МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

ДОКЛАД
о практическом задании по дисциплине АиСД
«Сравнение сбалансированных деревьев: AA-tree и Splay-tree»

направление подготовки 09.03.03 «Прикладная информатика»
профиль «Прикладная информатика в компьютерном дизайне»

Студент группы
Б9121-09.03.03пикд
Панкратова Екатерина Денисовна
_____
(подпись)
«_____» _____ 2023г.

Доклад защищен
с оценкой _____

Руководитель практики
Доцент ИМКТ А.С. Кленин
_____
(подпись)
«_____» _____2023г.

Регистрационный номер _____
«____» _____2023г.

г. Владивосток
2023

Содержание

СОДЕРЖАНИЕ 2

АННОТАЦИЯ 3

1. ВВЕДЕНИЕ 4

1.1. ГЛОССАРИЙ 4

1.2. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 4

1.3. НЕФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ 5

2. ТРЕБОВАНИЯ К ОКРУЖЕНИЮ 6

2.1. ТРЕБОВАНИЯ К АППАРАТНОМУ ОБЕСПЕЧЕНИЮ 6

2.2. ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ 6

2.3. ТРЕБОВАНИЯ К ПОЛЬЗОВАТЕЛЯМ 6

3. АРХИТЕКТУРА СИСТЕМЫ 7

4. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ 8

5. ПРОЕКТ 9

5.1. СРЕДСТВА РЕАЛИЗАЦИИ 9

5.2. СТРУКТУРЫ ДАННЫХ 9

5.3. АЛГОРИТМ AA-TREE 9

5.4. АЛГОРИТМ SPLAY-TREE 13

6. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ 16

6.1. ВЫЧИСЛИТЕЛЬНЫЙ ЭКСПЕРИМЕНТ 16

ЗАКЛЮЧЕНИЕ 17

СПИСОК ЛИТЕРАТУРЫ 18

Аннотация

В данной работе сравниваются два алгоритма AA-tree и Splay-tree с помощью автоматических тестов. Сравнение происходит по времени выполнения теста. Также проверяется производительность и корректность работы алгоритма.

Цель: описать реализацию данных алгоритмов и с помощью тестов сравнить их.

Полученные результаты:

1. Введение

AA-tree – модификация красно-черного дерева, предложенная Арне Андерссоном в 1993 году. Это сбалансированное дерево, используемое для эффективного хранения и извлечения упорядоченных данных.

Splay-tree – двоичное дерево поиска, созданное Робертом Тарьяном и Даниелем Слейтор в 1983 году. Поддерживается свойство сбалансированности. Позволяет находить те данные, которые использовались недавно.

Цель: описать реализацию данных алгоритмов и с помощью тестов сравнить их.

Задачи:

1. Изучить теоретический материал
2. Описать данные алгоритмы
3. Реализовать данные алгоритмы
4. Реализовать тесты к алгоритмам
5. Описать результаты тестирования

Сравнение этих деревьев позволит определить в каких ситуациях следует использовать то или иное дерево, что позволит получать максимально эффективный результат.

1.1. Глоссарий

AA-tree – модификация красно-черного дерева, предложенная Арне Андерссоном в 1993 году. Это сбалансированное дерево, используемое для эффективного хранения и извлечения упорядоченных данных.

Splay-tree – двоичное дерево поиска, созданное Робертом Тарьяном и Даниелем Слейтор в 1983 году. Поддерживается свойство сбалансированности. Позволяет находить те данные, которые использовались недавно.

Амортизированный анализ – это метод анализа заданного алгоритма сложности или того, насколько ресурсов, особенно времени или памяти, которые требуются для выполнения.

Бинарное дерево поиска — дерево, в котором узлы располагаются таким образом, что каждый узел с меньшим значением (относительно родителя) находится в левой части дерева, а с большим — в правой.

Горизонтальное ребро — ребро, соединяющее вершины с одинаковым уровнем.

Красно-чёрное дерево — один из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла. Сбалансированность достигается за счёт введения дополнительного атрибута узла дерева — «цвета». Этот атрибут может принимать одно из двух возможных значений — «чёрный» или «красный».

Сбалансированное дерево поиска – такое дерево, в котором высота левого и правого поддеревьев отличаются не более чем на единицу.

Уровень вершины — вертикальная высота соответствующей вершины.

Эвристика – совокупность исследовательских методов, способствующих открытию ранее неизвестного.

Эффективность алгоритма — это свойство алгоритма, которое связано с вычислительными ресурсами, используемыми алгоритмом.

1.2. Описание предметной области

Направление исследований: сравнение AA-tree и Splay-tree, нахождение их сильных и слабых сторон, определение в каких ситуациях следует использовать то или иное дерево.

