Отчет по лабораторной работе № 23 по курсу
Алгоритмы и структуры данных

Студент группы М8О-103Б-22 Ахметшин Булат Рамилевич, № по списку 2

Контакты www, e-mail, icq, skype ahmbulat04@yandex.ru

Работа выполнена: 21.04.2023 г.

Преподаватель: доцент каф. 806 Никулин С.П.

Входной контроль знаний с оценкой _____

Отчет сдан « » _____ 202 __ г., итоговая оценка ___

Подпись преподавателя _____

1. **Тема:** Динамические структуры данных, обработка деревьев. _____

2. **Цель работы:** Научиться реализовывать динамические структуры данных, такие как деревья, на языке программирования Си и работать. _____

3. **Задание** (*вариант № 2*): Составить программу на языке Си для построения и обработки упорядоченного двоичного дерева, содержащего узлы типа int. _____

4. **Оборудование** (*лабораторное*):
ЭВМ _____, процессор _____, имя узла сети _____ с ОП _____ Мб,
НМД _____ Мб. Терминал _____ адрес _____. Принтер _____
Другие устройства _____

Оборудование ПЭВМ студента, если использовалось:
Процессор Intel(R) Core(TM) i7-10510U с ОП 8 ГБ НМД SSD 512 ГБ . Монитор Встроенный 1920x1080
Другие устройства _____

5. **Программное обеспечение** (*лабораторное*):
Операционная система семейства _____, наименование _____ версия _____
интерпретатор команд _____ версия _____
Система программирования _____ версия _____
Редактор текстов _____ версия _____
Утилиты операционной системы _____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных _____

Программное обеспечение ЭВМ студента, если использовалось:
Операционная система семейства UNIX, наименование Ubuntu версия 22.04
интерпретатор команд GNU bash версия 5.1.16
Система программирования Visual Studio Code версия 1.77.3
Редактор текстов Sublime Text 3 версия 3211

Утилиты операционной системы Стандартные утилиты OS Linux
Прикладные системы и программы Редактор текста nano.
Местонахождение и имена файлов программ и данных на домашнем компьютере /home/bulat/Studying/prprm/l/l23

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

Основные структуры программы будут описаны следующим образом:

node - структура, реализующая узел бинарного дерева

```
typedef struct node {
    // Identification number of each node
    uint64_t id;
    // Data of node
    int64_t data;
    // Pointer to left node of b_three
    struct node* left;
    // Pointer to right node of b_tree
    struct node* right;
    // Pointer to node's ancestor
    struct node* p;
} node;
```

b_tree - реализация бинарного дерева

```
typedef struct b_tree{
    // Count of nodes
    int64_t n;
    // Id of last added node
    int64_t last_id;
    // Pointer to root node of b_tree
    node* root;

} b_tree;
```

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

   (a) Реализовать структуру данных бинарное дерево, вершинами которого будут являться структуры типа node - узлы с идентификаторами, значениями и указателями на предка, младшего и старшего братьев.

   (b) Релизовать структуру данных строка, написать алгоритмы парсинга поступающих комманд

