|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Кафедра прикладной математики, механики, управления и программного обеспечения** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | | | | | |
| по лабораторной работе №2  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9120-09.03.04прогин | | | |
|  |  |  | | Перепечин В.В. | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | |  |
|  | | | | | | |  |  | ст. преподаватель | | | |
|  | | | |  |  | |  |  |  |  | О.А. Крестникова | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2021 | | | | | | | | | | | | |

1. **Неформальная постановка задачи**

Реализовать пакет подпрограмм для работы с красно-чёрным деревом поиска, в котором элементы представлены в виде структуры кода направления.

## Основные операции:

1. Создать дерево
2. Очистить дерево
3. Создать узел
4. Добавить узел
5. Удалить узел
6. Найти узел по значению
7. Найти min/max по значению в дереве
8. Обход дерева
9. Описание типа + спецификация подпрограмм + тесты

# Спецификация класса Key

Класс Key предназначен для хранения кода направления.

## Инициализация

Инициализирует класс входными значениями.  
*Входные данные:* строка вида “00.00.00”  
*Выходные данные:* Экземпляр класса Key

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Инициализация класса верной строкой | “01.02.03” | Key(01,02,03) |
| Инициализация класса не верной строкой | “foo” | Ошибка |

## Check

Сравнивает два экземпляра класса Key.  
Возвращает:

* 1, если значение первого больше второго.
* 0, если они равны.
* -1, если значение второго больше первого.

*Входные данные:* два экземпляра класса Key  
*Выходные данные:* Целое число 1, 0 или -1

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Сравнение экземпляров, где значение первого больше | Key(01,02,03), Key(99,02,03) | 1 |
| Сравнение экземпляра и null | Key(01,02,03), null | Ошибка |

# Спецификация класса List

Класс List является динамические односвязным списком. Хранит прошлый элемент и экземпляр класса Key.

## Remove

Удаляет последний элемент списка  
*Входные данные:* отсутствуют  
*Выходные данные:* текущий список

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Удаление элемента из списка с 2 элементами | [“01.02.03”,”24.44.12”] | [“01.02.03”] |
| Удаление элемента из списка с 1 элементом | [“01.02.03”] | null |

# Спецификация класса Node

Класс Node является узлом дерева. Хранит левый, правый элемент, родителя, цвет, экземпляр класса Key и экземпляр класса List.

## GetLevel

Возвращает чёрную высоту поддерева, где наш узел является корнем.  
*Входные данные:* отсутствуют  
*Выходные данные:* целое неотрицательное число

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Измерение высоты поддерева с высотой 3 |  | 3 |
| Измерение высоты поддерева с высотой 0 |  | 0 |

## AddEqual

Добавляет в конец экземпляра класса List элемент, содержащий такое же значение, что и сам узел.  
*Входные данные:* отсутствуют  
*Выходные данные:* отсутствуют

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Добавление, если список пуст | Null, “01.02.03” | [“01.02.03”] |
| Добавление, если список не пуст | [“01.02.03”], “01.02.03” | [“01.02.03”, “01.02.03”] |

## DeleteEqual

Удаляет их экземпляра класса List последний элемент.  
*Входные данные:* отсутствуют  
*Выходные данные:* отсутствуют

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Удаление, если список пуст | Null | Ошибка |
| Удаление, если список не пуст | [“01.02.03”, “01.02.03”] | [“01.02.03”] |
| Удаление, если в списке 1 элемент | [“01.02.03”] | null |

## HasEqual

Возвращает статус списка. Если он пуст, то false. Иначе true.  
*Входные данные:* отсутствуют  
*Выходные данные:* true или false

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Если список пуст | Null | false |
| Если список не пуст | [“01.02.03”, “01.02.03”] | true |

# Спецификация класса RedBlackTree

Класс RedBlackTree является структурой красно-черного дерева. Хранит экземпляр класса Node – корень.

