Отчет по курсовой работе по курсу “Алгоритмы и структуры данных (углубленный курс)”

Саитов Ирек, группа М4138

1. Постановка задачи

В данной курсовой работе необходимо было:

- Реализовать Красно-черное дерево, с поддержкой основных операций: поиска, нахождения минимума и максимума, вставки значения по ключу, также удаления ключа и связанного с ним значения.
- Провести тестирование корректности и провести анализ зависимости высоты получаемого дерева от количества значений.
- В качестве результата предоставить исходный код и краткий отчет с подробностями реализации, результатами тестов и графиками зависимости.

2. Подробности реализации

Красно-черное дерево было реализовано на языке программирования *Python*, в классе *RBTree*, а узел в классе *RBNode*, также был задействован вспомогательный класс *Color* для удобства кодирования дерева. Исходный код размещен на GitHub. Выполняется поддержка следующих операций:

- Поиск:
  - Метод *find(key)* производит поиск ключа в дереве, и если такой ключ найден, то возвращает узел, в котором хранится этот ключ.
- Вставка:
  - Метод *add_key(key)* вызывает инициализацию нового узла с ключом key и передаёт этот узел в метод вставки узла *add_node(node)*.
  - Метод *add_node(node)* осуществляет вставку нового узла в дерево.
  - Метод *_add_fix(node)* модифицирует структуру дерева путём поворотов _right_rotate(node) или _left_rotate(node) и окрашиванием узлов, чтобы все свойства красно-черного дерева выполнялись после вставки.
- Maximum:

- Метод *max_key(node)* находит узел с максимальным значением ключа в поддереве с корнем в узле node.

- Minimum:
  - Метод *min_key(node)* находит узел с минимальным значением ключа в поддереве с корнем в узле node.
- Удаление:
  - Метод *delete_key(key)* вызывает функцию поиска узла *find(key)* и передаёт найденный узел, как параметр для функции удаления узла *_delete_node*(*find(key)*)
  - Метод *_delete_fix(node)* восстанавливает структуру красно-черного дерева после удаления узла путём поворотов *_turn_left(node)* или *_turn_right(node)* и перекрашиванием узлов.
- Height:
  - Метод *tree_black_height()* возвращает число равное черное высоте дерева. Основано на таком свойстве красно-черного дерева, что черная высота этого дерева одинакова для любого простого пути от корня к листку.
  - Метод *tree_height(node, l_height, r_height)* находит максимальную высоту дерева. Эта функция основана на рекурсивном обходе дерева.

Вспомогательные функции и методы:

- Родственники:
  - Метод *_brother(node)* возвращает брата узла node
  - Метод *_uncle(node)* возвращает дядю узла node.
  - Метод *_grandfather(node)* возвращает дедушку узла node.

  Эти методы используются для удобства кодирования. В особенности позволяют избежать дублирования кода в функциях *_add_fix(node)* и *_delete_fix(node)*.
- Повороты:
  - Метод *_turn_left(node)* осуществляет поворот узла node налево, при этом происходит переподвешивание ребёнков узлов node и node --> rightChild.
  - Метод *_turn_right(node)* осуществляет поворот узла node направо, при этом происходит переподвешивание ребёнков узлов node и node --> leftChild.

  Эти методы используются для поддержания свойств красно-черного дерева.

3. Подробности тестирования

Работа с программой осуществляется через консоль. В качестве тестов корректности работы структуры данных были реализованы юнит-тесты для основных функций. Также функция вставки тестируется на рандомном множестве неповторяющихся ключей. А при тесте удаления после каждого удаленного узла в консоли выводится текущий список узлов в дереве. В конце тестов функций, которые меняют структуру дерева (т.е. вставка и удаление), результат записывается в файл f.txt и выполняется проверка корректности свойств измененного дерева. Результат тестов функций minimum, maximum, поиск можно посмотреть в консоли.

