

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА
по дисциплине «Алгоритмы и структуры данных»
Тема: Случайные БДП - вставка и исключение. Демонстрация

Студент гр. 7381

Дорох С.В.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2017

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Дорох С.В.

Группа 7381

Тема работы: Случайные БДП - вставка и исключение. Демонстрация.

Исходные данные:

На вход программе подаются следующие данные: элементы для построения бинарного дерева поиска, номер действия, необходимого для работы с программой, и элемент для работы с деревом.

Дата сдачи реферата:

Дата защиты реферата:

Студент

Дорох С.В.

Преподаватель

Фирсов М.А.

Содержание

Содержание	3
Аннотация	4
Введение	5
Постановка задачи	6
Алгоритм работы программы	7
Функции и структуры данных	8
Пользовательский интерфейс	11
Тестирование	13
Заключение	17

Аннотация

В данной курсовой работе реализовано БДП и основные функции для работы с ним: вставка и удаление элемента из дерева, поиск заданного элемента, поиск минимального и максимального элемента дерева.

В работе бинарное дерево поиска создавалось с целью, чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с ней действий. Реализован удобный и понятный для изучения структуры данных БДП интерфейс. В качестве языка программирования для создания программы выбран язык программирования C++.

Введение

Данная курсовая работа основана на работе с бинарным деревом поиска(БДП).

БДП — двоичное дерево поиска, для которого выполняются следующие дополнительные условия: оба поддерева – левое и правое – являются двоичными деревьями поиска, у всех узлов левого поддерева произвольного узла значения меньше, нежели значения ключей данных самого узла, у всех узлов правого поддерева произвольного узла значения ключей данных больше нежели ключи данных самого узла.

Постановка задачи

Реализация демонстрации по бинарным деревьям поиска . "Демонстрация" - визуализация структур данных, алгоритмов, действий. Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий. Особенно подробно должна быть продемонстрирована работа функций вставки исключения и построение БДП.

Алгоритм работы программы

На вход программе подаются элементы, которые в последующем станут узлами дерева. Далее происходит проверка введенных элементов на корректность и построение дерева, сопровождающееся подробным выводом на экран. После построения дерева в программу подаётся номер действия: 1 - вставка элемента, 2 - исключение элемента, 3 – завершение программы. В случае введения некорректного действия – программа выводит сообщение об ошибке и предлагает ввести данные ещё раз. Когда пользователь введёт номер действия – ему необходимо будет ввести с клавиатуры элемент для работы с деревом.

Описание алгоритма вставки:

Элементы в дерево вставляются как в обычное упорядоченное дерево – справа от узла дерева – узлы с большим значением, а слева – узлы с меньшим значением.

Описание алгоритма исключения:

Происходит поиск заданного элемента, если данный элемент имеется в дереве, то он удаляется в соответствии с определёнными правилами: если у элемента нет левого сына, то правый сын становится на место родителя, аналогично с отсутствием правого сына; если же у родителя два сына, то происходит поиск наименьшего элемента в правом поддереве и он становится родителем.

Функции и структуры данных

Для выполнения лабораторной работы был написан класс BST.

`struct Node` – структура, представляющая собой узел дерева.

Содержит в себе:

`Type data` – значение узла типа `Type`;

`Node* left` – указатель на левого сына типа `Node`;

`Node* right` – указатель на правого сына типа `Node`.

`struct Trunk` – структура, представляющая собой одно ответвление от узла.

Содержит в себе:

`Trunk *prev` – указатель на предыдущую структуры типа `Trunk`;

`string branch` – строка, представляющая из себя внешний вид ответвления.

`Node* makeEmpty(Node* t)` – метод, удаляющий дерево.

Принимаемые аргументы:

`Node* t` – указатель на узел дерева.

Возвращаемое значение: 0 – в случае если узел пуст, или элемент дерева удалён.

`Node* insert (Type x, Node* t)` – метод, вставляющий элемент в дерево.

Принимаемые аргументы:

`Type x` – элемент для вставки типа `Type`;

`Node* t` – узел дерева типа `Node`.

Возвращаемое значение: ссылка на новый узел дерева.

`Node* findMin(Node* t)` – метод нахождения узла с минимальным значением.

Принимаемые аргументы:

`Node* t` – узел дерева типа `Node`.

Возвращаемое значение: ссылку на минимальный элемент дерева.