## Insert

Добавляет в дерево новый элемент  
*Входные данные:* экземпляр класса Key  
*Выходные данные:* отсутствуют

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Вставка в пустое дерево | “01.02.03” | [“01.02.03”] |
| Вставка в не пустое дерево | “01.02.03” | […,“01.02.03”, …] |
| Вставка элемента с существующим экземпляром класса Key | “01.02.03” | […,“01.02.03”[2], …] |

## InserFixUp

Процедура балансировки дерева после вставки.  
*Входные данные:* экземпляр класса Node  
*Выходные данные:* отсутствуют

## leftRotate

Выполняет левосторонний поворот.  
*Входные данные:* экземпляр класса Node  
*Выходные данные:* отсутствуют

## rightRotate

Выполняет правосторонний поворот.  
*Входные данные:* экземпляр класса Node  
*Выходные данные:* отсутствуют

## Delete

Удаление элемента из дерева  
*Входные данные:* экземпляр класса Key  
*Выходные данные:* false,если элемент не найден. Иначе true.

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Удаление элемента, содержащегося в дереве | “01.02.03”, […,“01.02.03”, …] | true |
| Удаление элемента, не содержащегося в дереве | “01.02.03”, [… , …] | false |
| Удаление элемента, содержащегося в дереве не в единственном экземпляре | “01.02.03”, […,“01.02.03”[3], …] | true |

## Minimum

Нахождение минимального элемента в поддереве с заданным корнем.  
*Входные данные:* экземпляр класса Node  
*Выходные данные:* экземпляр класса Node

## Transplant

Меняет местами два экземпляра класса Node в дереве  
*Входные данные:* два экземпляра класса Node  
*Выходные данные:* отсутствуют

## Delete

Удаление элемента из дерева  
*Входные данные:* экземпляр класса Node  
*Выходные данные:* отсутствуют

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Удаление элемента, содержащегося в дереве | “01.02.03”, […,“01.02.03”, …] | [… , …] |
| Удаление элемента, не содержащегося в дереве | “01.02.03”, [… , …] | [… , …] |
| Удаление элемента, содержащегося в дереве не в единственном экземпляре | “01.02.03”, […,“01.02.03”[3], …] | […,“01.02.03”[2], …] |

## DeleteFixUp

Описание  
*Входные данные:* отсутствуют  
*Выходные данные:* отсутствуют

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
|  |  |  |
|  |  |  |

## DisplayTree

Процедура балансировки дерева после удаления.  
*Входные данные:* экземпляр класса Node  
*Выходные данные:* отсутствуют

## InCenterrightDisplay

Выводит значения элементов в симметричном порядке  
*Входные данные:* экземпляр класса Node  
*Выходные данные:* отсутствуют

## InCenterleftDisplay

Выводит значения элементов в обратном симметричном порядке  
*Входные данные:* экземпляр класса Node  
*Выходные данные:* отсутствуют

## InOrderDisplay

Выводит значения элементов по порядку  
*Входные данные:* экземпляр класса Node  
*Выходные данные:* отсутствуют

## InReverseDisplay

Выводит значения элементов в обратном порядке  
*Входные данные:* экземпляр класса Node  
*Выходные данные:* отсутствуют

## Find

Поиск элемента в дереве  
*Входные данные:* экземпляр класса Node  
*Выходные данные:* экземпляр класса Node

|  |  |  |
| --- | --- | --- |
| Описание тестовой ситуации | Входные данные | Выходные данные |
| Поиск элемента, содержащегося в дереве | “01.02.03”, […,“01.02.03”, …] | Node(01,02,03) |
| Поиск элемента, не содержащегося в дереве | “01.02.03”, [… , …] | Пустой узел |

1. Текст программы

# Текст файла List.cs

**using** System**;**

**namespace** BlackRedTree

**{**

**public** class List

**{**

**public** List prev**;**

**public** Key val**;**

**public** List**(**Key data**)**

**{**

val **=** data**;**

prev **=** **null;**

**}**

**public** List**(**List link**,** Key data**)**

**{**

val **=** data**;**

prev **=** link**;**

**}**

**public** List Remove**()**

**{**

**if** **(!**prev**)**

**return** **null;**

prev **=** prev**.**prev**;**

**return** **this;**

**}**

**public** static **implicit** **operator** bool**(**List p**)**

**{**

**return** p **!=** **null;**

**}**

**~**List**()**

**{**

List p **=** **this;**

**while** **(**p**)**

p**.**Remove**();**

**}**

**}**

**}**

# Текст файла Key.cs

**using** System**;**

**namespace** BlackRedTree

**{**

**public** class Key

**{**

**private** int**[]** f **=** **{** 0**,** 0**,** 0 **};**

**public** Key**(**int k1**,** int k2**,** int k3**)**

**{**

f **=** **new** int**[]** **{** k1**,** k2**,** k3 **};**

**}**

**public** Key**(**string str**)**

**{**

f**[**0**]** **=** f**[**1**]** **=** f**[**2**]** **=** 0**;** // save for TryParse

