Санкт-Петербургский государственный политехнический университет

Кафедра “Прикладная математика”

**Отчет по лабораторной работе 4**

**“Алгоритмы и структуры данных”**

Студент группы № 5030102/20001

ФИО: Тишковец Сергей Евгеньевич

Выполнил (дата) 30.11.2023

Оглавление

[**Постановка задачи 3**](#_Toc149322136)

[**Описание алгоритма 3**](#_Toc149322137)

[**Текст программы 5**](#_Toc149322138)

[**Описание тестирования 11**](#_Toc149322139)

# **Постановка задачи**

*Вариант 4:*

Задано бинарное дерево, содержащее слова разной длины. Назовем шириной узла число символов, требуемое для печати данных этого узла. Записать в узлах дерева ширину соответствующих поддеревьев. При печати вычисленные данные печатать под словами.

# 

# **Описание алгоритма**

Структура tree\_t:

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

* длина слова
* ссылка на левый дочерний элемент
* ссылка на родительский узел
* ссылка на правый дочерний элемент
* сумму длин слов всех узлов поддерева с корнем в этом узле
* фактическое слово, хранящееся в узле.

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 длины слова.
3. Обновляется длина поддерева всех узлов на пути от нового узла к корневому.

Функция для уничтожения бинарного дерева:

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

Функция для поиска узла с заданным словом в бинарном дереве:

1. Сравниваются длина слова и символы слова, чтобы найти соответствующий узел.
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 нужном поддереве, меняются местами слово и длина с удаляемым узлом, а затем удаляется максимальный узел.

Функция для печати бинарного дерева:

Рекурсивно обходится всё дерево и печатается слово, под ним – его длина и длина поддерева каждого узла.

# **Текст программы**

#pragma warning(disable:4996)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define max(a, b) ((a > b) ? a : b)

typedef struct tree\_t {

int length; //length of word

struct tree\_t\* left;

struct tree\_t\* parent;

struct tree\_t\* right;

int subtreeWigth; //sum of word lengths of sons, grandsons, etc.

char\* word;

} tree\_t;

typedef enum {

NONE = 0,

LEFT = 1,

RIGHT = 2

}way\_t;

void NodeAdd(tree\_t\*\* tree, char\* buff) {

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

int length = strlen(buff);

int i, subtreelength;

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

if (!newnode)

return;

newnode->left = NULL;

newnode->right = NULL;

newnode->parent = NULL;

newnode->length = length;

newnode->subtreeWigth = 0;

newnode->length = length;

newnode->word = (char\*)malloc((length + 1) \* sizeof(char));

if (!newnode->word)

return;

strcpy(newnode->word, buff);

if (!(\*tree)) {

(\*tree) = newnode;

return;

}

while (ptr) {

if (length < ptr->length) {

if (ptr->left)

ptr = ptr->left;

else

break;

}

else if (length > ptr->length) {

if (ptr->right)

ptr = ptr->right;

else

break;

}

else { //node with such length exists

free(newnode);

return;

}

}

//put new node

if (length < ptr->length)

ptr->left = newnode;

else

ptr->right = newnode;

newnode->parent = ptr;

//update subtreelengths

subtreelength = 0;

temp = newnode;

while (temp->parent) {

temp = temp->parent;

temp->subtreeWigth += length;

}

newnode->subtreeWigth += length;

}

void DestroyTree(tree\_t\* tree) {

if (tree) {

DestroyTree(tree->left);

DestroyTree(tree->right);

free(tree->word);

free(tree);

}

}

tree\_t\* NodeFind(tree\_t\* tree, char\* buff) {

tree\_t\* temp = tree;

int length = strlen(buff);

while (temp) {

if (length < temp->length)

temp = temp->left;

else if (length > temp->length)

temp = temp->right;

//check if words are equal

else {

if (strcmp(temp->word, buff) != 0)

return NULL;

return temp; //equal

}

}

return NULL;

}

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

if (!vertex)

return;

int temp, i;

tree\_t\* maximum = vertex, \*parent, \*child, \*temp2; //go to left child of vertex and then to right

way\_t parentway;

char\* help;

//no childs, free the element

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

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

free(vertex->word);

free(vertex);

(\*tree) = NULL;

return;

}

parent = vertex->parent;

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

if (parentway == LEFT)

parent->left = NULL;

else

parent->right = NULL;

//update subtreewidth

temp2 = vertex;

while (temp2->parent) {

temp2 = temp2->parent;

temp2->subtreeWigth -= vertex->length;

}

free(vertex->word);

free(vertex);

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);

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;

//update subtreewidth

temp2 = vertex;

while (temp2->parent) {

temp2 = temp2->parent;

temp2->subtreeWigth -= vertex->length;

}

free(vertex->word);

free(vertex);

return;

}

//node has two childs

maximum = maximum->right;

while (maximum->left)

maximum = maximum->left;

temp = maximum->length;

maximum->length = vertex->length;

vertex->length = temp;

help = (char\*)realloc(vertex->word, (max(vertex->length, maximum->length) + 1) \* sizeof(char)); //need more memory

if (help) {

vertex->word = help;

help = NULL;

}

else {

free(vertex->word);

return;

}

for (i = 0; i <= max(vertex->length, maximum->length); ++i)

vertex->word[i] = maximum->word[i];

temp2 = maximum;

while (temp2->parent != vertex->parent) {

temp2 = temp2->parent;

temp2->subtreeWigth += (maximum->length - vertex->length);

}

//and now need to delete max

//max can have one child on the right

if (maximum->right) {

child = maximum->right;

parent = maximum->parent;

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

if (parentway == LEFT)

parent->left = child;

else

parent->right = child;

child->parent = parent;

}

//max has no childs

else {

parent = maximum->parent;

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

if (parentway == LEFT)

parent->left = NULL;

else

parent->right = NULL;

}

//update subtreewidth

temp2 = maximum;

while (temp2->parent) {

temp2 = temp2->parent;

temp2->subtreeWigth -= maximum->length;

}

free(maximum->word);

free(maximum);

return;

}

/\* printing tree rotated on 90 degrees. Root of tree on the left, leaves on the right

structure of printing:

1. word

2. length of word

3. length of subtree

\*/

void PrintTree(tree\_t\* tree, int level) {

if (tree) {

int i;

PrintTree(tree->left, level + (tree->length)); //tree->length - length of word

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

printf(" ");

for (i = 0; i < tree->length; ++i)

printf("%c", tree->word[i]);

printf("\n");

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

printf(" ");

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

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

printf(" ");

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

PrintTree(tree->right, level + (tree->length));

}

}

int main(void) {

tree\_t\* tree = NULL;

NodeAdd(&tree, "MERCEDES");

NodeAdd(&tree, "BMW");

NodeAdd(&tree, "AUDI");

NodeAdd(&tree, "VOLKSWAGEN");

NodeAdd(&tree, "RENAULT");

NodeAdd(&tree, "TOYOTA");

NodeAdd(&tree, "HONDA");

NodeAdd(&tree, "LAMBORGHINI");

NodeDelete(&tree, NodeFind(tree, "RENAULT"));

PrintTree(tree, 1);

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:

NodeAdd(&tree, "MERCEDES");

NodeAdd(&tree, "BMW");

NodeAdd(&tree, "AUDI");

NodeAdd(&tree, "VOLKSWAGEN");

NodeAdd(&tree, "RENAULT");

NodeAdd(&tree, "TOYOTA");

NodeAdd(&tree, "HONDA");

NodeAdd(&tree, "LAMBORGHINI");

NodeDelete(&tree, NodeFind(tree, "RENAULT"));

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

Автоматически созданное описание

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