Реализация данного исследования будет состоять из написания алгоритмов деревьев, написания одинаковых тестов для них. С помощью автоматических тестов будут исследованы оба алгоритма.

Эффективность алгоритма будет оцениваться по времени, затраченному на тот или иной тест. Алгоритм, который выполнил тест быстрее другого алгоритма побеждает.

История создания алгоритмов.

AA-tree было придумано Арне Андерсоном, который решил, что для упрощения балансировки дерева нужно ввести понятие уровня вершины. Если представить себе дерево растущим сверху вниз от корня (то есть «стоящим на листьях»), то уровень любой листовой вершины будет равен 1. В своей работе Арне Андерсон приводит простое правило, которому должно удовлетворять AA-tree: к одной вершине можно присоединить другую вершину того же уровня, но только одну и только справа.

Таким образом, введенное понятие уровня вершины не всегда совпадает с реальной высотой вершины (расстояния от земли), но дерево сохраняет балансировку при следовании правилу «одна правая связь на одном уровне».

Splay-tree было придумано в середине восьмидесятых, когда Роберт Тарьян и Даниель Слейтор предложили несколько красивых и эффективных структур данных. Все они имеют несложную базовую структуру и одну-две эвристики, которые их постоянно локально подправляют.

Splay-tree — это самобалансирующееся бинарное дерево поиска. Дереву не нужно хранить никакой дополнительной информации, что делает его эффективным по памяти. После каждого обращения, даже поиска, Splay-tree меняет свою структуру.

1.3. Неформальная постановка задачи

Алгоритмы деревьев должны принимать на ввод числа и выдавать построенное дерево (обход: корень-левый потомок-правый потомок). Они должны соответствовать описанию данных деревьев и выполнять функции поиска, вставки и удаления элемента, функция вывода дерева. А также специфических функций для каждого дерева:

Для AA-tree:

Для Splay-tree: правый и левый развороты дерева, функция splay (самобалансирование дерева).

Основной набор тестов должен определять время выполнения того или иного набора команд и проверять, какой алгоритм справляется лучше. Также следует написать тесты, проверяющие корректность работы алгоритма, его производительность.

2. Требования к окружению

2.1. Требования к аппаратному обеспечению

Требования для данной работы минимальные, подойдет любой стандартный пользовательский компьютер или ноутбук.

2.2. Требования к программному обеспечению

Подойдет любая операционная система. Потребуется компилятор, установленный заранее или онлайн, поддерживающий C++.

Тестирование проводилось на ноутбуке с операционной системой Windows 10. Использовались онлайн компилятор Replit и установленный компилятор Visual Studio Code с расширением C/C++ v1.13.9.

2.3. Требования к пользователям

Требования к пользователям минимальные: для изменения и понимания работы следует знать язык программирования C++, для запуска тестов следует уметь работать с отдельными файлами и командной строкой или компилятором.

Если пользователь имеет все перечисленные навыки, он сможет полностью повторно провести тестирование системы и получить схожие результаты.

3. Архитектура системы

Система состоит изначально из 4 файлов:

1. Main.cpp – файл с главной функцией, который запускает тестирование
2. AA-tree.h – файл с алгоритмом AA-tree
3. Splay-tree.h – файл с алгоритмом Splay-tree
4. Tests.txt – файл с тестовыми данными

В результате работы файла Main.cpp создается еще один файл с результатами тестирования – Results.txt.

4. Функциональные требования

Система должна:

- Хранить информацию в виде чисел
- Иметь возможность удаления, поиска, вставки элемента
- Отвечать требованиям конкретного алгоритма
- Выполнять тестирование алгоритмов
- Выдавать отчет о тестировании

5. Проект

5.1. Средства реализации

В качестве языка программирования был выбран C++, так как он быстрее Python, подходит для целей работы и при этом более удобный и современный, чем C.

В качестве среды разработки были выбраны Replit, так как писать код в браузере удобно, и Visual Studio Code, как один из самых популярных средств, так как в браузере имеется задержка при тестировании, влияющая на результаты тестов.

5.2. Структуры данных

В данной работе используется структура данных node – узел дерева. В ней содержатся поля типа public: ключ – значение в узле (тип int), указатели на левого и правого потомков.

5.3. Алгоритм AA-tree

Свойства AA-дерева:

1. Уровень каждого листа равен 1.
2. Уровень каждого левого ребенка ровно на один меньше, чем у его родителя.
3. Уровень каждого правого ребенка равен или на один меньше, чем у его родителя.
4. Уровень каждого правого внука строго меньше, чем у его прародителя.
5. Каждая вершина с уровнем больше 1 имеет двоих детей.

