Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

Алгоритмы работы со словарями

по дисциплине «**Алгоритмы и структуры данных**»

Выполнил студент гр. 5130904/30003

Новиков Вадим Дмитриевич

Руководитель

Шемякин Илья Александрович

«___» _____ 2024 г.

Санкт-Петербург

2024 г.

1. Введение. Общая постановка задачи

Тема: Алгоритмы работы со словарями

1. Для разрабатываемого словаря реализовать основные операции:
   - `INSERT` (ключ, значение) — добавить запись с указанным ключом и значением.
   - `SEARCH` (ключ) — найти все записи с указанным ключом.
   - `DELETE` (ключ, значение) — удалить запись с указанным ключом и значением.
2. Предусмотреть обработку и инициализацию исключительных ситуаций, связанных, например, с проверкой значения полей перед инициализацией и присваиванием.
3. Программа должна быть написана в соответствии со стилем программирования: C++ Programming Style Guidelines (http://geosoft.no/development/cppstyle.html).
4. Тесты должны учитывать как допустимые, так и не допустимые последовательности входных данных.

Вариант 1.1.2.
Англо-русский словарь. АВЛ-дерево

Разработать и реализовать алгоритм работы с англо-русским словарем, реализованным как АВЛ-дерево.

Узел АВЛ-дерева должен содержать:

- Ключ — английское слово
- Показатель (фактор) сбалансированности
- Информационная часть — ссылка на список, содержащий переводы английского слова, отсортированные по алфавиту (переводов слова может быть несколько).

2. Основная часть работы

2.1. Описание алгоритма решения и используемых структур данных.

АВЛ-дерево — один из первых видов сбалансированных двоичных деревьев поиска, изобретённый в 1962 году советскими учёными Адельсон-Вельским и Ландисом [1]. Аббревиатура АВЛ образована первыми буквами фамилий его создателей. Особенность АВЛ-дерева заключается в том, что для любого его узла высота правого поддерева отличается от высоты левого поддерева не более чем на единицу. Это обеспечивается следующим образом:

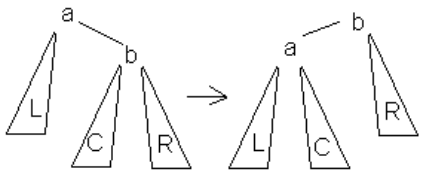
1. Особая структура узлов. Помимо ключа и указателей на узлы родителя, левого и правого ребёнка они содержат показатель (фактор) сбалансированности — разность высот правого и левого поддеревьев. Он может принимать 3 значения: -1, 0 или 1 в зависимости от того, какое из поддеревьев имеет большую высоту. Если фактор сбалансированности имеет отличное от представленных значение (-2 или 2), дерево является несбалансированным и ему требуется перебалансировка.
2. Адаптация алгоритмов вставки и удаления элементов. Ситуация несбалансированности чаще всего возникает при редактировании дерева, то есть при вставке или удалении элементов, поэтому соответствующие алгоритмы включают в себя механизм перебалансировки, основанный на правом и левом поворотах. Более подробно эти алгоритмы описаны в главе 2.2.

2.2. Анализ алгоритма

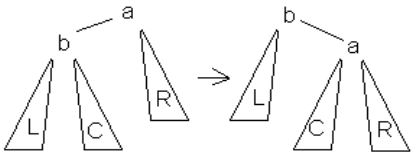
2.2.1. Алгоритмы балансировки узлов

Как уже было сказано ранее, при вставке и удалении элементов в АВЛ-дерево возможно возникновение ситуации, когда фактор сбалансированности у некоторых узлов оказывается равен -2 или 2, то есть происходит разбалансировка поддерева. Для исправления этой ситуации применяются алгоритмы поворота: малый левый, малый правый, большой левый и большой правый [2]. Обозначим фактор балансировки bf.

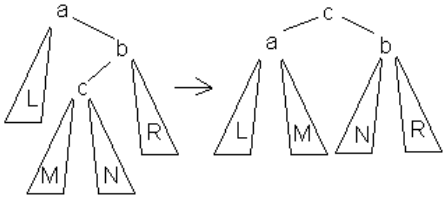
1. Малый левый поворот узла a вокруг его правого ребёнка b используется в случае, когда $bf(a)=2$ и $bf(b)\geq 0$. При этом узел b становится родителем узла a, а узел a становится левым ребёнком узла b.



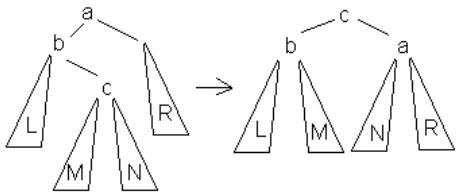
2. Малый правый поворот является симметричной копией левого и используется в случае, когда $bf(a)=-2$ и $bf(b)\leq 0$.



3. Большой левый поворот сводится к двум простым поворотам — сначала правый поворот узла b, затем левый поворот узла a. Применяется, когда $bf(a)=2$ и $bf(b)<0$.



4. Большой правый поворот также представляет собой комбинацию двух простых поворотов — сначала правый поворот вокруг узла *b*, затем левый поворот вокруг узла. Применяется, когда $bf(a)=-2$ и $bf(b)>0$.



