

# Лабораторная работа № 2 по курсу дискретного анализа: сбалансированные деревья

Выполнил студент группы 08-208 МАИ *Ширяев Никита*.

## Условие

Реализовать декартово дерево с возможностью поиска, добавления и удаления элементов.

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.

Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь следующий формат:

- + word 34 — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.
- - word — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.
- word — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавле

## Метод решения

Декартово дерево (Treap) — это структура данных, объединяющая в себе бинарное дерево поиска и кучу. Более строго, это бинарное дерево, в узлах которого хранятся пары  $(x, y)$ , где  $x$  — это ключ, а  $y$  — это приоритет. Основными операциями декартова дерева являются merge и split. Операция split позволяет сделать следующее: разрезать исходное дерево  $T$  по ключу  $k$ . Возвращать она будет такую пару деревьев  $\langle T_1, T_2 \rangle$ , что в дереве  $T_1$  ключи меньше  $k$ , а в дереве  $T_2$  все остальные:  $\text{split}(T, k) \rightarrow \langle T_1, T_2 \rangle$ . Рассмотрим вторую операцию с декартовыми деревьями — merge. С помощью этой операции можно слить два декартовых дерева в одно. Причём, все ключи в первом(левом) дереве должны быть меньше, чем ключи во втором(правом). В результате получается дерево, в котором есть все ключи из первого и второго деревьев:  $\text{merge}(T_1, T_2) \rightarrow T$ . Используя эти 2 функции, мы можем реализовать основные операции с деревом:

- Операция поиска. Используем операцию *split* два раза: сначала по нашему ключу *x*, а потом правое дерево по ключу *x+1*. Так мы получим три дерева, в первом все элементы строго меньше *x*, в третьем строго больше *x*, а второе дерево может быть или пустым, или содержать единственный элемент *x*. Для поиска можно просто проверить, что второе дерево не пустое, и вывести его значение, в противном случае будем выводить "NoSuchWord". После этого применяем операцию *merge* два раза, чтобы вернуться к исходному дереву. Все остальные операции построены аналогично.
- Операция вставки. Проверяем, что второе дерево пустое, значит такого элемента пока не существует, а значит, мы можем создать в этом дереве ноду с полученным от пользователя ключом и значением. В противном случае выводим "Exist".
- Операция удаления. Проверяем, что второе дерево не пустое, значит, мы можем удалить этот элемент, сначала удаляем ноду, а потом указателю на ноду присваиваем значение *null*. Если дерево оказалось пустым, то выводим "NoSuchWord".

## Описание программы

Для хранения элемента была создана структура, состоящая из двух полей: ключа и значения.

```
char *key;
int priority;
uint64_t value;
node *left, *right
```

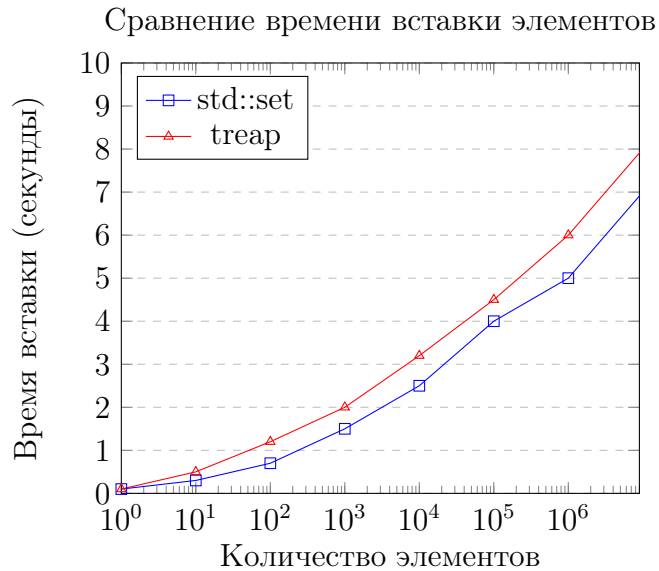
Также были реализованы функции вставки и удаления, использующие функции *merge* и *split*. При добавлении узла ключу ставится

## Дневник отладки

**Ошибка:** Ключи, отличающиеся только регистром, считались разными, что противоречило условию задачи.

**Способ устранения:** Написать функцию *toLowerCase*, которая будет переводить ключ в нижний регистр перед сравнением. Таким образом, все ключи будут сравниваться без учета регистра.

## Тест производительности



Оценка сложности:  $O(\log n)$  где,  $n$  - количество элементов в дереве.

## Выводы

В данной лабораторной работе было предложено изучить некоторые виды алгоритмов сбалансированных деревьев. В ходе выполнения было реализовано декартово дерево. Операции вставки, поиска и удаления выполняются за временную сложность  $O(\log n)$ , где  $n$  — количество элементов. Также были изучены дополнительные операции merge и split, которые помогают реализовать операции поиска, вставки и удаления.