Связь с красно-чёрным деревом

В отличие от красно-черных деревьев, к одной вершине можно присоединить вершину только того же уровня, только одну и только справа (другими словами, красные вершины могут быть добавлены только в качестве правого ребенка). На картинке ниже представлен пример красно-чёрного дерева.

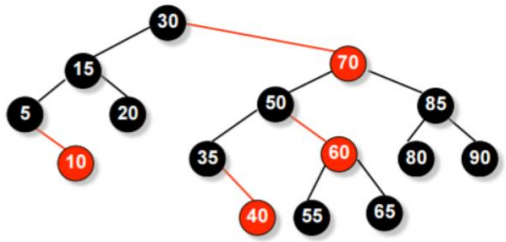

Рисунок 1

Теперь рассмотрим то же дерево, но с информацией об уровне каждой вершине. Горизонтальные ребра обозначают связи между ребрами одного уровня.

Рисунок 2

На практике в AA-дереве вместо значения цвета для балансировки дерева в вершине хранится информация только о ее уровне.

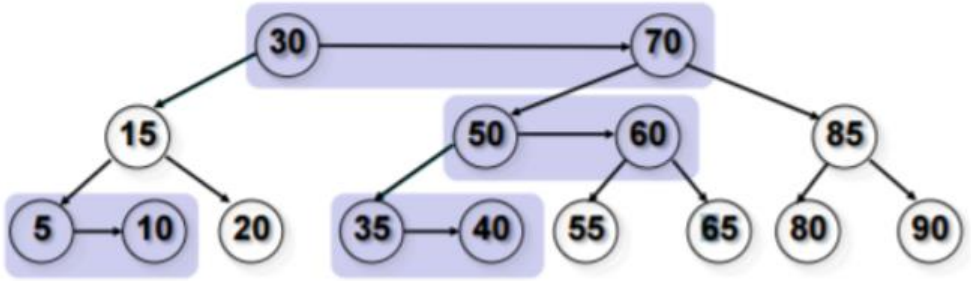

Рисунок 3

Для поддержки баланса красно-черного дерева необходимо обрабатывать 7 различных вариантов расположения вершин:

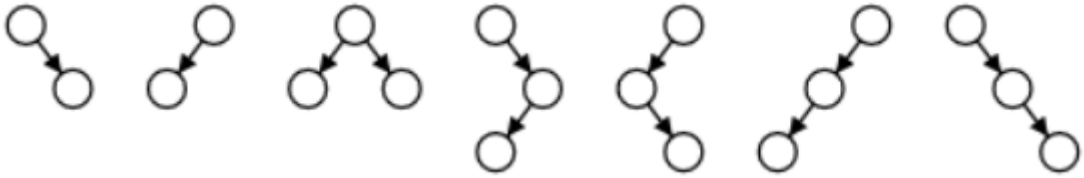

Рисунок 4

В АА-дереве из-за строгих ограничений необходимо обрабатывать только два вида возможных расположений вершин, чтобы проверить соблюдается ли главное правило «одна правая горизонтальная связь». То есть мы должны проверить нет ли левой горизонтальной связи, как на первом рисунке ниже и нет ли двух последовательных правых горизонтальных связей, как на правом рисунке.

Рисунок 5

В АА-дереве разрешены правые ребра, не идущие подряд, и запрещены все левые горизонтальные ребра. Эти более жесткие ограничения, аналогичные ограничениям на красно-черных деревьях, приводят к более простой реализации балансировки АА-дерева.

Для балансировки АА-дерева нужны следующие две операции: Skew и Split.

Skew

Skew(t) — устранение левого горизонтального ребра. Делаем правое вращение, чтобы заменить поддерево, содержащее левую горизонтальную связь, на поддерево, содержащее разрешенную правую горизонтальную связь.

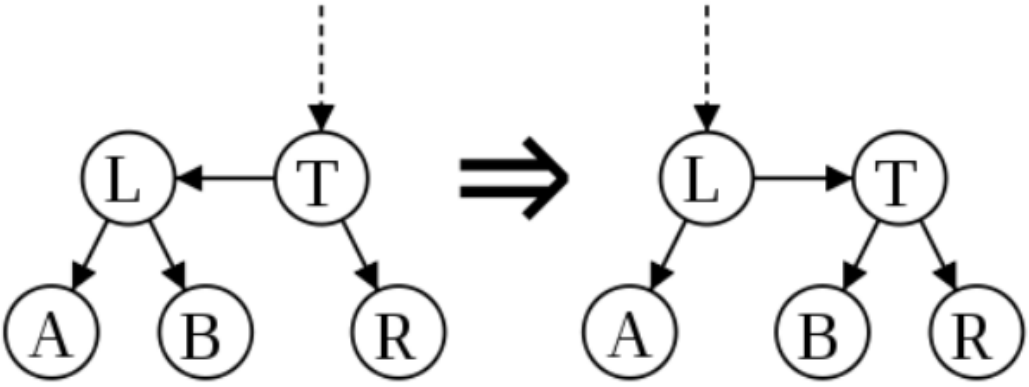

Рисунок 6

Split

Split(t) — устранение двух последовательных правых горизонтальных ребер. Делаем левое вращение и увеличиваем уровень, чтобы заменить поддерево, содержащее две или более последовательных правильных горизонтальных связи, на вершину, содержащую два поддерева с меньшим уровнем.

