**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра систем автоматизированного проектирования**

**ОТЧЕТ**

**по лабораторной работе №2**

**«Алгоритмы сортировки сравнением»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3353 |  | Шушков .В.А. |
| Преподаватель |  | Пестерев Д.О. |

Санкт-Петербург

2024

#### Тема лабораторной работы: «Самобалансирующие двоичные деревья поиска»

#### Цель лабораторной работы:

реализация самобалансирующихся деревьев поиска и экспериментальная проверка оценок высоты данных деревьев.

#### Задачи лабораторной работы:

для каждого вида дерева:

1. Реализовать функции вставки, удаления, поиска элемента. Для самобалансирующихся деревьев реализовать последующую балансировку.
2. Получить зависимость высоты дерева от количества ключей в этом дереве.
3. Сравнить практические результаты с теоретическими. Вывести итог на графики.
4. Реализовать обходы в глубину и ширину для каждого дерева.
5. **Теоретическая часть  
   Дерево АВЛ**
6. Определение

**АВЛ-дерево** - это самобалансирующееся бинарное дерево поиска, которое отличается от обычного тем, что балансировка поддеревьев всегда сохраняется в пределах определенного значения. В частности, разница в высотах левого и правого поддерева на любом уровне дерева не должна превышать 1. Это свойство дает определенные преимущества и недостатки.

Преимущества AVL - дерева заключаются в том, что оно всегда остается сбалансированным, благодаря чему его поддеревья равномерно распределены. Это обеспечивает более быстрый поиск, вставку и удаление элементов. Даже в худшем случае высота дерева остается логарифмической, что исключает «вымирание» дерева в виде линейной структуры, как может быть в обычном бинарном дереве поиска. Однако за счет частых операций балансировки и поворотов добавление и удаление элементов может занимать больше времени.

1. Описание алгоритмов вставки/удаления с последующей балансировкой для АВЛ-дерева  
   Добавление элементав AVL-дерево происходит аналогично добавлению в бинарное дерево поиска, при этом сохраняются все его свойства. Однако, после вставки элемента важно выполнить балансировку дерева с помощью поворотов. Поэтому стоит рассматривать не сами алгоритмы добавления и удаления, а алгоритмы поворотов, которые восстанавливают баланс:
2. **Правый поворот:** Этот поворот выполняется, если баланс узла превышает 1 и дисбаланс возник из-за вставки элемента в левое поддерево левого ребенка. Во время правого поворота левый ребенок проблемного узла становится новым корнем поддерева, а сам проблемный узел перемещается вправо. При этом правое поддерево нового корня становится левым поддеревом старого корня. Это уменьшает высоту левого поддерева, восстанавливая баланс.
3. **Левый поворот:** Левый поворот используется, если баланс узла меньше -1, и дисбаланс возник из-за вставки элемента в правое поддерево правого ребенка. В этом случае правый ребенок проблемного узла становится новым корнем поддерева, а сам проблемный узел перемещается влево. При этом левое поддерево нового корня становится правым поддеревом старого корня, что восстанавливает баланс правого поддерева.
4. **Лево-правый поворот:** Этот поворот применяется, если баланс узла превышает 1, но дисбаланс возник из-за вставки элемента в правое поддерево левого ребенка. Сначала выполняется левый поворот на левом ребенке, чтобы привести поддерево к форме, подходящей для правого поворота. Затем выполняется правый поворот на самом проблемном узле, чтобы восстановить баланс.
5. **Право-левый поворот:** Этот поворот используется, если баланс узла меньше -1, а дисбаланс вызван вставкой элемента в левое поддерево правого ребенка. Сначала выполняется правый поворот на правом ребенке, чтобы выровнять поддерево. После этого выполняется левый поворот на проблемном узле для окончательной балансировки дерева.

**Удаление элемента:** Удаление элемента из AVL-дерева происходит аналогично удалению из обычного бинарного дерева поиска. После выполнения базовой операции удаления необходимо проверить балансировку дерева и, если нужно, выполнить соответствующие повороты для восстановления баланса.

Таким образом, после каждой вставки или удаления выполняется балансировка дерева с помощью ротаций, что позволяет поддерживать логарифмическую сложность операций.

