**Поиск с возвращением**

1. Идея поиска с возвращением (backtracking)

Поиск с возвратом, бэктрекинг ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) backtracking) — общий [метод](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4) нахождения решений задачи, в которой требуется [полный перебор](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D0%B1%D0%BE%D1%80) всех возможных вариантов в некотором [множестве](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE) М. Как правило, позволяет решать задачи, в которых ставятся вопросы типа: «Перечислите все возможные варианты …», «Сколько существует способов …», «Есть ли способ …», «Существует ли объект…» и т. п.

Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше. Данный алгоритм позволяет найти все решения поставленной задачи, если они существуют. Для ускорения метода стараются вычисления организовать таким образом, чтобы как можно раньше выявлять заведомо неподходящие варианты. Зачастую это позволяет значительно уменьшить время нахождения решения.

Можно свести задачу к дереву решений, поиск решения – обход дерева.

1. Пример на задаче о ферзях

Исходная формулировка: «Расставить на стандартной 64-клеточной шахматной доске 8 [ферзей](https://ru.wikipedia.org/wiki/%D0%A4%D0%B5%D1%80%D0%B7%D1%8C) так, чтобы ни один из них не находился под боем другого». Подразумевается, что ферзь бьёт все клетки, расположенные по вертикалям, горизонталям и обеим диагоналям.

Все возможные комбинации: 64^8.

Сложность полного перебора: О(N^N).

С учётом того, что нельзя ставить нового ферзя на ту же строку/столбец, каждый раз уменьшается количество допустимых вариантов: O(N!).

Примерный алгоритм решения задачи:

while (не все решения) {

while (множество позиций не пусто) {

движемся вперёд (ставим ферзя)

if (решение) => фиксируем ферзя

else => ищем следующее место

}

} - шаг поиска с возвратом

1. Подходы метода ветвей и границ по отсечению вариантов

На основании знаний о задаче в дереве решений можно отсечь подмножество решений, не содержащих оптимальных решений.

В основе метода ветвей и границ лежит идея последовательного разбиения множества допустимых решений на подмножества (стратегия “разделяй и властвуй”). На каждом шаге метода элементы разбиения подвергаются проверке для выяснения, содержит данное подмножество оптимальное решение или нет. Проверка осуществляется посредством вычисления оценки снизу для целевой функции на данном подмножестве. Если оценка снизу не меньше рекорда — наилучшего из найденных решений, то подмножество может быть отброшено. Проверяемое подмножество может быть отброшено еще и в том случае, когда в нем удается найти наилучшее решение. Если значение целевой функции на найденном решении меньше рекорда, то происходит смена рекорда. По окончанию работы алгоритма рекорд является результатом его работы.

Если удается отбросить все элементы разбиения, то рекорд — оптимальное решение задачи. В противном случае, из неотброшенных подмножеств выбирается наиболее перспективное (например, с наименьшим значением нижней оценки), и оно подвергается разбиению. Новые подмножества вновь подвергаются проверке и т.д.

