Министерство науки и высшего образования РФ

ФГБОУ ВПО «Пермский государственный национальный исследовательский университет»

Кафедра прикладной математики и информатики

**Индивидуальное задание №2**

**«Динамические структуры данных»**

***по дисциплине «Алгоритмизация и программирование 2»***

|  |  |
| --- | --- |
|  | **Выполнил:** |
|  | Студент 1-го курса механико-математического ф-та  \_\_\_\_\_\_\_\_\_\_\_ Семенов С.П.,  группа ПМИ-1,2. |
|  | **Проверил:** |
|  | Доцент кафедры прикладной математики и информатики ПГНИУ, к.т.н.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Перескокова О.И. |
|  | «\_\_\_»\_\_\_\_\_\_\_\_\_ 2021 г. |

Пермь 2021

# Постановка задачи

Имеется список идентификаторов некоторой программы. Все идентификаторы хранятся в текстовом файле, по одному в строке и начинаются с первой позиции строки. Необходимо построить расширяющееся дерево поиска для данного списка идентификаторов. Программа должна выдавать полный отчет о построении дерева, т.е. распечатывать дерево так, чтобы видна была структура дерева после каждого включения и удаления вершины.

## Формат входных данных:

Все входные данные содержаться в текстовом файле input.txt, по одному в строке и начинаются с первой позиции строки.

## Формат выходных данных:

После выполнения каждой операции, в консоль выводится измененное дерево. После удаления последнего узла дерева, в консоль выводится соответствующее сообщение и программа завершает работу.

# Определение идеи алгоритма,

# выбор методов решения и структур данных

# Для решения задачи была выбрана система программирования Microsoft VS.

Узел дерева описывается как структура TNode

struct TNode {

string Data; - Поле данных

TNode\* Left; - Указатель на левого потомка

TNode\* Right; - Указатель на правого потомка

TNode\* Parent; - Указатель на родителя

}; typedef TNode\* PNode;

**Идея алгоритма:**

Функция Splay состоит из набора операций *Zig, Zig-Zig и Zig-Zag*, выполняющихся в зависимости от текущего расположения узла дерева, до тех пор, пока рассматриваемый нами узел не станет корнем дерева.

*Zig* выполняется в том случае, когда узел является правым потомком корня дерева (Выполняем левый поворот от узла. *Zag* работает аналогичным образом).

*Zig-Zig* выполняется в том случае, когда узел является правым потомком правого потомка прародителя рассматриваемого узла. (Выполняем сначала правый поворот от родителя узла, после выполняем правый поворот от узла. *Zag-Zag* выполняется аналогично).

*Zig-Zag* выполняется в том случае, когда узел является правым потомком левого потомка прародителя рассматриваемого узла (Выполняем сначала левый поворот от узла, а затем правый поворот. *Zag-Zig* выполняется аналогично).

Inset выполняется в точности как у обычного bst-дерева, но в конце функции вызываем дополнительно Splay, тем самым, перемещая добавленный узел в корень дерева.

Find так же выполняется как и в обычном bst-дереве, с перемещением найденного узла в корень дерева. Но если корень дерева с нужным значением в дереве найден не был, то Splay нужно вызвать от родителя, чей правый или левый потомок во время поиска оказался nullptr,

Delete выполняется следующим образом:

1. Выполняем Splay от узла, который необходимо удалить.
2. Удаляем выбранный узел, разделяя тем самым дерево на 2 поддерева.
3. Находим узел с максимальным значением в левом поддереве и делаем его корнем левого поддерева.
4. Правое дерево делаем правым потомком у корня левого поддерева. Тем самым у нас получается одно дерево, без удаленного узла
5. В случае, когда у удаляемого узла после выполнения Splay нет правого или левого потомка, то делаем корнем нового дерева левый или правый потомок удаляемого узла. Сам узел так же удаляется.

