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

Студент группы М80-103Б-21 Барсуков Егор Алексеевич, № по списку 1

Контакты e-mail, telegram: @corsider

Работа выполнена: «18» февраля 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:** Составить программу на языке Си для построения и обработки дерева общего вида или упорядоченного двоичного дерева, содержащего узлы типа int.

**Задание** (*вариант №* **32** )**:** Определить число вершин дерева, степень которых совпадает со степенью дерева.

1. **Оборудование** (студента):

Процессор *Intel Core i5-1135G7 @ 4x 2.4GH* с ОП *16384* Мб, НМД *512* Гб. Монитор *1920x1080*

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

Операционная система семейства: *linux*, наименование: *ubuntu*, версия *20.04*

интерпретатор команд: *bash* версия *5.0.17(1)*

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

Утилиты операционной системы --

Прикладные системы и программы --

Местонахождение и имена файлов программ и данных на домашнем компьютере --

**6. Идея, метод, алгоритм**

Хранение дерева будет реализовано с помощью структуры. Этот вариант позволяет очень удобно работать с “сыновьями” вершин. Будет реализовано управление памятью с помощью malloc, realloc.

**7. Сценарий выполнения работы**

- Изучить деревья и особенности работы с ними

- Написание программы

- Исправление возможных ошибок

**8. Распечатка протокола**

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

struct node \*\* children;

int count;

int key;

} node;

//functions

node \*make\_node(int x) {

node \*n = (node \*)malloc(sizeof(node));

n->key = x;

n->count = 0;

n->children = NULL;

return n;

}

void free\_node(node \*n) {

for (int i = 0; i < n->count; i++) {

if (n->children[i] != NULL) {

free\_node(n->children[i]);

}

}

//deleted

free(n->children);

n->children = NULL; //deleting field

free(n);

n = NULL; //deleting node

}

node \*find\_node(node \*n, int key) {

if (n->key == key) {

return n;

}

if (n->count > 0) {

for (int i = 0; i < n->count; i++) {

node \* nn = find\_node(n->children[i], key);

if (nn) {

return nn;

}

}

return NULL; //no more children

} else {

return NULL; //no more children

}

}

void add\_node(node \*n, int parent, int x) {

node \*pnt = find\_node(n, parent);

if (pnt) {

if (pnt->count > 0) {

//adding memory for new child

pnt->children = (node \*\*)realloc(pnt->children, sizeof(node \*) \* (pnt->count + 1));

//adding child

int c = pnt->count;

pnt->children[c] = make\_node(x);

//incrementing count

pnt->count++;

} else {

//adding child, malloc

pnt->children = (node \*\*)malloc(sizeof(node \*));

pnt->children[0] = make\_node(x);

pnt->count = 1;

}

} else {

printf("%s", "[ERROR] NO PARENTS FOR THIS KEY WERE FOUND\n");

}

}

void print\_node\_tabs(node \*n, int depth) {

for (int i = 0; i < depth; i++) {

printf("\t");

}

printf("%d", n->key);

printf("\n");

depth++;

for (int i = 0; i < n->count; i++) {

int depthN = depth;

print\_node\_tabs(n->children[i], depthN);

}

}

void print\_node(node \*n, int depth) {

for (int i = 0; i < depth; i++) {

if (i < depth - 1) {

printf("| ");

} else {

printf("|=");

}

}

printf("%d", n->key);

printf("\n");

depth++;

for (int i = 0; i < n->count; i++) {

int depthN = depth;

print\_node(n->children[i], depthN);

}

}

void print\_tree(node \*n) {

print\_node(n, 0);

}

node \*find\_parent(node \*n, int key) {

if (n->count > 0) {

for (int i = 0; i < n->count; i++) {

if (n->children[i]->key == key) {

return n;

} else {

node \*p = find\_parent(n->children[i], key);

if (p != NULL) {

return p;

}

}

}

return NULL;

/\*

for (int i = 0; i < n->count; i++) {

node \*p = find\_parent(n->children[i], key);

if (p) {

return p;

}

}

\*/

} else {

return NULL;

}

}

void remove\_node(node \*n, int key) {

node \*nn = find\_node(n, key);

node \*pnt = find\_parent(n, key);