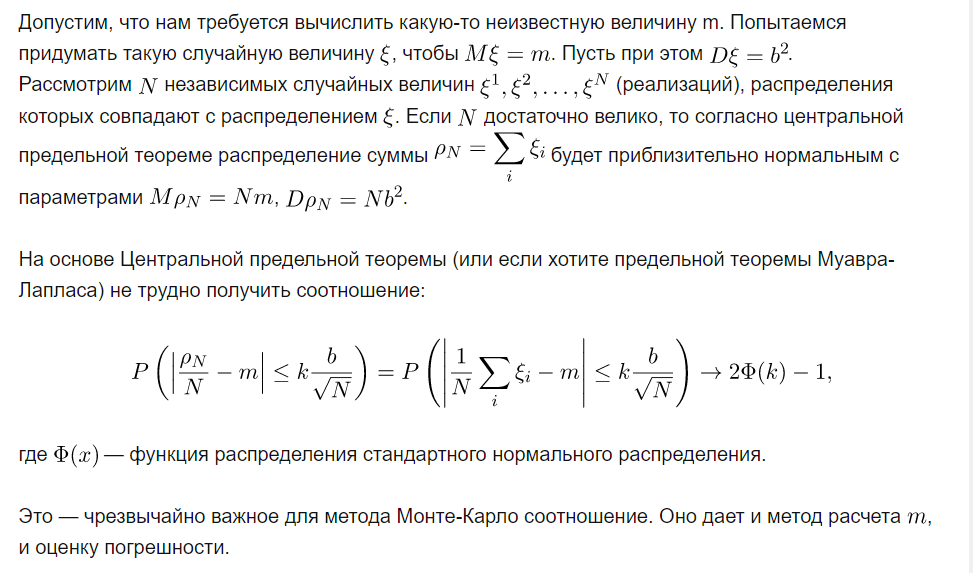
1. Метод Монте-Карло

Методы Монте-Карло — группа численных методов для изучения [случайных процессов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81). Суть метода заключается в следующем: процесс описывается математической моделью с использованием [генератора случайных величин](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB), модель многократно обсчитывается, на основе полученных данных вычисляются [вероятностные](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C) характеристики рассматриваемого процесса.

Например, чтобы узнать методом Монте-Карло, какое в среднем будет расстояние между двумя случайными точками в круге, нужно взять координаты большого числа случайных пар точек в границах заданной окружности, для каждой пары вычислить расстояние, а потом для них посчитать [среднее арифметическое](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B5%D0%B4%D0%BD%D0%B5%D0%B5_%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5).

Метод Бюффона для определения числа Пи (тот самый с киданием иголки на плоскость) – также метод Монте-Карло.

Основная идея метода:



1. Пример МК на размере дерева

Задача: найти количество ребер в дереве

Идея решения: идём по случайно выбранному ребру, предполагая, что каждые из “ветвей” дерева расположены зеркально, и считаем количество рёбер при этом условии. Находим среднее арифметическое от результатов нескольких экспериментов.

Фулл:

51:14

[Лекция](https://youtu.be/hLsYyYbbjsE)

1. Пример МК на площади фигуры

Задача: вычислить площадь случайной фигуры

Идея решения: фигуру вписываем в прямоугольник на плоскость. В область прямоугольника случайно бросаем точки. Смотрим, какие точки попали в область фигуры, и смотрим их соотношение к общему количеству точек.

Фулл:

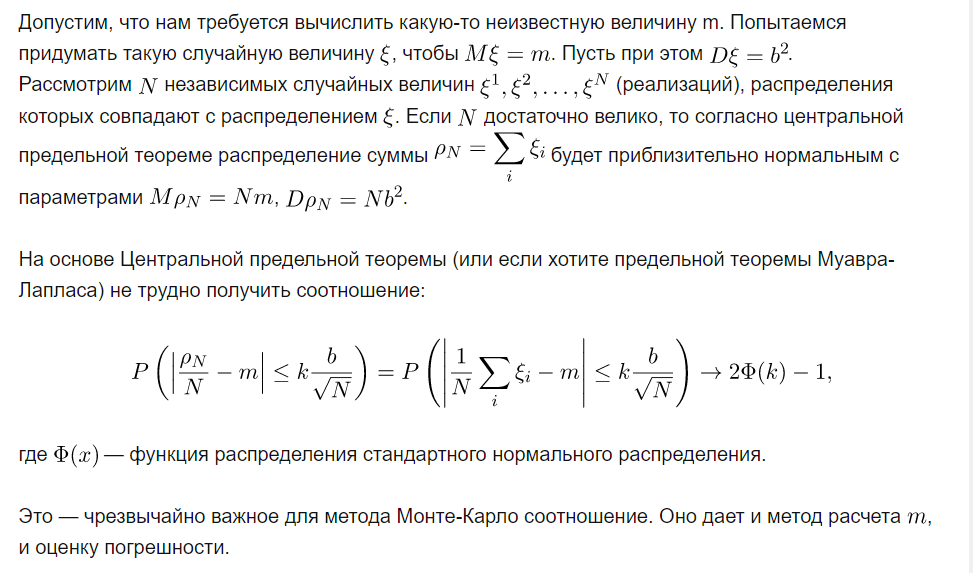
1:03:00

[Лекция](https://youtu.be/hLsYyYbbjsE)