1. **Расчет верхней оценки высоты дерева:** Для вычисления минимального числа узлов в AVL-дереве с заданной высотой h используется рекуррентная формула: **N(h)=N(h-1)+N(h-2)+1 ,** где N(0) = 1 и N(1) = 2 . Решая рекуррентное соотношение, получаем, что минимальное кол-во узлов для дерева высоты h равно N(h)=F(h+2)-1  
   Cогласно этой ф-ле , высота AVL – дерева растет логарифмически относительно кол-ва узлов, т.е. h=O(log n). Высота дерева ограничена значением , где ϕ — это золотое сечение, примерно равное 1.618. Тогда получаем, что высота AVL-дерева зависит от количества узлов логарифмически:

**Бинарное дерево**

1. **Определение**

Бинарное дерево поиска – это структура данных, представляющая собой дерево, в котором выполняются несколько ключевых условий:

1)Каждый узел может иметь не более двух потомков: левый и правый.

2)Все значения в левом поддереве узла меньше его значения, а все значения в правом поддереве – больше.

Эти свойства определяют преимущества и недостатки бинарного дерева поиска. Одним из главных достоинств является то, что в среднем время выполнения основных операций (поиск, вставка, удаление) составляет логарифмическое время. Кроме того, дерево не требует дополнительной сортировки, так как по своей структуре оно уже отсортировано. Однако при ухудшении структуры дерева (например, в случае образования сильно неравномерных поддеревьев) операции могут становиться очень неэффективными.

1. **Описание алгоритмов вставки/удаления с последующей балансировкой  
   Добавление элемента** в бинарное дерево поиска происходит следующим образом:

1)Если дерево пусто, новый элемент становится корнем.

2)В противном случае, мы сравниваем значение нового элемента с текущим узлом. Если значение нового элемента больше, идем по правому поддереву, если меньше – по левому.

3)Этот процесс повторяется до тех пор, пока элемент не найдет свое место в дереве.  
**Удаление элемента** происходит по следующему алгоритму:

1)Если дерево пусто, функция возвращает пустое значение.

2)Если элемент, который нужно удалить, меньше или больше текущего, рекурсивно продолжаем поиск в левом или правом поддереве.

3)Когда найден узел для удаления, существует несколько случаев:

* + Если у узла нет детей, его просто удаляем.
  + Если у узла есть один ребенок, то узел заменяется его потомком.
  + Если у узла есть два ребенка, нужно найти минимальный элемент в правом поддереве, заменить им данные текущего узла и рекурсивно удалить минимальный узел.

1. **Расчет верхней оценки высоты дерева**  
   Поскольку бинарное дерево поиска не имеет встроенных механизмов балансировки, высота дерева может сильно изменяться в зависимости от ситуации, начиная от оптимального случая и заканчивая худшим. Для более точного анализа рассчитаем зависимость высоты дерева от количества элементов в среднем случае. Важно отметить, что асимптотика для лучшего и худшего случая отличается от среднего только на постоянную величину.

Средний случай предполагает, что элементы добавляются в дерево таким образом, что оно оказывается почти сбалансированным. Это близко к лучшему случаю, когда дерево идеально сбалансировано, но с некоторыми отклонениями в высоте поддеревьев. Рассмотрим это на примере:

Когда элементы добавляются в дерево, оно делится на два поддерева, но с небольшими различиями в их размерах. Из этого анализа можно сделать вывод, что в лучшем случае высота дерева будет log2​(n), так как дерево будет структурировано максимально сбалансированно, как показано в примере.

Проверим это для дерева с 15 элементами. По формуле, что в итоге равно 4. Это подтверждает, что в лучшем случае высота дерева будет примерно log2​(n). В среднем же мы можем ожидать, что высота дерева будет близка к log2​(n), с добавлением некоторой константы C, которую можно уточнить с помощью практических экспериментов.

