Лабораторная работа № 2 по курсу дискретного анализа: сбалансированные деревья

Выполнил студент группы 08-207 МАИ *Боев Савелий*.

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 264 - 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 добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord».

Команд **! Save** и **! Load** в тестах нет.

Метод решения

Декартово дерево, также известное как **Treap**, - это структура данных, комбинирующая свойства двоичного дерева поиска и кучи (приоритетной очереди). В декартовом дереве каждый узел содержит ключ и приоритет, и они упорядочены по обоим этим значениям. При этом ключи узлов удовлетворяют условию двоичного дерева поиска, а приоритеты узлов образуют максимальную кучу.

Основные операции, которые необходимо реализовать в декартовом дереве, включают:

- **Поиск**: 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.
- **Вставка**: создайте новый узел с заданным ключом и приоритетом. С помощью операции разделения (**split**) разделите текущее дерево на два поддерева по ключу вставляемого узла. Затем объедините эти поддеревья и новый узел с помощью операции слияния (**merge**). Иначе вывод “**Exit**”
- **Удаление**: используя операцию разделения (**split**), разделите текущее дерево на два поддерева. Затем, используя операцию разделения еще раз, разделите правое поддерево на два поддерева по ключу удаляемого узла. В результате получатся три поддерева: левое поддерево, поддерево между ключами левого и правого поддеревьев, и правое поддерево. Удалите узел, соответствующий ключу, и объедините левое и

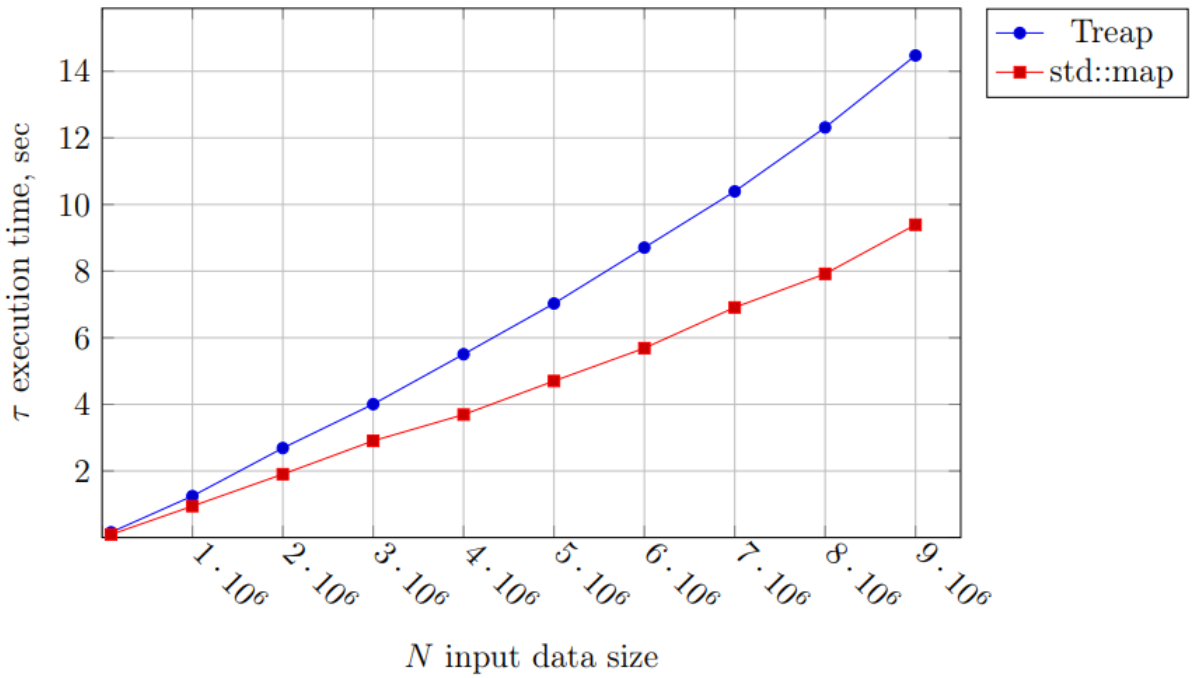
правое поддеревья с помощью операции слияния. Если дерево пустое, то "**NoSuchWord**".

Реализована структура декартового дерева (**bst**) и основные операции: **merge**, **split**, поиск (**search**), вставка (**insert**) и удаление (**remove**). Код также содержит функцию **main**, которая читает вводные данные и вызывает соответствующие операции дерева в зависимости от команд.

Описание программы

В данной программе содержится один файл main.cpp, в котором реализованы операции поиска, вставки и удаления.

Тест производительности



Оценка сложности операций вставки, поиска и удаления: O(h), где h — высота дерева, так как дерево является сбалансированным, то сложность операций можно представить как O(logn), где n — количество элементов. Значит, сложность всей программы оценивается как O(n·logn). Для сравнения используется стандартная библиотека std::map, которая реализована на основе красно-черного дерева.

Выводы

В данной лабораторной работе было предложено изучить некоторые виды алгоритмов сбалансированных деревьев. Мной был реализован алгоритм декартово дерево. Операции вставки, поиска и удаления выполняются за временную сложность O(logn), где n — количество элементов. Также мной были изучены дополнительные операции **merge** и **split**, которые помогают реализовать операции поиска, вставки и удаления. Я считаю, что эта лабораторная работа оказалась достаточно полезной, ведь сбалансированные деревья применяется, когда необходимо осуществлять быстрый поиск элементов, чередующихся со вставками новых элементов и удалениями существующих.