Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт Информационных технологий и анализа данных

АБСТРАКТНЫЕ СТРУКТУРЫ ДАННЫХ.

ОТЧЕТ  
по лабораторной работе №5  
Вариант №10

по дисциплине «Программирование»

Выполнил

Студент группы ЭВМб-23-3 \_\_\_\_\_\_\_\_\_\_\_\_Г. М. Распутин  
  
Принял

\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедева Е.Н.

Иркутск 2024

# Бинарное дерево поиска

## Условие задачи.

* *10в:*  Создать структуру (структуры), соответствующие индивидуальному заданию, с использованием бинарного дерева. Создать функции для выполнения следующих операций над двоичным деревом: добавление элемента; поиск элемента по ключу; удаление элемента по ключу; обход дерева в ширину и печать элементов; обход дерева в глубину и печать элементов, прямой, симметричный (по возрастанию и убыванию) и обратный. Функции обработки, указанные в индивидуальном варианте задания. (3A) Удалить все элементы, у которых "специальность" совпадает с заданным значением специальности.

## Таблица внешних спецификаций всей программы

Таблица 1.1 – Внешние спецификации всей программы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Имя | Назначение | Тип | Вх/Вых. | Диапазон |
| 1 | Choise | Выбор тест-функции, чтобы проверить работоспособность программы | Целочисленное | Вход | От 1 до 4 |
| 2 | Yes | Переменная, которая запускает функцию main ещё раз, если пользователь согласится | Целочисленные | Вход | От 0 до 1 |

## **Таблица спецификаций структур**

Таблица – Спецификация структуры дерева Tree

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Имя поля | Назначение поля | Тип поля | Диапазон |
| 1 | Data | Хранение данных элемента | Data | Символы ASCII кодировки |
| 2 | parent | Хранение указателя на родителя узла | Tree\* | Tree\* |
| 3 | left | Хранение указателя на левого потомка узла | Tree\* | Tree\* |
| 4 | right | Хранение указателя на правого потомка узла | Tree\* | Tree\* |

Таблица – Спецификация структуры данных Data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Имя поля | Назначение поля | Тип поля | Диапазон |
| 1 | fio | ФИО специалиста. | Строка | Символы ASCII кодировки |
| 2 | tabel\_num | Табельный номер | Вещественное | От -9\*10^4 до От 9\*10^4 |
| 3 | experience | Стаж работы | Целое | От 1 до 90 |
| 4 | salary | Зарплата | Целое | От 1000  до 2 147 483 |
| 5 | speciality | Специальность | Строка | Символы ASCII кодировки |

## **4. Схема иерархии функций**

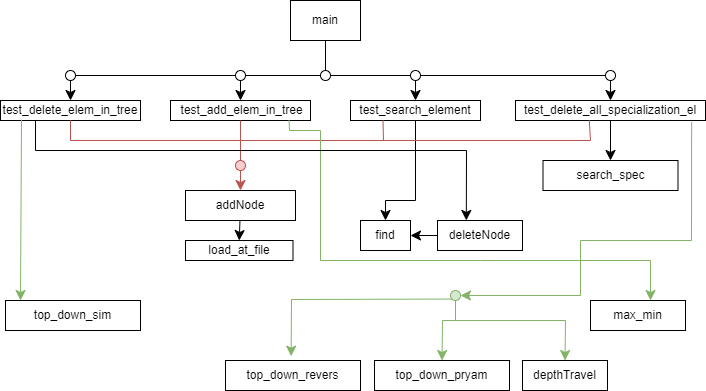
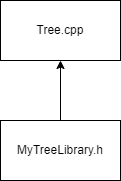


Рисунок 1 - Схема иерархии функций.