   (c) Написать итоговый код программы, протестировать

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)

```
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l23/a2$ script logs/proto_1
Script started, output log file is 'logs/proto_1'.
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l23/a2$ ls
l23-2012.djvu  main    README.md  string.h  tree.h
logs           main.c  string.c   tree.c    отчет23.pdf
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l23/a2$ cat tree.h
#ifndef M_TREE_H

#define M_TREE_H

#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>

typedef enum {
    N_LEFT,
    N_RIGHT,
    N_ROOT,
    INVALID
} SIDE;

// Inary tree node structure
typedef struct node {
    // Identification number of each node
    uint64_t id;
    // Data of node
    int64_t data;
    // Pointer to left node of b_three
    struct node* left;
    // Pointer to right node of b_tree
    struct node* right;
    // Pointer to node's ancestor
    struct node* p;
} node;

// Binary tree
typedef struct b_tree{
    // Count of nodes
    int64_t n;
    // Id of last added node
    int64_t last_id;
    // Pointer to root node of b_tree
    node* root;

} b_tree;

// Creates empty b_tree
b_tree create_empty_b_tree();
```

```
void copy_b_tree(b_tree* t, b_tree* const t_);

void copy_b_tree_n_update(b_tree* t, b_tree* const t_);

void __copy_b_subtree(node** root, node* const to_copy, node* const acc);

void init_b_tree(b_tree* t, int64_t data);

// Checks if root pointer is NULL
int8_t is_empty_b_tree(b_tree* const t);

// Adds node py pointer to its ancestor by ancestors id
void add_node_by_node(b_tree* t, node* p, int64_t data, SIDE s);

// Adds node py pointer to its ancestor by ancestors id
void add_node_by_id(b_tree* t, int64_t id, int64_t data, SIDE s);

//      Deletes node by id
void delete_node_by_id(b_tree* t, int64_t id);

//      Deletes node by pointer
void delete_node_by_node(b_tree* t, node* v);

// Returns pointer to node of b_tree by id
node* node_by_id(node* const root, int64_t id);

// Returns id node data
int64_t data_by_id(b_tree* const t, int64_t id);

// Checks if given node is in b_tree
int8_t node_is_in_b_tree(node* const root, node* const v);

// Removes all nodes from b_tree and sets it empty
void clear_tree(b_tree* t);

// Node
// {
//     id = id;
//     data = data;
//     left = NULL;
//     right = NULL;
// }
node* new_node(int64_t id, int64_t data, node* const p);

void copy_node(node* v, node* const v_);

void print_b_tree(node* const t, uint8_t tab, SIDE side);

int8_t is_B_tree(node* const root);

#define log_exception(M, ...)                                           \
{                                                                       \
    fprintf(stderr, "(%s:%d) " M "\n", __FILE__, __LINE__, ##__VA_ARGS__); \
}

#define exception_exit(M, ...)          \
{                                       \
    log_exception(M, ##__VA_ARGS__);    \
    exit(-1);                           \
}

#endifbulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l23/a2$ cat tree.c
#include "tree.h"

b_tree create_empty_b_tree() {
    b_tree t;
    t.n = 0;
    t.last_id = -1;
    t.root = NULL;
    return t;
}

int8_t is_empty_b_tree(b_tree* const t) {
    return t->root == NULL;
}

void copy_b_tree(b_tree* t, b_tree* const t_) {
    clear_tree(t);
    __copy_b_subtree(&(t->root), t_->root, NULL);
    t->n = t_->n;
    t->last_id = t_->last_id;
}

void copy_b_tree_n_update(b_tree* t, b_tree* const t_) {
    copy_b_tree(t, t_);
```

```
}

void __copy_b_subtree(node** root, node* const to_copy, node* const acc) {
    if (to_copy == NULL)
        return;
    *root = new_node(to_copy->id, to_copy->data, acc);
    __copy_b_subtree(&(*root)->left, to_copy->left, *root);
    __copy_b_subtree(&(*root)->right, to_copy->right, *root);
}

void init_b_tree(b_tree* t, int64_t data) {
    if (!is_empty_b_tree(t)) {
        exception_exit("Initializing not empty b_tree");
    }
    t->root = new_node(0, data, NULL);
    t->last_id = 0;
    t->n = 1;
}

void add_node_by_id (b_tree* t, int64_t id, int64_t data, SIDE s) {
    // Create new node
    if (is_empty_b_tree(t)) {
        init_b_tree(t, data);
    }
    node* v = node_by_id(t->root, id);
