Лабораторная работа № 2 по курсу: Дискретный анализ

Выполнил студент группы М8О-307Б-17 МАИ *Лопатин Александр.*

Задача

- **Постановка задачи:**

  Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер, от 0 до $2^{64} - 1$. Разным словам может быть поставлен в соответствие один и тот же номер.

- **Вариант дерева:**

  Красно-черное дерево

- **Вариант ключа:**

  Регистронезависимая последовательность букв английского алфавита длиной не более 256 символов.

- **Вариант значения:**

  Числа от 0 до $2^{64} - 1$.

Информация

Красно-чёрным называется бинарное поисковое дерево, у которого каждому узлу сопоставлен дополнительный атрибут — цвет и для которого выполняются следующие свойства:

- Каждый узел промаркирован красным или чёрным цветом
- Корень и конечные узлы (листья) дерева — чёрные
- У красного узла родительский узел — чёрный
- Все простые пути из любого узла x до листьев содержат одинаковое количество чёрных узлов
- Чёрный узел может иметь чёрного родителя

Метод решения

Вставка

Чтобы вставить узел, мы сначала ищем в дереве место, куда его следует добавить. Новый узел всегда добавляется как лист, поэтому оба его потомка являются NIL-узлами и предполагаются черными. После вставки красим узел в красный цвет. После этого смотрим на предка и проверяем, не нарушается ли красно-черное свойство. Если необходимо, мы перекрашиваем узел и производим поворот, чтобы сбалансировать дерево.

Удаление

При удалении вершины могут возникнуть три случая в зависимости от количества её детей:

- Если у вершины нет детей, то изменяем указатель на неё у родителя на nil.
- Если у неё только один ребёнок, то делаем у родителя ссылку на него вместо этой вершины.
- Если же имеются оба ребёнка, то находим вершину со следующим значением ключа. У такой вершины нет левого ребёнка (так как такая вершина находится в правом поддереве исходной вершины и она самая левая в нем, иначе бы мы взяли ее левого ребенка. Иными словами сначала мы переходим в правое поддерево, а после спускаемся вниз в левое до тех пор, пока у вершины есть левый ребенок). Удаляем уже эту вершину описанным во втором пункте способом, скопировав её ключ в изначальную вершину.

Генератор тестов

Сначала создаются тесты для создания базы данных и выгрузки её на жесткий диск, потом создаются тесты на выгрузку этих баз данных и поиск в них элементов. tests.py - создание баз данных; SearchTests.py - поиск элементов в базах данных.

tests.py

```
import random

for i in range(1000, 11000, 1000):
    file = open('tests/test'+str(i)+'.txt','w')
    for k in range(1, i + 1):
        file.write('+ ')
        for j in range(1, random.randint(2,100)):
            file.write(str(chr(random.randint(97, 122))))
        file.write(' ' + str(random.randint(1,1000000)))
        file.write('\n')
    file.write("! Save tests/database" + str(i) + ".txt")
    file.close()
```