Подробно тут (похожий способ): [Геометрический алгоритм Монте-Карло интегрирования](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%9C%D0%BE%D0%BD%D1%82%D0%B5-%D0%9A%D0%B0%D1%80%D0%BB%D0%BE#%D0%93%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9C%D0%BE%D0%BD%D1%82%D0%B5-%D0%9A%D0%B0%D1%80%D0%BB%D0%BE_%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)

1. Оценка сложности выполнения МК
2. Ограничения и условия применимости МК

Вероятность того, что найдём верное решение задачи:



Сурс: <https://habr.com/ru/post/274975/>

**Графы**

1. Графы и структуры данных
2. Графы: определения и примеры. Упорядоченный граф
3. Представление графа: матрица инцидентности, матрица смежности, список пар, структура смежности (списки инцидентности)

**Раскраска**

Когда говорят о раскраске графов, почти всегда подразумевают под этим раскраску их вершин, то есть присвоение цветовых меток вершинам графа так, чтобы любые две вершины, имеющие общее ребро, имели разные цвета. Так как графы, в которых есть петли, не могут быть раскрашены таким образом, они не являются предметом обсуждения.

**Хроматическое число графа** - минимальное количество цветов покрасок.

Соседей вершины нельзя красить в тот же цвет.

**Полный граф** - такой граф, у которого есть путь из каждой вершины в каждую. Очевидно, что хроматическое число полного графа совпадает с количеством вершин.

1. Алгоритм полного перебора

Пусть дан граф и дана его раскраска. Чтобы проверить её корректность, нужно обойти все ребра и проверить, если одинаковые цвета на концах. Значит, сложность проверки - линейная.

Самый простой вариант - генерация всех возможных три-раскрасок и проверка каждой на корректность. Сложность такого алгоритма - O(3^n), т.к. это число возможных три-раскрасок графа. Степенную сложность не убрать никак, на данный момент самый эффективный алгоритм - O(1.232^n).

1. Перебор с учётом выбора только из 2-х цветов

Пусть есть некоторая вершина. Мы красим её в первый цвет. Для инцидентных вершин выбор идет не из 3-х цветов, а из 2-х. Сложность такого алгоритма - O(2^n).

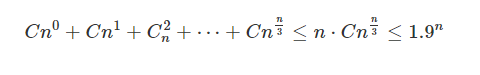
Другими словами, используется знание, что инцидентные данной вершины нельзя красить в цвет данной.

1. Перебор подмножеств размера <= n/3

Два соображения:

1. Если граф уже раскрашен, то вершины разделяются на множества – первого, второго и третьего цвета. Очевидно, что внутри этих множеств не существует рёбер. Если есть n вершин и 3 множества, очевидно, что одно из этих множеств будет иметь размер вершин.
2. Пусть нам заранее известно одно из этих множеств. Если это так, то все оставшиеся вершины будут краситься гораздо проще - с линейной сложностью.

Найдём такое множество. Способов выбрать множество заданной мощности конкретного цвета (это сочетания):



– радиус шара Хэмминга. После каждого шага докрашиваем до полной раскраски и проверяем, является ли она корректной.

Сложность алгоритма в таком случае: O(1,9^n).