    if (v == NULL) {
        exception_exit("There is no node in tree %p with id %ld\n", t, id);
    }
    add_node_by_node(t, v, data, s);
}

void add_node_by_node(b_tree* t, node* p, int64_t data, SIDE s) {
    if (p == NULL) {
        exception_exit("Trying to access NULL node pointer\n");
    }
    node* v = p;
    // Add new node to ancestor with given id
    if (s == N_LEFT) {
        if (v->left == NULL) {
            v->left = new_node(t->last_id + 1, data, v);
        } else {
            exception_exit("%ld node already has left child - %ld node", v->id, v->left->id);
        }
    } else if (s == N_RIGHT) {
        if (v->right == NULL) {
            v->right = new_node(t->last_id + 1, data, v);
        }
        else {
            exception_exit("%ld node already has right child - %ld node", v->id, v->right->id);
        }
    } else if (s == N_ROOT) {
        exception_exit("Root is already exist: add_node_by_node(%p, %p, %ld, N_ROOT), p id: %ld\n", t, p, data, p->id);
    } else {
        exception_exit("Invalid side've been given\n");
    }
    t->last_id++;
    t->n++;
}

node* node_by_id(node* const root, int64_t id) {
    node* const i = root;
    if (i == NULL || i->id == id) {
        return i;
    }
    node* l = i->left,* r = i->right;
    l = node_by_id(l, id);
    r = node_by_id(r, id);
    if (l != NULL) {
        return l;
    }
    if (r != NULL) {
        return r;
    }
    return NULL;
}

node* new_node(const int64_t id, int64_t data, node* const p) {
    node* v = (node*)malloc(sizeof(node));
    v->id= id;
    v->data = data;
    v->left = NULL;
    v->right = NULL;
    v->p = p;
    return v;
}
```

```
void copy_node(node* v, node* const v_) {
    v->data = v_->data;
    v->id = v_->id;
    v->left = NULL;
    v->right = NULL;
}

void delete_node_by_id(b_tree* t, const int64_t id) {
    node* v = node_by_id(t->root, id);
    delete_node_by_node(t, v);
}

void delete_node_by_node(b_tree* t, node* v) {
    if (v == NULL) {
        return;
    }
    node* i = v;

    node* l = v->left,* r = v->right;

    if (!(l == NULL)) {
        delete_node_by_node(t, l);
    }
    if (!(r == NULL)) {
        delete_node_by_node(t, r);
    }
    if (!(i->p == NULL)) {
        if (i->p->left == i) {
            i->p->left = NULL;
        } else {
            i->p->right = NULL;
        }
    }
    free(i);
    --t->last_id;
    --t->n;
    if (t->n == 0) {
        t->root = NULL;
    }
}

int64_t data_by_id(b_tree* const t, int64_t id) {
    node* v = node_by_id(t->root, id);
    if (v == NULL) {
        exception_exit("There is no node with given id in the b_tree\n")
    }
    return v->data;
}

int8_t node_is_in_b_tree(node* const root, node* const v) {
    if (v == NULL || root == NULL) {
        return 0;
    }
    if (root == v) {
        return 1;
    }
    return node_is_in_b_tree(root->left, v) \
        || node_is_in_b_tree(root->right, v);
}

void clear_tree(b_tree* t) {
    if (!is_empty_b_tree(t)) {
        delete_node_by_node(t, t->root);
    } else {
        t->n = 0;
        t->last_id = -1;
        t->root = NULL;
    }
}

void print_b_tree(node* const root, uint8_t tab, SIDE side) {
    if (root == NULL) {
        return;
    }
    printf("%ld", root->id);
    for (uint8_t i = 0; i <= tab; ++i) {
        printf("\t");
        printf("|");
    }
    if (side == N_LEFT) {
        printf("L: ");
    } else if (side == N_RIGHT) {
        printf("R: ");
    } else {
```

```
        printf("Root: ");
    }
    printf("%ld\n", root->data);
    print_b_tree(root->left, tab + 1, N_LEFT);
    print_b_tree(root->right, tab + 1, N_RIGHT);
    //printf("\n");
}

int8_t is_B_tree(node* const root) {
    if (root == NULL) {
        return 1;
    }
    if ((root->left == NULL) == (root->right == NULL)) {
        if (root->left == NULL && root->right == NULL) {
            return 1;
        }
        return is_B_tree(root->left) && is_B_tree(root->right);
    }
    return 0;
}bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l23/a2$ cat string.h
#ifndef STRING_H

#define STRING_H

#define ADD_MEMORY_COEFFICIENT 2
#define INIT_VALUE ' '

#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>

// Структура Строка
struct string {