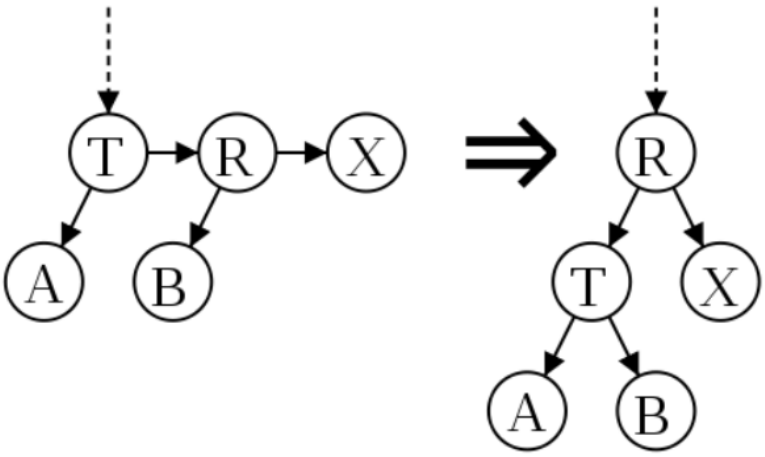

Рисунок 7

Вставка элемента

Вставка нового элемента происходит как в обычном дереве поиска, только на пути вверх необходимо делать ребалансировку, используя skew() и split().

Пример вставки нового элемента (на рис. уровни разделены горизонтальными линиями):

