Федеральное агентство по науке и образованию РФ

Санкт-Петербургский государственный электротехнический университет “ЛЭТИ”

Кафедра МО ЭВМ

Отчет

По учебному проекту

«Визуализация красно-черного дерева»

**Выполнили:**

Комаров Алексей Юрьевич

Попов Георгий Сергеевич

Плосков Александр Андреевич

**Группа:** 4304

**Факультет:** КТИ

**Преподаватель:** Шолохова О.М.

Санкт-Петербург

2016

Оглавление

[1. Постановка задачи 3](#_Toc455490011)

[2. Описание структуры данных 3](#_Toc455490012)

[a. Свойства красно-черных деревьев 3](#_Toc455490013)

[b. Вращения 3](#_Toc455490014)

[c. Поиск вершины 3](#_Toc455490015)

[d. Добавление вершины 4](#_Toc455490016)

[e. Удаление вершины 4](#_Toc455490017)

[3. Описание пользовательского интерфейса 5](#_Toc455490018)

[a. Вид интерфейса программы 5](#_Toc455490019)

[b. Описание элементов интерфейса 5](#_Toc455490020)

[4. Описание способов визуализации основных алгоритмов 6](#_Toc455490021)

[a. Описание визуализации алгоритма поиска 6](#_Toc455490022)

[b. Визуализация алгоритма поиска 6](#_Toc455490023)

[c. Описание визуализации алгоритма вставки 8](#_Toc455490024)

[5. Основные этапы разработки 9](#_Toc455490025)

[6. Тестирование приложения 10](#_Toc455490026)

# Постановка задачи

Создание приложения, позволяющего создавать красно-черные деревья любой сложности, редактировать их, а также визуализировать на их основе алгоритмы вставки / поиска / удаления элементов такого дерева. Приложение должно обладать понятным, лаконичным, но эффективным пользовательским интерфейсом.

# Описание структуры данных

## Свойства красно-черных деревьев

Красно-черное дерево (*red-black tree*) — это двоичное дерево поиска, вершины которого разделены на красные (*red*) и черные (*black*). Таким образом каждая вершина хранит один дополнительный бит — её цвет.

При этом выполняются определённые требования, которые гарантируют, что глубины любых двух листьев отличаются не более чем в два раза.

Каждая вершина красно-черного дерева имеет поля *color* (цвет), *key* (ключ), *left* (левый ребенок), *right* (правый ребенок) и *p* (родитель). Если у вершины отсутствует ребенок или родитель, то соответствующее поле содержит *NIL*.

Двоичное дерево поиска называется красно-черным, если оно обладает следующими свойствами:

* Каждая вершина — либо черная, либо красная.
* Каждый лист (NIL) — чёрный.
* Если вершина красная, оба её ребенка чёрные.
* Все пути, идущие вниз от корня к листьям, содержат одинаковое количество чёрных вершин.

Рассмотрим произвольную вершину *x* красно-черного дерева и пути, ведущие вниз от неё к листьям. Все они содержат одно и то же число чёрных вершин. Число чёрных вершин в любом из них (саму вершину *x* не считаем) будем называть чёрной высотой (black-height) вершины *x*. Чёрной высотой дерева будем считать чёрную высоту его корня.

## Вращения

Вращения представляют собой локальную операцию (меняются несколько указателей) и сохраняет свойство упорядоченности.

ЛЕВОЕ ВРАЩЕНИЕ возможно в любой вершине x, правый ребёнок которой (назовем его y) не является листом (NIL). После вращения: y оказывается сверху, x становится левым ребёнком y, а бывший левый ребёнок y — правым ребенком x.

ПРАВОЕ ВРАЩЕНИЕ возможно в любой вершине x, левый ребёнок которой (назовем его y) не является листом (NIL). После вращения: y оказывается сверху, x становится правым ребёнком y, а бывший правый ребёнок y — левым ребенком x.

## Поиск вершины

Процедура поиска выполняется следующим образом: в текущей вершине сравниваем ключи искомой вершины с ключом этой вершины. Если ключи совпадают — значит нашли вершину, иначе если ключ в этой вершине больше искомого, то идем к левому ребенку, иначе — к правому. Процедура заканчивается при нахождении вершины или если мы находимся в листе (NIL).

## Добавление вершины

Добавление вершины в красно-черное дерево производится за время O(log n). Сначала спустимся вниз по дереву так, как это происходит при поиске, пока мы не окажемся в вершине, не имеющий сына. Добавим вершину x с той стороны, куда мы должны были бы двинуться, если бы вершина имела детей. Красим новую вершину в красный цвет. После этого надо восстановить свойства красно-черного дерева, для чего приходится перекрасить некоторые вершины и произвести вращения.