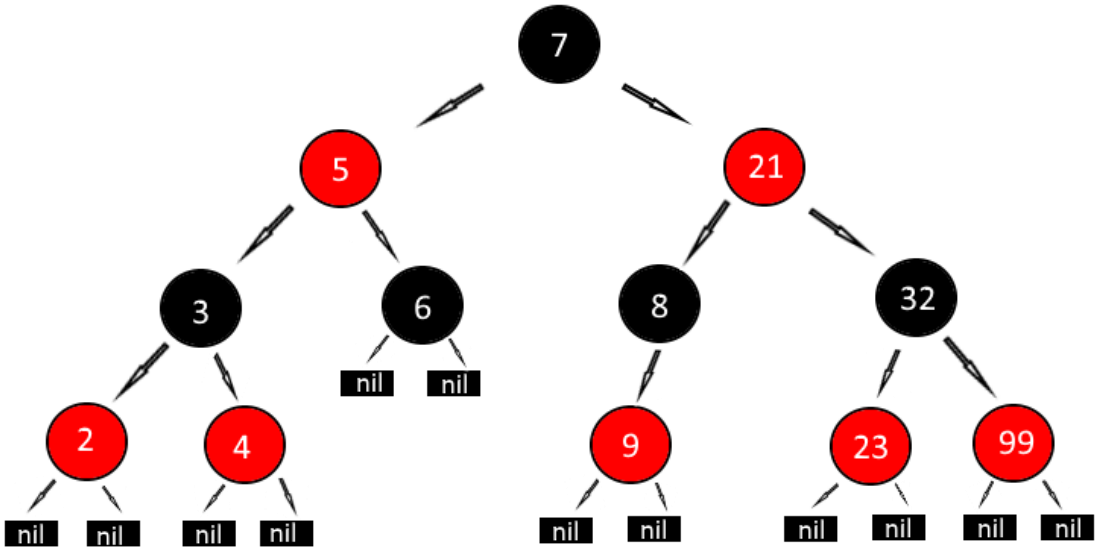


Рис. 1 Визуальное представление результата test_add

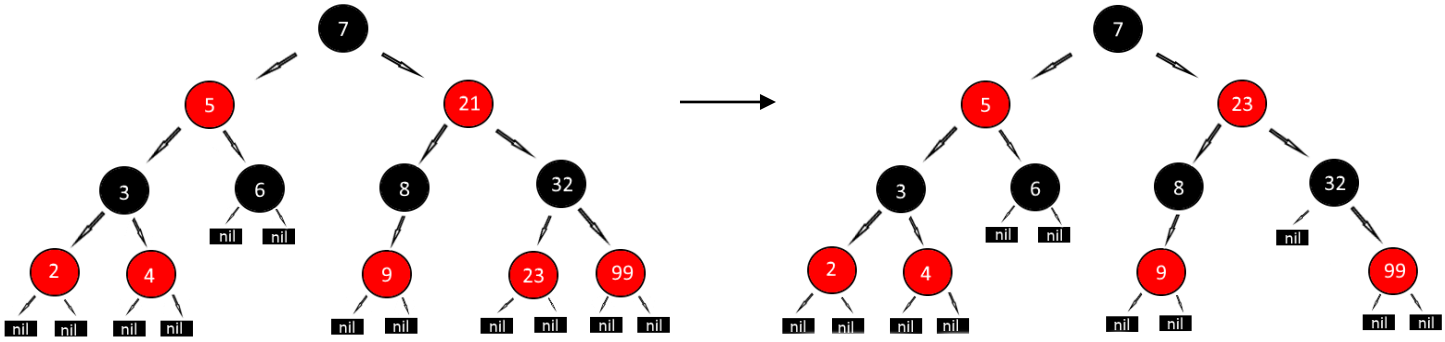


Рис. 2 Визуальное представление результата test_delete удаления узла 21

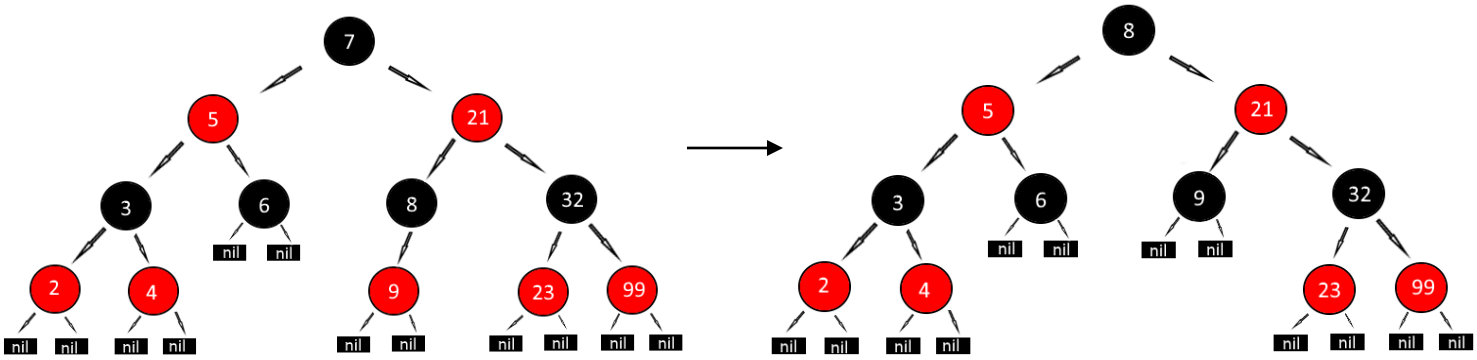


