# Курсовой проект по курсу дискретного анализа: Поиск ближайшего соседа

Выполнил студент группы 08-307 МАИ *Сысоев Максим.*

## Условие

Дано множество точек в многомерном пространстве. В каждом запросе задается точка, Вам необходимо вывести номер ближайшей к ней точки из исходного множества в смысле простого евклидова расстояния. Если ближайших точек несколько, то выведите номер любой из них. Читать сразу все запросы и обрабатывать их одновременно запрещено.

**Входные данные** В первой строке задано два числа n и d ($1 <= n <= 10^5$, $1 <= d <= 10$) — количество точек в множестве и размерность пространства.
Каждая из следующих n строк содержит d целых чисел $x_{ij}(|x_{ij}| <= 10^8)$ — j-я координата i-й точки.
Далее следует целое число q ($1 <= q <= 10^6$) — количество запросов.
Каждая из следующих q строк содержит d целых чисел $y_{ij}(|y_{ij}| <= 10^8)$ — j-я координата i-го запроса.

**Выходные данные** Для каждого запроса выведите в отдельной строке единственное число — индекс ближайшей точки из множества. Если таких несколько, выведите любую. Индексация точек множества начинается с 1.

## Метод решения

Для решения задачи существуют два типа методов: точные и приближённые. В угоду задаче, был выбран точный метод, который хоть и работает медленее, но не подводит в результатах, что критично в рамках курсового проекта. Из точных методов выделяют два:
1) Полный перебор
2) k-d tree
Про полный перебор: Можно просто перебрать все объекты из обучающей выборки, посчитать для каждого из них расстояние до тестового объекта и затем найти минимум. Однако несмотря на то что сложность такого поиска линейная, она также зависит и от размерности пространства признаков. Иначе говоря, сложность будет $(ND)$. Алгоритм поиска ближайших соседей часто используется в рекомендательных и поисковых системах, поэтому размерности N и D могут быть слишком большими, следовательно такая сложность не подходит. Было выбрано решение через k-d tree. Оно делит всё пространство на определённые участки, в которых и происходит поиск.

Для рассчёта расстояния между двумя точками существует множество метрик. Я выбрал две: Евклидово и Манхэтонское расстояния. В общем случае, это метрика Минского: $\rho(x, y) = (\sum_i |x_i - y_i|^p)^{1/p}$
Где при p = 1 получаем Манхэтонское расстояние, а при p = 2 - Евклидово.

Манхэтонское выигрывает у Евклидового при больших размерностях в виду меньшей чувствительности к выбросам. Для размерностей $d >= 200$ используется Манхэтонское, для меньших - Евклидово.

Поиск представляет собой обход дерева с некоторыми условиями:
1) Если node == nullptr - return
2) Если расстояние от точки до текущей node меньше текущего минимума - запоминаем и минимум, и минимальную точку.
3) Идти влево или вправо: если у точки запроса по текущему измерению меньше, чем node - идём сначало влево, иначе - вправо.
4) Когда посетили левое поддерево или правое: смотрим расстояние до bounding box и если оно меньше текущего минимального расстояния - существует шанс, что там есть ближайшая точка. Иначе, там точно нет точки, которая ближе чем текущая ближайшая.

## Описание программы

Выполнение лабораторной состояло из нескольких этапов:

1. Изучить алгоритм
2. Написать код
3. Написать Makefile
4. Протестировать программу

## Тест производительности

Приведу несколько тестов. Для начала посмотрим зависимость от N - кол-во входных данных. Затем посмотрим на зависимость от dim - размерность и, наконец, посмотрим на зависимость от q - кол-во запросов.
Первые три таблицы учитывают построение дерева и поиск в нём. Но, так как в классических задачах, таких как рекомендательные системы и информационный поиск, деревья можно не строить каждый раз (У нас есть заранее известные фильмы, либо вебстраницы. Деревья можно строить раз в некоторый промежуток времени, в оффлайне), то приведу также сравнительные тесты, когда оценивается только поиск в дереве против брутфорса.

