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

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: разработать и описать реализацию данных алгоритмов. Протестировать данные алгоритмы с помощью автоматических тестов и сравнить результаты.

Задачи:

1. Изучить теоретический материал по Splay-tree и AA-tree
2. Описать Splay-tree и AA-tree
3. Реализовать Splay-tree
4. Придумать тесты, подходящие для обоих деревьев
5. Реализовать тесты к Splay-tree
6. Реализовать AA-tree
7. Реализовать тесты к AA-tree
8. Описать результаты тестирования Splay-tree и AA-tree

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.

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 ситуаций, когда эффективнее тот или иной алгоритм.

Эффективность алгоритма будет оцениваться по времени, затраченному на тот или иной тест.

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.

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. Постановка задачи

Код должен быть оформлен в виде библиотеки (отдельного модуля) на выбранном языке программирования.

Должны быть представлены автоматические тесты.

Язык программирования должен позволять максимально эффективную реализацию, то есть реализация алгоритма не должна быть существенно медленнее реализации на C++.

Оформление и структура кода должны подчиняться обычным критериям качества.

2. Функциональные требования

Алгоритмы должны:

- Быть оформлены в виде библиотеки (.h)
- Иметь стандартные функции деревьев (удаления, поиска, вставки элемента)
- Иметь дополнительные функции, присущие AA-tree и Splay-tree
- Хранить данные

Тесты должны:

- Выполнять автоматическое тестирование алгоритмов
- Определять время выполнения алгоритма

3. Проект

4.1. 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 был выбран Visual Studio Code, как один из самых популярных средств написания кода.

4.2. 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.

4.3. 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, красные вершины могут быть добавлены только в качестве правого ребенка.

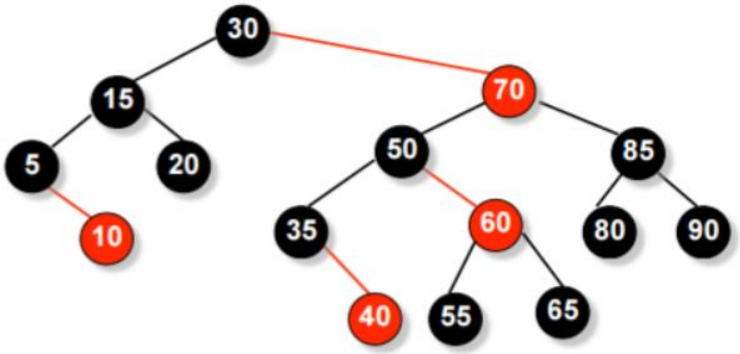

Рисунок 1. Пример красно-черного дерева

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

Рисунок 2. Пример дерева с информацией об уровне вершины

В AA-tree вместо значения цвета в вершине хранится информация только о ее уровне.

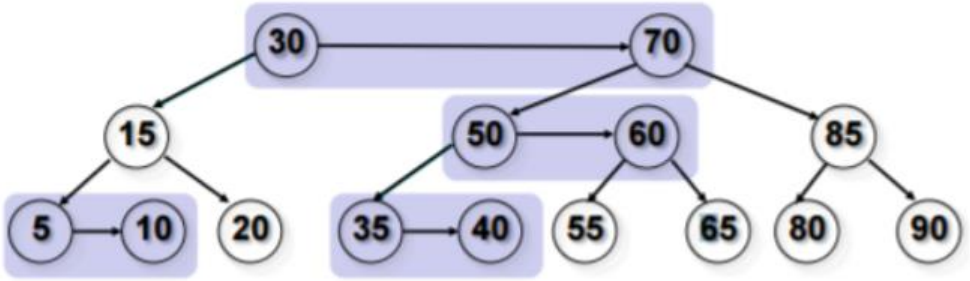

Рисунок 3. AA-tree

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

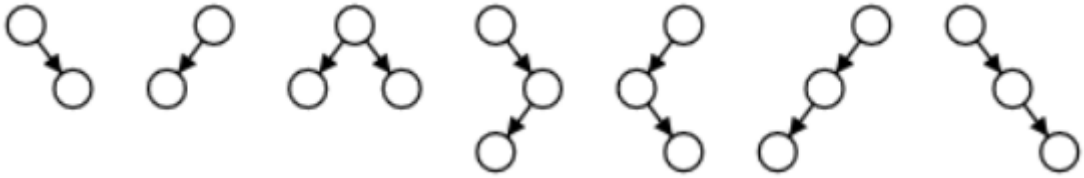

Рисунок 4. Варианты расположения вершин в красно-черном дереве

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 надо проверить нет ли левой горизонтальной связи или двух последовательных правых горизонтальных связей.

