Отчет по лабораторной работе №23 по курсу практикум на ЭВМ

Студент группы М8О-101Б-20 Ядров Артем Леонидович, № по списку 28

Контакты www, e-mail, icq, skype temayadrow@gmail.com

Работа выполнена: « » _____202___г.

Преподаватель: доцент каф. 806 Никулин Сергей Петрович

Входной контроль знаний с оценкой _____

Отчет сдан « » _____202 __ г., итоговая оценка _____

Подпись преподавателя _____

1. Тема: Динамические структуры данных. Обработка деревьев.

2 **Цель работы:** Научиться работать с динамическими структурами данных и обрабатывать деревья

3 **Задание** (*вариант № 26*)**:** Определить степень дерева

4 **Оборудование** (лабораторное):
ЭВМ Intel Pentium G2140, процессор ____3.30 GHz____, имя узла сети__Cameron____с ОП____8096____Мб,
НМД ___7906____ Мб. Терминал __ASUS___ адрес_____dev/pets/3____ Принтер ____HP Laserjet 6P_____
Другие устройства _____

Оборудование ПЭВМ студента, если использовалось:
Процессор __Intel core i5-7300HQ 2.50 GHz___ с ОП __8096__ Мб, НМД __131072__ Мб. Монитор _____ASUS_____
Другие устройства _____

5 Программное обеспечение (лабораторное):
Операционная система семейства __Unix___, наименование _____Ubuntu_____ версия ___18.15.0____
интерпретатор команд _____bash_____ версия ____4.4.20____
Система программирования _____CLion_____ версия _____2020.3_____
Редактор текстов _____emacs_____ версия ____25.2.2_____
Утилиты операционной системы _____cat, gcc_____
Прикладные системы и программы_____
Местонахождение и имена файлов программ и данных___stud/208104_____

Программное обеспечение ЭВМ студента, если использовалось:
Операционная система семейства ____Unix____, наименование __Fedora_____ версия ____33_____
интерпретатор команд _____bash_____ версия __5.0.17____
Система программирования ____CLion_____ версия __2020.3_____
Редактор текстов _____emacs_____ версия ____25.2.2_____
Утилиты операционной системы _____cat, gcc_____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных на домашнем компьютере ___home/Temi4_____

6. Идея, метод, алгоритм решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)

Опишем следующие структуры:

- struct tnode {
  float value;
  struct tnode *son;
  struct tnode *brother;
  struct tnode *parent;
  };
  Структура узла дерева. Хранит значение узла, указатель на старшего сына, указатель на следующего брата, указатель на родителя.
- typedef struct {
  node *root;
  } Tree;
  Структура самого дерева. Хранит указатель на корень.

С помощью этих структур реализуем функции для работы с деревом:

- node *create_node(float f, node *par)
  Функция создания узла дерева. Создает указатель на узел дерева, помещает в него значение и родителя, а затем возвращает указатель на созданный узел
- Tree *create_tree(float f)
  Функция создания дерева. Создает указатель на дерево, затем создает корень дерева с помощью описанный выше функции. Возвращает указатель на созданное дерево.
- node *search_tree(node *t, float f)
  Функция поиска узла по значению. Работает путем поиска в глубину и возвращает первый найденный узел с значением f (т. е. самый «глубокий»).
- void add_node_in_tree(Tree *tree, float par_f, float f)
  Функция добавления узла дерева. Для создания узла ищется родитель узла с помощью функции search_tree. Затем создается узел с помощью функции create_node. Если у родителя нет сына, то создаваемый узел становится сыном родителя. В противном случае становится самым младшим братом старшего сына родителя.
- void delete_node(Tree *t, float f)
  Функция удаления узла дерева. С помощью функции search_tree находится удаляемый узел. Старшим сыном родителя выбранного узла становится следующий брат выбранного узла (если братьев нет, то, как и положено, указатель на сына становится NULL). Указатель на выбранный узел освобождается.
- void print_tree(node *t, int x)
  Функция печати узла дерева. Печать осуществляется в порядке обхода КЛП. Братья находятся слева от родителя на одной вертикальной линии.
- int node_degree(node *t)
  Функция возвращает степень узла дерева. Проходит старшего сына, а затем всех его братьев.
- int max(int a, int b)
  Возвращает максимальное значение двух чисел.
- int task(node *t, int mx)
  Рекурсивная функция. Производит обход в глубину. Сначала обрабатывается сам узел (вычисляется степень узла), затем старший сын, а затем брат. Возвращает максимум из текущего максимума (mx), степени узла, степени старшего сына узла и степени брата узла.