## **Таблица спецификаций функций**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Имя функции** | **Назначение** | **Тип результата** | **Параметры** |
| **1** | test\_delete\_elem\_in\_tree | Тестирование удаления узла в дереве и их отображение с помощью обхода в глубину симметрично | Текст | - |
| **2** | test\_add\_elem\_in\_tree | Тестирование добавления узла в дерево и отображение узлов с помощью обхода от максимальному к минимальному | Текст | - |
| **3** | test\_search\_element | Тестирование функции find и вывод найденного элемента с помощью cout | Текст | - |
| **4** | test\_delete\_all\_specialization\_el | Тестирование функции search\_spec и вывод элементов с помощью оставшихся обходов дерева(обратного, прямого и в глубину) | Текст | - |
| **5** | addNode | Добавление элемента в дерево | Tree\* | data, Data, по значению, информация элемента |
| **6** | load\_at\_file | Создание случайной даты для последующего использования её в addNode | Data | m, целое, по значению, число, которое будет прибавляться к случайно сгенерированному табельному номеру  show\_names, целое, при значении 1 будет отображать имена и специализацию добавляемой даты  random\_tabel\_num, целое, по значению, при значении 0 табельный номер будет строго равен m |
| **7** | find | Поиск элемента дерева по заданному ключу | Tree\* | tree, Tree\*, по указателю, дерево в котором ищется конкретный узел  key, целое, по значению, ключ, по которому ищется узел |
| **8** | deleteNode | Удаление элемента по заданному ключу | Tree\* | tree, Tree\*, по указателю, дерево в котором ищется и удаляется конкретный узел  key, целое, по значению, ключ, по которому ищется и удаляестя узел |
| **9** | search\_spec | Поиск и удаление всех элементов с заданной специальностью | Tree\* | root, Tree\*, по указателю, дерево в котором ищутся и удаляются конкретные узлы  tree, Tree по указателю, дерево, которое рекурсивно передается в функцию для более удобного поиска  spec, текстовое, по значению, заданная специальность |
| **10** | max\_min | Вывод всех элементов дерева от максимального к минимальному | Текстовый | tree, Tree\*, по указателю, дерево, по которому происходит обход |
| **11** | depthTravel | Вывод всех элементов дерева в глубину | Текстовый | tree, Tree\*, по указателю, дерево, по которому происходит обход |
| **12** | top\_down\_pryam | Вывод всех элементов дерева прямым обходом | Текстовый | tree, Tree\*, по указателю, дерево, по которому происходит обход |
| **13** | top\_down\_revers | Вывод всех элементов дерева обратным обходом | Текстовый | tree, Tree\*, по указателю, дерево, по которому происходит обход |
| 14 | top\_down\_sim | Вывод всех элементов дерева симметричным обходом | *­* Текстовый | tree, Tree\*, по указателю, дерево, по которому происходит обход |

## **6. Схема иерархии модулей**



## **7. Словесное описание основных алгоритмов**

**Алгоритм добавления элемента в дерево:**

1. Принимаем на вход указатель на дерево (tree), указатель на родительский узел (parent) и данные для нового узла (d).

2. Проверяем, пустое ли дерево (tree == NULL).

3. Если дерево пустое, создаем новый узел, присваиваем ему переданные данные (d), указываем родителя (parent), и указатели на левого и правого потомков устанавливаем как NULL.

4. Если дерево не пустое, сравниваем данные нового узла (d) с данными текущего узла в дереве (tree->data.tabel\_num).

5. Если данные нового узла меньше, чем данные текущего узла, вызываем рекурсивно функцию addNode для левого потомка текущего узла (tree->left) и передаем в нее указатель на текущий узел (tree) в качестве родителя.

6. Если данные нового узла больше или равны данным текущего узла, вызываем рекурсивно функцию addNode для правого потомка текущего узла (tree->right) и передаем в нее указатель на текущий узел (tree) в качестве родителя.

7. Возвращаем указатель на измененное дерево.

**Алгоритм поиска элемента в дереве:**

1. Проверить, является ли дерево пустым. Если да, вывести сообщение "Узел не найден" и вернуть NULL.

2. Если ключ текущего узла равен искомому ключу, вернуть текущий узел.

3. Если искомый ключ меньше ключа текущего узла, рекурсивно вызвать функцию поиска для левого поддерева.

4. Если искомый ключ больше ключа текущего узла, рекурсивно вызвать функцию поиска для правого поддерева.

5. Вернуть результат рекурсивного вызова функции.

**Алгоритм удаления элемента в дереве:**

Если текущий узел null, вернуть null.

Если значение текущего узла равно искомому, вернуть текущий узел.

Если искомое значение меньше значения текущего узла, установить левого потомка текущим узлом и перейти к шагу 1.