Рисунок 5. Варианты расположения вершин в AA-tree

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

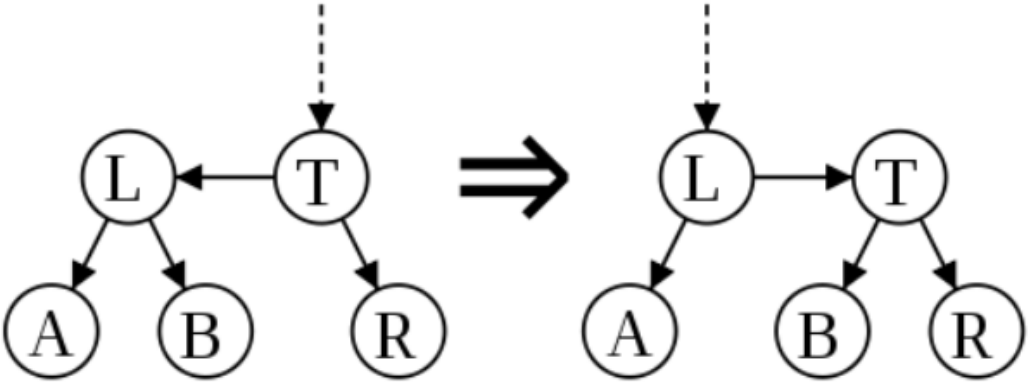

Рисунок 6. Пример операции Skew()

Split

Split() — 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.

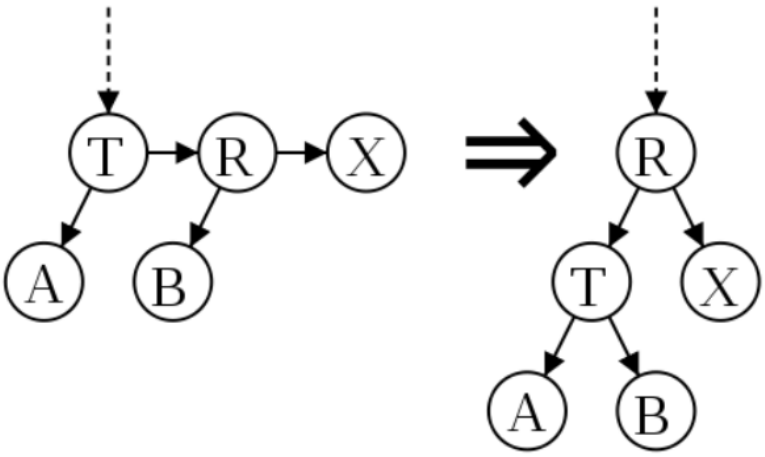

Рисунок 7. Пример операции Split()

