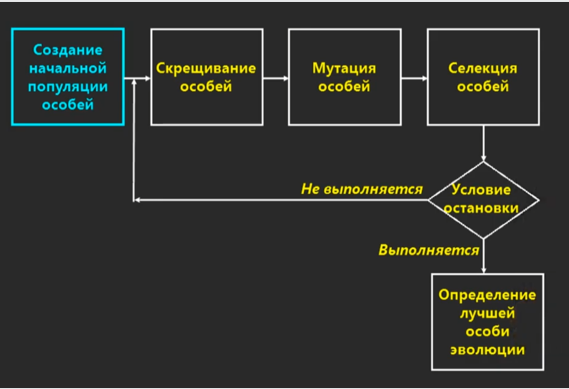
 



Генетический алгоритм



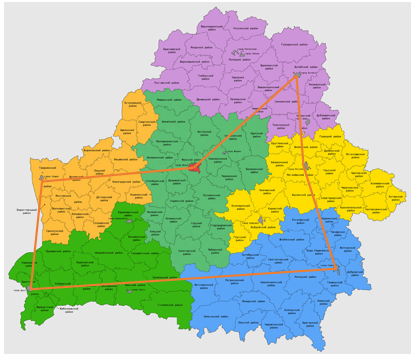
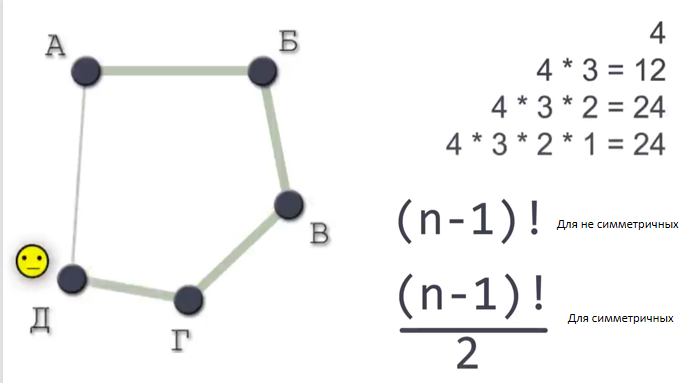


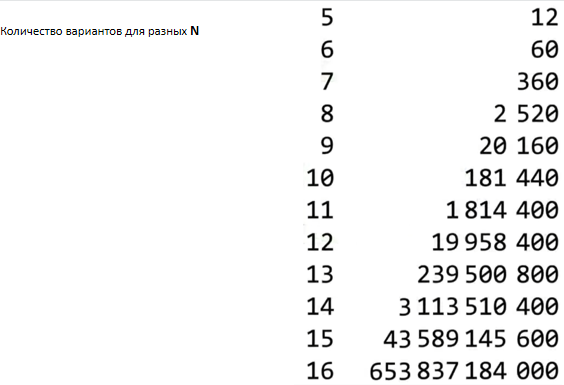




# **23. Недетерминированные полиномиальные задачи (NP-задачи). Задача коммивояжера. Метод ветвей и границ (алгоритм Литтла).**

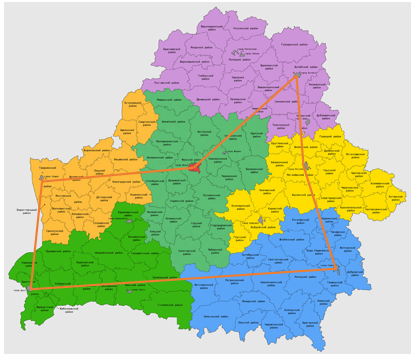
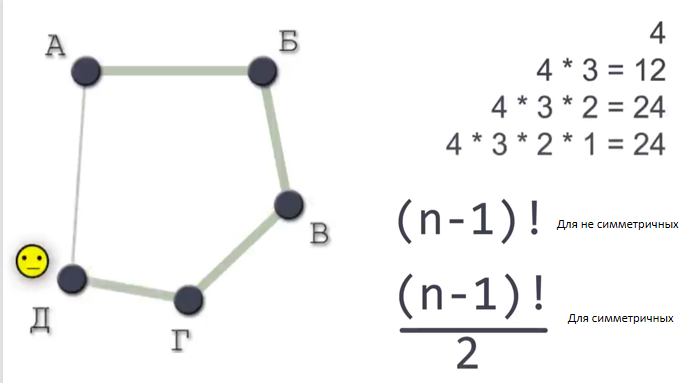
Задача коммивояжера

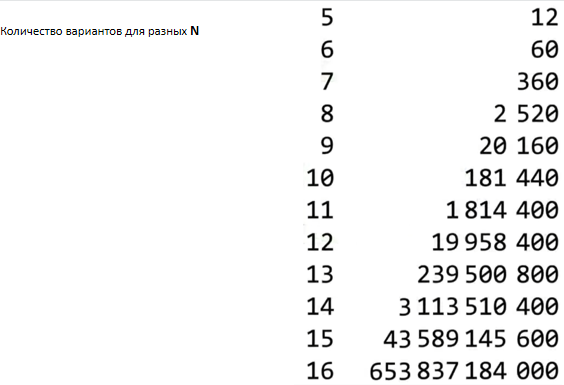
 

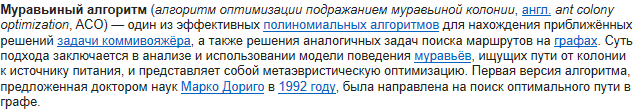


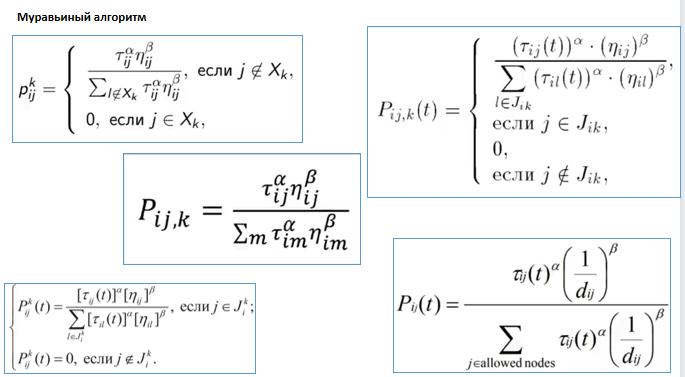
# **24. Недетерминированные полиномиальные задачи (NP-задачи). Задача коммивояжера. Муравьиный алгоритм.**

