Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ» по теме "Поиск ближайших соседей"

Студент: Меджидли И. И. о
Группа: М8О-301Б-21
Дата: 04.04.2024
Оценка:
Подпись:

Москва, 2024

Условие

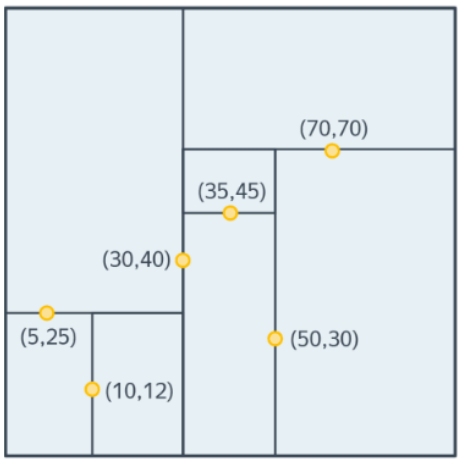
Дано множество точек в многомерном пространстве. В каждом запросе задается точка, Вам необходимо вывести номер ближайшей к ней точки из исходного множества в смысле простого евклидова расстояния. Если ближайших точек несколько, то выведите номер любой из них. Читать сразу все запросы и обрабатывать их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

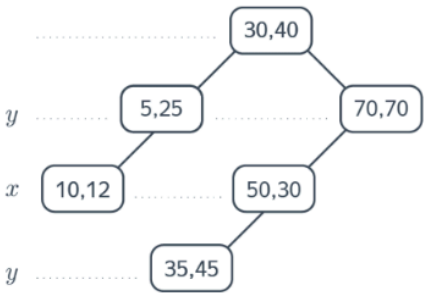
Входные данные В первой строке задано два числа n и d($1 <= n <= 10^5$,$1 <= d <= 10$) — количество точек в множестве и размерность пространства.Каждая из следующих n строк содержит d целых чисел x_{ij}($x_{ij} <= 10^8$) — j-я координата i-й точки. Далее следует целое число q($1 <= q <= 10^6$) — количество 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 приближенные. Я буду решать точным методом, а именно с помощью k-d дерева.Оно делит все пространство на определенные участки, в которых и происходит поиск, причем деление происходит по одной оси. При движении вниз по дереву оси, по которым точки делят пространство, циклически сменяют друг друга. Например, в двумерном пространстве корень будет отвечать за деление по x-координате, его сыновья за деление по y-координате, а внуки — снова за x-координату, и т. д. При добавлении вершин каждый раз будет браться точка, являющаяся медианой по одной оси в наборе оставшихся точек.





Для расчета расстояния между двумя точками существует множество метрик. Я выбрал Евклидову. Формула для нее в общем случае: ρ(x, y) = $\sqrt{\left(\sum_{i}^{p}\left|x_i - y_i\right|\right)}$, где d — размерность пространства.

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 минимальное расстояние, и минимальную точку.
3) Идти влево или вправо: если значение diff, равное разности координаты точки запроса по текущему измерению и соответствующей координаты точки node, меньше нуля – идём налево влево, иначе направо.
4) Когда поиск в выбранном поддереве закончен, то мы сравниваем $|diff|$ и минимальное расстояние до точки запроса: если $|diff|$ меньше, то нужно пройтись и по второму поддереву, потому что потенциально ближайший сосед может находиться в этом поддереве.

Исходный код

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <memory>

using namespace std;

struct Point {
    vector<long double> info;
    size_t index;
    Point() : index(0) {}
    Point(const vector<long double>& dims, size_t idx) : info(dims), index(idx) {}
};

static double Euclidean(const Point &a, const Point &b) {
    double sum = 0;
    for (size_t i = 0; i < a.info.size(); ++i) {
        double diff = a.info[i] - b.info[i];
        sum += diff * diff;
    }
    return sqrt(sum);
}

class KDTree{
    struct Node {
    Point point;
    unique_ptr<Node> right = nullptr;
    unique_ptr<Node> left = nullptr;
    Node(const Point& p) : point(p), left(nullptr), right(nullptr) {}
    };

    unique_ptr<Node> root = nullptr;
    int dim = 0;