`Node* findMax(Node* t)` – метод нахождения узла с максимальным значением.

Принимаемые аргументы:

`Node* t` – узел дерева типа `Node`.

Возвращаемое значение: ссылку на максимальный элемент дерева.

`Node* remove(Type x, Node*t)` – метод удаления узла дерева.

Принимаемые аргументы:

`Type x` – элемент для удаления типа `Type`;

`Node* t` – узел дерева типа `Node`.

Возвращаемое значение: возвращает узел, вставший на место удалённого.

`void showTrunks(Trunk *p)` – вспомогательный метод для печати бинарного дерева поиска.

Принимаемые аргументы:

`Trunk *p` – указатель на структуры типа `Trunk`.

Возвращаемое значение: функция ничего не возвращает.

`void inorder(Node* t, Trunk* prev, bool isRight)` – метод, печатающий бинарное дерево.

Принимаемые аргументы:

`Node* t` – узел дерева типа `Node`.

`Trunk* prev` – указатель на структуру типа `prev`, предыдущее ответвление дерева.

`bool isRight` – булева переменная, отвечающая за то, является ли элемент правым сыном.

Возвращаемое значение: метод ничего не возвращает.

`Node* find*(Type x, Node* t)` – метод, ищущий определённый узел дерева.

Принимаемые аргументы:

Type x – элемент для поиска типа Type;

Node* t – узел дерева типа Node.

Возвращаемое значение: метод возвращает указатель на найденный элемент или 0 в случае не нахождения элемента.

BST() – конструктор, присваивающий корню дерева NULL.

~BST() – деструктор, удаляющий дерево.

void insert(Type x) – метод, осуществляющий вставку заданного элемента в дерево.

Принимаемые аргументы:

Type x – искомое значение.

Возвращаемое значение: метод ничего не возвращает.

void remove(Type x) – метод, удаляющий узел с заданным значением.

Принимаемые аргументы:

Type x – удаляемое значение.

Возвращаемое значение: метод ничего не возвращает.

void display () – метод, выводящий дерево в древовидной форме в стандартный поток вывода.

Принимаемые аргументы: ничего не принимает.

Возвращаемое значение: метод ничего не возвращает.

int search (Type x) – метод, ищущий заданный элемент в дереве.

Принимаемые аргументы:

Type x – заданный элемент.

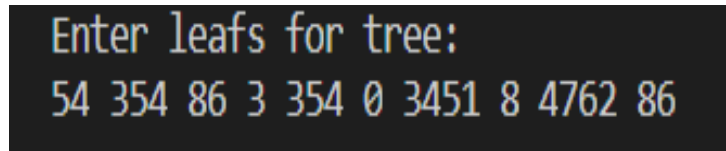
Возвращаемое значение: 1 – если элемент найден, 0 – не найден.

Пользовательский интерфейс

Работа пользователя с интерфейсом начинается с ввода элементов.

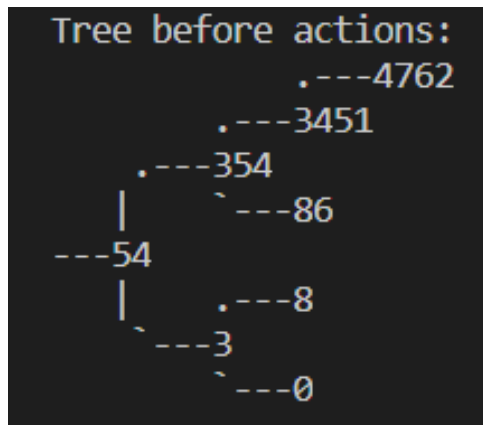
Enter leafs for tree: (ответ пользователя)

После ввода пользователем элементов произойдёт демонстрация построения дерева. Ввод элементов представлен на рисунке 1, вывод дерева представлен на рисунке 2.



```
Enter leafs for tree:
54 354 86 3 354 0 3451 8 4762 86
```

Рисунок 1- ввод элементов



```
Tree before actions:
      .---4762
       .---3451
        .---354
         |      `---86
        ---54
         |      .---8
         |      `---3
         |      `---0
```

Рисунок 2 - вывод дерева

Далее пользователю предлагается выбрать действие, которое он хочет выполнить.1

Entry rules:

- 1 - Insert element;
- 2 - Delete element;
- 3 - Exit.