char* s;
uint64_t memory_size;
uint64_t last_char;
};

typedef struct string string;

// Процедура конструирования строки из одного символа INIT_VALUE
void construct_empty(string* s);

string not_allocated_string();

// Процедура конструирования строки из n символов
void construct_from_n(string* s, uint64_t n);

// Процедура конструирования строки из другой строки
void construct_from_s(string* s, string s_);

// Процедура конструирования строки из массива символов
void construct_from_char_pointer(string* s, const char* p);

// Процедура деструктирования строки
void destruct(string* s);

// Аннулировать строку
void annul(string* s);

// Процедура присвоения памяти
void appropriate_memory(string* s, string* s_);

// Процедура присвоения строки
void appropriate_string(string* s, string s_);

// Процедура копирования строки (без учёта памяти)

// Пример: <'abc'; '1234'> -> <'123'; '1234'>
// Пример: <'abcd'; '123'> -> <'123 '; '123'>
void copy_string(string* s, string s_);

// Процедура добавления памяти в строку
void add_memory(string* s, uint64_t n);

// Процедура добавления символа в строку
void add_char(string* s, char c);

// Конкатенация строк
void add_string(string* s, string s_);

int8_t equal_string(string* const s, string* const s_);

int8_t equal_charp(string* const s, const char* p);
```

```
// Считать строку
int8_t read_line(string* s);

// Считать все строки
void read(string* s);

// Вывести строку
void print(string s);

// Получить i-тый символ строки
char* at(string* s, uint64_t i);

// Макросы

// Максимум
#define max(a, b) (a > b ? a : b)
// Минимум
#define min(a, b) (a < b ? a : b)
// Логирование
#define log_info(M, ...) fprintf(stderr, "(%s:%d) " M "\n",\
        __FILE__, __LINE__, ##__VA_ARGS__)

#endifbulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l23/a2$ cat string.c
#include "string.h"

// Процедура конструирования строки из одного символа ' '
void construct_empty(string* s) {
    construct_from_n(s, 1);
}

string not_allocated_string() {
string s;
annul(&s);
return s;
}

// Процедура конструирования строки из n символов
void construct_from_n(string* s, uint64_t n) {
    // Один последний символ зарезервирован под '\0'
s->s = (char*)malloc(sizeof(char) * n + 1);
if (!(s->s == NULL)) {
s->memory_size = n;
s->last_char = 0;
        // Строка инициализирована символом INIT_VALUE, объявленном в string.h
for (uint64_t i = 0; i < n; ++i)
s->s[i] = INIT_VALUE;
        s->s[n] = '\0';
}
    else {
        log_info("REFUZE_MEMORY_ALLOCATION\n");
        exit(-1);
    }
}

// Процедура конструирования строки из другой строки
void construct_from_s(string* s, string s_) {
construct_from_n(s, s_.memory_size);
copy_string(s, s_);
}

// Процедура конструирования строки из массива символов
void construct_from_char_pointer(string* s, const char* p) {
if (p == NULL) {
        log_info("NULL_POINTER_ACCES\n");
        exit(-1);
    }
construct_from_n(s, 1);
uint64_t i = 0;
while (p[i] != '\0') {
add_char(s, p[i]);
++i;
}
    add_char(s, '\0');
}

// Процедура деструктирования строки
void destruct(string* s) {
if (s->s) {
free(s->s);
s->last_char = 0;
s->memory_size = 0;
}
    else {
    log_info("DEALLOCATE_NULL_POINTER");
        exit(-1);
```

```
    }
}

// Аннулировать строку
void annul(string* s) {
s->s = NULL;
s->last_char = 0;
s->memory_size = 0;
}

// Процедура присвоения памяти
void appropriate_memory(string* s, string* s_) {
destruct(s);
s->s = s_->s;
s->memory_size = s_->memory_size;
s->last_char = s_->last_char;
annul(s_);
}

// Процедура копирования строки (без учёта памяти)

// Пример:
// <'abc', '1234'> -> <'123', '1234'>
// <'abcd', '123'> -> <'123 ', '123'>
// <'abc',  '123'> -> <'123',  '123'>
void copy_string(string* s, string s_) {
if (s->s && s_.s) {
uint64_t i = 0;
while (i < s->memory_size && i < s_.last_char) {
s->s[i] = s_.s[i];
++i;
}
while (i < s->memory_size) {
s->s[i] = ' ';
++i;
}
        s->last_char = min(s->memory_size, s_.last_char);
}
else {
        log_info("NULL_POINTER_ACCES");
        exit(-1);
    }
}

// Процедура присвоения строки
void appropriate_string(string* s, string s_) {
construct_from_s(s, s_);
copy_string(s, s_);
}


// Процедура добавления памяти в строку
void add_memory(string* s, uint64_t n) {