Вставка элемента

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

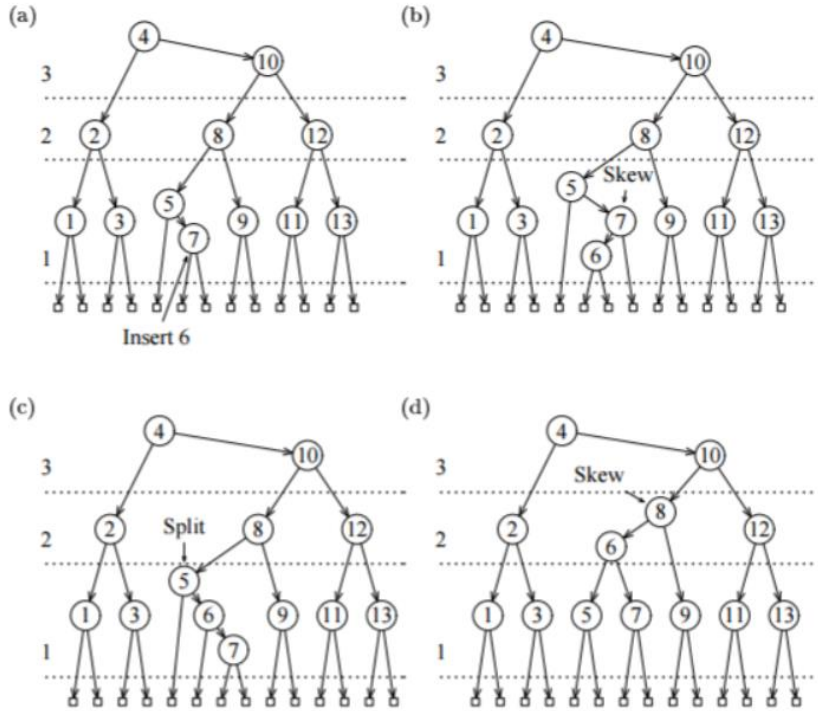

Рисунок 8.1. Пример вставки элемента

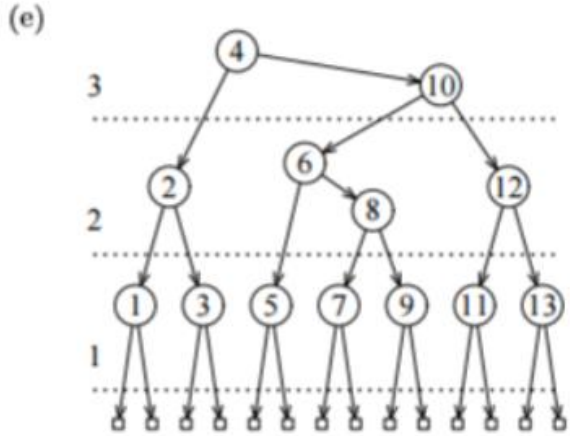

Рисунок 8.2. Пример вставки элемента

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». «Предшественник» находится в начале последнего левого ребра, после которого идут все правые ребра. «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.

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.