В противном случае установить правого потомка текущим узлом и перейти к шагу 1.

После того, как мы нашли узел, который необходимо удалить, у нас возможны три случая.

Случай 1: У удаляемого узла нет правого потомка. В этом случае мы просто перемещаем левого ребенка (при его наличии) на место удаляемого узла.

Случай 2: У удаляемого узла есть правый потомок, у которого, в свою очередь нет левого потомка. В этом случае нам надо переместить правого потомка удаляемого узла (6) на его место.

Случай 3: У удаляемого узла есть правый потомок, у которого есть левый потомок.

Мы должны поместить на место удаляемого узел со значением, меньшим или равным любому узлу справа от него. Поэтому мы берем крайний левый узел правого поддерева и перемещаем его на место удаляемого.

Алгоритм:

1. Находим узел с ключом key в дереве tree.

2. Если узел не найден (deletedTree == NULL), возвращаем исходное дерево.

3. Если у узла нет потомков (deletedTree->left == NULL && deletedTree->right == NULL):

- Если узел является корнем дерева, возвращаем NULL.

- Иначе, удаляем узел из родительского узла.

4. Если у узла нет правого потомка (deletedTree->right == NULL):

- Заменяем удаляемый узел его левым потомком.

5. Если у правого потомка удаляемого узла нет левого потомка:

- Заменяем удаляемый узел его правым потомком.

6. Если у правого потомка удаляемого узла есть левый потомок:

- Находим крайний левый потомок правого поддерева.

- Заменяем удаляемый узел данными крайнего левого потомка.

- Переносим правого потомка крайнего левого потомка на место перемещённого корня.

Возвращаем измененное дерево tree после выполнения всех шагов.

**Алгоритм удаления заданной специальности:**1. Начать с корневого элемента дерева.  
2. Проверить, не является ли текущий элемент NULL. Если является, вернуться.  
3. Рекурсивно вызвать функцию для правого поддерева.  
4. Рекурсивно вызвать функцию для левого поддерева.  
5. Проверить, совпадает ли значение специальности текущего элемента с заданным значением специальности.  
6. Если совпадает, вывести информацию о текущем элементе (ФИО, специальность, номер табельного номера) и удалить элемент из дерева с помощью функции deleteNode.  
7. Вернуться к шагу 2 для следующего элемента в дереве.  
8. Завершить выполнение функции.  
  
Этот алгоритм рекурсивно обходит все элементы дерева и удаляет те, у которых значение специальности совпадает с заданным значением.

## **8. Таблица тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Назначение теста | Входные данные | Выходные данные |
| 1 | Тестирование удаления узла в дереве и их отображение с помощью обхода в глубину симметрично |  | Вывод от минимального элемента к максимальному(симметричный вывод): 100, 101, 102,  Удалим узел 102:  Вывод от минимального элемента к максимальному(симметричный вывод): 100, 101, |
| 2 | Тестирование добавления узла в дерево и отображение узлов с помощью обхода от максимальному к минимальному |  | Вывод от максимального к минимальному: 100, 12, 11,  Создадим ещё один узел: 4. Спец.: Программист, Табельный номер : 103  фио: Алексеев Феликс Миронович  Вывод от максимального к минимальному: 103, 100, 12, 11, |
| 3 | Тестирование функции find и вывод найденного элемента с помощью cout |  | Ищем ФИО специалиста с табельным номером 101:  Терентьев Леонид Евгеньевич |
| 4 | Тестирование функции search\_spec и вывод элементов с помощью оставшихся обходов дерева(обратного, прямого и в глубину) |  | Григорий Распутин Максимович - Электрик с номером 9999 был удалён.  Все табельные номера в обратном порядке: 17, 14, 13, 106, 1008, 1005, 1002, 1001, 100,  Все табельные номера в прямом порядке: 100, 13, 14, 17, 1001, 106, 1002, 1005, 1008,  Все табельные номера в ширину:  1. 100  2. 13  3. 1001  4. 14  5. 106  6. 1002  7. 17  8. 1005  9. 1008 |

## **9. Программа на языке С++**

<https://replit.com/@GieorghiiRasput/bigverybigtree#main.cpp>

//В коде очень много строк, поэтому я загрузил его на replit и отправил файлом на Moodle

## 