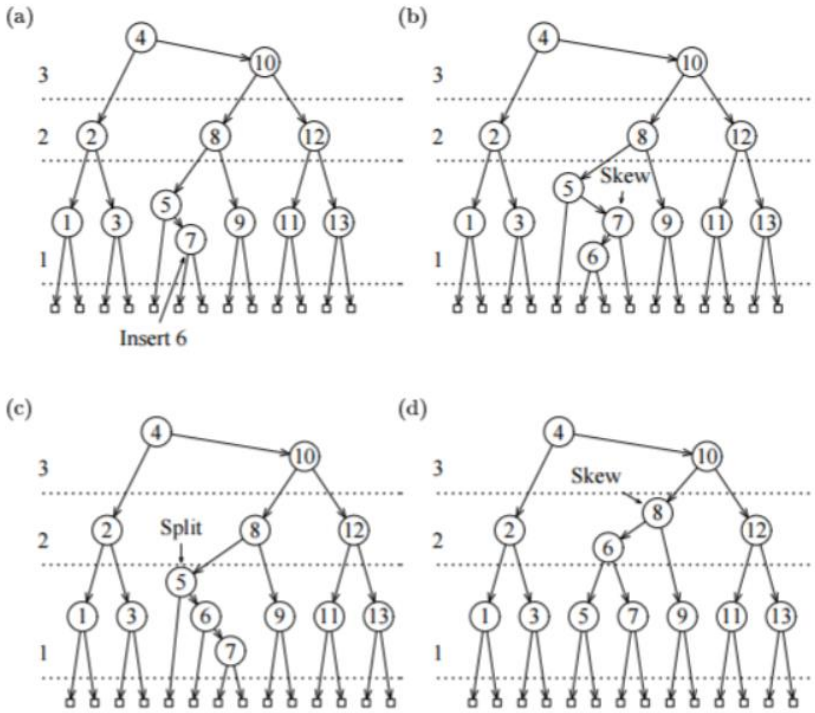

Рисунок 8

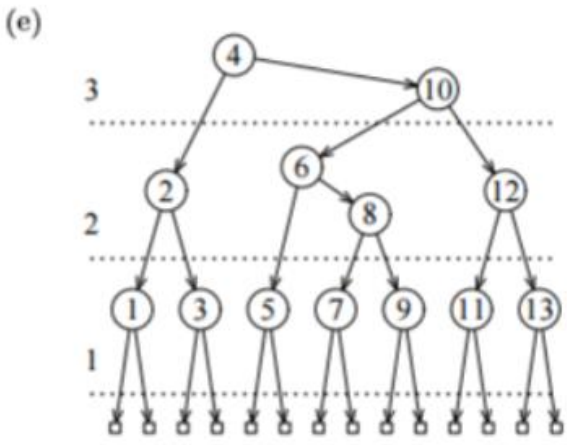

Рисунок 9

Удаление вершины

Как и в большинстве сбалансированных бинарных деревьев, удаление внутренней вершины можно заменить на удаление листа, если заменить внутреннюю вершину на ее ближайшего «предшественника» или «преемника», в зависимости от реализации. «Предшественник» находится в начале последнего левого ребра, после которого идут все правые ребра. По аналогии, «преемник» может быть найден после одного правого ребра и последовательности левых ребер, пока не будет найден указатель на NULL. В силу свойства всех узлов уровня более чем 1, имеющих двух детей, предшественник или преемник будет на уровне 1, что делает их удаление тривиальным. Ниже представлена рекурсивная реализация алгоритма.

Будем использовать дополнительную функцию decreaseLevel(), она будет обновлять уровень вершины, которую передают в функцию, в зависимости от значения уровня дочерних вершин.

Чтобы сохранять баланс дерева необходимо делать skew(), split() и decreaseLevel() для каждой вершины.

Пример удаления вершины (на рис. уровни разделены горизонтальными линиями):

