Иследование ассимптотик основных операций AVL и рандомизированных деревьев поиска, их сравнение, анализ

Бодров Марк и Базанов Дмитрий Б03-903

24 мая 2020 г.

1 Введение

Эта работа - наш совместный семестровый проект по информатике. Смысл её заключается в том, чтобы своими руками написать эти деревья, отладить и в дальнейшнем исследовать. Код проекта лежит здесь: https://github.com/zugzvangg/AVL-and-Random-Trees-project.

По итогу были написаны 2 дерева, интерфейс к ним и несколько тестов, сведённых в таблицы, обсчитанных и представленных в этом отчёте.

2 Что за деревья?

2.1 Рандомизированное дерерво

Рандомизированное дерерво называется так из-за особенности вставки элемента. Смысл вот в чём: если мы возьмём из любого идеально сбалансированного дерева ключи и перемешаем их, после чего попроюуем составить дерево, то его высота будет примерно равно $2log_2n$, в то время как у идеально сбалансированного высота log_2n(что в 2 раза увеличит время поиска, вставки, удаления и прочих функций, они как раз работают за log_2n). Заметим, что в этом случае корнем может с одинаковой вероятностью оказаться любой из исходных ключей. Что, если мы заранее не знаем, какие у нас будут ключи(может быть мы вводим их просто перебирая $for(int\ i = 0;\ i < n;\ i++)$), а затем наоборот от меньшего к большему, что вынуждает на каждой итерации выполнять повороты? Раз любой ключ (в том числе и тот, который мы сейчас должны вставить в дерево) может оказаться корнем с вероятностью $1/(n+1)$ (n - колличество элеметов до вставки в дерево), то мы выполняем с указанной вероятностью вставку в корень, а с вероятностью $1 - 1/(n+1)$ — рекурсивно вставляем в правое или левое поддерево в зависимости от значения ключа в корне(классическая реализация рандомизированного дерева).

2.2 AVL-дерево

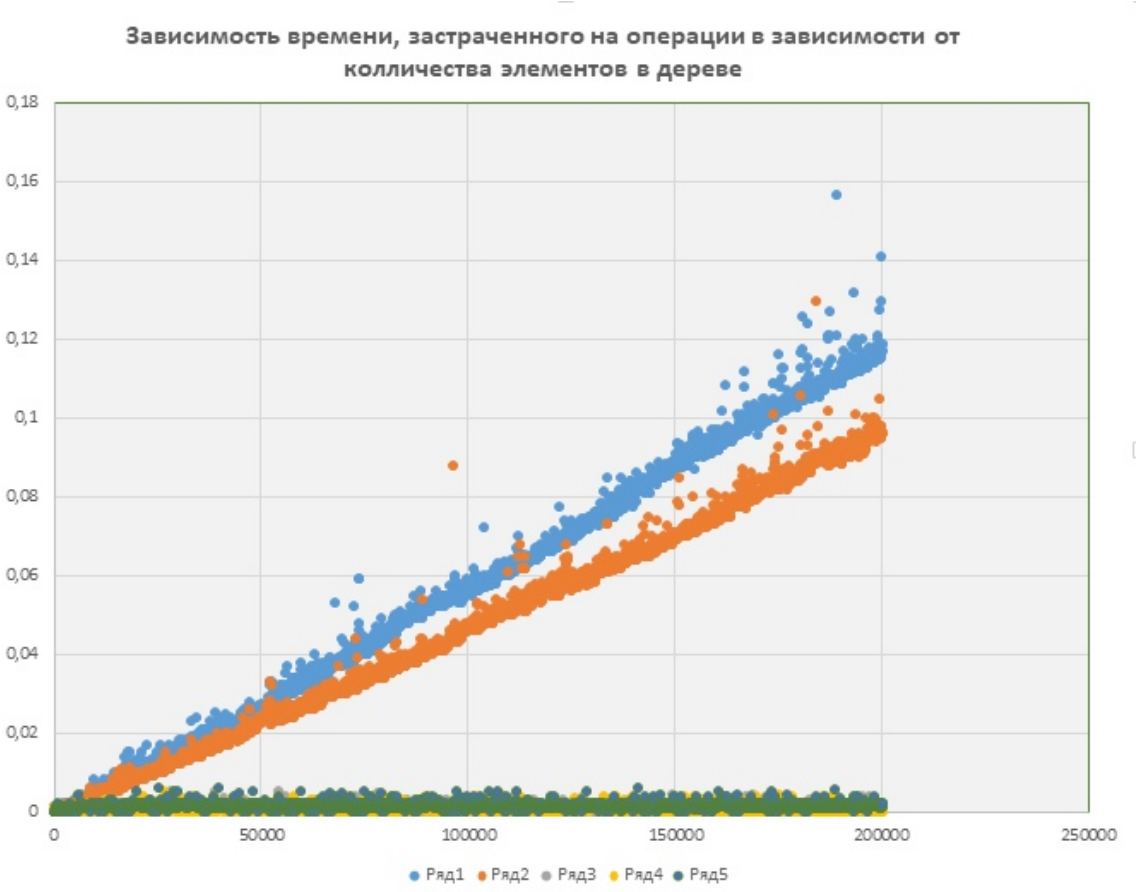
AVL дерево является идеально сбалансированным деревом. То есть разница высот двух любых его вершин не превосходит 1, что позволяет выполнять уже вышеперечисленные операции ровно за log_2n, но на пово-

