

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**

«Сбалансированные деревья: AA-дерево и Splay-дерево»

направление подготовки 09.03.03 «Прикладная информатика»

профиль «Прикладная информатика в компьютерном дизайне»

|  |  |
| --- | --- |
| Доклад защищен  с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_  Регистрационный номер \_\_\_\_\_\_\_  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г. | Студент группы  Б9121-09.03.03пикд  Панкратова Екатерина Денисовна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022г.  Руководитель практики  Доцент ИМКТ А.С. Кленин  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г. |

г. Владивосток

2022

Оглавление

[Глоссарий 3](#_Toc121580071)

[Введение 4](#_Toc121580072)

[Глава 1. AA-tree. 5](#_Toc121580073)

[История создания 5](#_Toc121580074)

[Алгоритм 5](#_Toc121580075)

[Эффективность 10](#_Toc121580076)

[Преимущества 10](#_Toc121580077)

[Недостатки 10](#_Toc121580078)

[Тесты 10](#_Toc121580079)

[Глава 2. Splay-tree. 11](#_Toc121580080)

[История создания 11](#_Toc121580081)

[Алгоритм 11](#_Toc121580082)

[Эффективность 13](#_Toc121580083)

[Преимущества 13](#_Toc121580084)

[Недостатки 13](#_Toc121580085)

[Тесты 14](#_Toc121580086)

[Список литературы 15](#_Toc121580087)

[Приложение 17](#_Toc121580088)

# Глоссарий

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.

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

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».

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

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

# 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 и тестов к ним.

Задачи:

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

# Глава 1. AA-tree.

## История создания

AA-дерево 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-дерево: 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».

## Алгоритм

Уровень вершины (Level) — вертикальная высота соответствующей вершины.

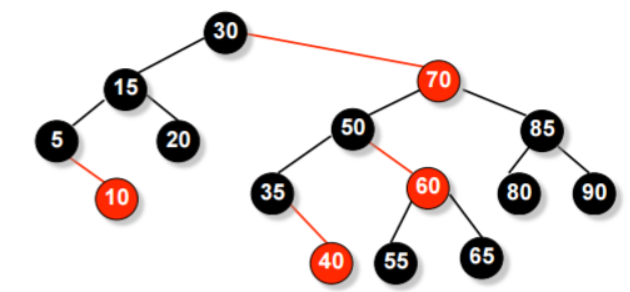
Горизонтальное ребро — ребро, соединяющее вершины с одинаковым уровнем.

Свойства АА-дерева:

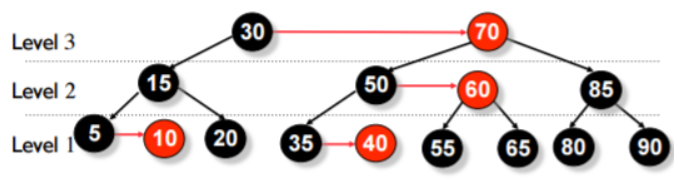
1. Уровень каждого листа равен 1.
2. Уровень каждого левого ребенка ровно на один меньше, чем у его родителя.
3. Уровень каждого правого ребенка равен или на один меньше, чем у его родителя.
4. Уровень каждого правого внука строго меньше, чем у его прародителя.
5. Каждая вершина с уровнем больше 1 имеет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

Связь с красно-чёрным деревом.

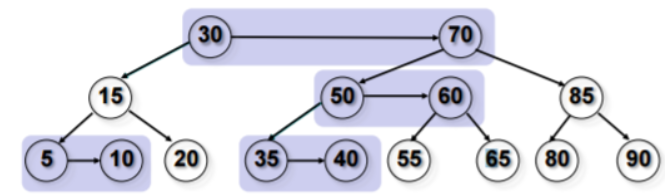
В отличие от красно-черных деревьев, к одной вершине можно присоединить вершину только того же уровня, только одну и только справа (другими словами, красные вершины могут быть добавлены только в качестве правого ребенка). На картинке ниже представлен пример красно-чёрного дерева.



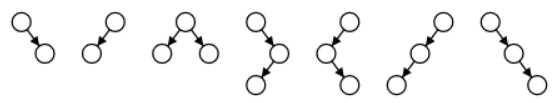
Теперь рассмотрим то же дерево, но с информацией об уровне каждой вершине. Горизонтальные ребра обозначают связи между ребрами одного уровня.