**Учитывается время построения дерева.**

Dim = 2, q = 10

| Method | $N = 10^3$ | $N = 10^4$ | $N = 10^5$ | $N = 10^6$ | $N = 10^7$ |
|---|---|---|---|---|---|
| Bruteforce | 3 ms | 84 ms | 603 ms | 3493 ms | 21985 ms |
| K-D-Tree | 3 ms | 69 ms | 1645 ms | 11073 ms | 106226 ms |

N = 1000, q = 10

| Method | $Dim = 2$ | $Dim = 10$ | $Dim = 100$ | $Dim = 200$ | $Dim = 500$ | $Dim = 1000$ |
|---|---|---|---|---|---|---|
| Bruteforce | 1 ms | 7 ms | 77 ms | 148 ms | 365 ms | 694 ms |
| K-D-Tree | 3 ms | 9 ms | 83 ms | 151 ms | 357 ms | 723 ms |

N = 1000, Dim = 2

| Method | $q = 10^3$ | $q = 10^4$ | $q = 10^5$ | $q = 10^6$ | $q = 10^7$ |
|---|---|---|---|---|---|
| Bruteforce | 189 ms | 1921 ms | 20118 ms | 226458 ms | 2261701 ms |
| K-D-Tree | 11 ms | 98 ms | 917 ms | 12896 ms | 133130 ms |

Из этих данных можно сделать следующие выводы:
1) Если запросов мало, а данных много - лучше отдать предпочтение простому перебору. Однако, редко когда в реальном мире можно высказать предположение о малом кол-ве запросов.
2) Оба метода зависят линейно от размерности и работают одинаково, если два других параметра неизменны.
3) Поиск в дереве проходит намного быстрее перебора, даже у учётом постройки дерева.

**Не учитывается время построения дерева.**

Dim = 2, q = 10

| Method | $N = 10^3$ | $N = 10^4$ | $N = 10^5$ | $N = 10^6$ | $N = 10^7$ |
|---|---|---|---|---|---|
| Bruteforce | 4 ms | 37 ms | 319 ms | 2919 ms | 25703 ms |
| K-D-Tree | 0 ms | 0 ms | 0 ms | 0 ms | 1 ms |

N = 1000, q = 10

| Method | $Dim = 2$ | $Dim = 10$ | $Dim = 100$ | $Dim = 200$ | $Dim = 500$ | $Dim = 1000$ |
|---|---|---|---|---|---|---|
| Bruteforce | 2 ms | 9 ms | 83 ms | 204 ms | 435 ms | 1004 ms |
| K-D-Tree | 0 ms | 7 ms | 100 ms | 174 ms | 473 ms | 999 ms |

N = 1000, Dim = 2

| Method | $q = 10^3$ | $q = 10^4$ | $q = 10^5$ | $q = 10^6$ | $q = 10^7$ |
|---|---|---|---|---|---|
| Bruteforce | 274 ms | 2516 ms | 24947 ms | 318182 ms | 2674481 ms |
| K-D-Tree | 16 ms | 123 ms | 1069 ms | 14831 ms | 122635 ms |

Отсюда можно сделать вывод, что метод через k-d-tree в среднем работает почти всегда лучше, особенно когда речь идёт о обработке большого кол-ва запросов.
Сложность перебора можно оценить как: $O(N * Dim * Q)$
Сложность поиска в дереве*: $O(Q * Dim * logN)$

Сложность построения дерева*: $O(N * Dim * logN)$
* При условии, что дерево сбалансированное или что на вход не приходят данные, заставляющие дерево работать за $O(n)$.

## Дневник отладки

Ошибка WA на 2 тесте: Изменить условие выбора доп.обхода поддерева
Ошибка TL на 4 тесте: Простой обход всего дерева - плохая идея. Считаем расстояние до bounding box, чтобы отсечь ненужные поддеревья

## Выводы

Выбор в пользу этого алгоритма пал потому, что изучал машинное обучение и хотелось самостоятельно реализовать алгоритм kNN. В ходе выполнения изучил устройство k-d дерева, изучил множество метрик для измерения расстояния между двумя точками и реализовал неплохо масштабируемый код, который при желании можно расширять.