роты на каждой итерации тоже тратятся временные ресурсы. Мы опытным путём, с помощью таймеров замерим и усредним, а затем сравним практическую стоимость операций вставки, удаления, поиска произвольного элемента, а также минимального и максимального.

3 Обработаем AVL-дерево

Мы написали тесты к каждой из операций(вставка, удаление, нахождение произвольного, минимального и максимального элемента в дереве). Все этит тесты были объединены в один большой тест, который проводил циклические тесты сразу всех пяти операций на объёмах данных, варьирующихся от 10^2 до $2*10^5$ элементов. Больше делать было довольно сложно, так как эти $2*10^5$ считались довольно долго, да и далее считать было довольно бессмысленно, так как тенденция на этих числах была понятна. Устройство тестов вышло понятным и также лежит здесь.

Собственно, **результаты**:



Оранжевым изображён график удаления из дерева именно n элемен-

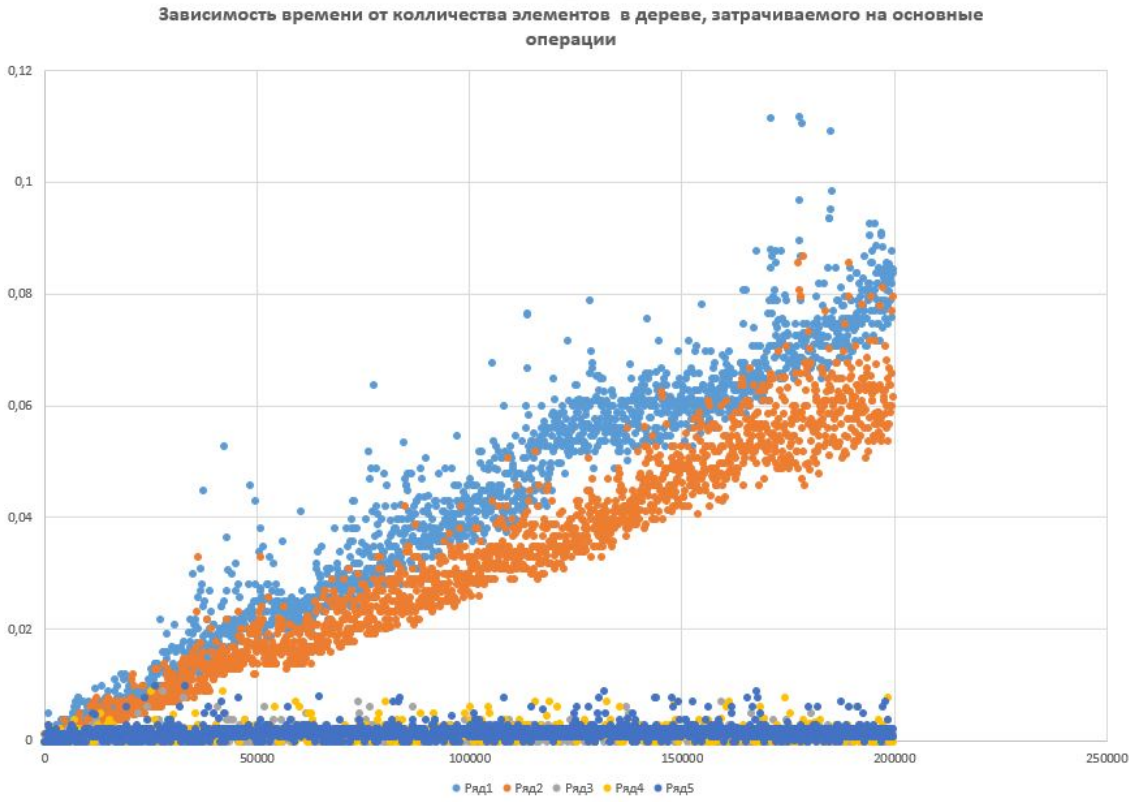
тов, а не одного. Так же дело обстоит со временем вставки в дерево, обозначенного синим цветом. Видим что, во-первых, с ростом числа элементов, вставка работает медленнее, чем удаление. С чем это связано? Почти каждую вставку элемента a в AVL дереве происходит балансировка. В удалении подобных операций не происходит, поэтому рост разницы в практической стоимости операций понятен. Визуально, а также по апроксимации некоторым графиком, у