В заключение, верхняя оценка для высоты бинарного дерева поиска в среднем случае будет

**Красно-черное дерево**

1. **Определение**

Красно-черное дерево — это самобалансирующаяся структура данных, являющаяся улучшением обычного бинарного дерева поиска. Это дерево сохраняет все свойства бинарного дерева поиска, а именно:

* Каждый узел имеет один из двух цветов: красный или черный.
* Корневой узел всегда черного цвета.
* У красного узла не может быть красного потомка.
* Все листовые узлы (NULL-элементы) всегда черные.
* На всех путях от корня до листа должно быть одинаковое количество черных узлов.

Такие свойства обеспечивают балансировку дерева и помогают поддерживать эффективность операций поиска, вставки и удаления, гарантируя, что они выполняются за логарифмическое время.

1. **Описание алгоритмов вставки/удаления с последующей балансировкой**Добавление элемента в красно-черное дерево происходит аналогично вставке в обычное бинарное дерево поиска. Однако после вставки элемента необходимо выполнить балансировку дерева, что включает повороты и изменение цветов узлов для восстановления его свойств.

**1.Правый поворот:**  
Этот поворот применяется, когда дерево нарушает баланс из-за того, что левое поддерево слишком высоко. Суть поворота заключается в том, что элемент, находящийся в нижней части левого поддерева, поднимается наверх, а элемент, вызвавший дисбаланс, перемещается в правую сторону.

**2.Левый поворот:**  
Если баланс нарушен из-за слишком высокого правого поддерева, то выполняется левый поворот. В этом случае элемент, находящийся в нижней части правого поддерева, поднимается, а нарушающий баланс элемент перемещается влево.

**3.Лево-правый поворот:**  
Если элемент был вставлен в правую часть левого поддерева, то сначала выполняется левый поворот на нижнем уровне, чтобы подготовить поддерево, а затем выполняется правый поворот на более высоком уровне для восстановления общего баланса дерева.

**4.Право-левый поворот:**  
Этот поворот используется, если элемент был добавлен в левую часть правого поддерева. Вначале выполняется правый поворот на нижнем уровне, после чего выполняется левый поворот на более высоком уровне.

**5.Перекраска элемента :**

Если нарушается одно из свойств дерева (например, корень не черный, красный узел имеет красного ребенка, или в каждом пути от корня до листа разное количество черных узлов), требуется выполнить перекраску. Процесс начинается с проверки, является ли родитель вставленного элемента красным. Если да, то необходимо перекрасить его, поскольку одно из свойств нарушено. Далее возможны четыре различных варианта действий, пока существует родитель и его цвет остается красным.

1. Если родитель элемента — левый ребенок, и у него есть дядя, который тоже красный, тогда родитель и дядя перекрашиваются в черный, а дедушка становится красным. В этом случае больше не нужно рассматривать сам элемент, а переключаемся на его дедушку.
2. Если дядя либо отсутствует, либо черный, и элемент является правым ребенком, то необходимо рассматривать не сам элемент, а его родителя. Родителю выполняется левый поворот, после чего его цвет становится черным, а дедушка перекрашивается в красный. Далее выполняется правый поворот для дедушки.
3. Если родитель элемента — правый ребенок и у него есть красный дядя, то снова выполняется перекраска: родитель и дядя становятся черными, а дедушка — красным. В этом случае, как и в первом варианте, рассматривается не сам элемент, а дедушка.
4. Если дядя отсутствует или его цвет черный, то проверяется, является ли элемент левым ребенком. Если да, то для родителя выполняется правый поворот. Затем родитель становится черным, а дедушка — черным, после чего выполняется левый поворот для дедушки.

Таким образом, повороты выполняются только во втором и четвертом вариантах. Во втором варианте это может быть либо лево-правый поворот, либо просто правый. В четвертом варианте возможны либо право-левый поворот, либо просто левый.

**Алгоритм удаления:**

Как и при добавлении элемента, сначала выполняется стандартная операция удаления для бинарного дерева поиска, а затем происходит изменение цветов в соответствии с алгоритмом. Этот процесс начинается только в случае, если элемент, который заменил удаленный, имеет черный цвет.

