Статическая задача в $\mathbb{R}^d$ с $O(\log^{d-1}(n))$

Андрей Козлов

22 января 2015 г.

Постановка задачи

Требуется реализовать дерево запросов принадлежности множества точек данному прямоугольнику с предподсчетом за $O(n\log(n))$ и запросом за $O(\log^{d-1}(n))$, где n — количество точек.

Также требуется сравнить время работы запроса на дереве со временем работы наивного алгоритма (запрос за $O(n)$), и дерева запросов без использования ссылок (запрос за $O(\log^d(n))$).

Реализация

Задача решалась для точек с координатами в целых числах.

Реализован абстрактный полиморфный класс `AbstractRangeTree` и его наследники `RangeTree2D` и `RangeTreeKD`, для двумерного и многомерного случаев соответсвенно.

Проверка корректности

Тесты на проверку корректоности проводились следующий образом:

1. генерируется прямоугольник `rectangle`;
2. генерируется набор точек `pointsInside` внутри прямоугольника `rectangle`;
3. генерирутеся набор точек `pointsOutside` снаружи прямоугольника `rectangle`.
4. наборы `pointsInside` и `pointsOutside` смешиваются и по ним строится дерево `tree`;
5. на дереве `tree` выполняется запрос на прямоугольник `rectangle`, ответ сравнивается с набором точек `pointsInside`.

Данные тесты проводились при различных значениях числа точек снаружи и внутри прямоугольника.

Сравнение производительности

В таблице 1 приведены результаты тестирования асимптотик операций запроса. Среднее значение бралось по результатам тысячи запусков.

Мощности компьютера, на котором проводились испытания, не хватает для продолжения испытаний многомерного дерева запросов со ссылками. "Выпавшее" значение объясняется активной сборкой мусора виртуальной машиной.

Это подтверждается тем, что данный эффект повторяется при том же значении n при повторных запусках с теми же опциями виртуальной машины, а также тем, что при увеличении или уменьшении максимального размера кучи `JVM` данный эффект проявляется при меньших или больших значениях n соответственно.

Тем не менее заметен прирост производительности относительно дерева без ссылок, в чем можно убедиться на графиках 1 и 2.

number of points n	$O(\log^{d-1}(n))$, $d=2$	$O(\log^{d}(n))$, $d=2$	$O(n)$, $d=2$	$O(\log^{d-1}(n))$, $d=3$	$O(\log^{d}(n))$, $d=3$	$O(n)$, $d=3$
10000	0.21	0.42	0.26	0.42	0.18	0.21
20000	0.36	0.19	0.46	0.10	0.30	0.39
30000	0.20	0.34	0.72	0.15	0.18	0.80
40000	0.28	0.28	1.44	0.12	0.20	0.82
50000	0.31	0.48	1.16	0.13	0.28	1.05
100000	0.68	0.65	2.14	0.26	0.48	2.48
200000	1.16	1.64	5.66	0.51	0.85	4.88
300000	1.57	2.37	7.54	1.39	1.83	7.50
400000	2.30	3.49	14.37	57.51	2.35	9.92
500000	3.27	4.52	14.66	1.40	2.76	12.48
600000	3.31	5.35	16.09		3.20	16.69
700000	3.90	6.23	23.76		3.79	18.41
800000	3.90	5.95	19.78		4.78	26.68
900000	4.05	6.75	21.49		5.83	24.46

Таблица 1: Среднее время выполнения запроса (мс)

Сравнение производительности запросов на двумерном дереве при малых значениях K

K — количество точек в ответе, размерность точек в данном разделе $d = 2$.

Тесты строятся следующим образом: генерируется набор точек с координатами в диапазоне $[-10000, 10000]$, набор прямоугольников с координатами $[-100, 100]$. Считаем, что точки распределены равномерно. Таким образом, вероятность p попадания точки в любой из прямоугольников не превышает 10^{-4}.