# Тестирование программы

Все тесты будут сверяться с визуализатором Splay-Tree. Указанные входные данные будут последовательно вводиться в визуализатор. Так же и функции, описанные в тестах, будут последовательно вводиться в визуализатор. В программу исходные данные будут приходить из файла, а функции вводиться последовательно через консоль.

Тест 1.

Входные данные :

ONE

TWO

THREE

FIVE

FG7

HH

BBC

KLM

ABC

Выполняемые операции:

Delete(FIVE)

Delete(HH)

Find(ONE)

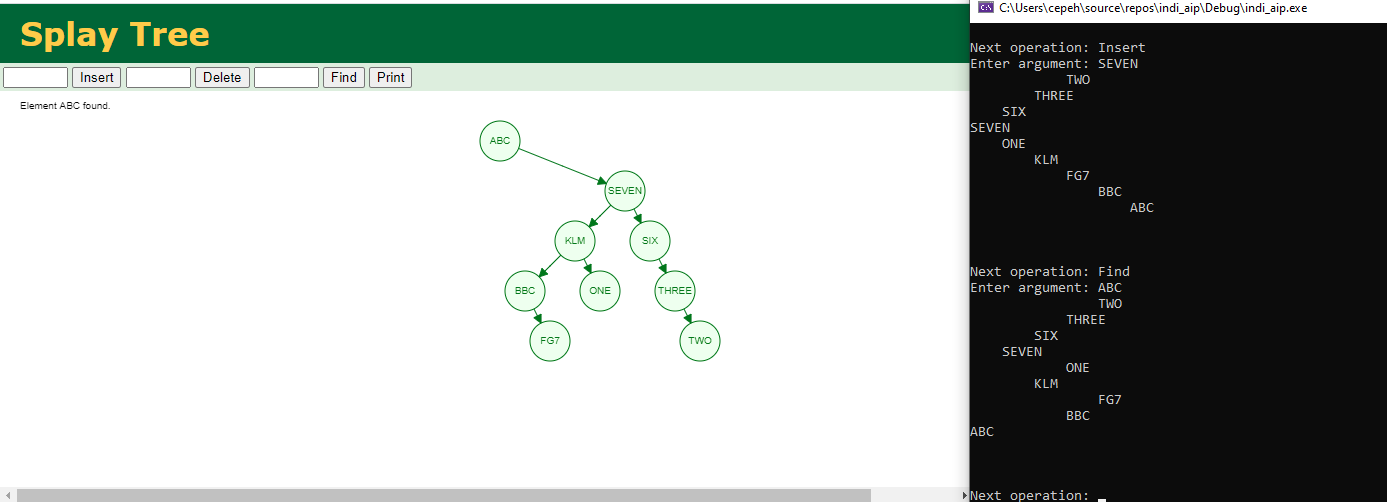
Find(SIX)

Insert(SIX)

Insert(SEVEN)

Find(ABC)

Получившееся дерево:



Тест 2.

Входные данные :

AFD

GHJ

TTE

ANNA

ALLA

NIK

MAR

FILL

ZERO

A

B

C

D

E

F

G

CAT

DOG

KIR

GOLD

SEL

POIN

FR

Выполняемые операции:

Delete(DOG)

Delete(CAT)

Delete(A)

Delete(B)

Delete(G)