    void build(vector<Point>& points, int l, int r, int depth = 0) {
```

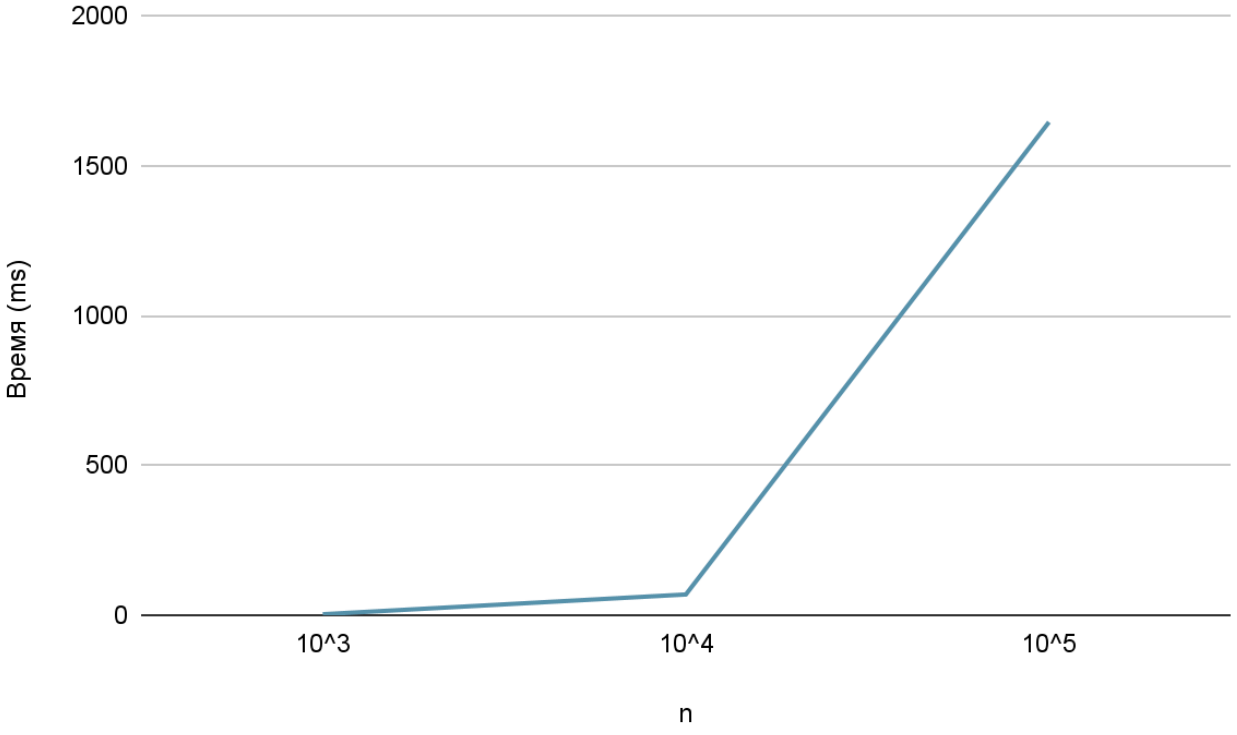
```
37        if (l > r) return;
38        size_t cd = depth % dim;
39        int medianIndex = l + (r - l) / 2;
40        sort(points.begin() + l, points.begin() + r + 1, [&cd] (Point& a, Point& b) {
41        return a.info[cd] < b.info[cd];
42        });
43        insert(root, 0, points[medianIndex]);
44        if (l != r) {
45            build(points, l, medianIndex - 1, depth + 1);
46            build(points, medianIndex + 1, r, depth + 1);
47        }
48        return;
49    }
50
51    void insert(unique_ptr<Node> &node, unsigned int depth, Point &request) {
52    if (node == nullptr){
53        node = make_unique<Node>(request);
54        return;
55    }
56    unsigned int cur_d = depth % dim;
57    if (request.info[cur_d] < node->point.info[cur_d]) {
58        insert(node->left, depth + 1, request);
59    }
60    else {
61        insert(node->right, depth + 1, request);
62    }
63    return;
64 }
65
66    void nearestNeighborSearch(const unique_ptr<Node>& node, const Point &target, size_t depth, double
  &bestDist, Point &bestPoint) const {
67        if (!node) return;
68        double dist = Euclidean(node->point, target);
69        size_t axis = depth % dim;
70
71        if (dist < bestDist) {
72            bestDist = dist;
73            bestPoint = node->point;
74        }
```

