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

Студент группы М80-103Б-21 Первухин Алексей Сергеевич, № по списку 18

Контакты pervukhin.alexey@mail.ru, telegram @alioxa

Работа выполнена: «1» апреля 2022г.

Преподаватель: каф. 806 Севастьянов Виктор Сергеевич

Отчет сдан « » \_\_\_\_\_\_\_\_\_20\_\_ г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. **Тема:** Динамические структуры данных. Обработка деревьев.

1. **Цель работы:** Научиться работать с деревьями
2. **Задание:** Определить глубину дерева общего вида
3. **Оборудование** (студента):

Процессор *Intel Core i5-8265U @ 8x 3.9GH* с ОП *7851* Мб, НМД *1024* Гб. Монитор *1920x1080*

1. **Программное обеспечение (**студента**):**

Операционная система семейства: *linux*, наименование: *ubuntu*, версия *18.10 cosmic*

интерпретатор команд: *bash* версия *4.4.19*.

Система программирования --GNU версия --**,** редактор текстов *emacs* версия *25.2.2*

1. **Идея, метод, алгоритм решения задачи**

Изучить принцип построения и обработки деревьев, реализовать добавление и удаление узлов дерева, написать функцию для обхода дерева с определением его глубины.

**7. Сценарий выполнения работы**

Написал программу и проверил ее работу на примерах.

**8. Распечатка протокола**

#include <stdio.h>  
#include <stdlib.h>  
#include <stdbool.h>  
  
typedef struct TNode {  
 struct TNode \*parent;  
 int value;  
 int available\_cnt;  
 int CountChild;  
 struct TNode \*\*children;  
} TNode;  
  
int maxchild = 5;  
TNode\* genesis\_node;  
  
struct TStackHead{  
 struct TStackList{  
 struct TStackList \*Next;  
 TNode \*Node;  
 int depth;  
 } \*Head;  
};  
  
typedef struct TStackHead TStackHead;  
typedef struct TStackList TStackList;  
  
TNode\* find\_vertex(TNode\* u, int vertex) {  
 if (u->value == vertex)  
 return u;  
 TNode\* vertex\_address = NULL;  
 for (int i = 0; i < u->CountChild; ++i) {  
 vertex\_address = find\_vertex(u->children[i], vertex);  
 if (vertex\_address != NULL && vertex\_address->value == vertex)  
 break;  
 }  
 return vertex\_address;  
}  
  
void delete\_node(TNode\* v) {  
 for (int i = v->CountChild - 1; i >= 0; --i) {  
 delete\_node(v->children[i]);  
 }  
 TNode\* v\_parent = v->parent;  
 int del\_index = 0;  
 for (int i = 0; i < v\_parent->CountChild; ++i) {  
 if (v\_parent->children[i]->value == v->value) {  
 del\_index = i;  
 free(v);  
 break;  
 }  
 }  
 for (int i = del\_index + 1; i < v\_parent->CountChild; ++i) {  
 v\_parent->children[i - 1] = v\_parent->children[i];  
 }  
 v\_parent->CountChild--;  
}  
  
void print\_tree(TNode\* u, int space\_cnt, int deep) {  
 printf("%d\n", u->value);  
 for (int i = 0; i < u->CountChild; ++i) {  
 for (int j = 0; j < space\_cnt; ++j) {  
 if (deep != 0 && j % (maxchild \* deep) == 0)  
 printf("|");  
 else  
 printf(" ");  
 }  
 printf("|\n");  
 for (int j = 0; j < space\_cnt; ++j) {  
 if (deep != 0 && j % (maxchild \* deep) == 0)  
 printf("|");  
 else  
 printf(" ");  
 }  
 printf("+");  
 printf("----");  
 print\_tree(u->children[i], space\_cnt + 5, deep++);  
 }  
}  
  
void create\_node(TNode\* u, int v) {  
 TNode \*nd = malloc(sizeof(TNode));  
 nd->parent = u;  
 nd->value = v;  
 nd->CountChild = 0;  
 nd->available\_cnt = maxchild;  
 nd->children = malloc(sizeof(TNode) \* maxchild);  
 if(u->CountChild == u->available\_cnt - 1) {  
 u->children = realloc(u->children, sizeof(TNode) \* u->available\_cnt \* 2);  
 u->available\_cnt \*= 2;  
 }  
 u->children[u->CountChild] = nd;  
 u->CountChild++;  
}  
  