string**[]** arr **=** str**.**Split**(**'.'**);**

int**.**TryParse**(**arr**[**0**],** **out** f**[**0**]);**

int**.**TryParse**(**arr**[**1**],** **out** f**[**1**]);**

int**.**TryParse**(**arr**[**2**],** **out** f**[**2**]);**

**}**

**public** static **implicit** **operator** Key**(**string str**)**

**{**

**return** **new** Key**(**str**);**

**}**

**public** static **implicit** **operator** string**(**Key p**)**

**{**

**return** p**.**f**[**0**].**ToString**(**"00"**)** **+** '.' **+** p**.**f**[**1**].**ToString**(**"00"**)** **+** '.' **+** p**.**f**[**2**].**ToString**(**"00"**);**

**}**

**private** static int Check**(**Key a**,** Key b**)**

**{**

**if** **(**a**.**f**[**0**]** **<** b**.**f**[**0**])**

**return** **-**1**;**

**else** **if** **(**a**.**f**[**0**]** **>** b**.**f**[**0**])**

**return** 1**;**

**if** **(**a**.**f**[**1**]** **<** b**.**f**[**1**])**

**return** **-**1**;**

**else** **if** **(**a**.**f**[**1**]** **>** b**.**f**[**1**])**

**return** 1**;**

**if** **(**a**.**f**[**2**]** **<** b**.**f**[**2**])**

**return** **-**1**;**

**else** **if** **(**a**.**f**[**2**]** **>** b**.**f**[**2**])**

**return** 1**;**

**return** 0**;** // a = b

**}**

**public** static bool **operator** **!=(**Key a**,** Key b**)**

**{**

**return** Check**(**a**,** b**)** **!=** 0**;**

**}**

**public** static bool **operator** **==(**Key a**,** Key b**)**

**{**

**return** Check**(**a**,** b**)** **==** 0**;**

**}**

**public** static bool **operator** **<(**Key a**,** Key b**)**

**{**

**return** Check**(**a**,** b**)** **==** **-**1**;**

**}**

**public** static bool **operator** **>(**Key a**,** Key b**)**

**{**

**return** Check**(**a**,** b**)** **==** 1**;**

**}**

**}**

**}**

# Текст файла Node.cs

**using** System**;**

**namespace** BlackRedTree

**{**

**public** class Node

**{**

**public** static **implicit** **operator** string**(**Node node**)**

**{**

int k **=** 0**;**

List p **=** node**.**List**;**

**while** **(**p**)**

**{**

k**++;**

p **=** p**.**prev**;**

**}**

**return** node**.**data **+** **(**k **!=** 0 **?** " [" **+** **(**1 **+** k**).**ToString**()** **+** ']' **:** ""**);**

**}**

**public** int GetLevel**()**

**{**

int k **=** 0**;**

Node n **=** **this;**

**while** **(**n**)**

**{**

**if** **(**n**.**Color **==** Color**.**Black**)**

k**++;**

n **=** n**.**parent**;**

**}**

**return** k**;**

**}**

//Dynamic list

**public** List List **=** **null;**

**public** void AddEqual**()**

**{**

List **=** **new** List**(**List**,** data**);**

**}**

**public** void DeleteEqual**()**

**{**

List **=** List**.**Remove**();**

**}**

**public** bool HasEqual**()**

**{**

**return** List**;**

**}**

//Black Red Tree

**public** Color Color**;**

//Node AVL

**public** Node left**;**

**public** Node right**;**

**public** Node parent**;**

**public** Key data**;**

//Initialization

**public** Node**(**Key data**)**

**{**

**this.**data **=** data**;**

**}**

**public** Node**(**Color Color**)**

**{**

**this.**Color **=** Color**;**

**}**

**public** Node**(**Key data**,** Color Color**)**

**{**

**this.**data **=** data**;** **this.**Color **=** Color**;**

**}**

**public** Node**(**Node parent**,** Color Color**)**

**{**

**this.**parent **=** parent**;**

**this.**Color **=** Color**;**

**}**

//Func

**public** static **implicit** **operator** bool**(**Node x**)** **{** **return** x **!=** **null;** **}**

**public** int Height**()** **{** **return** **this.**parent **?** **this.**parent**.**Height**()** **+** 1 **:** 1**;** **}**