1. Пока узел не является корнем и его цвет не черный, проверяется, является ли элемент левым ребенком. Если это так и если брат элемента красный, то мы перекрашиваем брата в черный, а родителю присваиваем красный цвет. Затем выполняем левый поворот и обновляем информацию о том, что брат теперь правый ребенок родителя.
2. Если у брата нет левого ребенка или его левый ребенок черный, а также если у брата нет правого ребенка или его правый ребенок черный, то мы меняем цвет брата на красный и начинаем рассматривать родителя.
3. Если ни одно из условий выше не выполнено, то проверяется, есть ли у брата правый ребенок, или его правый ребенок черный. Если у брата нет правого ребенка или его цвет черный, то нужно проверить наличие левого ребенка. Если он есть, его цвет становится черным. После этого меняем цвет брата на красный, выполняем правый поворот для брата и снова проверяем брата. После того как правый ребенок брата будет проверен, мы меняем цвет брата на родительский, а родителю присваиваем черный цвет. Затем проверяем, есть ли у брата правый ребенок, и если есть, его цвет становится черным. В завершение выполняем левый поворот для родителя и продолжаем с корнем дерева.

4, 5 и 6. Эти шаги аналогичны первым трем, но с зеркальной логикой. Например, если элемент является правым ребенком, вместо левого поворота будет выполнен правый поворот.

По завершению всех операций цвет корня устанавливается в черный.

1. **Расчет верхней оценки высоты дерева**Для оценки высоты красно-черного дерева можно использовать свойства его структуры:

Пусть bh — это черная высота дерева, то есть количество черных узлов на пути от корня до любого листа.

Максимальная высота дерева h ограничена значением h≤2⋅bh, потому что на пути от корня к листу количество черных узлов одинаково, а красные узлы могут чередоваться, не увеличивая высоту.

Для минимального количества узлов при черной высоте bh (идеальное черное дерево) имеем:

Пусть n — количество узлов в дереве. Из свойства дерева, что на каждом пути от корня к листу количество черных узлов одинаково, можем записать:

Решая относительно bh, получаем:

Используя ограничение для максимальной высоты дерева, имеем:

Таким образом, верхняя оценка высоты красно-черного дерева будет:

Это значит, что даже в худшем случае высота красно-черного дерева остается логарифмической относительно количества узлов в дереве.

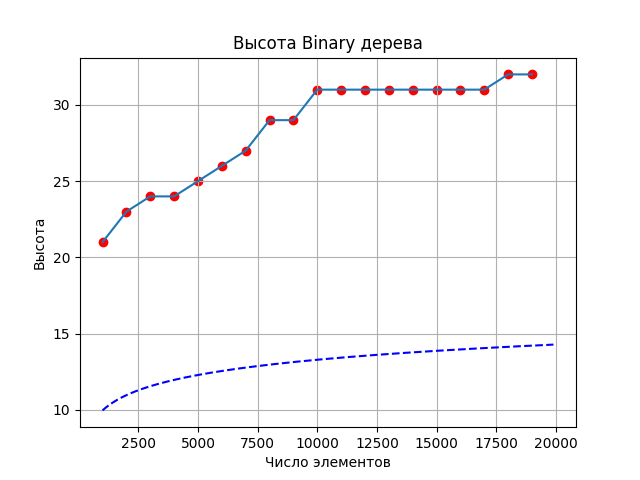
**Практическая часть.**

# **Зависимость высоты для бинарного дерева поиска.**

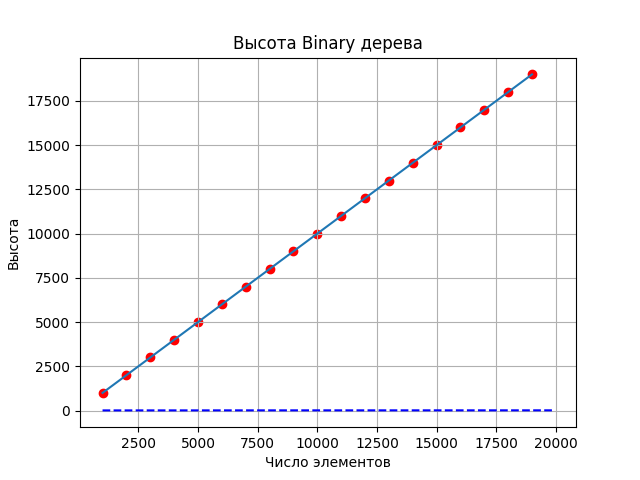
Таблица 1. Экспериментально полученные значения для зависимости высоты бинарного дерева поиска от количества ключей