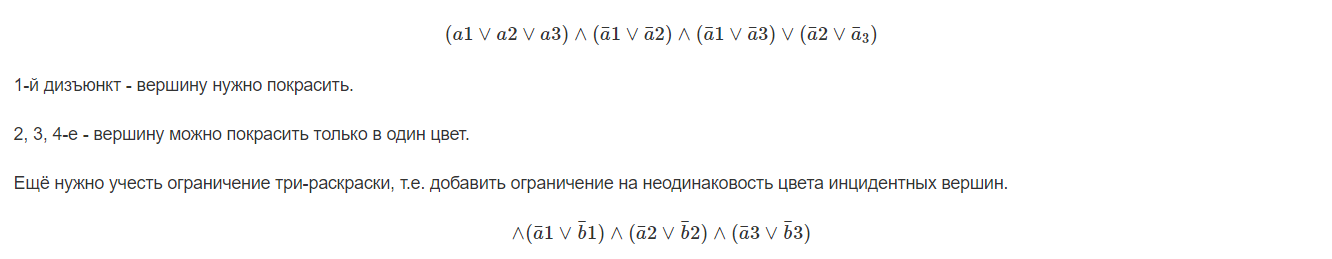
1. Вероятностный алгоритм. Сведение к задаче выполнимости

Посмотрим все цвета для вершин, в которые можно их покрасить. Можно для каждой вершины случайным образом выкинуть один цвет и красить из оставшихся. В таком случае каждая вершина может быть покрашена в один из двух цветов.

Такую задачу можно свести к задаче выполнимости булевых формул (2SAT).

Сведение такое:

Пусть каждый цвет обозначается переменной a\_1, a\_2, a\_3. Нужно покрасить все вершины в какие-то цвета. Это значит, что нужно выбрать хотя бы (и только) один из этих цветов. Т.е. для одной вершины верно следующее:



Вычеркнув неиспользуемые цвета, получаем задачу [2SAT](http://neerc.ifmo.ru/wiki/index.php?title=2SAT), где каждый дизъюнкт содержит не больше чем два литерала. Эту задачу можно решить за полиномиальное время.

Задачу 2SAT возможно свести к графу, и решением будет поиск компонент связности.

Если найдено решение задачи, то граф точно может быть раскрашен, но обратное неверно - возможно, мы вычеркнули нужные цвета для раскраски. Вероятность того, что раскраска выживет после вычеркивания -

Если прогнать алгоритм раз, получим вероятность ошибки После ещё 100 прогонов алгоритма вероятность ошибки – .

Сложность алгоритма: O(1,45^n).

1. Алгоритм перебора инцидентных вершин

Мы можем перебирать множества инцидентных вершин достаточно эффективно: если одна вершина попала в первое множество, то инцидентные гарантированно туда не попадут. Алгоритм, основанный на этой идее, даёт сложность O(1,45^n).

1. Применение раскраски на практике
2. Задача планирования. Пусть есть список заказов, которые начинаются и заканчиваются в определенное время. Нужно понять, какое минимальное количество ресурсов нужно выделить для решения этих заказов. Сводится так: пересечение заказов по времени - ребро графа. Хроматическое число графа и есть ответ.
3. Есть карта - отображение информации, которое предполагает, что между объектами есть границы (например - политическая карта мира). Задача - определить минимальное число цветов раскрасить карту, чтобы были видны границы между объектами. Задача очевидным образов сводится к раскраске графа.
4. Задача оптимального распределения переменных по регистрам.

Сурс: [Конспект лекции по раскраскам](http://se.moevm.info/doku.php/courses:algorithms_building_and_analysis:materials:graph_coloring_notes)

**Кратчайшие пути. Дейкстра**

1. Кратчайшие пути от фиксированной вершины
2. Случай неотрицательных весов: алгоритм Дейкстры

Алгоритм Дейкстры перебирает все возможные пути в ширину, пока не достигнет цели.

Вершины добавляются в очередь с приоритетом, выбирается вершина с наименьшим весом, пока очередь непуста и не достинута цель. Обновляется расстояние до вершины, если был найден лучший путь.