string s_;
construct_from_n(&s_, s->memory_size * n);
copy_string(&s_, *s);
appropriate_memory(s, &s_);
}

// Добавить символ
void add_char(string* s, char c) {
if (s->s) {
if (s->last_char >= s->memory_size) {
add_memory(s, ADD_MEMORY_COEFFICIENT);
}
s->s[s->last_char] = c;
        if (c != '\0')
            s->last_char++;
s->s[s->last_char] = '\0';
}
else {
        log_info("NULL_POINTER_ACCES");
        exit(-1);
    }
}

// Конкатенация строк
void add_string(string* s, string s_) {
    string S;
    construct_from_n(&S, s->last_char + s_.last_char);
    while (S.last_char < s->last_char) {
        S.s[S.last_char] = s->s[S.last_char];
        S.last_char++;
    }
    while (S.last_char - s->last_char < s_.last_char) {
```

```
        S.s[S.last_char] = s_.s[S.last_char - s->last_char];
        S.last_char++;
    }
    add_char(&S, '\0');
    appropriate_memory(s, &S);
}

int8_t equal_string(string* const s, string* const s_) {
if (s->last_char != s_->last_char) {
return 0;
}
for (uint64_t i = 0; i < s->last_char; ++i) {
if (*at(s, i) != *at(s_, i))
return 0;
}
return 1;
}

int8_t equal_charp(string* const s, const char* p) {
string s_;
construct_from_char_pointer(&s_, p);
return equal_string(s, &s_);
}

// Считать строку
int8_t read_line(string* s) {
char c;
while (1) {
//scanf("%c", &c);
c = getchar();
        if (c == EOF || c == '~') {
            add_char(s, '\0');
            return 0; // Конец потока
        }
        if (c == '\n') {
add_char(s, c);
            return 1; // Конец строки
        }
add_char(s, c);
}
}

// Считать все строки
void read(string* s) {
    while (read_line(s)) {}
}

// Вывести строку
void print(string s) {
printf("%s", s.s);
}

// Получить i-тый символ строки
char* at(string* s, uint64_t i) {
if (i < s->memory_size) {
return &s->s[i];
    }
    else {
    log_info("OCCUPIED_MEMORY_ACCES");
        exit(-1);
    }
}bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l23/a2$ cat main.c
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include "string.h"
#include "tree.h"

typedef enum COMMAND {
    ADD_NODE,    // add node $id$ $value$ $side$ or add root $value$
    DELETE_NODE, // delete $id$
    PRINT_TREE,  // print
    // Calculating function checks if tree is a B-tree
    IS_B_TREE,   // calc
    EXIT,        // exit
    UNKNOWN      // unknown command
} COMMAND;

void print_menu() {
    printf("Menu\n");
    printf("1. Add node\n");
    printf("2. Delete node\n");
    printf("3. Print tree\n");
    printf("4. Check if three is B-tree\n");
    printf("5. Exit\n\n");
```

```
    printf("Please, choose command: ");
}

COMMAND interpret_command(int8_t c) {
    if (c == 1) {
        return ADD_NODE;
    } else if (c == 2) {
        return DELETE_NODE;
    } else if (c == 3) {
        return PRINT_TREE;
    } else if (c == 4) {
        return IS_B_TREE;
    } else if (c == 5) {
        return EXIT;
    } else {
        return UNKNOWN;
    }
}

SIDE interpret_side(char c) {
    if (c == 'L' || c == 'l') {
        return N_LEFT;
    } else if (c == 'R' || c == 'r') {
        return N_RIGHT;
    } else {
        return INVALID;
    }
}

int main(int64_t argc, char** argv) {

    int64_t line = 1;

    b_tree t = create_empty_b_tree();
    char* pEnd;

    COMMAND c; int32_t in, exit = 0;

    while (! exit) {

        print_menu();

        scanf("%d", &in);
        c = interpret_command(in);

        if (c == ADD_NODE) {
            if (is_empty_b_tree(&t)) {
                printf("Tree is empty now. Please enter value of root node: ");
                int64_t val; scanf("%ld", &val);
                init_b_tree(&t, val);
            } else {
                printf("Enter id of ancestor node: ");
                int64_t id; scanf("%ld", &id);
                printf("... and value of new node: ");
                int64_t val; scanf("%ld", &val);
                printf("... and side. Left or right? [L/r] ");
                SIDE s; char s_;
                scanf(" %c", &s_);
                s = interpret_side(s_);
                if (node_by_id(t.root, id) == NULL) {
                    printf("There is no node in tree with id %ld\n", id);
                } else {
                    add_node_by_id(&t, id, val, s);