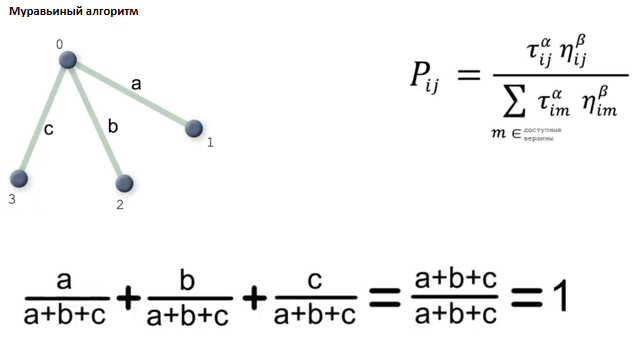
Задача коммивояжера

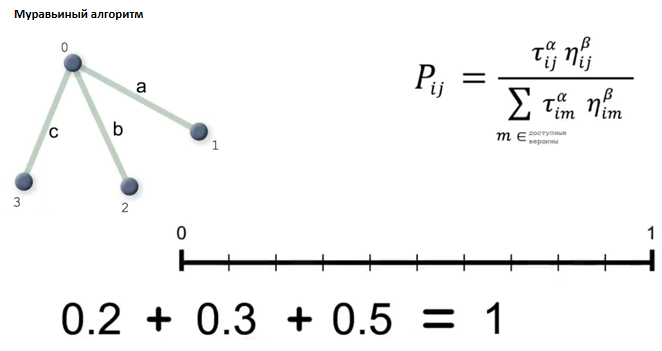
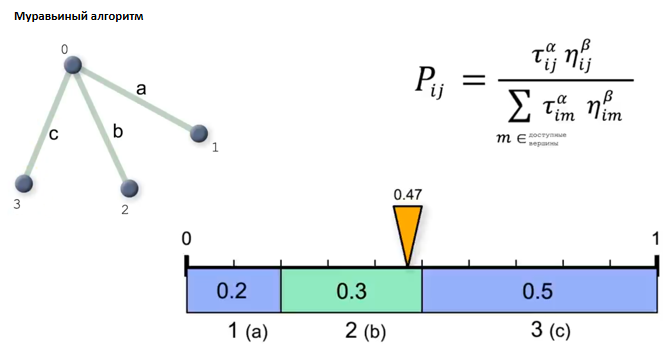
 

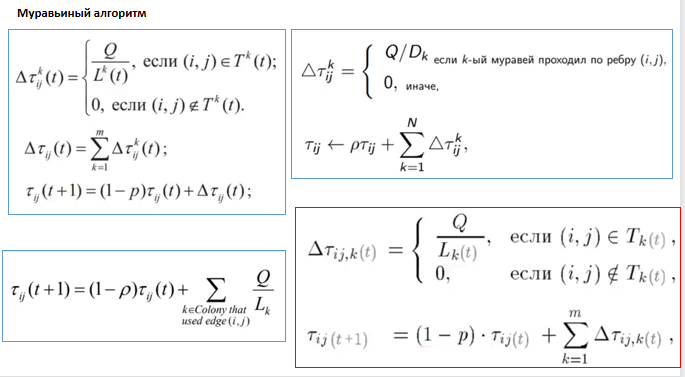


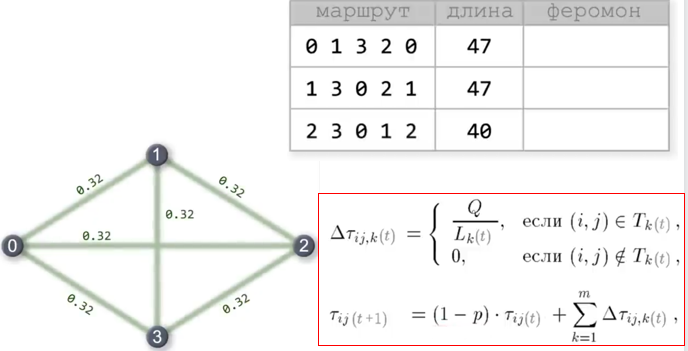


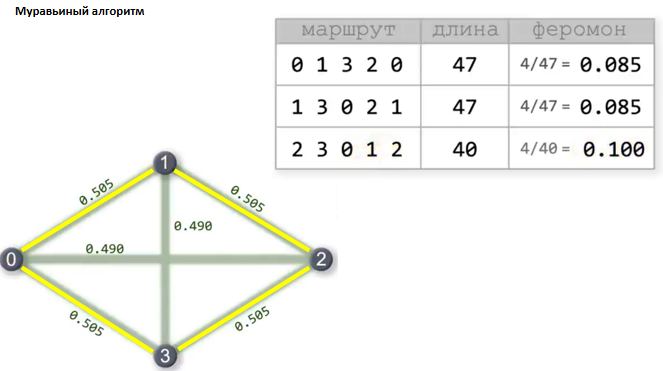










# **24. Алгоритмы с возвратом. Задача о ходе коня. Задача о ферзях.**

# **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.**