Алгоритмы запускаются с одинаковым количеством памяти. В таблице 2 приведены полученные значения времени выполнения запроса. Соответствующие зависимости изображены на графике 3.

number of points n	$O(\log(n))$	$O(n)$
10000	0.0052	0.21
25000	0.0048	0.32
50000	0.0052	0.47
75000	0.0060	0.54
100000	0.0046	1.48
250000	0.0074	1.48
500000	0.0118	2.87
750000	0.0116	4.25
1000000	0.0064	5.49

Таблица 2: Среднее время выполнения запроса (мс) при малых значениях K

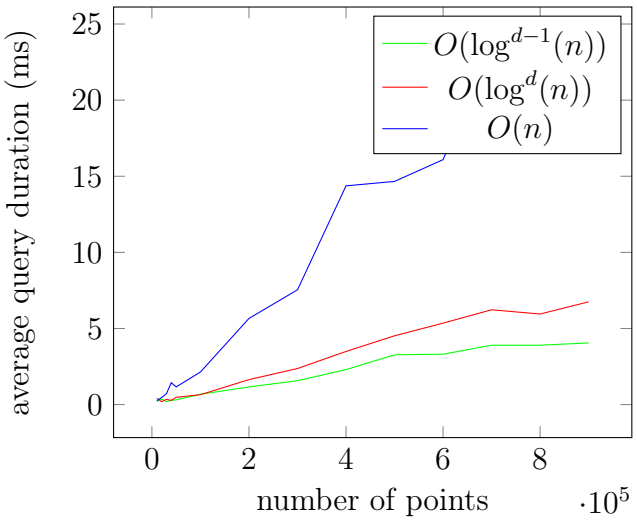

Рис. 1: $d = 2$

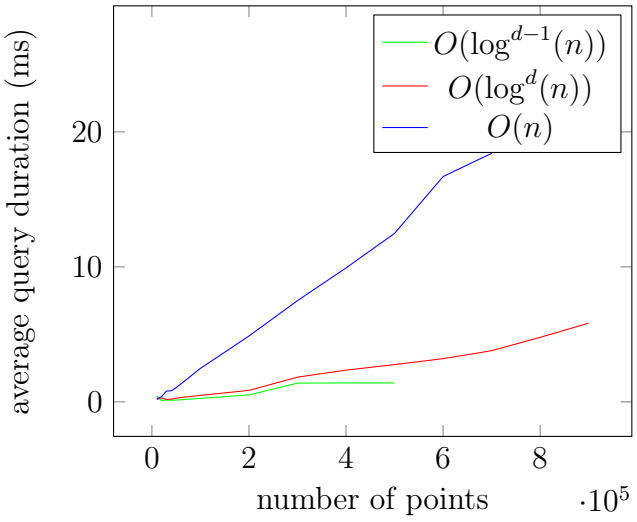

Рис. 2: $d = 3$

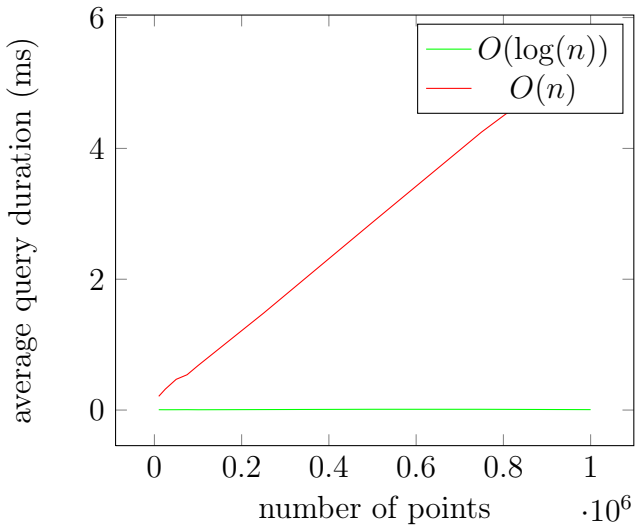

Рис. 3: $d = 2, p \leq 10^{-4}$