Санкт-Петербургский государственный политехнический университет

Кафедра “Прикладная математика”

**Отчет по лабораторной работе 5**

**“Алгоритмы и структуры данных”**

Студент группы № 5030102/20001

ФИО: Тишковец Сергей Евгеньевич

Выполнил (дата) 14.12.2023

Оглавление

[**Постановка задачи 3**](#_Toc153420310)

[**Описание алгоритма 3**](#_Toc153420311)

[**Текст программы 5**](#_Toc153420312)

[**Описание тестирования 10**](#_Toc153420313)

# **Постановка задачи**

*Вариант 9:*

Для дерева двоичного поиска с дополнительной информацией реализовать процедуру, которая находит K-й наименьший элемент и, если он оказывается четным, печатает элементы с порядковыми номерами меньшими K.

# **Описание алгоритма**

Структура tree\_t:

Эта структура представляет собой узел в бинарном дереве. Он содержит такие поля, как:

* целочисленное значение
* порядковый номер
* ссылка на левый дочерний элемент
* ссылка на родительский узел
* ссылка на правый дочерний элемент

Структура way\_t:

Содержит в себе 3 значения, которые будут использоваться для указания направления обхода в дереве.

Функция для присвоения значений порядка каждому узлу в дереве:

Рекурсивно пересекается бинарное дерево по порядку обновляя порядок каждого узла.

Функция для добавления узла в бинарное дерево:

1. Выделяется память для нового узла, идентифицируются его элементы.
2. Новый узел вставляется его в соответствующую позицию в дереве на основе его значения.
3. Если узел с таким значением уже существует, функция возвращает результат без внесения каких-либо изменений.

Функция для уничтожения бинарного дерева:

Рекурсивно уничтожается бинарное дерево, освобождается память, выделенная для каждого узла.

Функция для поиска узла в дереве с заданным порядковым значением:

1. Рекурсивно обходится всё бинарное дерево.
2. Если соответствующий узел найден, возвращается указатель на этот узел – в противном случае возвращается значение NULL.

Функция для поиска узла с заданным значением в бинарном дереве:

1. Итеративно обходится всё дерево и сравнивается целевое значение со значениями в каждом узле до тех пор, пока не будет найдено совпадение.
2. Если соответствующий узел найден, возвращается указатель на этот узел – в противном случае возвращается значение NULL.

Функция для удаления узла из бинарного дерева:

Обрабатывается 3 случая:

1. Если у узла нет дочерних элементов, то просто освобождается память для узла.
2. Если у узла есть один дочерний элемент, то узел заменяется своим дочерним элементом.
3. Если у узла есть два дочерних элемента, то находится максимальный узел в правом поддереве, затем он меняется местами с удаляемым узлом, после чего максимальный узел удаляется.

Функция для печати бинарного дерева:

Рекурсивно обходится всё дерево и печатаются значения каждого из узлов, исключая узлы со значениями порядка, большими или равными К.

# **Текст программы**

#pragma warning(disable:4996)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

typedef struct tree\_t {

int num;

int order; //order from the smallest number in tree to the biggest

struct tree\_t\* left;

struct tree\_t\* parent;

struct tree\_t\* right;

} tree\_t;

typedef enum {

NONE = 0,

LEFT = 1,

RIGHT = 2

}way\_t;

//will place order in appending function

void UpdateOrder(tree\_t\* tree, int\* order) {

if (tree) {

if (tree->left)

UpdateOrder(tree->left, order);

tree->order = \*order;

(\*order)++;

if (tree->right)

UpdateOrder(tree->right, order);

}

}

void NodeAdd(tree\_t\*\* tree, int num) {

tree\_t\* temp = (\*tree), \* newnode, \* ptr = (\*tree);

int order = 1;

newnode = (tree\_t\*)malloc(sizeof(tree\_t));

if (!newnode)

return;

newnode->left = NULL;

newnode->right = NULL;

newnode->parent = NULL;

newnode->num = num;

if (!(\*tree)) {

(\*tree) = newnode;

UpdateOrder((\*tree), &order);

return;

}

while (ptr) {

if (num < ptr->num) {

if (ptr->left)

ptr = ptr->left;

else

break;

}

else if (num > ptr->num) {

if (ptr->right)

ptr = ptr->right;

else

break;

}

else { //such node exists

free(newnode);

return;

}

}

//put new node

if (num < ptr->num)

ptr->left = newnode;

else

ptr->right = newnode;

newnode->parent = ptr;

UpdateOrder((\*tree), &order);

}

void DestroyTree(tree\_t\* tree) {

if (tree) {

DestroyTree(tree->left);