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



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



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.

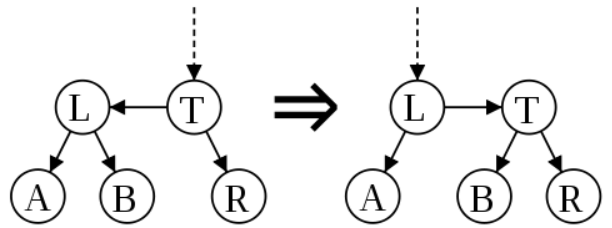


В AA-дереве разрешены правые ребра, не идущие подряд, и запрещены все левые горизонтальные ребра. Эти более жесткие ограничения, аналогичные ограничениям на красно-черных деревьях, приводят к более простой реализации балансировки AA-дерева.

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

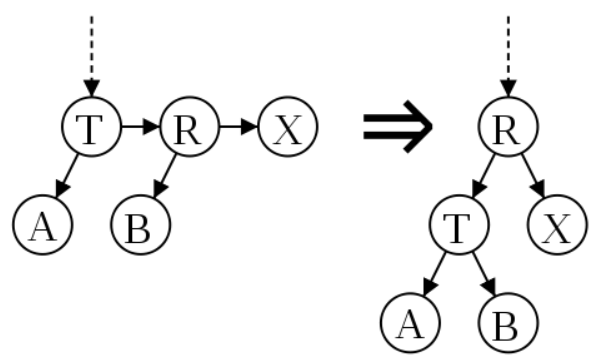
Skew

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



Split

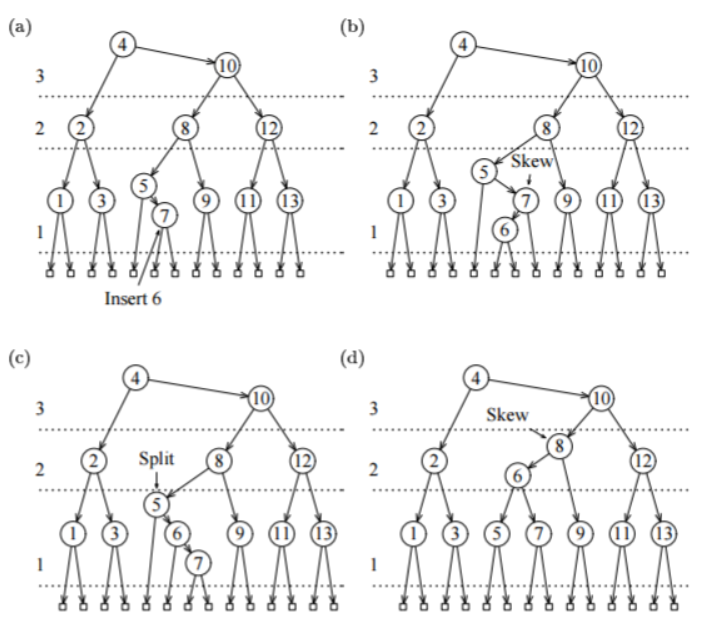
Split(t) — устранение двух последовательных правых горизонтальных ребер. Делаем левое вращение и увеличиваем уровень, чтобы заменить поддерево, содержащее две или более последовательных правильных горизонтальных связи, на вершину, содержащую два поддерева с меньшим уровнем.

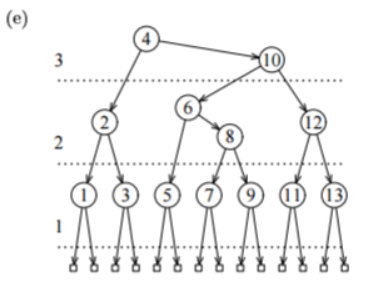


Вставка элемента.