                }
            }
        } else if (c == DELETE_NODE) {
            printf("Enter id of node to be deleted: ");
            int64_t id; scanf("%ld", &id);
            delete_node_by_id(&t, id);
        } else if (c == PRINT_TREE) {
            print_b_tree(t.root, 0, N_ROOT);
        } else if (c == IS_B_TREE) {
            printf("This tree is%s%c", (is_B_tree(t.root) ? " a B-tree" : " not a B-tree"), '\n');
        } else if (c == EXIT) {
            exit = 1;
        } else {
            printf("Wrong command.\n");
        }

        printf("-----------------------------------------\n");
    }

    clear_tree(&t);

    return 0;
```

```
}bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l23/a2$ gcc -g tree.c string.c main.c -o main
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l23/a2$ ./main
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 1
Tree is empty now. Please enter value of root node: 0
----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 1
Enter id of ancestor node: 0
... and value of new node: 1
... and side. Left or right? [L/r] l
----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 1
Enter id of ancestor node: 0
... and value of new node: 2
... and side. Left or right? [L/r] r
----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 1
Enter id of ancestor node: 1
... and value of new node: 3
... and side. Left or right? [L/r] l
----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 1
Enter id of ancestor node: 1
... and value of new node: 4
... and side. Left or right? [L/r] r
----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 3
0 |Root: 0
1 | |L: 1
3 | | |L: 3
4 | | |R: 4
2 | |R: 2
----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 4
This tree is a B-tree
----------------------------------------
Menu
```

```
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 1
Enter id of ancestor node: 2
... and value of new node: 5
... and side. Left or right? [L/r] l
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 3
0 |Root: 0
1 | |L: 1
3 | | |L: 3
4 | | |R: 4
2 | |R: 2
5 | | |L: 5
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 4
This tree is not a B-tree
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 2
Enter id of node to be deleted: 2
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 3
0 |Root: 0
1 | |L: 1
3 | | |L: 3
4 | | |R: 4
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 4
This tree is not a B-tree
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 2
Enter id of node to be deleted: 1
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit
```

```
Please, choose command: 3
0 |Root: 0
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 4
This tree is a B-tree
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 2
Enter id of node to be deleted: 0
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 3
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 1
Tree is empty now. Please enter value of root node: -10
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 1
Enter id of ancestor node: 0
... and value of new node: 1
... and side. Left or right? [L/r] l
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 4
This tree is not a B-tree
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 3
0 |Root: -10
1 | |L: 1
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 1
Enter id of ancestor node: 0
```

```
... and value of new node: 2
... and side. Left or right? [L/r] r
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 3
0 |Root: -10
1 | |L: 1
2 | |R: 2
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 4
This tree is a B-tree
-----------------------------------------
Menu
1. Add node
2. Delete node
3. Print tree
4. Check if three is B-tree
5. Exit

Please, choose command: 5
-----------------------------------------
bulat@bulat-Swift-SF314-58:~/Studying/prprm/l/l23/a2$ exit
exit
Script done.
```

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 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:** В ходе этой лабораторной работы я получил опыт реализации некоторых структур данных, работы с ними. _____

Недочёты при выполнении задания могут быть устранены следующим образом: _

Подпись студента _____