**}**

**}**

# Текст файла RedBlackTree.cs

**using** System**;**

**namespace** BlackRedTree

**{**

class RedBlackTree

**{**

**public** Node root**;**

**public** Node sentinel**;**

**public** RedBlackTree**()**

**{**

root **=** sentinel **=** **new** Node**(**root**,** Color**.**Black**);**

root**.**left **=** root**.**right **=** sentinel**;**

**}**

**~**RedBlackTree**()**

**{**

RecDestruct**(**root**);**

**}**

**private** void RecDestruct**(**Node x**)**

**{**

**if** **(!**x**)** **return;**

RecDestruct**(**x**.**left**);**

RecDestruct**(**x**.**right**);**

Delete**(**x**);**

**}**

#region Insert

**public** void Insert**(**Key itemKey**)**

**{**

Node z **=** **new** Node**(**itemKey**);**

**if** **(**root **==** sentinel**)**

**{**

root **=** z**;**

root**.**parent **=** sentinel**;**

root**.**left **=** sentinel**;**

root**.**right **=** sentinel**;**

root**.**Color **=** Color**.**Black**;**

**return;**

**}**

Node Y **=** sentinel**;**

Node X **=** root**;**

**while** **(**X **!=** sentinel**)**

**{**

Y **=** X**;**

**if** **(**z**.**data **<** X**.**data**)**

X **=** X**.**left**;**

**else** **if** **(**z**.**data **==** X**.**data**)**

**{**

X**.**AddEqual**();**

**return;**

**}**

**else**

X **=** X**.**right**;**

**}**

z**.**parent **=** Y**;**

**if** **(**Y **==** sentinel**)**

root **=** z**;**

**else** **if** **(**z**.**data **<** Y**.**data**)**

Y**.**left **=** z**;**

**else**

Y**.**right **=** z**;**

z**.**left **=** sentinel**;**

z**.**right **=** sentinel**;**

z**.**Color **=** Color**.**Red**;**

InsertFixUp**(**z**);**

**}**

**private** void InsertFixUp**(**Node z**)**

**{**

**while** **(**z **!=** root **&&** z**.**parent**.**Color **==** Color**.**Red**)**

**{**

Node y**;**

**if** **(**z**.**parent **==** z**.**parent**.**parent**.**left**)**

**{**

y **=** z**.**parent**.**parent**.**right**;**

**if** **(**y **!=** **null** **&&** y**.**Color **==** Color**.**Red**)**

**{**

z**.**parent**.**Color **=** Color**.**Black**;**

y**.**Color **=** Color**.**Black**;**

z**.**parent**.**parent**.**Color **=** Color**.**Red**;**

z **=** z**.**parent**.**parent**;**

**}**

**else**

**{**

**if** **(**z **==** z**.**parent**.**right**)**

**{**

z **=** z**.**parent**;**

leftRotate**(**z**);**

**}**

z**.**parent**.**Color **=** Color**.**Black**;**

z**.**parent**.**parent**.**Color **=** Color**.**Red**;**

rightRotate**(**z**.**parent**.**parent**);**

**}**

**}**

**else**

**{**

y **=** z**.**parent**.**parent**.**left**;**

**if** **(**y **!=** **null** **&&** y**.**Color **==** Color**.**Red**)**

**{**

z**.**parent**.**Color **=** Color**.**Black**;**

y**.**Color **=** Color**.**Black**;**

z**.**parent**.**parent**.**Color **=** Color**.**Red**;**