Сложность в случае простых структур данных: O(m + n log n)

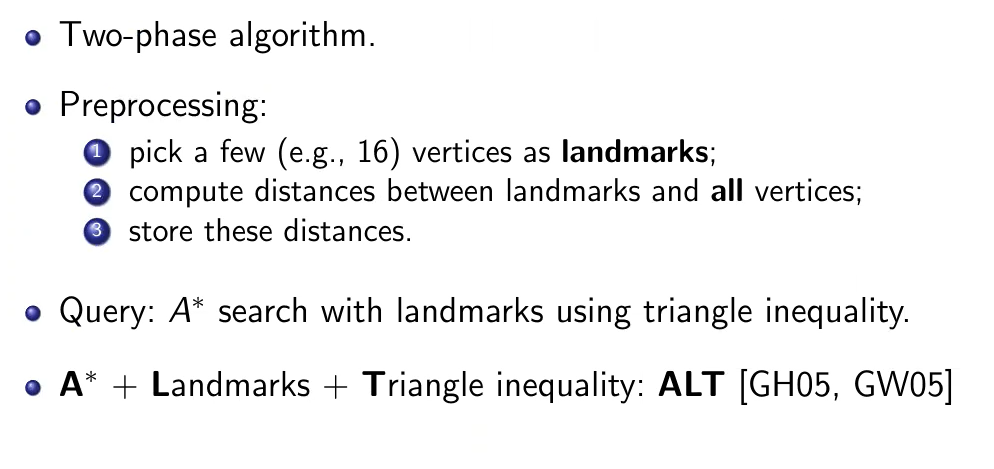
1. Алгоритм Флойда-Уоршелла вычисления расстояний между парами вершин, одновременное построение путей
2. Эвристические алгоритмы
3. Алгоритм A\*

Сравнение A\* и Дейкстры: <https://habr.com/ru/post/331192/>

В алгоритме А\* учитывается не только вес вершины, но и эвристическая функция. Сложность A\* зависит от эвристики.

1. Эвристические функции
2. Применение в ГИС (обзор алгоритма ALT)

Выбираются маркерные точки, для каждой пары которых заранее просчитываются функции. Найдём две ближайшие к цели маркерные точки.



**Минимальный разрез**

Разрез графа — множество рёбер, образующих [двудольный](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D1%83%D0%B4%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84) подграф, удаление которых делит граф на два или более компонентов, которые, в частности, могут быть изолированными узлами. А также линия, проходящая через все рёбра разреза графа.

Разрез является минимальным по некоторому критерию (минимальное число рёбер, вершин и т. д.).

Граф с n вершинами может иметь не более n(n-1)/2 различных минимальных разрезов.

1. Идея алгоритма на основе поиска потока
2. Примеры практических задач
3. Оценка надёжности сети (например, телекоммуникационной)
4. Кластеризация изображений (сегментирование)
5. Алгоритм Каргера

Алгоритм Каргера — является [вероятностным](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) [алгоритмом](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), позволяющим найти [минимальный разрез](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B8%D0%BC%D0%B5%D0%BD%D1%8C%D1%88%D0%B8%D0%B9_%D1%80%D0%B0%D0%B7%D1%80%D0%B5%D0%B7) [связного](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84) [графа](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)).

Идея алгоритма основана на [стягивании ребра](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%8F%D0%B3%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D1%80%D0%B5%D0%B1%D1%80%D0%B0) в неориентированном графе. Во время стягивания ребра происходит объединение двух вершин в одну, что уменьшает количество вершин графа на единицу. Все рёбра стягиваемых вершин соединяются со вновь образованной вершиной, порождая [мультиграф](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%B3%D1%80%D0%B0%D1%84). Алгоритм Каргера итеративно выбирает случайные рёбра и выполняет операцию до тех пор, пока не останется две вершины, которые и представляют собой разрез изначального графа. Если повторять алгоритм достаточное количество раз, то с высокой вероятностью может быть найден минимальный разрез.

Граф |V| = n, |E|, |C| = K (размер разреза). Допустим, что первое выбранное ребро попало в разрез.

P(первое ребро принадлежит разрезу) = |C|/|E|

Знаем, что степень любой вершины больше или равна K.

Предположим, что мы не знаем разрез. Оценим такую же вероятность.

Выразим |E|: |E| nK/2

|C|/|E| K/(Kn/2) = 2/n

P(ребро не принадлежит разрезу) = 1 - 2/n = (n-2)/n

P(на каждом шаге ребро не принадлежит разрезу) =

Оценка сложности:

1. Стягиваем: n
2. Сколько стягиваем: n
3. Количество раз (цикл): n^2

В результате: O(n^4).