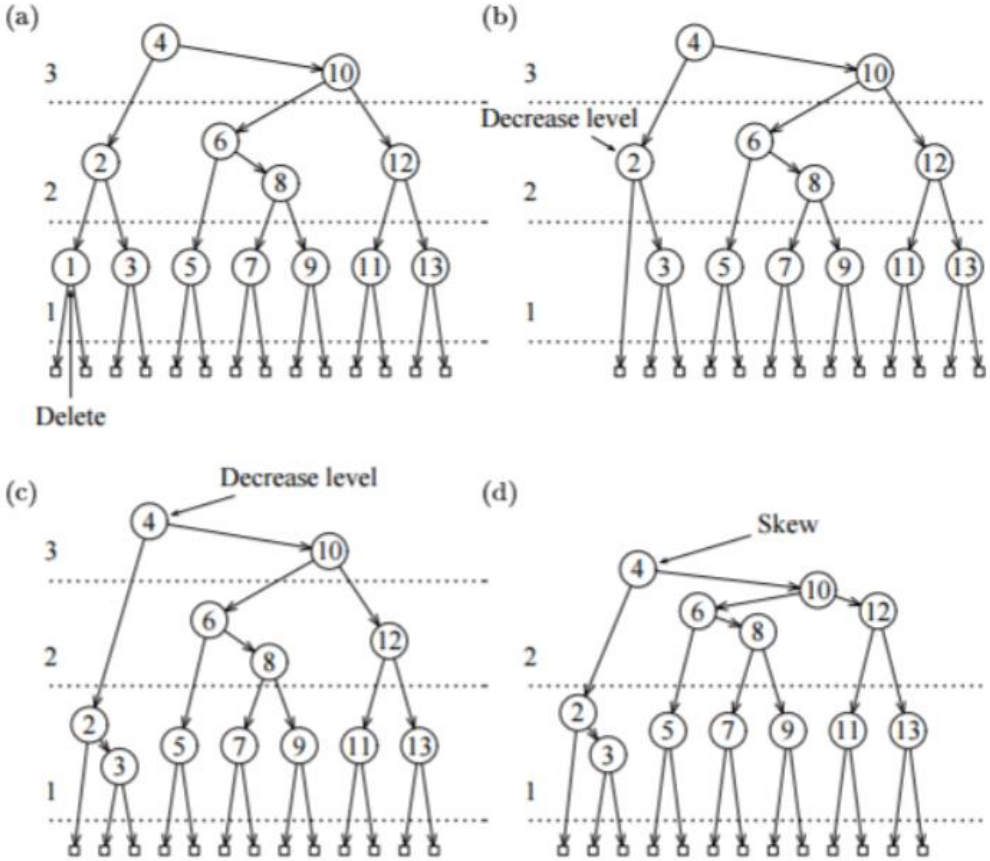

Рисунок 10

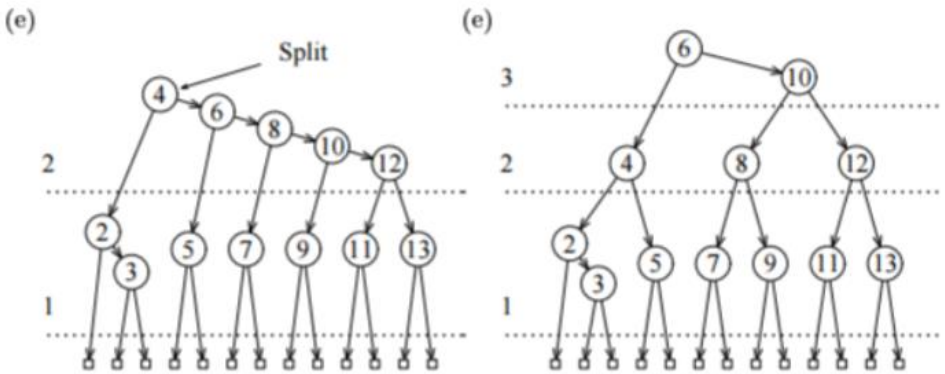

Рисунок 11

Эффективность

Оценка на высоту деревьев соответствует оценке для красно-черного дерева, 2·log2(n), так как АА-дерево сохраняет структуру красно-черного дерева. Следовательно все операции происходят за O (log n), потому что в сбалансированном двоичном дереве поиска почти все операции реализуются за O(n). Скорость работы АА-дерева эквивалентна скорости работы красно-черного дерева, но так как в реализации вместо цвета обычно хранят «уровень» вершины, дополнительные расходы по памяти достигают байта.

Преимущества

Введенное понятие уровня вершины не всегда совпадает с реальной высотой вершины, но дерево сохраняет балансировку при следовании правилу «одна правая связь на одном уровне», что делает это дерево одним из самых быстрых, но в то же время простых в реализации.

Для балансировки АА-дерева нужно всего две операции.

5.4. Алгоритм Splay-tree

Это дерево принадлежит классу «саморегулирующихся деревьев», которые поддерживают необходимый баланс ветвления дерева, чтобы обеспечить выполнение операций поиска, добавления и удаления за логарифмическое время от числа хранимых элементов. Это реализуется без использования каких-либо дополнительных полей в узлах дерева (как, например, в Красно-чёрных деревьях или АВЛ-деревьях, где в вершинах хранится, соответственно, цвет вершины и глубина поддерева). Вместо этого «расширяющие операции» (splay operation), частью которых являются вращения, выполняются при каждом обращении к дереву.

Splay (расширение)

Основная операция дерева. Заключается в перемещении вершины в корень при помощи последовательного выполнения трёх операций: Zig, Zig-Zig и Zig-Zag. Обозначим вершину, которую хотим переместить в корень за x, её родителя — p, а родителя p (если существует) — g.

Zig: выполняется, когда p является корнем. Дерево поворачивается по ребру между x и p. Существует лишь для разбора крайнего случая и выполняется только один раз в конце, когда изначальная глубина x была нечётна.

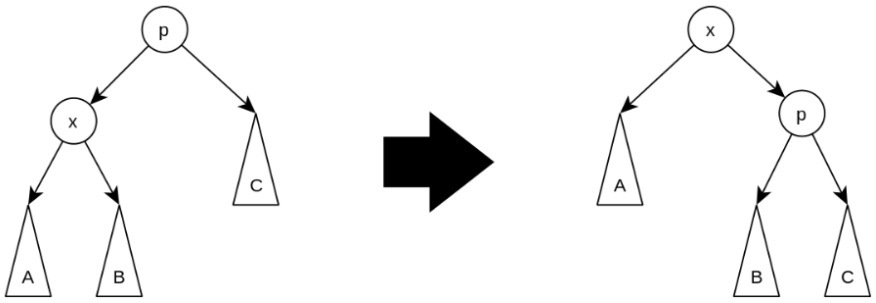

Рисунок 12

Zig-Zig: выполняется, когда и x, и p являются левыми (или правыми) сыновьями. Дерево поворачивается по ребру между g и p, а потом — по ребру между p и x.

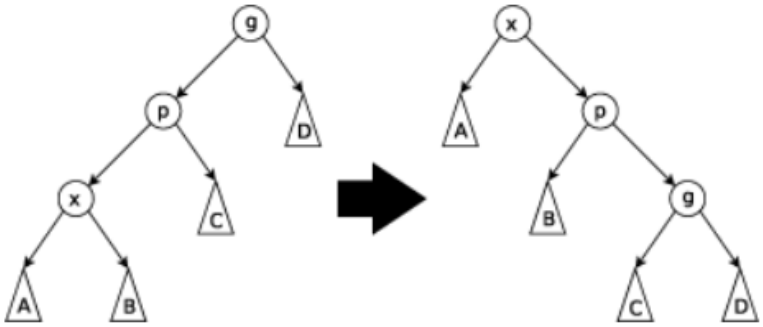

Рисунок 13

Zig-Zag: выполняется, когда x является правым сыном, а p — левым (или наоборот). Дерево поворачивается по ребру между p и x, а затем — по ребру между x и g.