z **=** z**.**parent**.**parent**;**

**}**

**else**

**{**

**if** **(**z **==** z**.**parent**.**left**)**

**{**

z **=** z**.**parent**;**

rightRotate**(**z**);**

**}**

z**.**parent**.**Color **=** Color**.**Black**;**

z**.**parent**.**parent**.**Color **=** Color**.**Red**;**

leftRotate**(**z**.**parent**.**parent**);**

**}**

**}**

**}**

root**.**Color **=** Color**.**Black**;**

**}**

#endregion

#region Rotates

**private** void leftRotate**(**Node X**)**

**{**

Node Y **=** X**.**right**;**

X**.**right **=** Y**.**left**;**

**if** **(**Y**.**left **!=** sentinel**)**

Y**.**left**.**parent **=** X**;**

Y**.**parent **=** X**.**parent**;**

**if** **(**X**.**parent **==** sentinel**)**

root **=** Y**;**

**else** **if** **(**X **==** X**.**parent**.**left**)**

X**.**parent**.**left **=** Y**;**

**else**

X**.**parent**.**right **=** Y**;**

Y**.**left **=** X**;**

X**.**parent **=** Y**;**

**}**

**private** void rightRotate**(**Node Y**)**

**{**

Node X **=** Y**.**left**;**

Y**.**left **=** X**.**right**;**

**if** **(**X**.**right **!=** sentinel**)**

X**.**right**.**parent **=** Y**;**

X**.**parent **=** Y**.**parent**;**

**if** **(**Y**.**parent **==** sentinel**)**

root **=** X**;**

**else** **if** **(**Y **==** Y**.**parent**.**right**)**

Y**.**parent**.**right **=** X**;**

**else**

Y**.**parent**.**left **=** X**;**

X**.**right **=** Y**;**

Y**.**parent **=** X**;**

**}**

#endregion

#region Delete

**public** bool Delete**(**Key key**)**

**{**

Node z **=** Find**(**key**);**

**if** **(**z **!=** sentinel**)**

**{**

Delete**(**z**);**

**return** **true;**

**}**

**return** **false;**

**}**

**private** Node Minimum**(**Node X**)**

**{**

Node Y **=** sentinel**;**

**while** **(**X **!=** sentinel**)**

**{**

Y **=** X**;**

X **=** X**.**left**;**

**}**

**return** Y**;**

**}**

**private** void Transplant**(**Node u**,** Node v**)**

**{**

**if** **(**u**.**parent **==** sentinel**)**

root **=** v**;**

**else** **if** **(**u **==** u**.**parent**.**left**)**

u**.**parent**.**left **=** v**;**

**else**

u**.**parent**.**right **=** v**;**

**if(**v**)**

v**.**parent **=** u**.**parent**;**

**}**

**public** void Delete**(**Node z**)**

**{**

**if** **(**z**.**HasEqual**())**

**{**

z**.**DeleteEqual**();**

**return;**

**}**

Node y **=** z**;**

Node x**;**

Color y\_original\_Color **=** y**.**Color**;**

**if** **(**z**.**left **==** sentinel**)**

**{**

x **=** z**.**right**;**

Transplant**(**z**,** z**.**right**);**

**}**

**else** **if** **(**z**.**right **==** sentinel**)**

**{**

x **=** z**.**left**;**

Transplant**(**z**,** z**.**left**);**

**}**

**else**

**{**

y **=** Minimum**(**z**.**right**);** //Без проверок возможна бесконечная рекурсия