После добавления вершины выполняются все свойства красно-черного дерева, кроме одного: красная вершина x может иметь красного родителя, и это нарушение единственно. Восстанавливается это свойство очень просто: пока вершина x не корень дерева и имеет красного родителя выполняем следующие действия:

* Если дядя вершины x красного цвета, то красим его и отца вершины x в черный цвет, а дедушку вершины x в красный цвет. Далее вершиной x становится дедушка.
* Если дядя вершины x черного цвета, то проверяем являются ли вершина x и его отец одинаковыми детьми (т.е. оба ЛЕВЫЕ или ПРАВЫЕ дети). Если не являются одинаковыми детьми, то делаем соответствующие вращение (становятся одинаковыми детьми). Красим отца в черный цвет. Далее делаем соответствующее вращение;

Будем полагать, что во всех красно-черных деревьях корень черный, и поддерживать это свойство.

## Удаление вершины

Как и другие операции, удаление вершины из красно-чёрного дерева требует времени . Удаление вершины несколько сложнее добавления.

Для начала находим вершину z в дереве, которую требуется удалить. Возможны три случая:

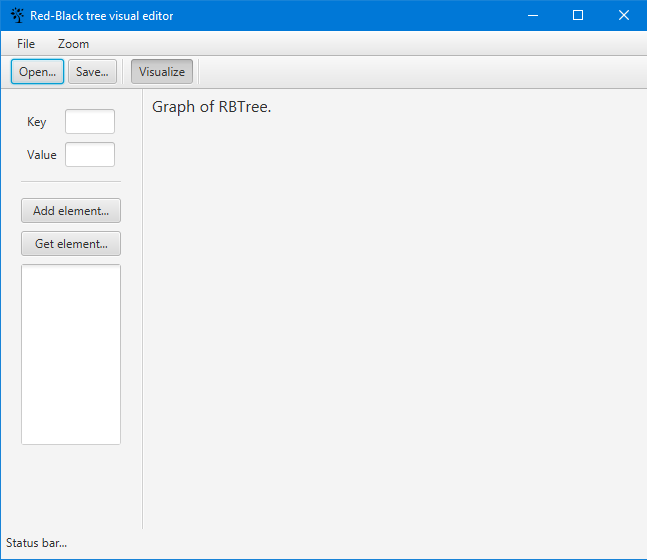
* Если у вершины z нет детей, для удаления z достаточно поместить NIL в соответствующее поле его родителя (вместо z).
* Если у z один ребенок, то можно вырезать z, соединив его родителя напрямую с его ребенком.
* Если у z двое детей, то мы находим следующий (в смысле порядка на ключах) за z элемент y; у него нет левого ребенка. Теперь можно скопировать ключ и дополнительные данные из вершины y в вершину z, а саму вершину y удалить выше описанным способом.

Если была удалена черная вершина, то надо восстановить свойства красно-черного дерева (при удалении красной вершины свойства дерева не нарушаются). Пусть вершина x — ребенок удаленной вершины. Пока вершина x не корень дерева, и она чёрная вершина выполняем следующие действия:

* Если у вершины x брат красного цвета, то делаем ВРАЩЕНИЕ (брат становится родителем отца), при этом брата красим в черный, а отца в красный.
* Если у вершины x брат красного цвета, то делаем ВРАЩЕНИЕ (брат становится родителем отца), при этом брата красим в черный, а отца в красный.
  + Если у брата оба ребенка черные, то красим брата в красный цвет. Теперь вершиной x будет его отец.
  + Если у брата его "одинаковый" ребёнок черного цвета, то красим брата в красный цвет и делаем вращение.
  + Иначе красим брата в цвет отца, отца красим в черный цвет. Делаем вращение и выходим из цикла.

# Описание пользовательского интерфейса

## Вид интерфейса программы



## Описание элементов интерфейса

Меню

* + File
    1. Exit – Закрытие приложения.
  + Zoom
    1. Zoom Reset – Вернуть стандартный масштаб дерева.
    2. Zoom In – Увеличение масштаба дерева.
    3. Zoom Out – Уменьшение масштаба дерева.