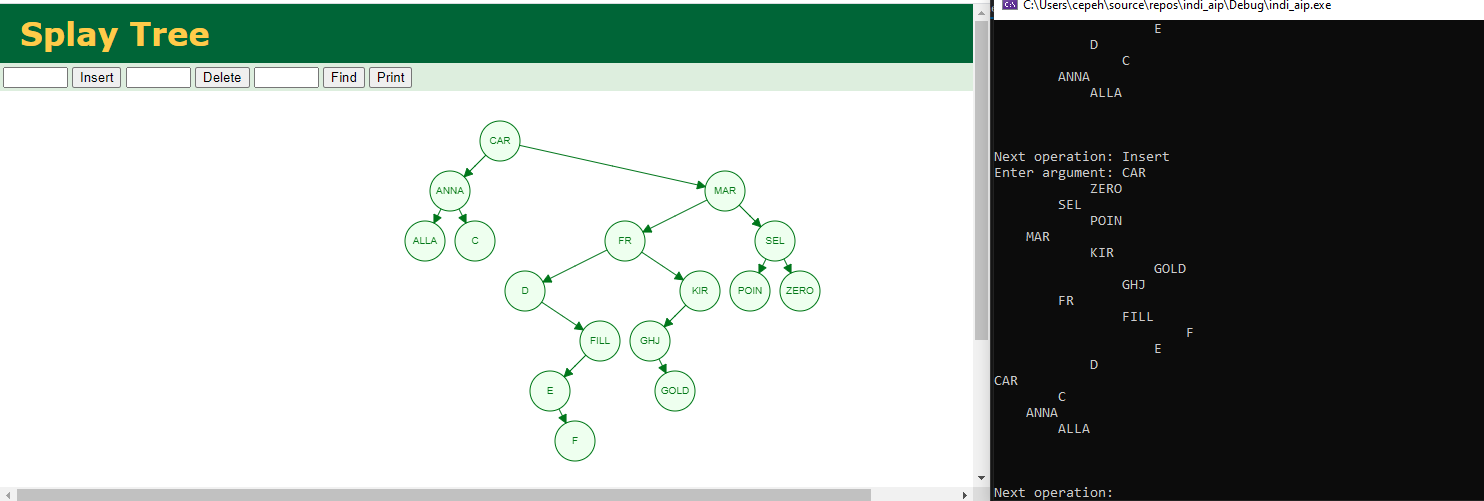
Delete(TTE)

Delete(AFD)

Delete(NIK)

Insert(CAR)

Получившееся дерево:



Тест 3.

Входные данные :

ALL

OIL

CAR

BOY

KID

MAN

Выполняемые операции:

Delete(ALL)

Delete(MAN)

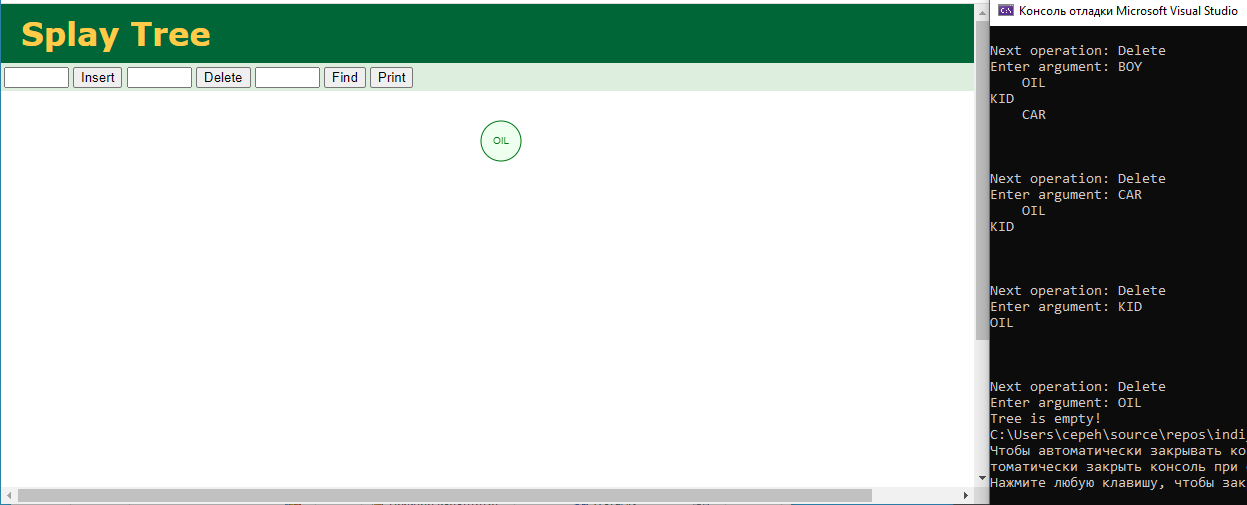
Delete(BOY)