y\_original\_Color **=** y**.**Color**;**

x **=** y**.**right**;**

**if** **(**y**.**parent **==** z**)**

x**.**parent **=** y**;**

**else**

**{**

Transplant**(**y**,** y**.**right**);**

y**.**right **=** z**.**right**;**

y**.**right**.**parent **=** y**;**

**}**

Transplant**(**z**,** y**);**

y**.**left **=** z**.**left**;**

y**.**left**.**parent **=** y**;**

y**.**Color **=** z**.**Color**;**

**}**

**if** **(**y\_original\_Color **==** Color**.**Black**)**

DeleteFixUp**(**x**);**

**}**

**private** void DeleteFixUp**(**Node X**)**

**{**

**while** **(**X **!=** root **&&** X**.**Color **==** Color**.**Black**)**

**{**

**if** **(**X **==** X**.**parent**.**left**)**

**{**

Node Y **=** X**.**parent**.**right**;**

**if** **(**Y**.**Color **==** Color**.**Red**)**

**{**

Y**.**Color **=** Color**.**Black**;**

X**.**parent**.**Color **=** Color**.**Red**;**

leftRotate**(**X**.**parent**);**

Y **=** X**.**parent**.**right**;**

**}**

**if** **(**Y**.**left**.**Color **==** Color**.**Black **&&** Y**.**right**.**Color **==** Color**.**Black**)**

**{**

Y**.**Color **=** Color**.**Red**;**

X **=** X**.**parent**;**

**}**

**else**

**{**

**if** **(**Y**.**right**.**Color **==** Color**.**Black**)**

**{**

Y**.**left**.**Color **=** Color**.**Black**;**

Y**.**Color **=** Color**.**Red**;**

rightRotate**(**Y**);**

Y **=** X**.**parent**.**right**;**

**}**

Y**.**Color **=** X**.**parent**.**Color**;**

X**.**parent**.**Color **=** Color**.**Black**;**

Y**.**right**.**Color **=** Color**.**Black**;**

leftRotate**(**X**.**parent**);**

X **=** root**;**

**}**

**}**

**else**

**{**

Node Y **=** X**.**parent**.**left**;**

**if** **(**Y**.**Color **==** Color**.**Red**)**

**{**

Y**.**Color **=** Color**.**Black**;**

X**.**parent**.**Color **=** Color**.**Red**;**

rightRotate**(**X**.**parent**);**

Y **=** X**.**parent**.**left**;**

**}**

**if** **(**Y**.**right**.**Color **==** Color**.**Black **&&** Y**.**left**.**Color **==** Color**.**Black**)**

**{**

Y**.**Color **=** Color**.**Red**;**

X **=** X**.**parent**;**

**}**

**else**

**{**

**if** **(**Y**.**left**.**Color **==** Color**.**Black**)**

**{**

Y**.**right**.**Color **=** Color**.**Black**;**

Y**.**Color **=** Color**.**Red**;**

leftRotate**(**Y**);**

Y **=** X**.**parent**.**left**;**

**}**

Y**.**Color **=** X**.**parent**.**Color**;**

X**.**parent**.**Color **=** Color**.**Black**;**

Y**.**left**.**Color **=** Color**.**Black**;**

rightRotate**(**X**.**parent**);**

X **=** root**;**

**}**

**}**

**}**

X**.**Color **=** Color**.**Black**;**

**}**

#endregion

#region Display

**public** void DisplayTree**(**bool all **=** **false)**

**{**

int k **=** Display**(**root**,** Console**.**BufferWidth **/** 2**,** Console**.**BufferWidth **/** 2**,** 0**,** 4**);**

Console**.**SetCursorPosition**(**0**,** k**-**1**);**

**if** **(**all**)**

**{**

Console**.**WriteLine**(**"InCenterrightDisplay"**);** InCenterrightDisplay**(**root**);** Console**.**WriteLine**(**'\n'**);**

Console**.**WriteLine**(**"InCenterleftDisplay"**);** InCenterleftDisplay**(**root**);** Console**.**WriteLine**(**'\n'**);**

Console**.**WriteLine**(**"InOrderDisplay"**);** InOrderDisplay**(**root**);** Console**.**WriteLine**(**'\n'**);**

Console**.**WriteLine**(**"InReverseDisplay"**);** InReverseDisplay**(**root**);** Console**.**WriteLine**(**'\n'**);**

**}**

**}**

**private** void InCenterrightDisplay**(**Node current**)** //Центрированный обход

**{**

**if** **(**current **==** sentinel**)**

**return;**

InCenterrightDisplay**(**current**.**right**);**

Console**.**Write**(**"(" **+** current **+** ") "**);**

InCenterrightDisplay**(**current**.**left**);**

**}**

**private** void InCenterleftDisplay**(**Node current**)** //Центрированный обход

**{**

**if** **(**current **==** sentinel**)**

**return;**

InCenterleftDisplay**(**current**.**left**);**

Console**.**Write**(**"(" **+** current **+** ") "**);**

InCenterleftDisplay**(**current**.**right**);**

**}**

**private** void InOrderDisplay**(**Node current**)** //Прямой обход

**{**

**if** **(**current **==** sentinel**)**

**return;**

Console**.**Write**(**"(" **+** current **+** ") "**);**

InOrderDisplay**(**current**.**left**);**

InOrderDisplay**(**current**.**right**);**

**}**

**private** void InReverseDisplay**(**Node current**)** //Обратный обход

**{**

**if** **(**current **==** sentinel**)**

**return;**

InReverseDisplay**(**current**.**left**);**

InReverseDisplay**(**current**.**right**);**

Console**.**Write**(**"(" **+** current **+** ") "**);**

**}**

**private** int Display**(**Node node**,** int offset **=** 10**,** int x **=** 20**,** int y **=** 0**,** int lineoffset **=** 2**)**

