Отчет по лабораторной работе № 23 по курсу “Фундаментальная информатика”

Студент группы М80-103Б-21 Катин Иван Вячеславович, № по списку 12

Контакты e-mail: ikatin.2003.sokol@gmail.com, telegram: @Dazz1e

Работа выполнена: «16» февраля 2022г.

Преподаватель: каф. 806 Севастьянов Виктор Сергеевич

Отчет сдан « » _____20__ г., итоговая оценка _____

Подпись преподавателя_____

Тема: Динамическое структуры данных. Обработка деревьев.

1. **Цель работы:** Научиться реализовать деревья, выделять и очищать память.
2. **Задание:** 13. Проверить является ли дерево линейным списком вершин.

3. **Оборудование** (студента):
   Процессор *Intel® Core™ i5-9300H CPU @ 2.40GHz × 8* с ОП 7,6 GiB, НМД *1024* Гб. Монитор *1920x1080*

4. **Программное обеспечение (**студента**):**
   Операционная система семейства: *linux*, наименование: *ubuntu*, версия *20.04.3 LTS*
   интерпретатор команд: *bash* версия *4.4.20(1)-release*.
   Система программирования -- CLion--**,** редактор текстов *emacs* версия *25.2.2*
   Утилиты операционной системы --
   Прикладные системы и программы – **LibreOffice**
   Местонахождение и имена файлов программ и данных на домашнем компьютере – *home*/dazzle

6. **Идея, метод, алгоритм.** Реализовать дерево через структуру, в структуре будет массив указателей на поддеревья, номер вершины, указатель на родителя, количество поддеревьев, размер массива.

7. **Сценарий выполнения работы**

1.Создание структуры.

2.Реализация функций:

- добавление вершины
- удаление вершины
- вывод дерева
- проверка дерева на линейный список вершин