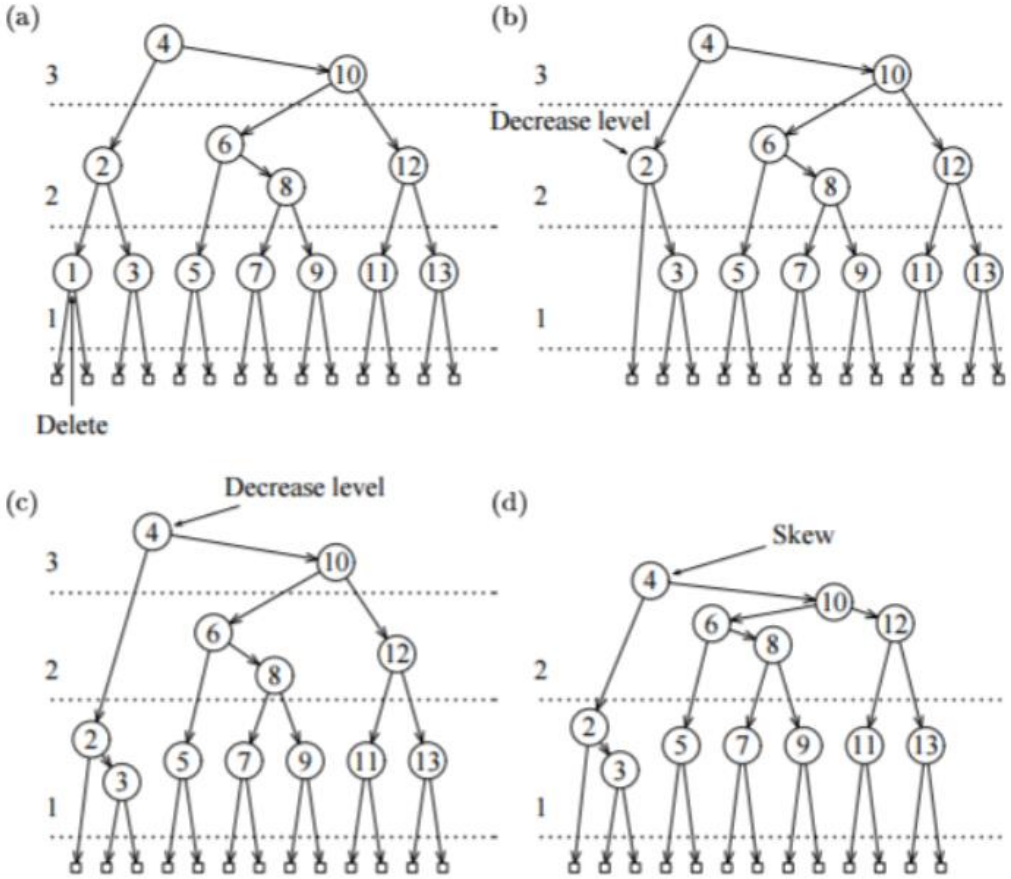

Рисунок 9.1. Пример удаления вершины

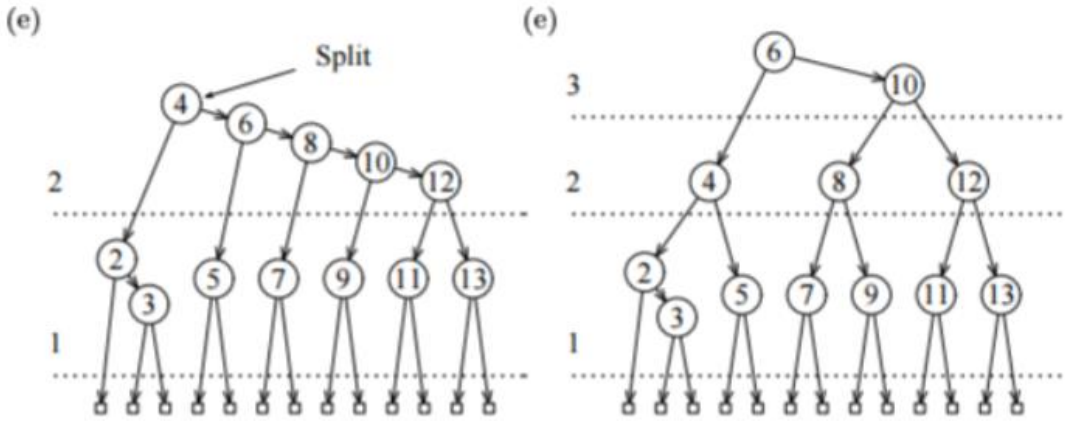

Рисунок 9.2. Пример удаления вершины

Преимущества

Дерево сохраняет балансировку при следовании правилу «одна правая связь на одном уровне», что делает это дерево быстрым и простым в реализации.

Для балансировки АА-дерева нужно всего две операции Skew и Split.

4.4. 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. 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.

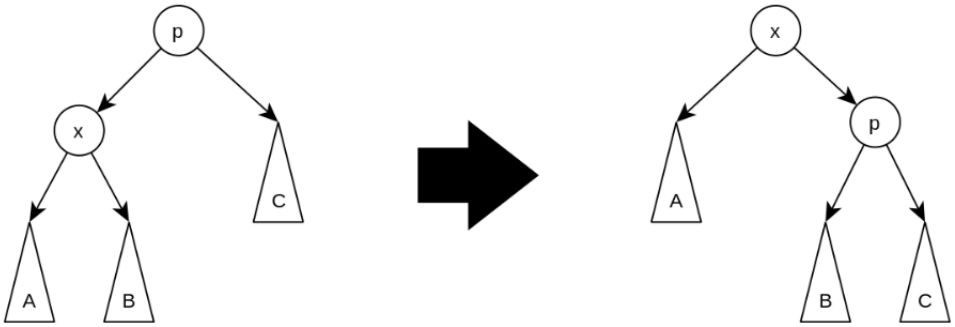

Рисунок 10. Пример операции Zig

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.