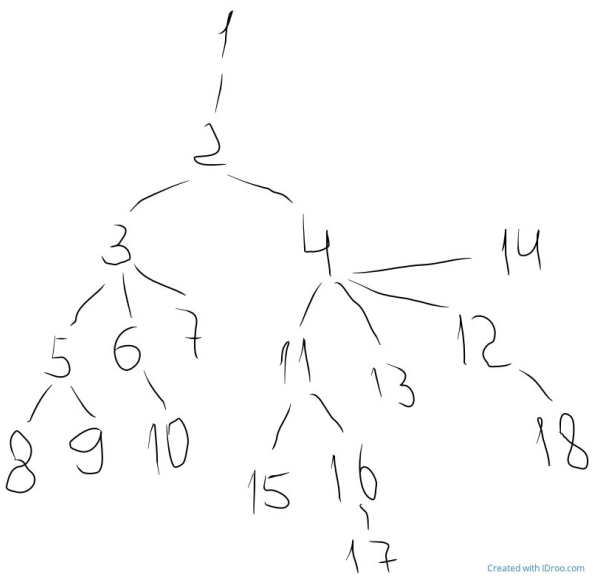
В основной части программы будем использовать меню, в котором есть 6 опций:

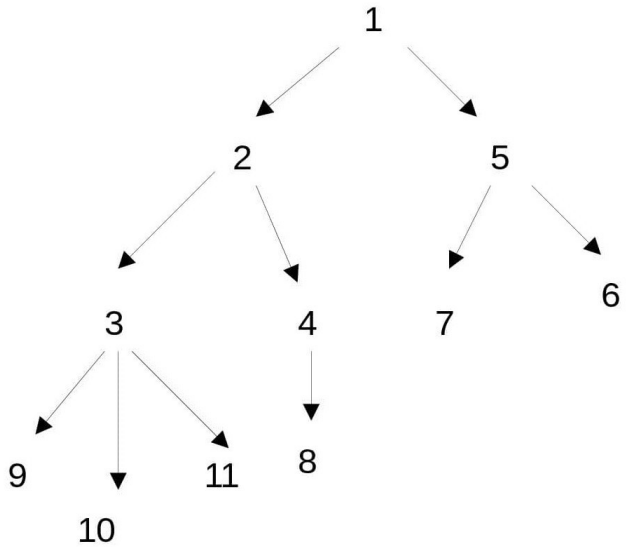
1. Создание дерева (Create tree)
   Запрашивает значение корня дерева, а затем создает дерево, вызывая функцию create_tree
2. Добавление узла в дерево (Add node to tree)
   Запрашивает значение добавляемого узла, а затем вызывает функцию add_node
3. Удаление узла дерева (Delete node from tree)
   Вызывает функцию delete_node
4. Выполнение задания (вычисление степени дерева) (Task)
   Вызывает функцию task от корня с максимальным значением 0, а затем выводит ответ.
5. Печать дерева (Print tree)
   Вызывает функцию print_tree
6. Выход (Exit)
   Выходит из меню

7 **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

Тесты:

- Дерево, состоящее из одного корня (1)
- Пустое дерево





Пункты 1-7 отчета составляются строго до начала лабораторной работы.