|  |  |  |
| --- | --- | --- |
| Номер | Количество ключей | Высота |
| 1 | 1000 | 21 |
| 2 | 2000 | 23 |
| 3 | 3000 | 24 |
| 4 | 4000 | 24 |
| 5 | 5000 | 27 |
| 6 | 6000 | 29 |
| 7 | 7000 | 29 |
| 8 | 8000 | 31 |
| 9 | 9000 | 31 |
| 10 | 10000 | 31 |
| 11 | 11000 | 31 |
| 12 | 12000 | 31 |
| 13 | 13000 | 31 |
| 14 | 14000 | 31 |
| 15 | 15000 | 31 |
| 16 | 16000 | 31 |
| 17 | 17000 | 31 |
| 18 | 18000 | 32 |
| 19 | 19000 | 32 |

На первый взгляд, по представленному графику трудно определить, насколько значения соответствуют теоретическому предсказанию O(log n). Для более точного сравнения теории и практики необходимо построить график логарифма. Однако, как показано в теории, график логарифма для бинарного дерева поиска может иметь некоторую константу, чтобы точно соответствовать полученному графику.



Изображение 4. График экспериментальных значений для бинарного дерева поиска с теоретическим графиком.



Изображение 5 . График экспериментальных значений , монотонно возрастающих.

Можно заметить, что график не совсем совпадает с теоретическим предсказанием. Однако важно учитывать, что в теоретической части была получена асимптотическая оценка, что означает, что на очень больших значениях график в среднем выровняется и частично будет соответствовать теоретическим значениям. Скачки на графике могут быть связаны с тем, что бинарное дерево поиска не является самобалансирующимся, и в некоторых случаях его значения могут колебаться.

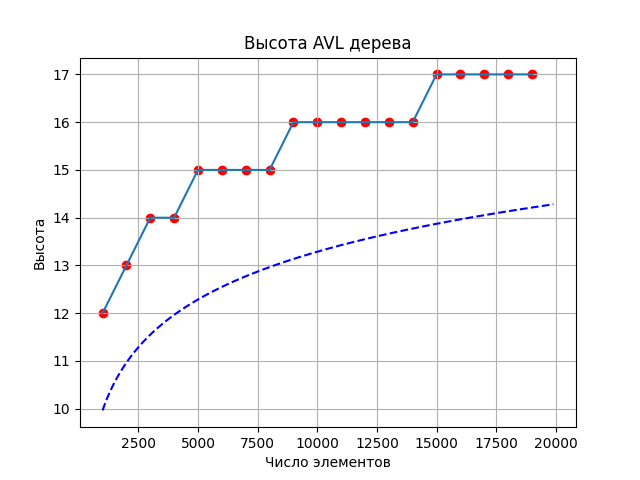
# **Зависимость высоты для AVL дерева**

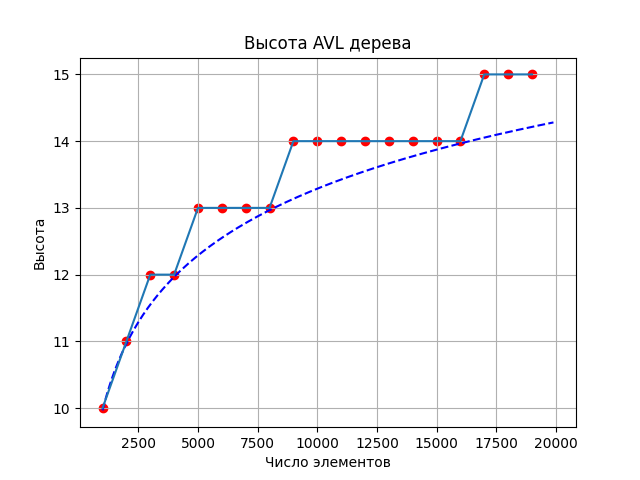
Таблица 2. Экспериментально полученные значения для зависимости высоты AVL дерева от количества ключей