```
75
76          double diff = target.info[axis] - node->point.info[axis];
77          const unique_ptr<Node>& first = (diff < 0) ? node->left : node->right;
78          const unique_ptr<Node>& second = (diff >= 0) ? node->left : node->right;
79
80          nearestNeighborSearch(first, target, depth + 1, bestDist, bestPoint);
81
82          if (fabs(diff) < bestDist) {
83              nearestNeighborSearch(second, target, depth + 1, bestDist, bestPoint);
84          }
85      }
86  public:
87      KDTree(vector<Point> &points, size_t d) : dim(d) {
88          build(points, 0, points.size() - 1, 0);
89      }
90
91      Point nearestNeighbor(const Point &target) const {
92          Point bestPoint;
93          double bestDist = numeric_limits<double>::max();
94          nearestNeighborSearch(root, target, 0, bestDist, bestPoint);
95          return bestPoint;
96      }
97  };
98
99  int main() {
100     ios::sync_with_stdio(false); cin.tie(0);
101     size_t n, q; int d;
102     cin >> n >> d;
103     vector<Point> points;
104     points.reserve(n);
105     for(size_t i = 0; i < n; ++i){
106         vector <long double> dimensions;
107         dimensions.resize(d);
108         for(size_t j = 0; j < d; ++j){
109             cin >> dimensions[j];
110         }
111         points.emplace_back(dimensions, i);
112     }
113     KDTree tree(points, d);
```

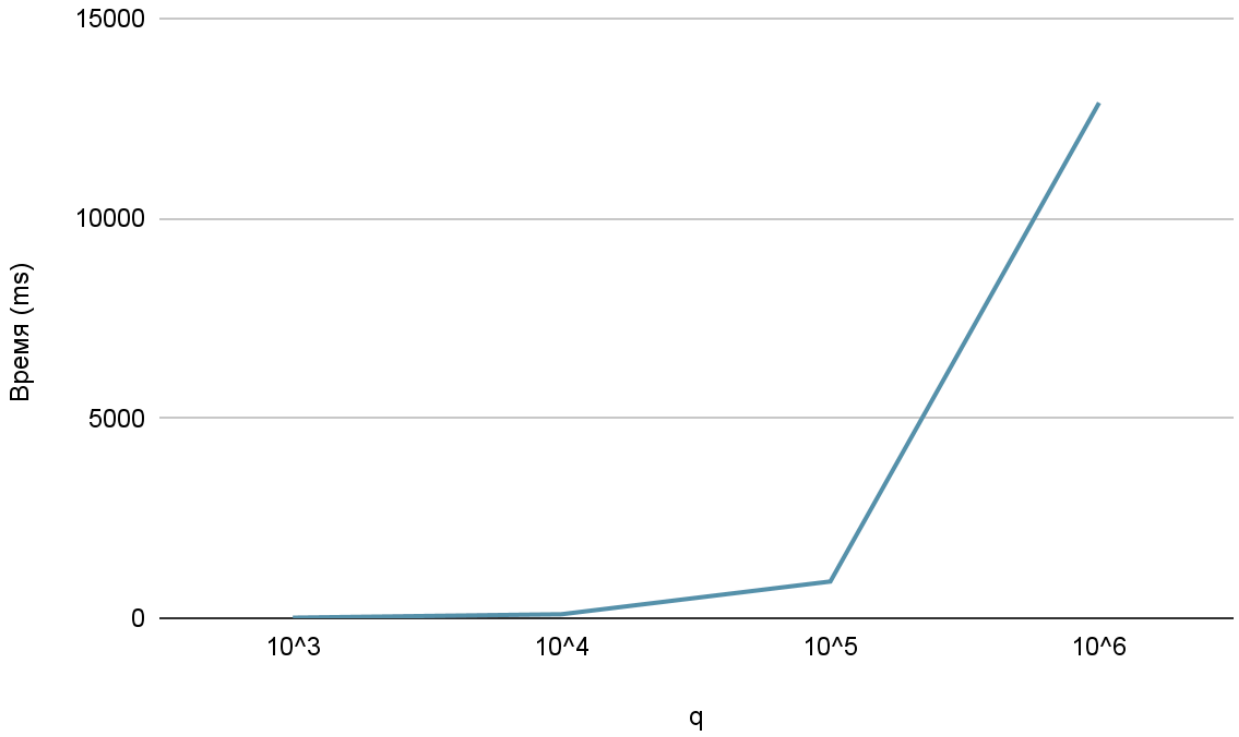
```
    cin >> q;
    for (size_t i = 0; i < q; ++i) {
        vector<long double> queryCoords(d);
        for (size_t j = 0; j < d; ++j) {
            cin >> queryCoords[j];
        }
        Point queryPoint(queryCoords, 0);
        Point nearest = tree.nearestNeighbor(queryPoint);
        cout << nearest.index + 1 << "\n";
    }
    return 0;
}
```

Тест производительности

Зависимость времени выполнения от размера множества n для размерности d = 2 и 10 запросов q:



Зависимость времени выполнения от количества запросов q при n = 1000 и d = 2:



Сложность перебора можно оценить как: O(N ∗ Dim ∗ Q)
Сложность поиска в дереве: O(Q ∗ Dim ∗ logN)
Сложность построения дерева*: O(N ∗ Dim ∗ logN)

Дневник отладки

Ошибка RE в 3 тесте: недостаток памяти. В функции build поменял типы l и r на int.
Ошибка WA в 3 тесте: при сортировке матрицы в функции build не учитывался последний элемент. Исправил границы вектора для сортировки.
Ошибка TL в 11 тесте: Исправил тем, что каждая медиана будет добавляться после поиска подходящего для нее места в дереве путем его обхода.

Результат работы программы

imedzhidli@imedzhidli:~/Desktop/DA/KP$ ./kpda
3 2
0 0
3 3
3 0
2
-1 0
2 1
1
3

```
imedzhidli@imedzhidli:~/Desktop/DA/KP$ ./kpda
10 2
0 0
1 4
3 3
2 -2
-3 1
1 1
10 5
-3 -3
-100 -100
90 -40
5
-1 -1
50 2
-7 -3
2 0
1 1
1
7
8
6
6
imedzhidli@imedzhidli:~/Desktop/DA/KP$
```

Как можно заметить, все работает верно.

Выводы

При выполнении курсового проекта по курсу «Дискретный анализ» я изучил k-d дерево, смог его построить и научился реализовывать с его помощью поиск ближайших соседей. Надеюсь, мне понадобятся знания, полученные в ходе выполнения данной работы.