Select an action for the item: (ответ пользователя: 1,2 или 3).

Enter an item to work with: (ответ пользователя)

В случае выбора вставки элемента происходит демонстрация работы функции вставки элемента, представлено на рисунке 3.

```

Enter an item to work with:
777
<----Inserting element---->
Element '777' is more than 54 , go to the right subtree.
Element '777' is more than 354 , go to the right subtree.
Element '777' is less than 3451 , go to the left subtree.

      .---4762
     .---3451
    |      `---777
   .---354
  |      `---86
---54
 |      .---8
 |      `---3
 |      `---0

```

Рисунок 3 - вставка элемента

В случае выбора исключения элемента происходит демонстрация работы функции исключения элемента, представлено на рисунке 4.

```

Enter an item to work with:
4762
<----Deleting element---->
Element '4762' is more than 54 , go to the right subtree.
Element '4762' is more than 354 , go to the right subtree.
Element '4762' is more than 3451 , go to the right subtree.
Element '4762' doesn't have left son. Putting the right son in the place of the parent.

      .---3451
     |      `---777
    .---354
   |      `---86
---54
 |      .---8
 |      `---3
 |      `---0

```

Рисунок 4 - исключение элемента

Обработка некорректных данных. При попытке ввода некорректных данных программа выводит соответствующее сообщение об ошибке. Если некорректные данные были обнаружены при вводе элементов дерева, то считывание элементов останавливается на некорректном, если же ошибка обнаружена при введении действия, то пользователю предлагается ввести номер заново, если некорректные данные обнаружены при вводе элемента для работы с деревом, то работа программы останавливается.

Тестирование

Для тестирования был использован bash-скрипт, использованный в первых 5 лабораторных работах с небольшими доработками. Данные тестирования представлены в таблице ниже. Ввиду большого объёма выводимой информации большая часть тестов искусственно урезана.

№	Входные данные	Выходные данные
1	54 354 86 3 354 0 3451 8 4762 86 1 777 3	Enter leafs for tree: 54 354 86 3 354 0 3451 8 4762 86 Tree before actions: <pre> .---4762 .---3451 .---354 `---86 ---54 .---8 `---3 `---0 </pre> Entry rules: 1 - Insert element; 2 - Delete element; 3 - Exit. Select an action for the item: 1 Enter an item to work with: 777 <----Inserting element----> Element '777' is more than 54 , go to the right subtree. Element '777' is more than 354 , go to the right subtree. Element '777' is less than 3451 , go to the left subtree.

		<pre> .---4762 .---3451 `---777 .---354 `---86 ---54 .---8 `---3 `---0 </pre> <p>Select an action for the item:</p> <p>3</p> <p>The end of program...</p>
2	23 17 41 42 77 97 2 97 3	<p>Enter leafs for tree:</p> <p>23 17 41 42 77 97</p> <p>Tree before actions:</p> <pre> .---97 .---77 .---42 .---41 ---23 `---17 </pre> <p>Entry rules:</p> <ul style="list-style-type: none"> 1 - Insert element; 2 - Delete element; 3 - Exit. <p>Select an action for the item:</p> <p>2</p> <p>Enter an item to work with:</p> <p>97</p> <p><----Deleting element----></p> <p>Element '97' is more than 23 , go to the right subtree.</p> <p>Element '97' is more than 41 , go to the right subtree.</p> <p>Element '97' is more than 42 , go to the right subtree.</p>

		<p>Element '97' is more than 77 , go to the right subtree.</p> <p>Element '97' doesn't have left son. Putting the right son in the place of the parent.</p> <pre> .---77 .---42 .---41 ---23 `---17 </pre> <p>Select an action for the item: 3 The end of program...</p>
3	13 7 -7 0 43 123 3	<p>Enter leafs for tree: 13 7 -7 0 43</p> <p>Tree before actions:</p> <pre> .---43 ---13 `---7 .---0 `----7 </pre> <p>Entry rules:</p> <ul style="list-style-type: none"> 1 - Insert element; 2 - Delete element; 3 - Exit. <p>Select an action for the item: 123 Error! Enter correct number of action. Select an action for the item: 3 The end of program...</p>
4	13 7 -7 qwe 777 3	<p>Enter leafs for tree: 13 7 -7 qwe 777</p> <p>Error! q - was not digit! Stop building the tree...</p> <p>Error! w - was not digit! Stop building the</p>

