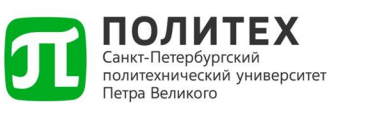
**Санкт-Петербургский политехнический университет Петра Великого  
  
Институт компьютерных наук и технологий  
Высшая школа программной инженерии**

****

**К У Р С О В А Я Р А Б О Т А**

**Алгоритмы работы со словарями**по дисциплине **«Алгоритмы и структуры данных»**

|  |  |
| --- | --- |
| Выполнил студент гр. | 3530904/10003  Майстренко Дмитрий |
| Руководитель | Шемякин Илья Александрович |
|  | **«\_\_» \_\_\_\_\_\_\_\_\_**2022 |

# **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 запись с указанным ключом
   * DELETE (ключ) – удалить запись с указанным ключом
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>).

1. Тесты должны учитывать как допустимые, так и недопустимые последовательности входных данных.

**Вариант 1.1.4.**

**Англо-русский словарь. Красно-чёрное дерево.**Разработать и реализовать алгоритм работы с англо-русским словарём, реализованным как красно-чёрное дерево.

Узел дерева должен содержать:

* Ключ – английское слово
* Информационная часть – ссылка на список, содержащий переводы английского слова, отсортированные по алфавиту (переводов слова может быть несколько).

# **Основная часть работы.**

## **Описание алгоритма решения и используемых структур данных**

Англо-русский словарь – класс, основная задача которого добавлять, удалять и хранить данные. Одной единицей данных является английское слово и соответствующий ему список русских слов, являющихся смысловыми эквивалентами.

### **Красно-чёрное дерево**

#### Общие сведения

Для хранения данных внутри класса была выбрана и самостоятельно реализована структура данных красно-чёрное дерево, которое является одним из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и логарифмическое время поиска, вставки и удаления элементов.

Красное-чёрное дерево имеет следующие свойства:

* 1. Узел может быть либо красным, либо чёрным (так как данное свойство имеет только два состояния, оно организовано переменной логического типа) и имеет двух потомков (однако потомок может быть не только узлом, но и листом – то есть не содержать данных никаких данных, кроме цвета);
  2. Корень – как правило чёрный;
  3. Все листья – чёрные;
  4. Оба потомка каждого красного узла – чёрные;
  5. Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число чёрных узлов.

Данные ограничения гарантируют путь от корня до самого дальнего листа не более, чем вдвое длиннее, чем до самого длинного и дерево примерно – не строго – сбалансированно.

Балансировка происходит после каждых вставки и удаления, причём косвенно рекурсивным методом – практически каждый пункт подразумевает при несоблюдении условия в нём переход к следующему.

Все рекурсивные вызовы функций хвостовые и преобразуются в циклы, так что алгоритм требует памяти О(1).

В дереве были также организованы некоторые алгоритмы, которые не участвуют в командах словаря. Это было реализовано для общности структуры данных. Также, было принято решение использовать шаблон для универсальности и переносимости.

#### Вставка

Вставка в красно-чёрное дерево ничем не отличается от вставки в обычное бинарное дерево поиска за исключением того, что в красно-чёрном дереве новые данные помещаются в лист, после чего он становится узлом и приобретает свои листы.

Однако после вставки происходит самобалансировка дерева, опирающаяся на свойства красно-чёрного дерева.

Определим,

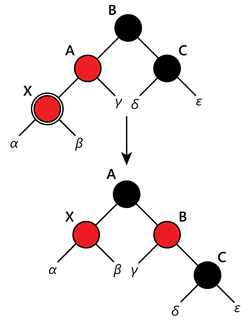
* Дедушка – родительский элемент родительского элемента данного узла;
* Дядя – дочерний элемент дедушки данного узла, не являющийся родительским;

Рис. 1

* Брат – второй потомок родителя данного узла;
* Поворот – изменение связей друг относительно друга четырёх элементов, а также смена цветов (Рис.1 – поворот вправо).

Балансировка дерева осуществляется, как уже было упомянуто, косвенной рекурсией. Выбор осуществлён в связи с тем фактом, что балансировка дочерних ряда узлов может нарушить балансировку родительских. В связи с чем требуется при балансировке идти снизу (от вставленного) вверх (к узлу), по пути рассматривая 5 различных ситуаций (текущий узел – узел, к которому применяется балансировка):

1. Текущий узел в корне дерева – перекрасить его в чёрный цвет;
2. Предок текущего узла чёрный – всё корректно (данный пункт нужен для исключения его из последующих ситуаций);
3. Родитель и дядя красные – перекрасить их в чёрный, а дедушку – в красный и вызов и перейти в ситуацию 1;
4. Родитель является красным, а дядя чёрным – выполнить поворот на родителе в сторону, зависящую от того, каким потомком (левым или правым) является текущий узел и его родитель (причём текущий и родитель должны быть разной стороны потомками);
5. Родитель является красным, а дядя чёрным – выполнить поворот на дедушке в сторону, зависящую от того, каким потомком (левым или правым) является текущий узел и его родитель (причём текущий и родитель должны быть одной стороны потомками);

Стоит отметить, что каждая ситуация (за исключением пятой), при невыполнении условия переходит в следующих пункт.

#### Удаление

Удаление происходит следующим образом: удаляемый элемент меняется местами с самым меньшим элементом правого поддерева или с самым большим левого. Затем, перемещённый удаляемый элемент удаляется и дерево балансируется. Такое перемещение упрощает балансировку – исключается ситуация, когда у текущего могут быть два потомка – и не нарушает свойство бинарного дерева поиска.

Далее, для балансировки рассматриваются 6 следующий ситуаций:

1. Текущий является корнем – ничего делать не требуется (данный пункт создан во избежание действий над корнем);
2. Брат является красным – меняем местами цвета родителя и брата и делаем поворот влево на родителе;
3. Родитель, брат и все потомки брата являются чёрными – перекрашиваем брата в красный;
4. Родитель - красный, а брат и все его потомки являются чёрными – меняем цвета брата и родителя;
5. Брат является чёрным, левый его потомок - красный, а правый - чёрный – делаем поворот на брате, в зависимости от того, каким потомком является текущий элемент;
6. Брат - чёрный, правый его потомок является красным – делаем поворот на родителе, в зависимости от того, каким потомком является текущий.

### **Коллекция переводов**

В качестве коллекции переводов было выбрано упорядоченное множество с уникальными элементами. Вставка гарантированно происходит за время О(n) и даёт нам возможность выполнять необходимое слияние словарей с аналогичной скоростью.

## **Анализ алгоритма**

В данной работе используются 2 структуры данных: красно-чёрное дерево и упорядоченное множество.

* Упорядоченное множество имеет скорость О(n) (вставка, доступ, удаление);
* Красно-чёрное дерево имеет скорость O(logn) (вставка, доступ, удаление);

На графике 1 представлено последовательное добавление элементов по возрастанию (от 0 до 999999) и график зависимости количества необходимых операций сравнения и присваивания от количества элементов в дереве. Из него явно видно, что скорость вставки растёт логарифмически, как и предсказывала теория.

График 1

На графике 2 представлено последовательное удаление элементов по возрастанию (от 0 до 999999) и график зависимости количества необходимых операций сравнения и присваивания от количества элементов в дереве. На нём мы не видим явного очертания логарифмической функции, в связи с тем, что удалялись элементы не по порядку количества необходимых операций. Однако при проведении линии, соединяющей «верхние» точки, мы снова получим логарифмический график, что также подтверждает теорию.

## **Описание спецификации программы (детальные требования)**

График 2