1. Оптимизация Штейна алгоритма Каргера

Изначально вероятность того, что случайно выбранное для стягивания ребро принадлежит разрезу, небольшая. Чем меньше остаётся рёбер, тем больше вероятность того, что случайно выбранное ребро будет принадлежать разрезу. Найдём точку с предположительно “опасным” ребром. Запомним это место и два раза пройдём из него случайным образом. Вероятность того, что “выживает” минимальный разрез:

P(выжил разрез) =

Общая вероятность успеха: P(успех) = 1/log n.

Логарифм появляется из-за того, что мы выходим случайным образом из вершины два раза, то получается дерево.

Оценка сложности:

T(n) = n^2 + 2T() ~ n^2 log n

Алгоритм итеративный.

**Строки**

**Редакционное восстановление**

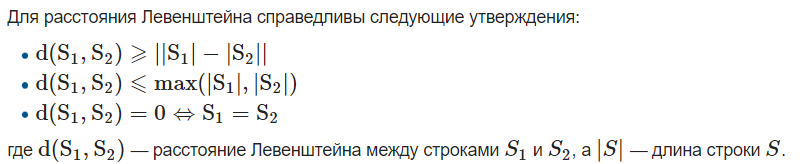
Хорошая статья по теме: <https://habr.com/ru/post/117063/>

1. Расстояние Левенштейна (редакционное расстояние)

Расстояние Левенштейна (редакционное расстояние, дистанция редактирования) — [метрика](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D1%80%D0%B8%D0%BA%D0%B0_(%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F)), измеряющая по модулю разность между двумя последовательностями символов. Она определяется как минимальное количество односимвольных операций (а именно вставки, удаления, замены), необходимых для превращения одной последовательности символов в другую. В общем случае, операциям, используемым в этом преобразовании, можно назначить разные цены.

Другими словами: Редакционное расстояние, или расстояние Левенштейна — метрика, позволяющая определить «схожесть» двух строк — минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Префикс строки – подстрока любой длины, начинающаяся с начала строки.



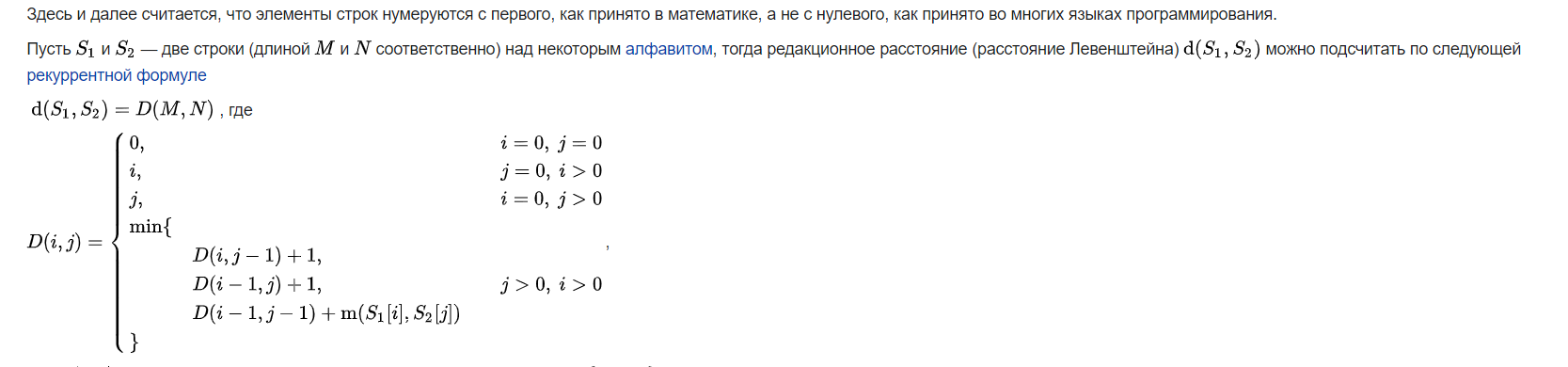
Если известны префикс длины i, префикс длины j (представьте, что там после min большая фигурная скобка):