|  |  |  |
| --- | --- | --- |
| Номер | Количество ключей | Высота |
| 1 | 1000 | 12 |
| 2 | 2000 | 13 |
| 3 | 3000 | 14 |
| 4 | 4000 | 14 |
| 5 | 5000 | 15 |
| 6 | 6000 | 15 |
| 7 | 7000 | 15 |
| 8 | 8000 | 15 |
| 9 | 9000 | 16 |
| 10 | 10000 | 16 |
| 11 | 11000 | 16 |
| 12 | 12000 | 16 |
| 13 | 13000 | 16 |
| 14 | 14000 | 16 |
| 15 | 15000 | 17 |
| 16 | 16000 | 17 |
| 17 | 17000 | 17 |
| 18 | 18000 | 17 |
| 19 | 19000 | 17 |

Изучив данный график зависимости высоты дерева от количества элементов, можно предположить, что он соответствует теоретическим значениям, поскольку каждый следующий отрезок графика длиннее предыдущего, что характерно для логарифмического графика. Однако для большей точности, как и в случае с бинарным деревом поиска, следует построить график логарифма и сравнить оба графика.

Изображение 5. График экспериментальных значений для AVL дерева с теоретическим графиком.



Как и было сказано ранее, график полностью совпал с ожидаемым, отсутствие скачков обусловлено свойствами AVL дерева.

Изображение 6. График экспериментальных значений , монотонно возрастающих.

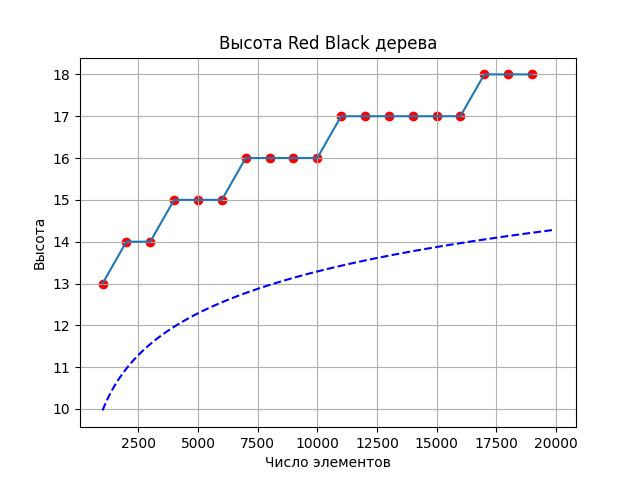
# **Зависимость высоты для красно-черного дерева.**

Таблица 3. Экспериментально полученные значения для зависимости высоты красно-черного дерева от количества ключей

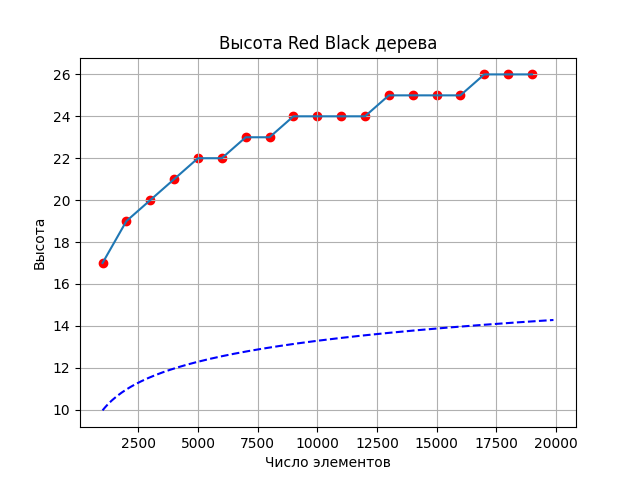
|  |  |  |
| --- | --- | --- |
| Номер | Количество ключей | Высота |
| 1 | 1000 | 13 |
| 2 | 2000 | 14 |
| 3 | 3000 | 14 |
| 4 | 4000 | 15 |
| 5 | 5000 | 15 |
| 6 | 6000 | 15 |
| 7 | 7000 | 16 |
| 8 | 8000 | 16 |
| 9 | 9000 | 16 |
| 10 | 10000 | 16 |
| 11 | 11000 | 17 |
| 12 | 12000 | 17 |
| 13 | 13000 | 17 |
| 14 | 14000 | 17 |
| 15 | 15000 | 17 |
| 16 | 16000 | 17 |
| 17 | 17000 | 18 |
| 18 | 18000 | 18 |
| 19 | 19000 | 18 |

Исходя из изображения , график красно-черного почти сходится с нашим теоретическим графиком , это связано с тем , что высота красно-черного самобалансируется.