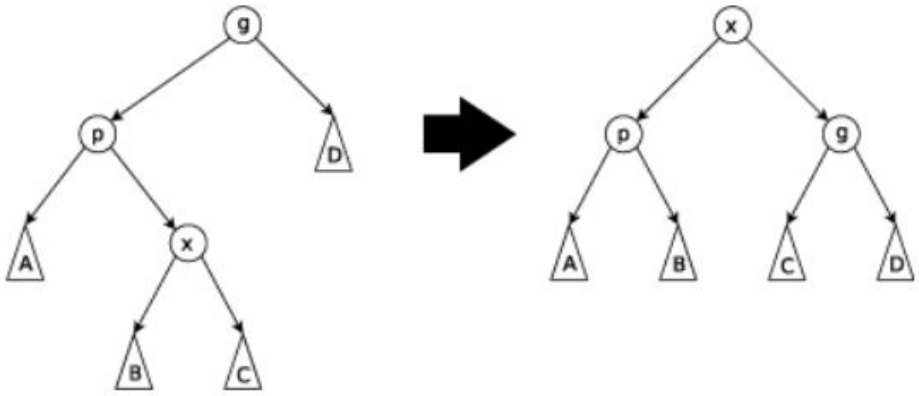

Рисунок 14

Search (поиск элемента)

Поиск выполняется как в обычном двоичном дереве поиска. При нахождении элемента запускаем Splay() для него.

Insert (добавление элемента)

Запускаем Split() от добавляемого элемента и подвешиваем получившиеся деревья за элемент к добавлению.

Delete (удаление элемента)

Находим элемент в дереве, делаем Splay() для него, делаем текущим деревом Merge() его детей.

Эффективность

Эффективность в среднем в расчёте на одну операцию с деревом составляет O (log n).

Алгоритм	Среднее	Худший случай
Пробел	O (n)	O (n)
Поиск	амортизированный O (log n)	амортизированный O (log n)
Вставить	амортизированный O (log n)	амортизированный O (log n)
Удалить	амортизированный O (log n)	амортизированный O (log n)

Рисунок 15

Преимущества

Хорошая производительность расширяемого дерева зависит от того факта, что оно самооптимизируется, поскольку часто используемые узлы будут приближаться к корню, где к ним можно будет получить доступ быстрее. Высота наихудшего случая - хотя и маловероятна - равна O (n), а средняя - O (log n). Наличие часто используемых узлов рядом с корнем является преимуществом для многих практических приложений и особенно полезно для реализации алгоритмов кэшей и сборки мусора.

Также можно отметить:

Сопоставимая производительность: производительность в среднем случае такая же эффективная, как и у других деревьев.

Небольшой объем памяти: в Splay-деревьях не требуется хранить какие-либо дополнительные данные.

Недостатки

Наиболее существенный недостаток расширяемых деревьев состоит в том, что высота расширяемых деревьев может быть линейной. Например, это произойдет после доступа ко всем n элементам в неубывающем порядке. Поскольку высота дерева соответствует наихудшему времени доступа, это означает, что фактическая стоимость одной операции может быть высокой. Однако амортизированная стоимость доступа в этом наихудшем случае является логарифмической, O (log n).

Также, когда шаблон доступа является случайным, дополнительные накладные расходы на расширение добавляют значительный постоянный фактор к стоимости по сравнению с менее динамичными альтернативами.

6. Реализация и тестирование

Привести данные о физических характеристиках текущей версии системы:

объём написанного автором кода в килобайтах и строках, отдельно по каждому языку программирования,

количество модулей, форм, экранов, страниц сайта и т. п.,

количество автоматических тестов,

количество и объём, в килобайтах, программных компонент,

фактическое быстродействие и затраты оперативной памяти, на нескольких примерах, сравнить с требованиями п. 7.2.

Указать методику тестирования: по белому или чёрному ящику, бета-тестирование, случайное тестирование. Описать процедуру тестирования (вручную или автоматически), его объём и результаты.

Сделать вывод об успешности реализации программной системы.

Если сложность описываемой системы невелика, данный раздел можно опустить, и перенести данные о характеристиках системы и её внедрении в «Заключение».

6.1. Вычислительный эксперимент

Данный раздел наиболее характерен для научно-исследовательских работ и часто является центральной частью таких работ.

Следует указать цели эксперимента, экспериментальную гипотезу (если есть). Типичными целями являются: демонстрация возможности (приблизительного) решения поставленной задачи разработанным алгоритмом, подбор оптимальных параметров алгоритма, оценка производительности и оптимизация, сравнение различных вариантов алгоритма, оценка качества прогнозирования, выполняемого алгоритмом.

Описать методику проведения эксперимента, обратить особое внимание на возможность её воспроизведения независимыми исследователями. При исследовании производительности — подробно описать использованную программно-аппаратную конфигурацию и методику измерений.