Рис. 3 Визуальное представление результата test_delete удаления узла 7

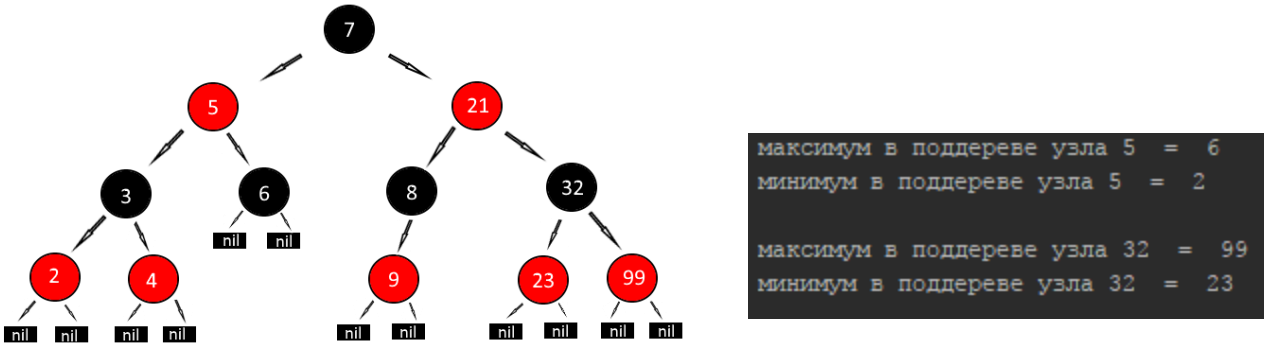


Рис. 4 Визуальное представление test_min_max для узлов 5 и 32

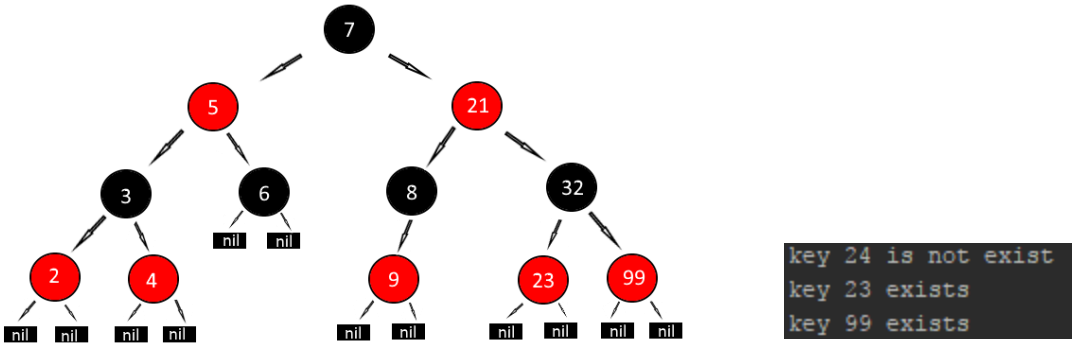
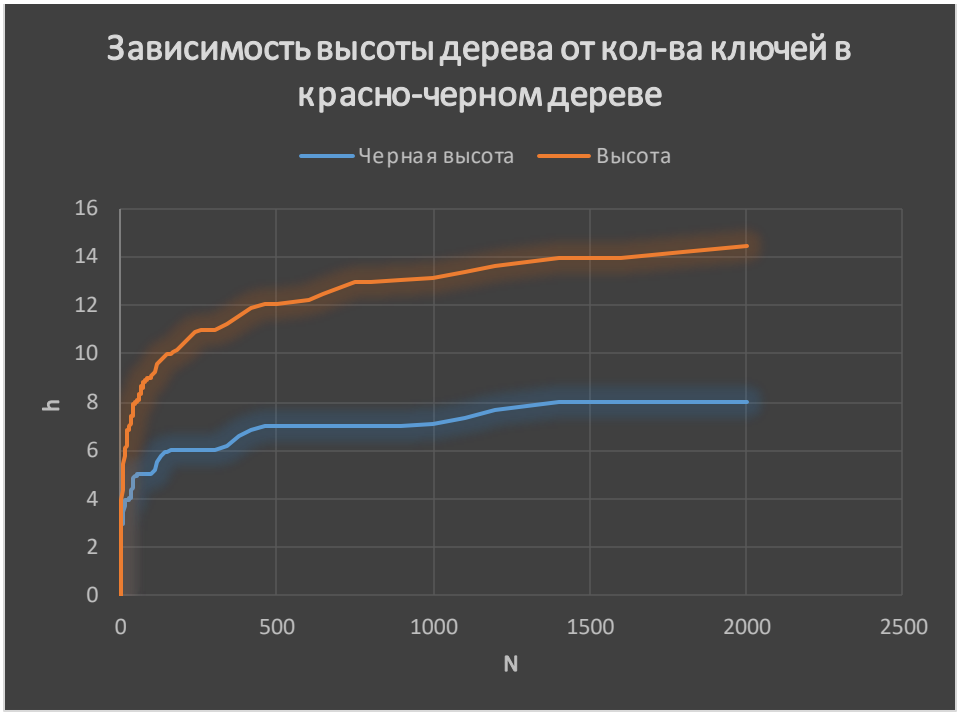