Панель инструментов

* + Open… – Открытие файла. Получение и обработка данных из файла с расширением “*.graph”*.
  + Save… – Сохранение данных о текущем состоянии дерева в файл с расширением “*.graph”*.
  + Visualize – Переключатель для выбора визуализации алгоритмов. Если выбран, то включен режим визуализации алгоритмов. Иначе будет выводится конечный результат алгоритмов.
* Левая панель
  + Key [*line\_key*] – Ввод ключа *line\_key* для вставки в красно-черном дереве. Поиск вершины в дереве ведется по ключу.
  + Value [*line\_value*] – Ввод данных *line\_value*, что будут храниться по ключу *line\_key*.
  + Add element... – Добавление элемента в красно-черное дерево с ключом *line\_key* и данными *line\_value*. Если *line\_key* = <пусто> или *line\_value =* <пусто>, то элемент не будет добавлен. Кнопка управления.
  + Get element... – Получение элемента в красно-черном дереве по ключу *line\_key*. Если *line\_key* = <пусто>, то поиск не будет произведен. Кнопка управления.
  + << – Шаг назад. Кнопка навигации. Доступна в режиме визуализации.
  + >> – Шаг вперед. Кнопка навигации. Доступна в режиме визуализации.
  + Текстовое поле – Вывод сообщений.
* Главная панель
  + Визуальное представление дерева.
* Статус-панель – панель для вывода статуса приложения. Изменяется во время выполнения определенных действий.

# Описание способов визуализации основных алгоритмов

## Описание визуализации алгоритма поиска

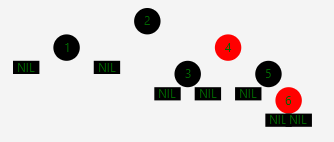
Поиск в красно-черном дереве аналогичен поиску в любом другом бинарном дереве. Визуализация этого алгоритма демонстрирует пользователю текущую вершину и все пройденные вершины на каждом шаге:

* Текущая вершина окрашивается в определенный цвет (отличный от других цветов)
* Пройденные вершины указываются в текстовом поле в порядке их использования

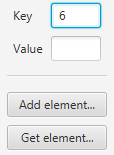
После завершения алгоритма выводится сообщение “End.” в текстовое поле. Найденная вершина окрашивается в определенный цвет. После повторного нажатия кнопки “>>” все цвета возвращаются в исходное состояние.

## Визуализация алгоритма поиска

Создадим начальное дерево, у которого ключи от 1 до 6 (см. *Описание пользовательского интерфейса*).



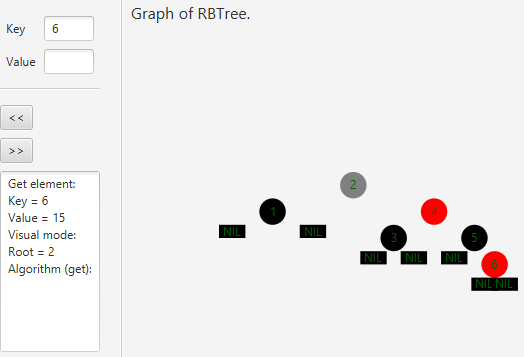
Введем в поле *Key* значение 6.



Перед нажатием управляющей кнопки “*Get element…”* убедимся, что переключатель *“Visualize”* активирован.

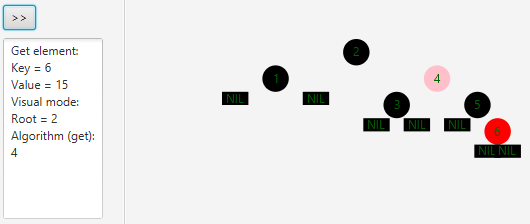


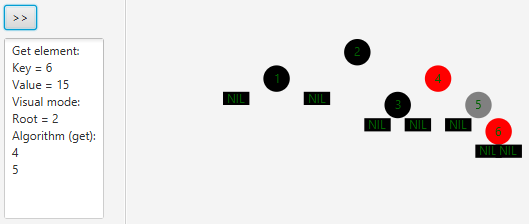
После нажатия управляющей кнопки “*Get element…”* интерфейс программы изменится, а именно кнопки *“Add element…”* и *“Get element…”* заменятся на кнопки для пошаговой визуализации: “>>” – следующий шаг, “<<” – предыдущий шаг.

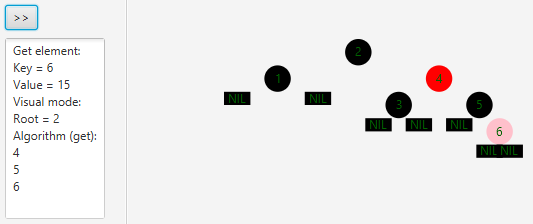


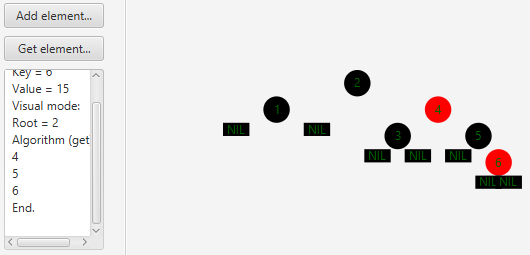
Как видно, изменился интерфейс, в текстовое поле добавились описание о выбранной вершине и начальное значение корня дерева.