TNode \*create\_genesis\_node(int v) {  
 TNode \*nd = malloc(sizeof(TNode));  
 nd->parent = NULL;  
 nd->value = v;  
 nd->CountChild = 0;  
 nd->available\_cnt = maxchild;  
 nd->children = malloc(sizeof(TNode) \* maxchild);  
 return nd;  
}  
  
int PushStack(TStackHead \*HeadS, TNode \*Node, int depth) {  
 if (!HeadS) //проверка ошибки если голова пустая  
 return 1;  
 TStackList\* New;  
 if (!(New = (TStackList\*)malloc(sizeof(TStackList)))) { //проверка, получилось ли выделить память  
 return 1;  
 }  
 New->Node = Node;//присваиваю значение  
 New->depth = depth;  
 if (!HeadS->Head) {//если нет первого элемента  
 New->Next = NULL;  
 }  
 else {  
 New->Next = HeadS->Head;//если первый есть, то new указывает на него  
 }  
 HeadS->Head = New;//первый элемент - new, new->next(указывает на тот что раньше был первый)  
 return 0;  
}  
  
TStackList\* PopStack(TStackHead \*HeadS) {  
 if (!HeadS || !HeadS->Head) { //проверка ошибок(если голова пустая или нет первого элемента)  
 return NULL;//ошибка  
 }  
 TStackList\* Temp = HeadS->Head;//временный элемент, чтобы достать из стека  
 HeadS->Head = HeadS->Head->Next;//голова указывает на второй(первого больше нет в стеке)  
 return Temp;  
}  
  
int Detour(TNode\* Tree) {  
 if (!Tree)  
 return -1;  
 TStackHead Head;  
 Head.Head = NULL;  
 if (PushStack(&Head, Tree, 1)) {  
 return -1;  
 }  
 int maxdepth = 0;  
 while(true) {  
 TStackList\* Temp = PopStack(&Head);  
 if (!Temp) {  
 break;  
 }  
 if (maxdepth < Temp->depth)  
 maxdepth = Temp->depth;  
 for (int i = 0; i < Temp->Node->CountChild;i++) {  
 if (PushStack(&Head, Temp->Node->children[i],Temp->depth + 1))  
 return -1;  
 }  
 free(Temp);  
 }  
 return maxdepth;  
}  
  
int main(void) {  
 bool is\_genesis = true;  
 int op;  
 int newval;  
 int ans;  
 int u, v;  
 printf("1 - Create new rib\n");  
 printf("2 - Delete a rib\n");  
 printf("3 - Print the tree\n");  
 printf("4 - Check the depth\n");  
 printf("5 - End a program\n");  
 while (true) {  
 scanf("%d", &op);  
 switch (op) {  
 case 1:  
 scanf("%d %d", &u, &v);  
 if (u < 0 || v < 0) {  
 printf("Incorrect data\n");  
 continue;  
 }  
 if (is\_genesis) {  
 is\_genesis = false;  
 genesis\_node = create\_genesis\_node(u);  
 create\_node(genesis\_node, v);  
  
 } else {  
 TNode \*u\_address = find\_vertex(genesis\_node, u);  
 if (u\_address == NULL) {  
 printf("There is no such vertex in the tree\n");  
 continue;  
 }  
 TNode \*v\_address = find\_vertex(genesis\_node, v);  
 if (v\_address == NULL) {  
 create\_node(u\_address, v);  
  
 } else {  
 printf("There is such vertex in the tree already\n");  
 }  
 }  
 break;  
 case 2:  
 scanf("%d", &u);  
 TNode\* vert = find\_vertex(genesis\_node, u);  
 TNode\* par = vert;  
 if (vert == NULL)  
 printf("Incorrect data\n");  
 else if (par->parent==NULL) {  
 printf("Can't delete the genesis. Want to change it?(1/0)\n");  
 scanf("%d", &ans);  
 if (ans == 1) {  
 printf("New value: ");  
 scanf("%d", &newval);  
 vert->value = newval;  
 for (int i=0;i<=vert->CountChild;i++)  
 delete\_node(vert->children[i]);  
 }  
 }  
 else  
 delete\_node(vert);  
 break;  
 case 3:  
 if (genesis\_node != NULL)  
 print\_tree(genesis\_node, 0, 0);  
 else  
 printf("The tree is empty");  
 break;  
 case 4:  
 printf("Depth: %d\n", Detour(genesis\_node));  
 break;  
 case 5:  
 return 0;  
 default:  
 printf("No such option");  
 }  
 }  
}

**9. Дневник отладки**

1. **Замечания автора** по существу работы
2. **Выводы**

В результате работы у меня получилось выполнить поставленную задачу и реализовать дерево общего вида. Работа была сложная, пришлось потратить много времени на изучение принципов и алгоритмов обработки дерева.

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_