Привести результаты эксперимента в виде набора таблиц и графиков. При построении графиков, особенно сравнительных, обратить внимание на корректный выбор масштаба по осям.

Проанализировать результаты, сделать выводы о достижении целей эксперимента. Если целью эксперимента был подбор оптимальных параметров или вариантов алгоритма, перечислить полученные рекомендации.

Заключение

Раздел не нумеруется. Начать раздел фразой «Таким образом, в процессе курсовой / дипломной / др. работы мною было …», за которой перечислить виды деятельности, выполненные в рамках работы. Отделить в списке учебную деятельность («изучено», «углублены знания / повышены навыки в области…») от производственной («разработано», «спроектировано», «реализовано», и т.п.)

В перечислении избегать общих выражений («изучена предметная область»), а вместо этого использовать конкретные («изучены основы банковского дела и схема работы на примере банка X»).

Кратко перечислить основные характеристики и достоинства разработанной системы, привести данные о её внедрении и достигнутом за счёт него эффекте, указать пути дальнейшего развития системы. В случае отсутствия п. 8 «Реализация», привести краткие сведения об объёме и сложности системы.

Список литературы

1. https://www.youtube.com/watch?v=nTbD-36EA78
2. https://www.nayuki.io/page/aa-tree-set
3. https://github.com/JuYanYan/AA-Tree
4. https://iq.opengenus.org/aa-trees/
5. https://auth.geeksforgeeks.org/roadBlock.php
6. https://habr.com/ru/post/110212/
7. https://en.wikipedia.org/wiki/AA_tree
8. https://neerc.ifmo.ru/wiki/index.php?title=AA-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
9. https://ppt-online.org/87041
10. http://proteus2001.narod.ru/gen/txt/8/aa.html
11. https://alphapedia.ru/w/AA_tree
12. https://studassistent.ru/charp/aa-derevo-c
13. https://www.youtube.com/watch?v=zo8khisctxA
14. https://habr.com/ru/company/JetBrains-education/blog/210296/
15. https://www.youtube.com/watch?v=Sf0-5pjSgyQ
16. https://www.youtube.com/watch?v=qMmqOHr75b8&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=67
17. https://www.youtube.com/watch?v=1HeIZNP3w4A&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=68
18. https://www.youtube.com/watch?v=ewRSYHStdSA&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=69
19. https://www.youtube.com/watch?v=MumJoiP84J0&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=70
20. https://www.youtube.com/watch?v=IBY4NtxmGg8
21. https://ru.wikipedia.org/wiki/Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
22. https://en.wikipedia.org/wiki/Splay_tree
23. https://www.javatpoint.com/splay-tree
24. https://www.geeksforgeeks.org/splay-tree-set-1-insert/
25. https://www.youtube.com/watch?v=So8szqIvIFs
26. https://www.youtube.com/watch?v=2eCKpEmkxIc
27. https://www.youtube.com/watch?v=D9BZk1giMws
28. https://github.com/PetarV-/Algorithms/blob/master/Data%20Structures/Splay%20Tree.cpp
29. https://habr.com/ru/company/otus/blog/535316/
30. https://neerc.ifmo.ru/wiki/index.php?title=Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE#.D0.9E.D0.BF.D0.B5.D1.80.D0.B0.D1.86.D0.B8.D0.B8_.D1.81.D0.BE_splay-.D0.B4.D0.B5.D1.80.D0.B5.D0.B2.D0.BE.D0.BC
31. https://www.youtube.com/watch?v=zvZEFqxmgOY
32. https://www.youtube.com/watch?v=RmbLpFBqPqo
33. https://www.youtube.com/watch?v=almow4O2Cmg
34. https://wiki.algocode.ru/index.php?title=Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
35. https://en.wikipedia.org/wiki/Splay_tree
36. https://m.vk.com/video-54530371_456245537?list=37500ba36cfaecbe1e&from=wall10393881_1291
37. https://algorithmtutor.com/Data-Structures/Tree/Splay-Trees/
38. https://www.youtube.com/watch?v=yldKfG_0rwU
39. https://www.youtube.com/watch?v=Ex20GVEGf_s
40. https://codeforces.com/blog/entry/18462
41. https://www.youtube.com/watch?v=cILoJnFhGV0

42. https://www.youtube.com/watch?v=MoHHCiQnfuQ
43. https://www.youtube.com/watch?v=AHWbu3B6UKA
44. https://intellect.icu/derevya-poiska-avl-derevo-splej-derevo-dekartovo-derevo-65
45. https://www.tutorialspoint.com/cplusplus-program-to-implement-splay-tree
46. https://www.sanfoundry.com/cpp-program-implement-splay-tree/
47. https://wikicsu.ru/wiki/Splay_tree