Допущен к выполнению работы. **Подпись преподавателя** _____

8 **Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).

```
[Temi4@localhost 23]$ cat tree.c
#include <stdio.h>
#include <stdlib.h>

struct tnode {
   float value;
   struct tnode *son;
   struct tnode *brother;
   struct tnode *parent;
};
typedef struct tnode node;
typedef struct {
   node *root;
} Tree;

struct Item {
   node *t;
   struct Item *next;
};

struct queue {
   struct Item *first;
   struct Item *last;
   int size;
};

struct queue *create_queue() {
   struct queue *q = (struct queue *) malloc(sizeof(struct queue));
   q->first = (struct Item *) malloc(sizeof(struct Item));
   q->last = (struct Item *) malloc(sizeof(struct Item));
   q->size = 0;
   return q;
}

int empty_queue(struct queue *q) {
   return q->size == 0;
}

int Push(struct queue *q, node *t) {
   if (!(q->last->next = (struct Item*) malloc(sizeof (struct Item)))){
      return 0;
   }
   q->last->t = t;
   q->last = q->last->next;
   q->size++;
   return 1;
}

int Pop(struct queue *q) {
   if (q->first == q->last) {
      return 0;
   }
   struct Item *pi = q->first;
   q->first = q->first->next;
   q->size--;
   free(pi);
   return 1;
}

node *Top(struct queue *q) {
   if (q->first != q->last) {
      return q->first->t;
   }
   return NULL;
}
```

```
node *create_node(float f, node *par) {
   node *t;
   t = (node *) malloc(sizeof(node));
   t->value = f;
   t->son = NULL;
   t->brother = NULL;
   t->parent = par;
   return t;
}

Tree *create_tree(float f) {
   Tree *t;
   t = (Tree *) malloc(sizeof(Tree));
   t->root = create_node(f, NULL);
   return t;
}

node *search_tree(node *t, float f) {
   if (t == NULL){
      return  t;
   }
   node *tree = NULL;
   if (t->value == f) {
      return t;
   }
   if (t->son != NULL) {
      tree = search_tree(t->son, f);
   }
   if (tree == NULL) {
      tree = search_tree(t->brother, f);
   }
   return tree;
}

void add_node_in_tree(Tree *tree, float par_f, float f) {
   node *t = tree->root;
   t = search_tree(t, par_f);
   if (t == NULL) {
      printf("%-.2f not contains in tree\n", par_f);
      return;
   }
   if (t->son == NULL) {
      t->son = create_node(f, t);
   } else {
      t = t->son;
      while (t->brother != NULL) {
         t = t->brother;
      }
      t->brother = create_node(f, t->parent);
   }
}

void delete_node(Tree *tree, float f) {
   node *t = tree->root;
   t = search_tree(t, f);
   if (t == NULL) {
      printf("%-.2f not contains in tree\n", f);
      return;
   }
   if (t->parent->son == t){
      t->parent->son = t->brother;
   }
   else{
      node *tr = t->parent->son;
      while (tr->brother != t){
         tr = tr->brother;
```

```
        }
        tr->brother = t->brother;
    }
    free(t);
}

void print_tree(node *t, int x) {
    if (t == NULL) {
        return;
    }
    for (int i = 0; i < x; i++) {
        printf("\t");
    }
    printf("%-.2f\n", t->value);
    print_tree(t->son, x + 1);
    print_tree(t->brother, x);
}

int task(Tree *T) {
    node *t = T->root;
    struct queue *q = create_queue();
    Push(q, t);
    int cur_lvl = 0;
    int prev_lvl = 0;
    node *end_cur_level = T->root;
    while (!empty_queue(q)) {
        t = Top(q);
        cur_lvl++;
        Pop(q);
        if (t == end_cur_level){
            if (end_cur_level->son != NULL){
                end_cur_level = end_cur_level->son;
                while (end_cur_level->brother != NULL){
                    end_cur_level = end_cur_level->brother;
                }
            }
            else {
                node *tree = end_cur_level->parent->son;
                end_cur_level = NULL;
                while (tree->brother != NULL) {
                    tree = tree->brother;
                    if (tree->son != NULL) {
                        end_cur_level = tree->son;
                    }
                }
            }
            if (prev_lvl < cur_lvl){
                prev_lvl = cur_lvl;
                cur_lvl = 0;
            }
            else{
                return 0;
            }
            end_cur_level = NULL;
        }
        node *tree = t->son;
        if (tree == NULL) {
            continue;
        }
        Push(q, tree);
        while (tree->brother != NULL) {
            Push(q, tree->brother);
            tree = tree->brother;
        }
    }
    return 1;
}
```

```
int main() {
   Tree *t = NULL;
   int choose, g = 1;
   while (g) {
      printf("1. Create tree\t 2. Add node to tree\t 3. Delete node from tree\t 4. Task\t 5. Print tree\t 6. Exit \n");
      scanf("%d", &choose);
      switch (choose) {
         case 1: {
            printf("Write tree's root\n");
            float f;
            scanf("%f", &f);
            t = create_tree(f);
            break;
         }
         case 2: {
            printf("Write tree node value\n");
            float f, par_f;
            scanf("%f", &f);
            printf("Write parent value\n");
            scanf("%f", &par_f);
            add_node_in_tree(t, par_f, f);
            break;
         }
         case 3: {
            printf("Write tree node value\n");
            float f;
            scanf("%f", &f);
            delete_node(t, f);
            break;
         }
         case 4: {
            if (task(t)) {
               printf("The width of the tree level rises\n");
            } else {
               printf("The width of the tree level doesn't rise\n");
            }
            break;
         }
         case 5: {
            print_tree(t->root, 0);
            break;
         }
         case 6: {
            g = 0;
            break;
         }
         default: {
            printf("Wrong answer\n");
         }
      }
   }
   return 0;
}[Temi4@localhost 23]$ gcc tree.c
[Temi4@localhost laabs]$ ./a.out
1. Create tree   2. Add node to tree     3. Delete node from tree        4. Task        5. Print tree   6. Exit
1
Write tree's root
1
1. Create tree   2. Add node to tree     3. Delete node from tree        4. Task        5. Print tree   6. Exit
2
Write tree node value
2
Write parent value
1
1. Create tree   2. Add node to tree     3. Delete node from tree        4. Task        5. Print tree   6. Exit
2
Write tree node value
5
```