D(i, j) = min D(i-1, j) + 1

D(i, j-1) + 1

D(i-1, i-j) (S1[i] != S2[j])

Забейте на то, что выше, вот норм рекуррентная формула:



1. Редакционное предписание (РП)

Редакционное предписание (англ. Editorial prescription) — последовательность действий, необходимых для получения из первой строки второй кратчайшим образом. Обычно действия обозначаются так: D (англ. delete) — удалить, I (англ. insert) — вставить, R (англ. replace) — заменить, M (англ. match) — совпадение.

1. Алгоритм Вагнера-Фишера

Строим матрицу D с помощью описанных выше формул. Остальное на вики, потому что мне лень расписывать. Можно еще посмотреть на лекции пример.

1. Вычисление редакционного расстояния методом ДП

По идее, +- всё, что было на лекции, есть в статье. В лекции можно посмотреть пример на матрице.

1. Восстановление РП по таблице (обратный ход в методе ДП)
2. Сведение задачи к путям в графе

**Алгоритм Кнута-Морриса-Пратта**

На лекции рассмотрен алгоритм со склеиванием строк паттерна и текста, есть алгоритм и без склеивания.

1. Основные определения

S[1…n] – текст, P[1…m] – шаблон, m n.

Задача: найти вхождения P в S.

Префикс-функция строки - максимальный префикс, который в том числе является суффиксом строки.

1. Задача точного поиска образца в строке
2. Наивный алгоритм

Постепенно сдвигается паттерн, ищутся совпадения. Сложность наивного алгоритма O(m(n-m+1)).

1. КМП

Алгоритм Кнута — Морриса — Пратта (КМП-алгоритм) — эффективный [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), осуществляющий [поиск подстроки в строке](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D0%BF%D0%BE%D0%B4%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8). Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать [асимптотически более эффективный](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) алгоритм невозможно.

Зачем нужно склеивать строки: если префикс стал суффиксом, то это значит, что P (который здесь является префиксом) совпал с какой-то подстрокой в S (суффиксом).

**Сложность: КМП, Левенштейн**

1. Виды сложности

Важно оценить скорость роста функции.

О(n): количество операций < = n

o(n): количество операций < n

Θ(n): количество операций = n

Ω(n): количество операций > = n

ω(n): количество операций > n

Пример линейной сложности O(n):

for (1…n)

s+=1;

Пример квадратичной сложности O(n^2):

for (1…m)

for (1…n)

s+=1;

Сложность по памяти.

1. Сложность КМП

N шагов вычисления префикс-функции. “Рост” и “падение” могут происходить не более N раз. Таким образом, алгоритм КМП имеет сложность O(N).

Сложность по памяти: или O(|P| +|S|) ~ O(|P|)

1. Сложность редакционного состояния

Память: O(M\*N) для строк длины M и N

Операции: обход точек в ширину O(N)

**Ахо-Корасик**

1. Задача поиска точного набора образцов
2. Trie

Префиксное дерево (также бор, луч, нагруженное дерево, [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) trie) — [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), позволяющая хранить [ассоциативный массив](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2), ключами которого являются строки. Представляет собой [корневое дерево](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%80%D0%BD%D0%B5%D0%B2%D0%BE%D0%B9_%D0%B3%D1%80%D0%B0%D1%84#%D0%9A%D0%BE%D1%80%D0%BD%D0%B5%D0%B2%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), каждое ребро которого помечено каким-то символом так, что для любого узла все рёбра, соединяющие этот узел с его сыновьями, помечены разными символами.

1. Задача о словаре
2. Алгоритм Ахо-Корасик

[Алгоритм Ахо-Корасик / Хабр](https://habr.com/ru/post/198682/)

**Динамическое программирование**

Динамическое программирование в теории управления и теории вычислительных систем — способ решения сложных задач путём разбиения их на более простые подзадачи. Он применим к задачам с оптимальной подструктурой, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходной.