Delete(CAR)

Delete(KID)

Delete(OIL)

Получившееся дерево:



Программа удаляет последний узел и завершает свою работу с соответствующим сообщением (Визуализатор не удаляет последний узел)

# Исходный код

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct TNode {

string Data;

TNode\* Left;

TNode\* Right;

TNode\* Parent;

}; typedef TNode\* PNode;

//Печать дерева

void PrintTree(PNode root, int lvl) {

if (!root) return;

PrintTree(root->Right, lvl + 1);

for (int i = 0; i < lvl; i++)

cout << " ";

cout << root->Data << endl;

PrintTree(root->Left, lvl + 1);

}

//Правый поворот

void RightRotation(PNode& root, PNode ptr) {

PNode rightChild = ptr->Right;

PNode father = ptr->Parent;

if (ptr->Parent->Parent != nullptr) {

PNode grandFather = father->Parent;

if (grandFather->Right != nullptr) {

if (ptr == grandFather->Right->Left)

grandFather->Right = ptr;

else grandFather->Left = ptr;

ptr->Parent = grandFather;

}

else {

grandFather->Left = ptr;

ptr->Parent = grandFather;

}

}

else ptr->Parent = nullptr;

ptr->Right = father;

father->Parent = ptr;

father->Left = rightChild;

if (rightChild != nullptr)

rightChild->Parent = father;

while (root->Parent != nullptr)

root = root->Parent;

}

//Левый поворот

void LeftRotation(PNode& root, PNode ptr) {

PNode leftChild = ptr->Left;

PNode father = ptr->Parent;

if (ptr->Parent->Parent != nullptr) {

PNode grandFather = father->Parent;

if (grandFather->Left != nullptr) {

if (ptr == grandFather->Left->Right)

grandFather->Left = ptr;

else grandFather->Right = ptr;

ptr->Parent = grandFather;

}

else {

grandFather->Right = ptr;

ptr->Parent = grandFather;

}

}

else ptr->Parent = nullptr;

ptr->Left = father;

father->Parent = ptr;

father->Right = leftChild;

if (leftChild != nullptr)

leftChild->Parent = father;

while (root->Parent != nullptr)

root = root->Parent;

}

//Функция splaying

void Splay(PNode& root, PNode ptr) {

while (ptr->Parent != nullptr) {

//zig & zag

if (ptr->Parent->Parent == nullptr) {

if (ptr == ptr->Parent->Right)

LeftRotation(root, ptr);

else RightRotation(root, ptr);

continue;

}

//zig-zig & zag-zig

if (ptr->Parent->Parent->Right != nullptr) {

if (ptr == ptr->Parent->Parent->Right->Right) {

LeftRotation(root, ptr->Parent);

LeftRotation(root, ptr);

continue;

}

else

if (ptr == ptr->Parent->Parent->Right->Left) {

RightRotation(root, ptr);

LeftRotation(root, ptr);

continue;

}

}

//zag-zag & zig-zag

if (ptr->Parent->Parent->Left != nullptr) {

if (ptr == ptr->Parent->Parent->Left->Left) {

RightRotation(root, ptr->Parent);

RightRotation(root, ptr);

}

else

if (ptr == ptr->Parent->Parent->Left->Right) {

LeftRotation(root, ptr);

RightRotation(root, ptr);

}

}

}

root = ptr;

}

//Вставка нового элемента

void InsertInTree(PNode& root, PNode& newElem, PNode parent) {

if (!root) {

newElem->Left = nullptr;

newElem->Right = nullptr;

if (parent == nullptr) newElem->Parent = nullptr;

else newElem->Parent = parent;

root = newElem;

return;

}

if (newElem->Data < root->Data)

InsertInTree(root->Left, newElem, root);

else if (newElem->Data > root->Data)

InsertInTree(root->Right, newElem, root);

