МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Бинарное дерево поиска»

Выполнил:

Студент группы 23ВВВ2

Монин Иван

Приняли:

доцент Юрова О.В.

доцент Митрохин М.А.

Пенза 2024

**Цель работы** – научиться создавать бинарные деревья и реализовать стандартные функции для них

**Лабораторное задание:**

**Задание 1**

Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.

**Задание 2**

Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

**Задание 3**\*

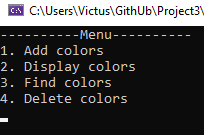
Изменить функцию добавления элементов для исключения добавления одинаковых символов.

**Задание 4**\*

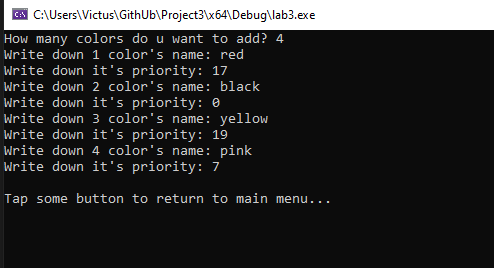
Оценить сложность процедуры поиска по значению в бинарном дереве.

**Задание 1**

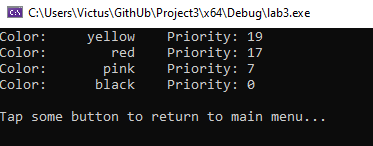
1. В начале предлагается меню с выбором:



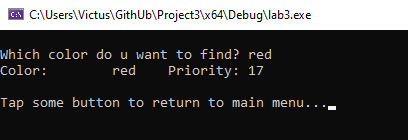
1. При выборе добавления цветов нужно указать кол-во добавляемых цветов, а затем ввести название цвета и приоритет



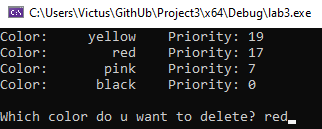
1. При выборе отображения цветов откроется список всех добавленных цветов



1. При поиске цвета необходимо ввести название



1. Во время удаления элемента так же нужно указать название цвета



**Листинг:**

**Задание 1**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <corecrt\_malloc.h>

#include <stdlib.h>

#include <string.h>

struct node {

char color[10];

int priority;

struct node\* last, \* next;

};

node\* find(char\* color, node\* head) {

struct node\* tmpHead = head;

struct node\* returningList = NULL;

struct node\* tmpRL = NULL;

while (tmpHead != NULL) {

if (strcmp(tmpHead->color, color) == 0) {

struct node\* tmp = (node\*)malloc(sizeof(struct node));

if (tmp == NULL) {

printf("Ошибка при распределении памяти\n");

exit(1);

}

strcpy(tmp->color, tmpHead->color);

tmp->priority = tmpHead->priority;

tmp->next = NULL;

if (returningList == NULL) {

returningList = tmp;

tmpRL = returningList;

}

else {

tmpRL->next = tmp;

tmpRL = tmp;

}

}

tmpHead = tmpHead->next;

}

return returningList;

}

void deleteElem(node\*\* head,node\* elems) {

struct node\* tempHead = \*head;

while (elems != NULL) {

while (tempHead != NULL) {

if (strcmp(tempHead->color, elems->color)==0) {

if (tempHead == \*head) {

if (tempHead->next != NULL) {

\*head = tempHead->next;

(\*head)->last = NULL;

}

else {

\*head = NULL;

break;

}

}

if (tempHead!=NULL && tempHead->last != NULL) {

struct node\* tmp = tempHead->last;

if (tempHead->next != NULL) {

tmp->next = tempHead->next;

tempHead = tempHead->next;

tempHead->last = tmp;

}

else {

tempHead = tempHead->next;

tmp->next = NULL;

}

}

}

else {

tempHead = tempHead->next;

}

}

elems = elems->next;

}

}

void addNewColor(char clr[10], int priority, node\*\* head) {

struct node\* temp = NULL;

struct node\* tmpHead = \*head;

if ((temp = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

strcpy(temp->color, clr);

temp->priority = priority;

temp->last = NULL;

temp->next = NULL;

while (1) {

if (tmpHead == NULL || tmpHead->priority < temp->priority) {

temp->next = tmpHead;

if (tmpHead != NULL && tmpHead->last != NULL) {

struct node\* lastElem = tmpHead->last;

temp->last = tmpHead->last;

lastElem->next = temp;

tmpHead->last = temp;

}

if (tmpHead == \*head) {

if (tmpHead != NULL)

tmpHead->last = temp;

\*head = temp;

}

break;

} else if(tmpHead->next!=NULL) tmpHead = tmpHead->next;

else

{

tmpHead->next = temp;

temp->last = tmpHead;

break;

}

}

}

void displayList(node\* head) {

while (head != NULL) {

printf("Color: %10s Priority: %d\n", head->color, head->priority);

head = head->next;