1. Примеры вычисления чисел Фибоначчи

Обычное решение задачи:

int fib (int n )

if (n == 1 || n == 2)

return 1;

return fib(n-1) + fib(n-2)

Решение задачи (динамическое программирование):

int fib (int n)

//Нужны только два последних числа, которые будем хранить в массиве

int f[2]

f[0] = f[1] = 1

for (i: 1…n)

//f = f\_{i-1}+f\_{i-2}

f = f[(i-1) mod 2] + f[(i-2) mod 2]

return f

1. Максимальная возрастающая последовательность: сведение к графу, выделение подзадачи

Найти наибольшую возрастающую последовательность в рамках набора чисел.

Возможных подмножеств у n чисел: 2^n

1. Задача о рюкзаке: без повторений (одномерный случай, с повторениями (двумерный случай)

Есть n предметов стоимостью c\_1, c\_2, … , c\_n и весами w\_1, w\_2, …, w\_n. W – максимальный вес, который можно положить в рюкзак.

Задача: максимизировать стоимость предметов.

1. C повторениями

Нет ограничения на количество предметов. Полный перебор: 2^n.

Найти A(W) - максимальная стоимость унесённых вещей при весе < W.

A(0) = 0

Пройдёмся по всем весам:

for w\_i … W:

A(w) = max\_(i, w > w\_i) {A(w - w\_i) + c }

Сложность: W\*n

1. Без повторений.

A(w, i) - максимальная стоимость первых i предметов в рюкзаке для веса < w. A(W, n) - ?

for i = 1 to n

for w = 1 to W

A(w, i) = max {A(w - w\_i,m i + 1) + c\_i, A(w, i -1)}

1. Задача о порядке перемножения матриц

**Задача коммивояжера**

Рассматривается n городов, связанных дорожной сетью. Пусть хij = 1, если путешественник переезжает из i-ого города в j-ый и хij =0, если j-ый город не посещается. Условно введем (n+1)-й город, совмещенный с 1-м городом, т.е. расстояния от (n+1)-го города до любого другого, отличного от первого, равны расстояниям от первого города. При этом, если из первого города можно лишь выйти, то в (n+1)-й город можно лишь прийти. Введем дополнительные целые переменные, равные номеру посещения этого города на пути. u1=0, un+1=n. Для того, чтобы избежать замкнутых путей, выйти из первого города и вернуться в (n+1) введем дополнительные ограничения, связывающие переменные xij и переменные ui (ui целые неотрицательные числа).

**Суть задачи:** Посетить все возможные точки, минимизировав расстояние.

1. Сложность полного перебора

O(n!)

1. Метод ветвей и границ: отсечка по текущему найденному пути, отсечка по весу МОД

Строится дерево всех возможных путей, лишние пути отсекаются по границе текущего минимального пути.

Минимальная оценка длины пути коммивояжера – вес остовного дерева графа. Можно построить минимальное остовное дерево ещё не просмотренной части графа. На основе этого

1. Локальный поиск: 2-окружение, имитация отжига

<https://habr.com/ru/post/209610/> – про имитацию отжига

[Про локальный поиск](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%BA%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA_(%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F))

<https://proglib.io/p/metod-k-blizhayshih-sosedey-k-nearest-neighbour-2021-07-19> – про ближайших соседей

Локальный поиск с 2-окружением – рассматриваем пары ребёр, заменяем их при необходимости.

1. Приближённое решение (2-приближённый алгоритм)

2-приближение означает, что результат получится не более чем в 2 раза худший, чем оптимальное решение.

Строим минимальное остовное дерево. Удваиваем каждое ребро графа. Строим в графе эйлеров цикл. Не будем выдавать на выход уже посещённые вершины. Уменьшаем пути за счёт неравенства треугольника. Длина цикла – удвоенная длина веса минимального дерева. Предположим, что не меняли путь. Оценка: две длины МОД. Вес пути меньше либо равен этой оценке.



Рис. 1 – Мало текста? Вы всегда можете помочь проекту, дополнив его!!