		<p>tree...</p> <p>Error! e - was not digit! Stop building the tree...</p> <p>Tree before actions:</p> <pre> ---13 `---7 `----7 </pre> <p>Entry rules:</p> <ul style="list-style-type: none"> 1 - Insert element; 2 - Delete element; 3 - Exit. <p>Select an action for the item:</p> <p>3</p> <p>The end of program...</p>
5	<p>86 654 87 43 980 32 0</p> <p>1</p> <p>qwe</p>	<p>Enter leafs for tree:</p> <p>86 654 87 43 980 32 0</p> <p>Tree before actions:</p> <pre> .---980 .---654 `---87 ---86 `---43 `---32 `---0 </pre> <p>Entry rules:</p> <ul style="list-style-type: none"> 1 - Insert element; 2 - Delete element; 3 - Exit. <p>Select an action for the item:</p> <p>1</p> <p>Enter an item to work with:</p> <p>qwe</p> <p>Error! Items for search can only be of one type(digits)!</p>

Заключение

В процессе работы были освоены и закреплены навыки работы с основными алгоритмами и структурами данных, используемыми при работе, бинарными деревьями поиска. При разрешении возникающих проблем в приоритете были поиск и использование наиболее оптимальных подходов. В процессе работы были изучены дополнительные источники, что позволило повысить знания по изучаемой и сопутствующим темам. Программа написана на языке C++ .

Приложение

Приложение А. Код основной программы.

```
#include "BST.hpp"
#include <iostream>
#include <cstring>
#include <sstream>

int menu(BST<int> &tree) {
    int value, action;
    std::cout << "Entry rules: " << std::endl;
    std::cout << "\t\t1 - Insert element;\n\t\t2 - Delete element;\n\t\t3
- Exit." << std::endl;
    while (action != 3) {
        std::cout << "Select an action for the item: " << std::endl;
        std::cin >> action;
        if(std::cin.fail()) {
            std::cout << "Error! Items for search can only be of one
type(digits)!" << std::endl;
            return 0;
        }
        switch(action) {
            case 1:
                std::cout << "Enter an item to work with:" << std::endl;
                std::cin >> value;
                if(std::cin.fail()) {
                    std::cout << "Error! Items for search can only be of
one type(digits)!" << std::endl;
                    return 0;
                }
                std::cout << "<----Inserting element---->" << std::endl;
                tree.insert(value);
                tree.display();
                break;

            case 2:
                std::cout << "Enter an item to work with:" << std::endl;
                std::cin >> value;
                if(std::cin.fail()) {
                    std::cout << "Error! Items for search can only be of
one type(digits)!" << std::endl;
                    return 0;
                }
                std::cout << "<----Deleting element---->" << std::endl;
                tree.remove(value);
                tree.display();
                break;

            case 3:
```

```

        std::cout << "The end of program..." << std::endl;
        break;

        default:
            std::cout << "Error! Enter correct number of action." <<
std::endl;
    }
}
return 0;
}

int main() {
    std::string list;
    std::cout << "Enter leafs for tree:" << std::endl;
    getline(std::cin, list);
    std::cout << std::endl;
    std::stringstream ss;
    for(size_t i = 0; i < list.size(); i++)    //Проверка элемента на
принадлежность к целому типу
        if(isalpha(list[i]))
            std::cout << "Error! " << list[i] << " - was not digit! Stop
building the tree..." << std::endl;
    ss.str(list);
    BST<int> tree;
    int value;
    std::cout << "\t\tWork with tree..." << std::endl;
    while(ss >> value) {
        tree.insert(value);
        tree.display();
    }
    std::cout << "Tree before actions:" << std::endl;
    tree.display();
    menu(tree);
}