Рис. 5 Визуальное представление результата test_find для ключей 24, 23 и 99

Для исследования зависимости высоты дерева от количества узлов в дереве были проведены 1000 опытов для каждого конкретного значения ключа в диапазоне от N = {0, 2000} с разным набором допустимых ключей. Затем взяты среднее арифметическое высоты и построены следующий график зависимости:



Исходя из результатов можно сделать вывод, что наблюдается логарифмическая зависимость высоты от количества ключей. Причем увеличение высоты дерева происходит скачкообразно в местах, которые являются степенью 2. Например, около N = 512 происходит рост вероятности построения более высокого дерева, это же можно увидеть у 1024.

Также можно вывести верхнюю границу для высоты красно-черного дерева:

$$2^{\frac{h-1}{2}} \leq N$$

$$\frac{h-1}{2} \leq \log_2 N$$

$$h - 1 \leq 2\log_2 N$$

$$h \leq 2\log_2 N + 1$$

где h – высота дерева, а N – кол-во узлов или ключей.

Сделанное предположение о получении более укомплектованного дерева (высота наименьшая) при динамическом расширении (например, для 40 узлов берутся значения ключей от 0 до 40; для 80 узлов – от 0 до 80) ключей неверно. Отсюда можно сделать вывод, что последовательность добавления узлов влияет на получаемую высоту красно-черного дерева, так как при каждом добавлении новый узел подвешивается к родителю листка, а уже затем происходит балансировка.

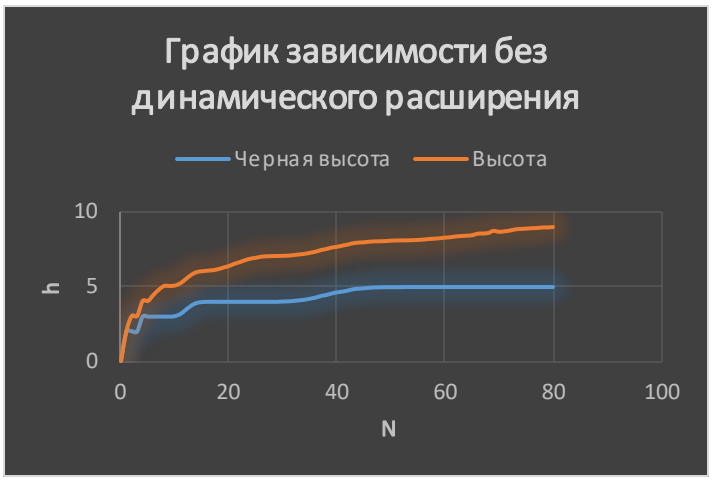


График зависимости без динамического расширения
Черная высота
Высота
10
5
0
h
0
20
40
60
80
100
N

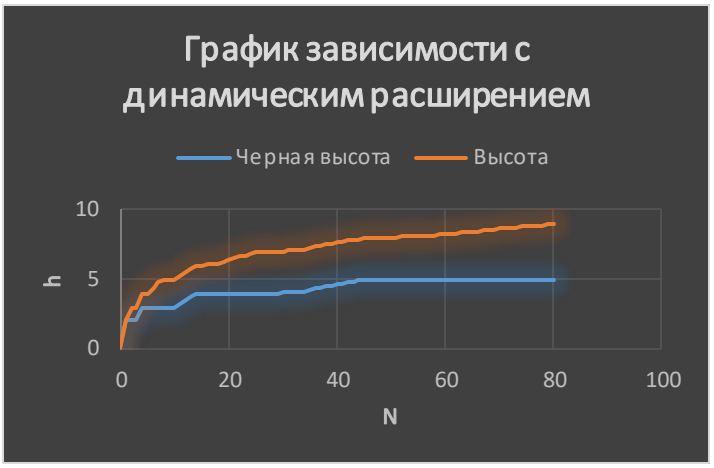


График зависимости с динамическим расширением
Черная высота
Высота
10
5
0
h
0
20
40
60
80
100
N