Анализ возможных ситуаций показывает, что выполнить перебалансировку узла *a* можно следующим образом:

Если $bf(a)=2$, то проверяем фактор сбалансированности правого ребёнка. Если он больше или равен 0, то выполняем левый поворот вокруг узла *a*, иначе выполняем большой левый поворот вокруг того же узла.

Если $bf(a)=-2$, то проверяем фактор сбалансированности левого ребёнка. Если он меньше или равен 0, то выполняем правый поворот вокруг узла *a*, иначе выполняем большой правый поворот вокруг того же узла.

Если $-1\le bf(a)\le 1$, ничего не делаем.

2.2.2. Алгоритм вставки

Алгоритм вставки в АВЛ-дерево работает по тому же принципу, что алгоритм вставки в несбалансированное бинарное дерево поиска. Единственное отличие заключается в том, что после вставки необходимо выполнить перебалансировку корневого узла.

Начинаем поиск места для вставки с корневого узла. Если вставляемый элемент меньше элемента в текущем узле, то переходим к его левому ребёнку, а если больше, переходим к правому. Повторяем до тех пор, пока не достигнем элемента, равного вставляемому или являющегося листом. В первом случае завершаем работу функции и сообщаем, что вставка не удалась, во втором случае выделяем память под новый элемент, связываем его с найденным листом и выполняем перебалансировку поддерева. Для этого поднимаемся вверх к корню и пересчитываем факторы балансировки у узлов. Если мы поднялись в узел *i* из левого поддерева, то уменьшаем его фактор балансировки на 1, если из правого — увеличиваем на 1 [3]. Если баланс вершины стал равен 2 или -2, выполняем один из четырёх поворотов и, если после этого баланс вершины стал равен 0, то останавливаемся, иначе продолжаем подъём.

2.2.3. Алгоритм поиска

Алгоритм поиска в АВЛ-дереве полностью идентичен алгоритму поиска в несбалансированном бинарном дереве поиска.

Начинаем поиск нужного элемента с корневого узла. Если вставляемый элемент меньше элемента в текущем узле, то переходим к его левому ребёнку, а если больше, переходим к правому. Повторяем до тех пор, пока не достигнем элемента, являющегося листом или равного искомому. В первом случае завершаем работу функции и сообщаем, что элемента нет в дереве, во втором случае предоставляем доступ к этому элементу.

2.2.4. Алгоритм удаления

Алгоритм удаления из АВЛ-дерева работает по тому же принципу, что алгоритм удаления из несбалансированного бинарного дерева поиска. Единственное отличие заключается в том, что после удаления необходимо выполнить перебалансировку.

Начинаем поиск удаляемого элемента с корневого узла. Если удаляемый элемент меньше элемента в текущем узле, то переходим к его левому ребёнку, а если больше, переходим к правому. Повторяем до тех пор, пока не достигнем элемента, являющегося «ребёнком» листа или равного удаляемому. В первом случае завершаем работу функции и сообщаем, что удаление не удалось. Во втором случае:

1. Если найденный элемент является листом, отвязываем от него родителя, если он есть.
2. Если найденный элемент имеет одного ребёнка, ставим этого ребёнка на его место (привязываем родителя элемента к ребёнку, а ребёнка к родителю).
3. Если найденный элемент имеет двух детей, заменяем его на наименьший элемент из его правого поддерева (или наибольший из левого поддерева).

Далее удаляем найденный элемент из памяти и выполняем перебалансировку. От удалённого узла поднимаемся вверх к корню и пересчитываем факторы балансировки. Если мы поднялись в узел i из левого поддерева, то увеличиваем его фактор балансировки на 1, если из правого — уменьшаем на 1. Если после этого баланс вершины стал равен 1 или -1, то подъём можно остановить, то как высота этого поддерева не изменилась. Если баланс стал равен 0, то высота поддерева уменьшилась, и подъём нужно продолжить Если баланс стал равен 2 или -2, выполняем одно из четырёх вращений и, если после этого баланс вершины стал равен нулю, продолжаем подъём, иначе останавливаемся.

2.3. Описание спецификации программы (детальные требования)

Программа, реализующая англо-русский словарь на основе АВЛ-дерева, написана на языке C++ с использованием стандарта C++14.

На основе АВЛ-дерева реализованы множество (`AvlTreeSet`) и словарь (`AvlTreeMap`). Множество хранит упорядоченные уникальные ключи, словарь хранит упорядоченные уникальные ключи, каждому из которых соответствует единственное значение. Созданы отдельные классы для множества, словаря, их узлов и стратегий обхода (итераторов). Итераторы являются двунаправленными, то есть имеется возможность обходить АВЛ-дерево как прямым, так и в обратным инфиксным обходом.

Сам англо-русский словарь реализован как словарь множеств, где ключом является строка, а значением — упорядоченное множество слов. Тип данных — `AvlTreeMap<std::string, AvlTreeSet<std::string>>`. Для данного типа реализован класс-обёртка `EngRusDictionary`, в котором определены методы для более удобного взаимодействия с такой композицией типов извне.

Реализовано 3 команды: `INSERT`, `SEARCH` и `DELETE`.

- `INSERT` (ключ, значение) — добавить запись с указанным ключом и значением.
- `SEARCH` (ключ) — найти все записи с указанным ключом.
- `DELETE` (ключ, значение) — удалить запись с указанным ключом и значением.

2.4.