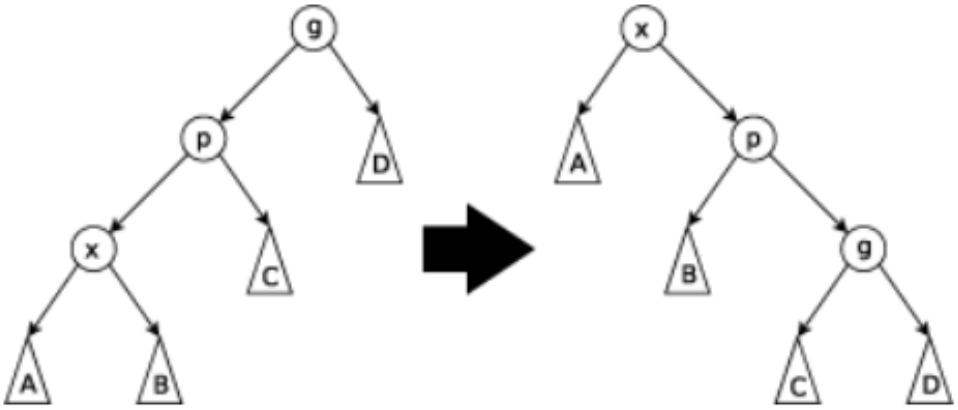

Рисунок 11. Пример операции Zig-Zig

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.

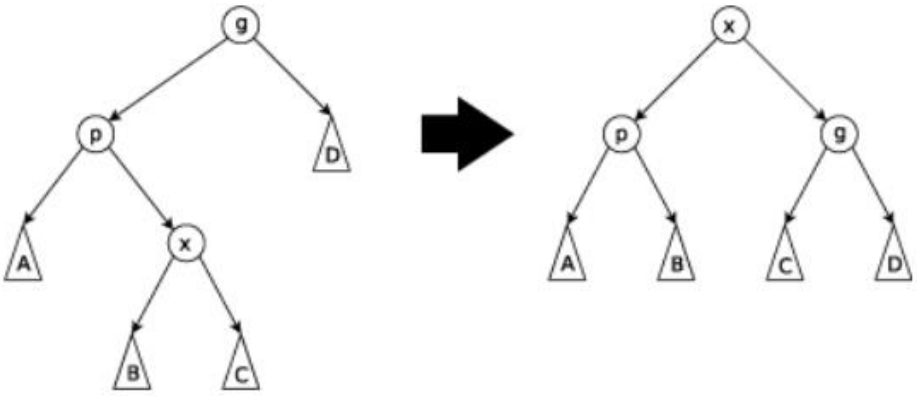

Рисунок 12. Пример операции Zig-Zag

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.

Delete (удаление элемента)

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

Преимущества

Хорошая производительность, так как это самооптимизирующееся дерево, часто используемые узлы будут приближаться к корню, где к ним можно будет получить доступ быстрее.

Небольшой объем занимаемой памяти, так как в Splay-деревьях не требуется хранить какие-либо дополнительные данные.

Недостатки

Наиболее существенный недостаток такого дерева состоит в том, что высота может быть линейной. Например, это произойдет после доступа ко всем элементам в неубывающем порядке. В таком случае эффективность алгоритма уменьшается.

Если доступ к элементам будет случайный, это увеличит время выполнения алгоритма из-за операций расширения.

4. 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».

5.1. 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=MumJoiP84J0&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=70
19. https://www.youtube.com/watch?v=IBY4NtxmGg8
20. https://ru.wikipedia.org/wiki/Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
21. https://en.wikipedia.org/wiki/Splay_tree
22. https://www.javatpoint.com/splay-tree
23. https://www.geeksforgeeks.org/splay-tree-set-1-insert/
24. https://www.youtube.com/watch?v=So8szqIvIFs
25. https://www.youtube.com/watch?v=2eCKpEmkxIc
26. https://www.youtube.com/watch?v=D9BZk1giMws
27. https://github.com/PetarV-/Algorithms/blob/master/Data%20Structures/Splay%20Tree.cpp
28. https://habr.com/ru/company/otus/blog/535316/
29. 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
30. https://www.youtube.com/watch?v=zvZEFqxmgOY
31. https://www.youtube.com/watch?v=RmbLpFBqPqo
32. https://www.youtube.com/watch?v=almow4O2Cmg
33. https://wiki.algocode.ru/index.php?title=Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
34. https://en.wikipedia.org/wiki/Splay_tree
35. https://m.vk.com/video-54530371_456245537?list=37500ba36cfaecbe1e&from=wall10393881_1291
36. https://algorithmtutor.com/Data-Structures/Tree/Splay-Trees/
37. https://www.youtube.com/watch?v=yldKfG_0rwU
38. https://www.youtube.com/watch?v=AHWbu3B6UKA
39. https://intellect.icu/derevya-poiska-avl-derevo-splej-derevo-dekartovo-derevo-65
40. https://wikicsu.ru/wiki/Splay_tree