Далее используем кнопки навигации *“<<”* , *“>>”* и прослеживаем алгоритм поиска. На каждом шаге текущая вершина окрашивается в светлый цвет и записывается ключ в текстовое поле.









По окончании визуализации алгоритма поиска все цвета вернулись в исходное состояние, кнопки навигации заменились на управляющие, в текстовое поле вывелось сообщение об окончании алгоритма.

## Описание визуализации алгоритма вставки

Вставка в красно-черном дереве состоит из двух шагов: вставка в нужное место новой вершины (красной), а далее балансировки полученного дерева при помощи раскраски и поворотов.

Визуализация достигается следующими способами:

* Первая часть алгоритма (поиск) визуализируется согласно предыдущему пункту.
* Поворот вершины визуализируется с помощью:
  + Обозначения вершины от которой выполняется поворот
  + Стрелки, определяющей направление поворота (влево или вправо)
* Смена цвета у вершин при балансировке.

После завершения алгоритма выводится сообщение “End.” в текстовое поле. Найденная вершина окрашивается в определенный цвет. После повторного нажатия кнопки “>>” все цвета возвращаются в исходное состояние.

# Основные этапы разработки

В данном разделе в виде таблицы будут перечислены основные этапы разработки. В таблице будут указаны даты, наименования и результаты (git commits) основных этапов разработки.

|  |  |  |
| --- | --- | --- |
| Дата | Этап разработки | Результаты |
| 24.06.2016 | Подготовка к разработке. Создание репозитория GitHub. Создание и назначение первых задач по разработке. | https://github.com/ JorgenPo/JavaPractice2016 |
| 24.06.2016 | Wiki страница с описанием пользовательского интерфейса. Прототип пользовательского интерфейса. Описание структуры данных (Красно – Черное дерево). Создана структура github проекта. | Wiki страницы. Структура проекта.  Прототип графического интерфейса. |
| 25.06.2016 | Wiki страница с требованиями к проекту. Создание страницы с описанием проекта. Готов интерфейс Red – Black Tree и node. Реализован класс Node. | Wiki страница с требованиями. Описание проекта. Интерфейс RBTree, Node, реализация Node. |
| 27.06.2016 | Доделан класс Node, реализован класс RBTree. Реализованы методы поиска и добавления вершины (добавление без балансировки). Добавлены классы графического интерфейса пользователя. | Класс RBTree. Методы добавления и поиска (без балансировки). Графический интерфейс пользователя (первая версия). |
| 29.06.2016 | Wiki страница с описанием способов визуализации алгоритмов (поиска и добавления вершины). Вторая версия пользовательского интерфейса. | Вторая версия GUI. Описания визуализации алгоритмов. |
| 30.06.2016 | Графическая визуализация дерева. Форматирование класса дерева (в соответствии со стандартами кода Java). Идет работа над GUI и сохранением / загрузкой дерева. | Графическая визуализация Red – Black Tree. |
| 1.07.2016 | Изменения графического интерфейса. Добавлено меню. Некоторые изменения класса дерева. Первая рабочая версия приложения. | Первая версия приложения. |
| 2.07.2016 | Добавлена иконка приложения. Добавлена возможность масштабирования дерева. Визуализация поиска вершины. Завершены методы загрузки и сохранения дерева. | Визуализация поиска. Методы загрузки и сохранения дерева. |
| 3.07.2016 | Появился метод поиска в ширину над деревом. Окончательно закончена работа над загрузкой и сохранения дерева на диске. Стабильная версия. | Первая стабильная версия. Можно сохранять и загружать дерево. |
| 4.07.2016 | Визуализация алгоритма вставки в дерево. Проведено тестирование и написан отчет. | Визуализация алгоритма вставки. Отчет и тестирование. |

# Тестирование приложения

|  |  |
| --- | --- |
| Input | Output |
| Key = Foo  Value = Boo  Click – <Add element…> |  |
| Key = Hello  Value = Tree  Click - <Add element…> |  |
| Key = Key  Value = Value  Click - <Add element…> |  |
| Попытка создания без Key или Value  >>> Вершина не создалась. Статут-панель изменилась. |  |
| Key = Foo  Click - <Get element…>  1.  Visualize OFF  2.  Visualize ON  См. 4.b. Визуализация алгоритма поиска. | * + 1. 2. |
| Click - <Save>  Write – tree  Click – Сохранить  >>> Данные о состояния дерева успешно сохранены в файл “tree.graph”.  Листинг файла. |  |
| Click - <Save>  Write – tree\_test  Click – Отмена |  |
| Key = 1  Value = 10  Click – Add element…  Click – <Open>  Choose “tree.graph”  Click – Открыть | Before opening    After opening |
| Click - <Save>  Choose – “tree.graph”  Click – Отмена |  |