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):





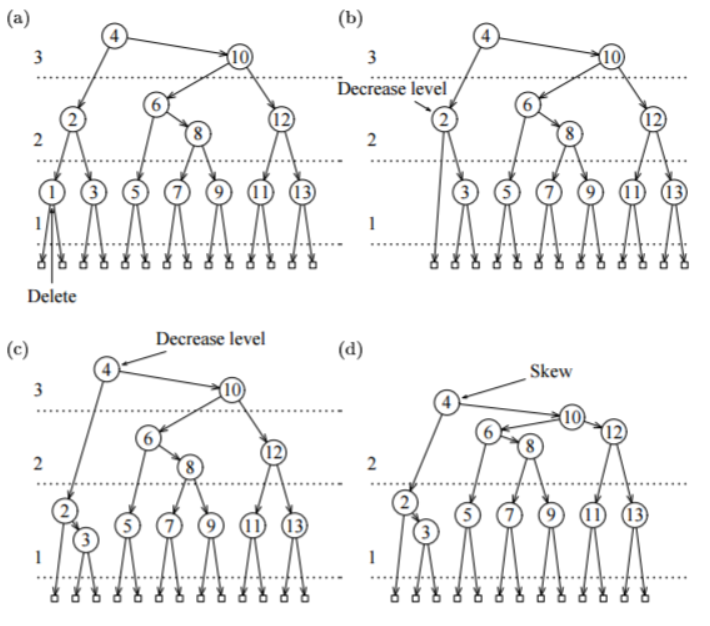
Удаление вершины.

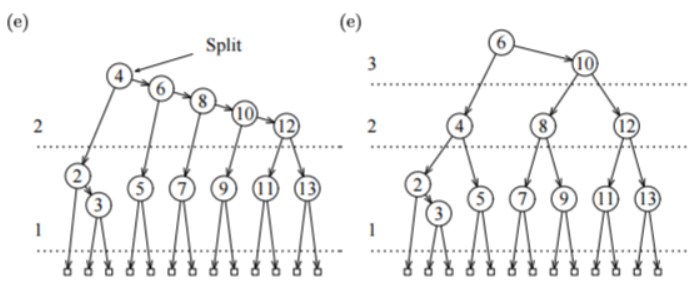
Как и в большинстве сбалансированных бинарных деревьев, удаление внутренней вершины можно заменить на удаление листа, если заменить внутреннюю вершину на ее ближайшего «предшественника» или «преемника», в зависимости от реализации. «Предшественник» находится в начале последнего левого ребра, после которого 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):





## Эффективность

Оценка на высоту деревьев соответствует оценке для красно-черного дерева, 2⋅log2(n), так как AA-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 AA-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.

## Недостатки

Надо найти и добавить в презентацию

## Тесты

Надо сделать и добавить в презентацию

# Глава 2. Splay-tree.

## История создания

В середине восьмидесятых 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-дерево — одна из таких структур.

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

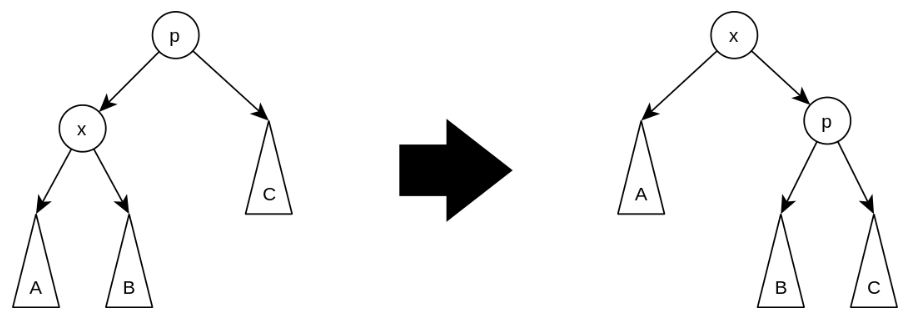
## Алгоритм

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.

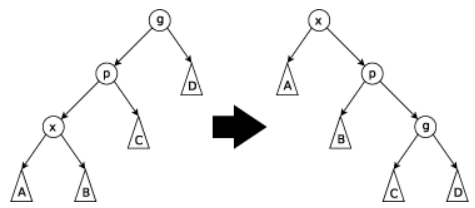
Splay (расширение)

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.

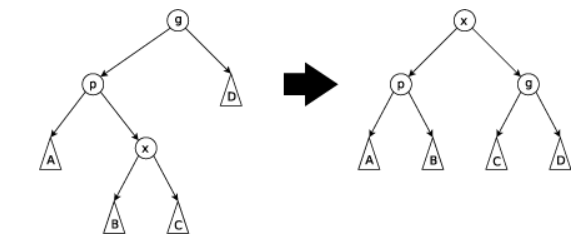
Zig: выполняется, когда p является корнем. Дерево поворачивается по ребру между x и p. Существует лишь для разбора крайнего случая и выполняется только один раз в конце, когда изначальная глубина x была нечётна.