Write parent value
1
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
3
Write parent value
2
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
4
Write parent value
2
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
9
Write parent value
3
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
10
Write parent value
3
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
11
Write parent value
3
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
4
Write parent value
8
8.00 not contains in tree
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
8
Write parent value
4
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
7
Write parent value
5
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
6
Write parent value
5
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
5
1.00
2.00
3.00
9.00
10.00
11.00
4.00
8.00
5.00

```
            7.00
            6.00
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
4
tree grade is 3
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
3
Write tree node value
3
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
5
1.00
      2.00
            4.00
                  8.00
      5.00
            7.00
            6.00
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
4
tree grade is 2
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
6
[Temi4@localhost laabs]$ ./a.out
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
1
Write tree's root
1
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
2
Write tree node value
2
Write parent value
1
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
2
Write tree node value
3
Write parent value
2
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
2
Write tree node value
4
Write parent value
2
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
2
Write tree node value
5
Write parent value
3
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
2
Write tree node value
6
Write parent value
3
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
2
Write tree node value
7
Write parent value
3
1. Create tree   2. Add node to tree   3. Delete node from tree   4. Task   5. Print tree   6. Exit
2
Write tree node value
8
```

Write parent value
5
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
9
Write parent value
5
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
10
Write parent value
6
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
11
Write parent value
4
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
13
Write parent value
4
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
12
Write parent value
4
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
14
Write parent value
4
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
15
Write parent value
11
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
16
Write parent value
11
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
17
Write parent value
16
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
2
Write tree node value
18
Write parent value
12
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
5
1.00
2.00
3.00
5.00

8.00
9.00
6.00
10.00
7.00
4.00
11.00
15.00
16.00
17.00
13.00
12.00
18.00
14.00
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
4
tree grade is 4
1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit
6

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. или дом.	Дата	Время	Событие	Действие по исправлению	Примечание

10 **Замечания автора** по существу работы_____

11 Выводы

Я научился работать с динамическими структурами и обрабатывать деревья

Недочёты при выполнении задания могут быть устранены следующим образом:

Подпись студента _____