Изображение 6. График экспериментальных значений для красно-черного дерева с теоретическим графиком.



В отличии от AVL дерева , высота красно-черного дерева может отличаться в узлах в 2 раза, это связано со свойствами красно-черного дерева

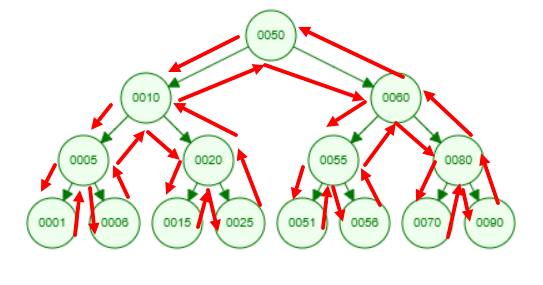


Изображение 7. График экспериментальных значений , монотонно возрастающих.

Однако подводя итоги по красно-черному дереву, можно сказать, что в целом теория вновь совпала с практикой.

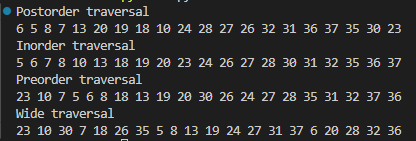
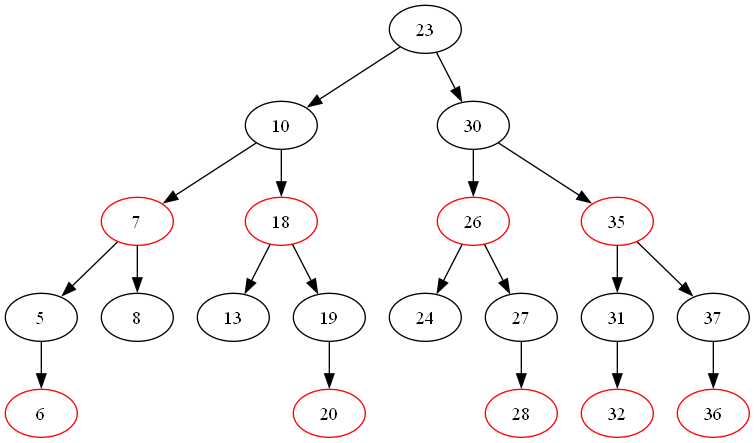
# **Реализация обхода деревьев**

Обход был реализован только для бинарного дерева поиска, так как остальные типы деревьев наследуют эти методы. Обход в глубину был выполнен с использованием рекурсии, начиная с корня и двигаясь внутрь дерева. Как показано на иллюстрации ниже, обход начинается с корня и сначала идет по левому поддереву до самого края. Затем процесс продолжается в правую часть, при этом уже посещенные элементы игнорируются.  
Изображение 7. Иллюстрация работы обхода в глубину.



Реализация обхода в ширину была выполнена аналогично, но для этого использовалась структура данных queue, которая позволяет добавлять элементы в конец. Алгоритм работает следующим образом: мы начинаем с корня, добавляем его в очередь, затем извлекаем первый элемент из очереди. Если у узла есть левый ребенок, он добавляется в очередь первым, затем, если есть правый ребенок, он добавляется после левого. Этот процесс продолжается для каждого узла до самого конца дерева.

Изображение 8. Иллюстрация работы обхода в ширину.



**Заключение**

В ходе выполнения лабораторной работы были изучены различные виды деревьев. Их преимущества и недостатки. Был реализован обход дерева вглубь и вширь, произведен анализ зависимости высоты дерева от количества ключей в нем. Были изучены основные операции дерева, такие как поиск, удаление, вставка и вывод. В результате работы была продемонстрирована эффективность каждого из деревьев для своей задачи.

**Ссылка на репозиторий GitHub**

https://github.com/K4taLizator/AiSD\_LR2.git