**{**

**if** **(**node **==** sentinel**)**

**return** y**;**

Console**.**SetCursorPosition**(**x**,** y **+** 1**);** //For change Color

Console**.**ForegroundColor **=** ConsoleColor**.**DarkBlue**;**

**if** **(**node**.**right **!=** sentinel **||** node**.**left **!=** sentinel**)**

**{**

Console**.**WriteLine**(**"│"**);**

Console**.**SetCursorPosition**(**x**,** y **+** 2**);**

**if** **(**node**.**right **!=** sentinel **&&** node**.**left **!=** sentinel**)**

Console**.**WriteLine**(**"┴"**);**

**if** **(**node**.**right **!=** sentinel **&&** node**.**left **==** sentinel**)**

Console**.**WriteLine**(**"└"**);**

**if** **(**node**.**right **==** sentinel **&&** node**.**left **!=** sentinel**)**

Console**.**WriteLine**(**"┘"**);**

**for** **(**int i **=** 0**;** i **<** offset **/** 2 **-** 1**;** i**++)**

**{**

**if** **(**node**.**right **==** sentinel **&&** node**.**left **!=** sentinel**)**

**{**

Console**.**SetCursorPosition**(**x **-** i **-** 1**,** y **+** 2**);**

Console**.**WriteLine**(**"─"**);**

**if** **(**i **==** offset **/** 2 **-** 2**)**

**{**

Console**.**SetCursorPosition**(**x **-** offset **/** 2 **+** 1**,** y **+** 2**);**

Console**.**WriteLine**(**"┌"**);**

Console**.**SetCursorPosition**(**x **-** offset **/** 2 **+** 1**,** y **+** 3**);**

Console**.**WriteLine**(**"│"**);**

**}**

**continue;**

**}**

**if** **(**node**.**right **!=** sentinel **&&** node**.**left **==** sentinel**)**

**{**

Console**.**SetCursorPosition**(**x **+** i **+** 1**,** y **+** 2**);**

Console**.**WriteLine**(**"─"**);**

**if** **(**i **==** offset **/** 2 **-** 2**)**

**{**

Console**.**SetCursorPosition**(**x **+** offset **/** 2**,** y **+** 2**);**

Console**.**WriteLine**(**"┐"**);**

Console**.**SetCursorPosition**(**x **+** offset **/** 2**,** y **+** 3**);**

Console**.**WriteLine**(**"│"**);**

**}**

**continue;**

**}**

Console**.**SetCursorPosition**(**x **-** i **-** 1**,** y **+** 2**);**

Console**.**WriteLine**(**"─"**);**

**if** **(**i **==** offset **/** 2 **-** 2**)**

**{**

Console**.**SetCursorPosition**(**x **-** offset **/** 2 **+** 1**,** y **+** 2**);**

Console**.**WriteLine**(**"┌"**);**

Console**.**SetCursorPosition**(**x **-** offset **/** 2 **+** 1**,** y **+** 3**);**

Console**.**WriteLine**(**"│"**);**

**}**

Console**.**SetCursorPosition**(**x **+** i **+** 1**,** y **+** 2**);**

Console**.**WriteLine**(**"─"**);**

**if** **(**i **==** offset **/** 2 **-** 2**)**

**{**

Console**.**SetCursorPosition**(**x **+** offset **/** 2**,** y **+** 2**);**

Console**.**WriteLine**(**"┐"**);**

Console**.**SetCursorPosition**(**x **+** offset **/** 2**,** y **+** 3**);**

Console**.**WriteLine**(**"│"**);**

**}**

**}**

**}**

Console**.**SetCursorPosition**(**x**-**3**,** y**);**

Console**.**ForegroundColor **=** node**.**Color **==** Color**.**Red **?** ConsoleColor**.**DarkRed **:** ConsoleColor**.**White**;**

Console**.**Write**(**node**.**data**);**

offset **/=** 2**;**

int k **=** Display**(**node**.**left**,** offset**,** x **-** offset**,** y **+** lineoffset**,** lineoffset**);**

int c **=** Display**(**node**.**right**,** offset**,** x **+** offset**,** y **+** lineoffset**,** lineoffset**);**

**return** k **>** c **?** k **:** c**;**

**}**

#endregion

**public** Node Find**(**Key key**)**

**{**

Node temp **=** root**;**

**while** **(**temp **!=** sentinel**)**

**{**

**if** **(**key **<** temp**.**data**)**

temp **=** temp**.**left**;**

**if** **(**key **>** temp**.**data**)**

temp **=** temp**.**right**;**

**if** **(**key **==** temp**.**data**)**

**return** temp**;**

**}**

**return** sentinel**;**

**}**

**}**

**}**