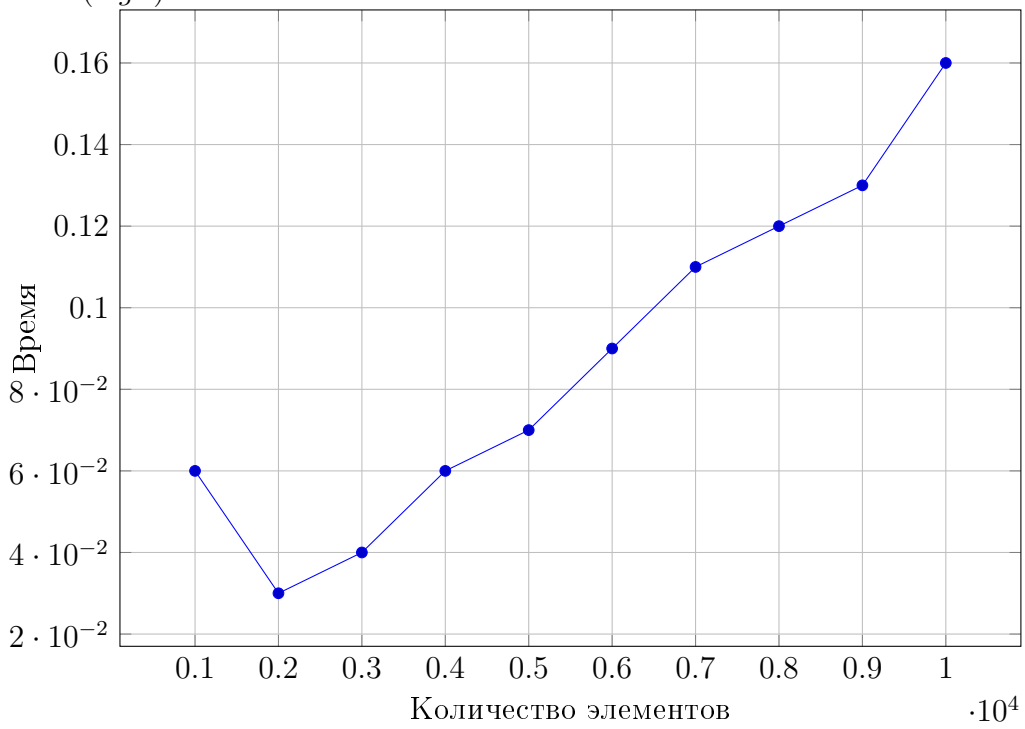
SearchTests.py

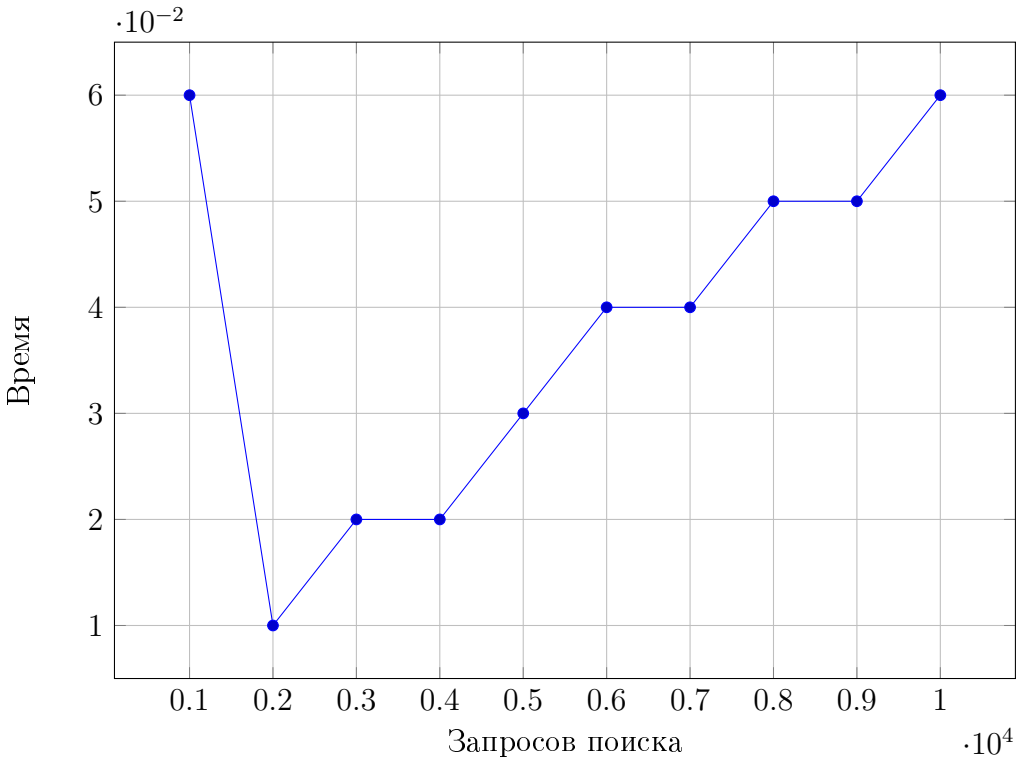
```
import random

for i in range(1000, 11000, 1000):
    file = open('tests/test'+str(i)+'.txt','w')
    file.write("! Load tests/database" + str(i) + ".txt")
    for k in range(1, i + 1):
        for j in range(1, random.randint(2,100)):
            file.write(str(chr(random.randint(97, 122))))
        file.write('\n')
    file.close()
```

Тест производительности

Сложность добавления, поиска и удаления элемента в красно-черном дереве составляет $O(logn)$.





Выводы

Хоть красно-черное дерево не является сбалансированным, оно гарантирует время удаления и вставки за O(log n) и, в отличе от АВЛ-дерева, требует лишь один дополнительный бит памяти на вершину (в АВЛ-дереве требуется два бита). Также можно заметить, что процедуру балансировки практически всегда можно выполнять параллельно с процедурами поиска, так как алгоритм поиска не зависит от атрибута цвета узлов.