//printf("%s %d\n", "Parent is ", pnt->key);

int index;

for (int i = 0; i < pnt->count; i++) {

if (pnt->children[i]->key == key) {

index = i;

break;

}

}

free\_node(nn);

for (int i = index; i < pnt->count-1; i++) {

pnt->children[i] = pnt->children[i+1];

}

pnt->count--;

}

int tree\_power(node \*n, int max) {

//printf("%s %d %s %d\n", "I recieved MAX", max, "node", n->key);

int k = n->count;

if (k > max) {

max = k;

//printf("%s %d %s %d\n", "New MAX", max, "was at node", n->key);

}

for (int i = 0; i < n->count; i++) {

k = tree\_power(n->children[i], max);

if (k > max) {

max = k;

}

}

return max;

}

int nodes\_with\_max\_power(node \*n, int max, int k) {

if (n->count == max) {

k++;

//printf("%s %d %s %d\n", "Node", n->key, "power", n->count);

}

for (int i = 0; i < n->count; i++) {

int s = nodes\_with\_max\_power(n->children[i], max, k);

k = s;

}

return k;

}

int main(void)

{

//making tree

int start = 32;

node \*n = make\_node(start);

add\_node(n, 32, 31);

add\_node(n, 32, 30);

add\_node(n, 32, 29);

add\_node(n, 30, 28);

add\_node(n, 30, 27);

add\_node(n, 28, 26);

add\_node(n, 28, 25);

add\_node(n, 31, 24);

add\_node(n, 28, 1);

//MENU

int cont = 0;

int tp;

while (cont < 6) {

//loop

tp = tree\_power(n, 0);

printf("%s\n", "What do you want to do?");

printf("%s\n", "1) Add new node");

printf("%s\n", "2) Print tree");

printf("%s\n", "3) Print node");

printf("%s\n", "4) Remove node");

printf("%s\n", "5) Number of nodes, power of which is equal to tree's power");

printf("%s\n", "6) Exit");

printf("%s", "Decide. ");

scanf("%d", &cont);

printf("\n");

if (cont == 1) {

int parent;

int val;

printf("%s\n", "Adding new node. Which one?");

printf("%s\n", "Enter parent node and new node's key value ([PARENT] [VALUE]):");

scanf("%d %d", &parent, &val);

printf("\n");

if (find\_node(n, parent)) {

if (find\_node(n, val)) {

printf("%s\n", "This key already exists");

} else {

add\_node(n, parent, val);

printf("%s\n", "Added. Now tree looks like this:");

printf("\n");

print\_tree(n);

}

} else {

printf("%s\n", "No such parent was found");

}

printf("\n");

} else if (cont == 2) {

printf("%s\n", "Printing tree...");

printf("\n");

print\_tree(n);

printf("\n");

} else if (cont == 3) {

int kkey;

printf("%s\n", "Which node and it's children to print?");

scanf("%d", &kkey);

printf("\n");

if (find\_node(n, kkey)) {

print\_node(find\_node(n, kkey), 0);

} else {

printf("%s\n", "No such node was found");

}

printf("\n");

} else if (cont == 4) {

int dkey;

printf("%s\n", "Select node's key to delete");

scanf("%d", &dkey);

printf("\n");

if (find\_node(n, dkey) && dkey != start) {

remove\_node(n, dkey);

printf("%s\n", "Now tree looks like this:");

printf("\n");

print\_tree(n);

} else if (dkey == start) {

printf("%s\n", "Can't delete root node");

} else {

printf("%s\n", "No node with such key was found");

}

printf("\n");

} else if (cont == 5) {

int k = 0;

int np = nodes\_with\_max\_power(n, tp, 0);

printf("%s %d\n", "Nodes with tree's power:", np);

printf("\n");

}

}

free\_node(n);

return 0;

}

**9. Дневник отладки** должен содержать дату и время сеансов отладки и основные события(ошибки в сценарии и программе,нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

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

1. **Замечания автора**
2. **Выводы**

Эта лабораторная работа мне очень понравилась, т.к. тема деревьев обширна и интересна. Я научился работать с деревьями. Уверен, что этот навык пригодится в будущем.

Подпись студента \_\_Барсуков Е.А.\_\_\_\_\_\_