нас выходит нечто среднее между зависимостью $O(n)$ и $O(nlog(n))$. Почему так происходит? Обычное время как вставки, так и удаления элемента равно $O(log(n))$. У нас на каждой итерации теста вставляется n элементов. Теоретическое время $O(nlogn)$. Тесты устроены так, что они прогоняют в дерево целые числа в порядке возрастания, поэтому они, как правило, идут в нижние слои дерева, но ассимптотика выходит несколько лучше теоретической. С поиском элементов, как видно, вопрос обстоит интереснее. Поиск минимального, максимального и произвольного элемента происходят будто бы за константное время, хотя в теории $O(logn)$. На самом деле оно всё же ближе к $logn$. Приблизим:

Наблюдаем, что логарифмический рост действительно можно себе представить. Значение времени области кучности максимального и ми-

нимального элемента повышаются с ростом колличества элементов, что логично. Произвольный же элемент имеет 2 области скопления, как видно из графика. Это обусловлено тем, что на произвольном элементе мы ищем всегда фиксированную константу, равную всегда $n-1$(почти максимальный элемент), но зависимости между графиком максимального элемента и произвольного не наблюдается, хотя они почти одинаковы. Обусловлено разными механизмами поиска, используемыми в программе. Так что время поиска случайного элемента практически константно. Все численные данные, опять же, лежат здесь. Далее у нас:

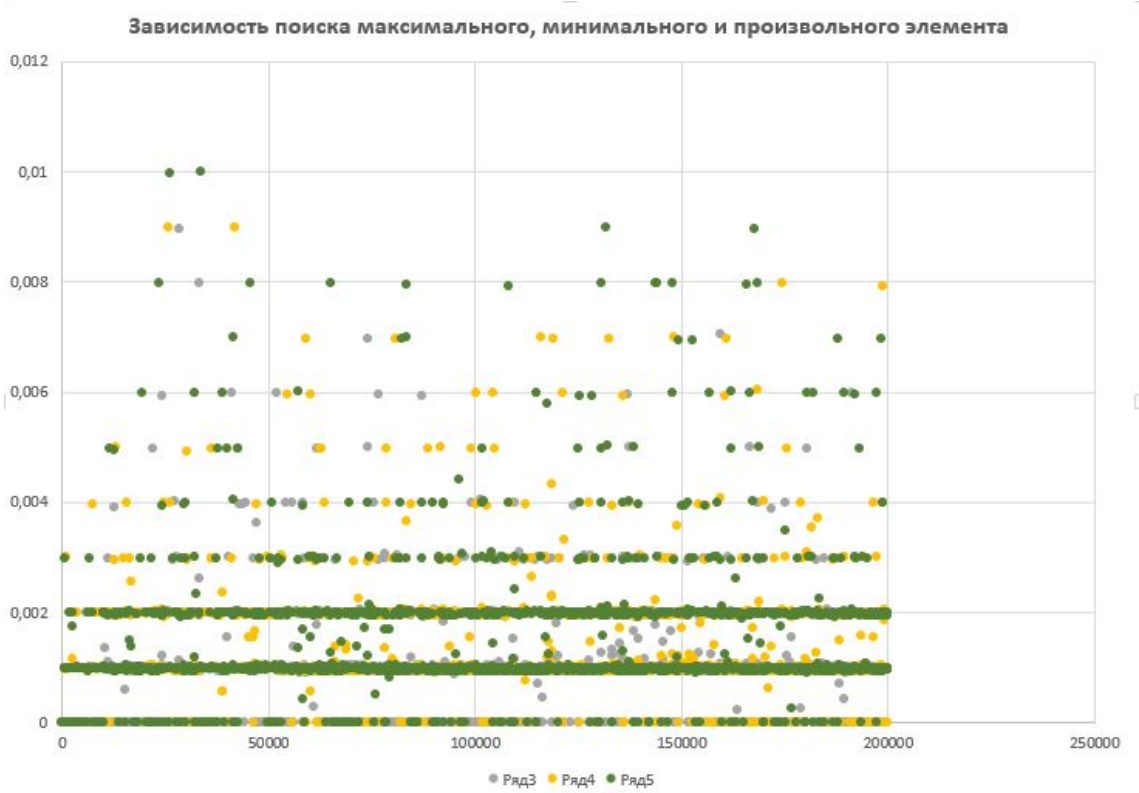
4 Обработаем рандомизированное дерево поиска

Обозначений цветов графиков те же. Тестирование происходило на точно тех же данных.



Первое отличие в то, что вставка и удаление элемента происходят с гораздо большим размахом, то есть они далеко не фиксированные. Обусловлено это как раз «рандомизированностью» дерева. То есть в зави-

исмости от случайного параметра мы вставляем элемент либо в корень дерева, либо опускаем вниз, что, очевидно, требует разного колличесва времени. График также хорошо апроксимируется чем-то средним между n и $nlogn$, но работает, что интересно, почти в 1,5 раза быстрее, если принять график за прямую. Это также объясняется тем, что некоторые элементы вставляются в корень, некоторые опускаются к листьям, что суммарно приводит к уменьшению времени работы в $O(1)$ раз. Посмотрим аналогично ближе на остальные операции:



Графики похожи. Интерес представляет то, что средние значения операция поднимаются выше, что обусловленно, конечно, тем, что дерево получается несколько менее сбалансированным, чем AVL в силу осбенностей вставки. Поэтому в среднем (проход по левому и правому краю примем примерно равными, исходя из графика)получаются больше, так же как и проход с поиском произвольного элемента.

5 А если STL контейнер?

STL set организован на красно-чёрном дереве. Мы сделали те же тесты для него, вот результат:

Зависимость очень похожая, работает он медленнее, чем наши самопальные контейнеры, связано это с тем, что set отсортирован, на что уходит много времени, в то время как операция извлечения минимального и максимального элемента у него гарантированно $O(1)$.

6 Выводы

В ходе данного «исследования» мы выяснили, что AVL дерево больше подходит, когда нам нужно иметь больший контроль над деревом. То есть быстрее доставать из него элементы, использовать другие операции, связанные с обработкой уже имеющихся данных. Random же лучше подходит, когда мы с большей частотой меняем заполнение дерева, когда оно более динамическое. Сравнение с сетом дало понять, что наши контейнеры функционируют нормально и работают за хорошее время. Авторы пошли ботать матан...