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.



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.



Search (поиск элемента)

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

Split (разделение дерева на две части)

Для разделения дерева по значению x найдем наименьший элемент, больший или равный x, и сделаем для него Splay(). После этого отсоединяем от корня левого ребёнка и возвращаем 2 получившихся дерева.

Insert (добавление элемента)

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

Merge (объединение двух деревьев)

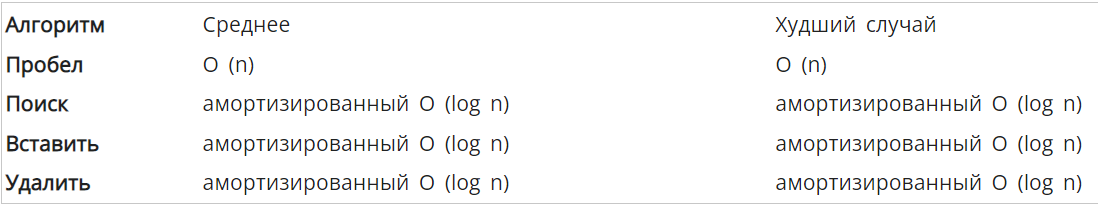
Для слияния деревьев T1 и T2, в которых все ключи T1 меньше ключей в T2, делаем Splay() для максимального элемента T1, тогда у корня T1 не будет правого ребенка. После этого делаем T2 правым ребенком T1.

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).



## 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 кешей 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.

## Тесты

Надо сделать

# 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. –
15. <https://habr.com/ru/company/JetBrains-education/blog/210296/>
16. <https://www.youtube.com/watch?v=Sf0-5pjSgyQ>
17. <https://www.youtube.com/watch?v=qMmqOHr75b8&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=67>
18. <https://www.youtube.com/watch?v=1HeIZNP3w4A&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=68>
19. <https://www.youtube.com/watch?v=ewRSYHStdSA&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=69>
20. <https://www.youtube.com/watch?v=MumJoiP84J0&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=70>
21. <https://www.youtube.com/watch?v=IBY4NtxmGg8>
22. <https://ru.wikipedia.org/wiki/Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE>
23. <https://en.wikipedia.org/wiki/Splay_tree>
24. <https://www.javatpoint.com/splay-tree>
25. <https://www.geeksforgeeks.org/splay-tree-set-1-insert/>
26. <https://www.youtube.com/watch?v=So8szqIvIFs>
27. <https://www.youtube.com/watch?v=2eCKpEmkxIc>
28. <https://www.youtube.com/watch?v=D9BZk1giMws>
29. <https://github.com/PetarV-/Algorithms/blob/master/Data%20Structures/Splay%20Tree.cpp>
30. <https://habr.com/ru/company/otus/blog/535316/>
31. <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>
32. <https://www.youtube.com/watch?v=zvZEFqxmgOY>
33. <https://www.youtube.com/watch?v=RmbLpFBqPqo>
34. <https://www.youtube.com/watch?v=almow4O2Cmg>
35. <https://wiki.algocode.ru/index.php?title=Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE>
36. <https://en.wikipedia.org/wiki/Splay_tree>
37. <https://m.vk.com/video-54530371_456245537?list=37500ba36cfaecbe1e&from=wall10393881_1291>
38. <https://algorithmtutor.com/Data-Structures/Tree/Splay-Trees/>
39. <https://www.youtube.com/watch?v=yldKfG_0rwU>
40. <https://www.youtube.com/watch?v=Ex20GVEGf_s>
41. <https://codeforces.com/blog/entry/18462>
42. <https://www.youtube.com/watch?v=cILoJnFhGV0>
43. <https://www.youtube.com/watch?v=MoHHCiQnfuQ>
44. <https://www.youtube.com/watch?v=AHWbu3B6UKA>
45. <https://intellect.icu/derevya-poiska-avl-derevo-splej-derevo-dekartovo-derevo-65>
46. <https://www.tutorialspoint.com/cplusplus-program-to-implement-splay-tree>
47. <https://www.sanfoundry.com/cpp-program-implement-splay-tree/>
48. <https://wikicsu.ru/wiki/Splay_tree>

# Приложение

**Приложение 1.**

а