DestroyTree(tree->right);

free(tree);

}

}

tree\_t\* KLowerNodeFind(tree\_t\* tree, int k) {

if (tree) {

if (tree->order == k)

return tree;

if (KLowerNodeFind(tree->left, k))

return tree->left;

if (KLowerNodeFind(tree->right, k))

return tree->right;

}

return NULL;

}

tree\_t\* NodeFind(tree\_t\* tree, int num) {

tree\_t\* temp = tree;

while (temp) {

if (num < temp->num)

temp = temp->left;

else if (num > temp->num)

temp = temp->right;

else// found

return temp;

}

return NULL;

}

void NodeDelete(tree\_t\*\* tree, tree\_t\* vertex) {

if (!vertex)

return;

int temp, order = 1;

tree\_t\* max = vertex, \* parent, \* child; //go to left child of vertex and then to right

way\_t parentway;

//no childs, free the element

if (vertex->left == NULL && vertex->right == NULL) {

if ((vertex == (\*tree))) {

free(vertex);

(\*tree) = NULL;

return;

}

parent = vertex->parent;

parentway = parent->left == vertex ? LEFT : RIGHT;

if (parentway == LEFT)

parent->left = NULL;

else

parent->right = NULL;

free(vertex);

UpdateOrder((\*tree), &order);

return;

}

//has one child, put child instead of itself

if ((vertex->left != NULL && vertex->right == NULL) || (vertex->left == NULL && vertex->right != NULL)) {

if (vertex == (\*tree)) { //no parent

if (vertex->left)

(\*tree) = vertex->left;

else

(\*tree) = vertex->right;

(\*tree)->parent = NULL;

free(vertex);

UpdateOrder((\*tree), &order);

return;

}

parent = vertex->parent;

parentway = parent->left == vertex ? LEFT : RIGHT;

if (vertex->left) //only left child

child = vertex->left;

else //only right child

child = vertex->right;

child->parent = vertex->parent;

if (parentway == LEFT)

parent->left = child;

else

parent->right = child;

free(vertex);

UpdateOrder((\*tree), &order);

return;

}

//node has two childs

max = max->right;

while (max->left)

max = max->left;

temp = max->num;

max->num = vertex->num;

vertex->num = temp;

//no need to delete max

//max can have one child on the right

if (max->right) {

child = max->right;

parent = max->parent;

parentway = parent->left == max ? LEFT : RIGHT;

if (parentway == LEFT)

parent->left = child;

else

parent->right = child;

child->parent = parent;

}

//max has no childs

else {

parent = max->parent;

parentway = parent->left == max ? LEFT : RIGHT;

if (parentway == LEFT)

parent->left = NULL;

else

parent->right = NULL;

}

free(max);

UpdateOrder((\*tree), &order);

return;

}

//printing tree rotated on 90 degrees without vertexes which order >= k

void PrintLowerKTree(tree\_t\* tree, int level, int k) {

if (tree) {

PrintLowerKTree(tree->left, level + 1, k);

if (tree->order < k) {

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

printf(" ");

printf("%d\n", tree->num);

}

PrintLowerKTree(tree->right, level + 1, k);

}

}

int main(void) {

const int k = 5; //order

tree\_t\* tree = NULL, \*find = NULL;

NodeAdd(&tree, -6);

NodeAdd(&tree, 12);

NodeAdd(&tree, 7);

NodeAdd(&tree, -1);

NodeAdd(&tree, 5);

NodeAdd(&tree, 2);

NodeAdd(&tree, -9);

NodeDelete(&tree, NodeFind(tree, 2));

find = KLowerNodeFind(tree, k);

if (find && (find->num) % 2 == 0)

PrintLowerKTree(tree, 1, k);

else printf("Not today");

DestroyTree(tree);

return 0;

}

# **Описание тестирования**

Для тестирования данной программы производился ее неоднократный запуск с вводом конкретных данных, при котором проверялось:

1. Стабильность работы программы при одинаковых входных данных;
2. Корректное выполнение всех заявленных процедур;
3. Корректное завершение программы;

Приведем пример тестирования на скриншотах, представленных ниже:

const int k = 5;

NodeAdd(&tree, -6);

NodeAdd(&tree, 12);

NodeAdd(&tree, 7);

NodeAdd(&tree, -1);

NodeAdd(&tree, 5);

NodeAdd(&tree, 2);

NodeAdd(&tree, -9);

NodeDelete(&tree, NodeFind(tree, 2));

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

При проведении тестирования такого рода никаких проблем обнаружено не было, что позволяет судить о корректности работы программы в целом.