8. Распечатка протокола (подклеить листинг окончательного варианта программы с тестовыми примерами,подписанныйпреподавателем).

```c
#include <stdio.h>
#include <malloc.h>
#include <stdbool.h>
#include <string.h>


typedef struct _node{
    struct _node* parent;
    int countChild;
    int maxCount;
    int val;
    struct _node** child;
} node;

node* head;

node* createNode(node* parent, int children) {
    node *n = malloc(sizeof(node));
    n->parent = parent;
    n->countChild=0;
    n->maxCount=10;
    n->val = children;
    n->child = malloc(sizeof(node)* 10);
    if(parent->countChild == parent->maxCount){
        parent->child = realloc(parent->child, sizeof(node) * parent->maxCount *
2);
        parent->maxCount = parent->maxCount * 2;
        parent->child[parent->countChild] = n;
        parent->countChild++;
    } else {
        parent->child[parent->countChild] = n;
        parent->countChild++;
    }
    return n;
}

node* create_tree(int c){
    node* n = malloc(sizeof(node));
    n->maxCount=10;
    n->parent = NULL;
    n->countChild = 0;
    n->child = malloc(sizeof(node)* 10);
    n->val = c;
    return n;
}

node* poisk(node* u, int v){
    if(u-> val == v) return u;
    node* temp = NULL;
    for(int i =0; i < u->countChild; i++){
        temp = poisk( u->child[i],v);
        if(temp != NULL && temp->val == v) break;
    }
    return temp;
}
```

```
void deleteNode(node* temp, bool fl){
    if(temp->parent != NULL && fl) {
        node **n = malloc(temp->parent->maxCount * sizeof(node));
        int j = 0;
        for (int i = 0; i < temp->parent->countChild; i++) {
            if (temp->parent->child[i]->val != temp->val) {
                n[j] = temp->parent->child[i];
                j++;
            }
            fl = false;
        }
        temp->parent->countChild--;
        free(temp->parent->child);
        temp->parent->child = n;
    } else if(fl && temp->parent == NULL){
        head = NULL;
        fl = false;
    }
    int raz = temp->countChild;
    for(int i = 0; i < raz; i++){
        deleteNode( temp->child[i],fl);
    }
    if(temp->countChild ==0) {
        free (temp->child);
        free(temp);
    }
}

int perevod(char*s){
    int res =0;
    for(int i =0; i < strlen(s); i++){
        if(s[i] - '0' >= 0 && s[i] - '0' <= 9) {
            res = res * 10 + s[i] - '0';
        } else {
            return -1;
        }
    }
    return res;
}

int dfs(node* n, int deep){
    if(n->countChild == 0) return deep;
    for(int i =0; i < n->countChild; i++){
        if(n->child != NULL) deep = dfs(n->child[i], deep+1);
    }
    return deep;
}

void otctup(int deep){
    for (int i = 0; i < deep; i++) {
        if(i == deep -1 && (deep == 3 || deep == 4)) printf("-|");
        else if(i == deep -1 && deep > 2) {
            for(int j =0; j < deep - 1; j++) {
                printf("-");
            }
            printf("|");
```

```
        }
        else if(i == deep -1 && deep != 1) printf("|");
        printf("-----");
    }
}

void printTree(node* n, int deep){
    printf("|");
    otctup(deep);
    printf("%d",n->val);
    printf("\n");
    for(int i = 0; i < n->countChild; ++i) {
        printTree(n->child[i], deep+1);
    }
}

bool list(node* curNode){
    bool res= (curNode->countChild == 1 || curNode->countChild == 0);
    if(curNode->countChild > 0) res = res & list(curNode->child[0]);
    return res;
}

int main() {
    bool first = true;
    char str[255];
    int u, v;
    while(true){
        printf("Введите 1 для добавление нового узла\n");
        printf("Введите 2 для вывода дерева\n");
        printf("Введите 3 для удаление узла\n");
        printf("Введите 4 для того, чтобы проверить является ли дерево линейным
списком вершин\n");
        scanf("%s", str);
        if(str == EOF) break;
        if(!strcmp(str,"1")){
            printf("Первое число - родитель, второе - потомок\n");
            scanf("%s", str);
            if((u = perevod(str)) == -1){
                printf("Ожидалось число...\n");
                continue;
            }
            scanf("%s", str);
            if((v = perevod(str)) == -1){
                printf("Ожидалось число...\n");
                continue;
            }
            if(first) {
                head = create_tree(u);
                first = false;
            }
            node* temp = poisk(head,u);
            if(temp == NULL){
                printf("Родитель не существует\n");
                continue;
            } else {
                node* proof = poisk(head,v);
                if(proof == NULL) {
                    createNode(temp, v);
```

```
                } else {
                    printf("Вершина уже существует\n");
                }
            }
        } else if(!strcmp(str,"2")){
            if(first){
                printf("У деререва нет вершин\n");
                continue;
            }
            printf("\n");
            printTree(head,0);
            printf("\n");
        } else if(!strcmp(str,"3")){
            if(first){
                printf("У деререва нет вершин\n");
                continue;
            }
            scanf("%s", str);
            if((v = perevod(str)) == -1){
                printf("Ожидалось число...\n");
                continue;
            } else {
                if(head->val == v){
                    first = true;
                }
                node* temp = poisk(head, v);
                if(temp == NULL){
                    printf("Вершина не существует\n");
                    continue;
                } else {
                    deleteNode(temp,true);
                }
            }
        } else if(!strcmp(str,"4")){
            if(first){
                printf("У деререва нет вершин\n");
            } else {
                if(list(head)) printf("true\n");
                else printf("false\n");
            }
        } else {
            printf("Неверный формат\n");
        }
    }
}
```

9. Дневник отладки

№	Лаб. или дом.	Дата	Время	Событие	Действие по исправлению	Примечание

10. Замечания автора

11. Выводы

Эта ЛР помогла разобраться со структурами, с типами данных. Благодаря этой ЛР ознакомился с выделением и освобождением памяти.

Подпись студента _____