}

//Вставка с последущим расширением

void Insert(PNode& root, string D) {

PNode newElem = new TNode;

newElem->Data = D;

InsertInTree(root, newElem, nullptr);

Splay(root, newElem);

PrintTree(root, 0);

cout << endl << endl << endl;

}

//Создать дерево из идентификаторов файла input

void CreateTree(PNode& root) {

ifstream input("input.txt");

string x;

while (input >> x)

Insert(root, x);

}

//Поиск элемента в дереве

PNode FindElement(PNode root, string x, PNode& prev) {

PNode ptr = root;

while (ptr) {

if (x == ptr->Data) return ptr;

if (x < ptr->Data) {

prev = ptr;

ptr = ptr->Left;

}

else if (x > ptr->Data) {

prev = ptr;

ptr = ptr->Right;

}

}

return nullptr;

}

//Поиск с расширением

void Find(PNode& root, string x) {

PNode prev = nullptr;

PNode ptr = FindElement(root, x, prev);

if (ptr == nullptr) {

cout << "Element " << x << " not found. Splaying parent of return ptr" << endl;

Splay(root, prev);

PrintTree(root, 0);

cout << endl << endl << endl;

return;

}

Splay(root, ptr);

PrintTree(root, 0);

cout << endl << endl << endl;

}

//Если у удаляемой вершины после операции splay нет

//правого поддерева, то оставляем дерево начиная с левого поддерева

void MakeLeftChildTreeRoot(PNode& root, PNode delElem) {

root = delElem->Left;

root->Parent = nullptr;

delete delElem;

PrintTree(root, 0);

cout << endl << endl << endl;

}

//Если у удаляемой вершины после операции splay нет

//левого поддерева, то оставляем дерево начиная с правого поддерева

void MakeRightChildTreeRoot(PNode& root, PNode delElem) {

root = delElem->Right;

root->Parent = nullptr;

delete delElem;

PrintTree(root, 0);

cout << endl << endl << endl;

}

//Удаление

void Delete(PNode& root, string x) {

PNode prev = nullptr;

PNode delElem = FindElement(root, x, prev);

if (delElem == nullptr) {

cout << "Element " << x << " not found. Splaying parent of return ptr" << endl;

Splay(root, prev);

PrintTree(root, 0);

cout << endl << endl << endl;

return;

}

Splay(root, delElem);

if (delElem->Right == nullptr && delElem->Left == nullptr) {

delete delElem;

cout << "Tree is empty!";

exit(1);

}

if (delElem->Right == nullptr) {

root = delElem;

MakeLeftChildTreeRoot(root, delElem);

}

else if (delElem->Left == nullptr) {

root = delElem;

MakeRightChildTreeRoot(root, delElem);

}

else {

PNode leftRoot = delElem->Left;

PNode rightRoot = delElem->Right;

leftRoot->Parent = nullptr;

rightRoot->Parent = nullptr;

delete delElem;

PNode largeLeft = leftRoot;

while (largeLeft->Right != nullptr)

largeLeft = largeLeft->Right;

Splay(leftRoot, largeLeft);

largeLeft->Right = rightRoot;

rightRoot->Parent = largeLeft;

largeLeft->Parent = nullptr;

root = largeLeft;

PrintTree(root, 0);

cout << endl << endl << endl;

}

}

int main() {

PNode root = nullptr;

CreateTree(root);

string function, argument;

cout << "Enter names of operations (Insert, Find, Delete). For stop enter Stop: ";

cin >> function;

while (function != "Stop") {

if (!(function == "Insert" || function == "Find" || function == "Delete")) {

cout << "Operation not found. Enter next operation" << endl;

cin >> function;

continue;

}

cout << "Enter argument: ";

cin >> argument;

if (function == "Insert") {

Insert(root, argument);

}

else if (function == "Find") {

Find(root, argument);

}

else if (function == "Delete") {

Delete(root, argument);

}

cout << "Next operation: ";

cin >> function;

}

}