```

Приложение Б. Код заголовочного файла БДП.

```
#ifndef _BST_HPP_
#define _BST_HPP_

#include <iostream>
#include <cstring>

template <class Type>
class BST {
    struct Node {
        Type data;
        Node* left;
        Node* right;
    };

    Node* root;

    Node* makeEmpty(Node* t) {                //Метод удаления дерева
        if(t == nullptr)
            return 0;
        makeEmpty(t->left);
        makeEmpty(t->right);
        delete t;
        return 0;
    }

    Node* insert(Type x, Node* t) {           //Метод вставки элемента в
    дерево
        if(t == nullptr) {
            t = new Node;
            t->data = x;
            t->left = t->right = 0;
            std::cout << std::endl;
        }
        else if(x < t->data) {
            std::cout << "Element '\" << x << \"' is less than " << t-
>data << " , go to the left subtree." << std::endl;
            t->left = insert(x, t->left);
        }
        else if(x > t->data) {
            std::cout << "Element '\" << x << \"' is more than " << t-
>data << " , go to the right subtree." << std::endl;
            t->right = insert(x, t->right);
        }

        return t;
    }
}
```

```

Node* findMin(Node *t) {                                     //Метод нахождения минимального
элемента в дереве
    if(!t)
        return 0;
    else if(!t->left)
        return t;
    else return findMin(t->left);
}

Node* findMax(Node* t) {                                     //Метод нахождения максимального
элемента в дереве
    if(t = nullptr)
        return 0;
    else if(!t->right)
        return t;
    else return findMax(t->right);
}

Node* remove(Type x, Node* t) {                             //Метод удаления элемента в дере-
be
    Node* temp;
    if(t == nullptr) {
        std::cout << "Element \' " << x << "\' wasn't found!" <<
std::endl;
        return 0;
    }
    else if(x < t->data) {
        std::cout << "Element \' " << x << "\' is less than " << t-
>data << " , go to the left subtree." << std::endl;
        t->left = remove(x, t->left);
    }
    else if(x > t->data) {
        std::cout << "Element \' " << x << "\' is more than " << t-
>data << " , go to the right subtree." << std::endl;
        t->right = remove(x, t->right);
    }
    else if (t->left && t->right) {
        std::cout << "Element \' " << x << "\' have two sons. Putting
the minimal elemnt in the right subtree in the place of the parent." <<
std::endl;
        temp = findMin(t->right);
        t->data = temp->data;
        t->right = remove(t->data, t->right);
    }
    else {
        temp = t;
        if(!t->left) {
            std::cout << "Element \' " << x << "\' doesn't have left
son. Putting the right son in the place of the parent." << std::endl;
            t = t->right;

```

```

        }
        else if(!t->right) {
            std::cout << "Element \'' << x << '\'' doesn't have right
son. Putting the left son in the place of the parent." << std::endl;
            t = t->left;
        }
        delete temp;
    }
    return t;
}

struct Trunk {
    Trunk *prev;
    std::string branch;
    Trunk(Trunk *prev, std::string branch) {
        this->prev = prev;
        this->branch = branch;
    }
};

void showTrunks(Trunk *p) {                                     //Вспомогательный метод
для печати ветвей бинарного дерева поиска
    if(p == nullptr)
        return;
    showTrunks(p->prev);
    std::cout << p->branch;
}

void inorder(Node* t, Trunk* prev, bool isRight) {             //Метод для
печати бинарного дерева
    if(t == nullptr)
        return;
    std::string prev_str = " ";
    Trunk *trunk = new Trunk(prev, prev_str);
    inorder(t->right, trunk, true);
    if(!prev)
        trunk->branch = "---";
    else if(isRight) {
        trunk->branch = ".---";
        prev_str = " |";
    }
    else {
        trunk->branch = "`---";
        prev->branch = prev_str;
    }
    showTrunks(trunk);
    std::cout << t->data << std::endl;

    if(prev)
        prev->branch = prev_str;

```

```

        trunk->branch = "    |";

        inorder(t->left, trunk, false);
        delete trunk;
    }

Node* find(Type x, Node* t) {           //Метод поиска элемента в дереве
    if(t = nullptr)
        return 0;
    else if(x < t->data)
        return find(x, t->left);
    else if(x > t->data)
        return find(x, t->right);
    else return t;
}

public:
    BST() {
        root = NULL;
    }

    ~BST() {
        root = makeEmpty(root);
    }

    void insert(Type x) {
        root = insert(x, root);
    }

    void remove(Type x) {
        root = remove(x, root);
    }

    void display() {                   //Метод отображения дерева
        if(!root) {
            std::cout << "BST is empty!" << std::endl;
            return;
        }
        inorder(root, nullptr, false);
        std::cout << std::endl;
    }

    int search (Type x) {             //Метод поиска элемента в дереве
        Node* temp = find(x, root);
        if(temp) {
            std::cout << "Element was found!" << std::endl;
            return 1;
        }
        else return 0;
    }
}

```

```
};  
#endif
```