}

}

void displayMenu(node\* head) {

char color[10];

int priority;

struct node\* findedElms;

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

printf("1. Add colors\n");

printf("2. Display colors\n");

printf("3. Find colors\n");

printf("4. Delete colors\n");

int choice;

scanf("%d", &choice);

system("cls");

switch (choice) {

case(1):

int count;

printf("How many colors do u want to add? ");

scanf("%d", &count);

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

printf("Write down %d color's name: ", i + 1);

scanf("%s", color);

printf("Write down it's priority: ");

scanf("%d", &priority);

addNewColor(color, priority, &head);

}

printf("\nTap some button to return to main menu...");

getchar();

system("cls");

displayMenu(head);

break;

case(2):

displayList(head);

printf("\nTap some button to return to main menu...");

getchar();

getchar();

system("cls");

displayMenu(head);

break;

case(3):

printf("\nWhich color do u want to find? ");

scanf("%s", color);

findedElms = find(color, head);

displayList(findedElms);

printf("\nTap some button to return to main menu...");

getchar();

getchar();

system("cls");

displayMenu(head);

break;

case(4):

char answer[10];

displayList(head);

printf("\nWhich color do u want to delete? ");

scanf("%s", color);

findedElms = find(color, head);

deleteElem(&head, findedElms);

printf("\nTap some button to return to main menu...");

getchar();

system("cls");

displayMenu(head);

break;

}

}

void main() {

struct node\* head = NULL;

displayMenu(head);

}

**Задание 2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Node {

char name[20];

struct Node\* next,\*prev;

} Node;

typedef struct Queue {

Node\* head;

Node\* tail;

} Queue;

void addElem(Queue\* q, char \*name) {

Node\* tmp = (Node\*)malloc(sizeof(Node));

strcpy(tmp->name,name);

tmp->next = NULL;

tmp->prev = NULL;

if (q->tail == NULL) {

q->head = tmp;

q->tail = tmp;

}

else {

q->tail->next = tmp;

tmp->prev = q->tail;

q->tail = tmp;

}

}

void deleteElem(Queue\* q) {

if (q->head == NULL) {

printf("Queue is empty!\n");

exit(1);

}

q->head = q->head->next;

if (q->head == NULL) {

q->tail = NULL;

}

}

void displayElems(Queue \*q) {

Node\* tmp = q->head;

while (tmp) {

printf("%s ", tmp->name);

tmp = tmp->next;

}

}

void displayMenu(Queue\* q) {

char name[20];

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

printf("1. Add name\n");

printf("2. Display names\n");

printf("3. Delete name\n");

int choice;

scanf("%d", &choice);

system("cls");

switch (choice) {

case(1):

int count;

printf("How many names do u want to add? ");

scanf("%d", &count);

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

printf("Write down %d name: ", i + 1);

scanf("%s", name);

addElem(q,name);

}

printf("\nTap some button to return to main menu...");

getchar();

system("cls");

displayMenu(q);

break;

case(2):

displayElems(q);

printf("\nTap some button to return to main menu...");

getchar();

getchar();

system("cls");

displayMenu(q);

break;

case(3):

deleteElem(q);

printf("\nThe element deleted.\nTap some button to return to main menu...");

getchar();

system("cls");

displayMenu(q);

break;

}

}

int main() {

Queue\* q = (Queue\*)malloc(sizeof(Queue));

q->head = NULL;

q->tail = NULL;

displayMenu(q);

return 0;

}

**Задание 2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Node {

char name[20];

struct Node\* next, \* prev;

} Node;

typedef struct Stack {

Node\* head;

Node\* tail;

} Stack;

void addElem(Stack\* q, char\* name) {

Node\* tmp = (Node\*)malloc(sizeof(Node));

strcpy(tmp->name, name);

tmp->next = NULL;

tmp->prev = NULL;

if (q->tail == NULL) {

q->head = tmp;

q->tail = tmp;

}

else {

q->tail->next = tmp;

tmp->prev = q->tail;

q->tail = tmp;

}

}

void deleteElem(Stack\* q) {

if (q->head == NULL) {

printf("Stack is empty!\n");

exit(1);

}

if (q->tail->prev != NULL) {

q->tail = q->tail->prev;

q->tail->next = NULL;

}

else q->tail = NULL;

if (q->tail == NULL) {

q->head = NULL;

}

}

void displayElems(Stack\* q) {

Node\* tmp = q->head;

while (tmp) {

printf("%s ", tmp->name);

tmp = tmp->next;

}

}

void displayMenu(Stack\* q) {

char name[20];

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

printf("1. Add name\n");

printf("2. Display names\n");

printf("3. Delete name\n");

int choice;

scanf("%d", &choice);

system("cls");

switch (choice) {

case(1):

int count;

printf("How many names do u want to add? ");

scanf("%d", &count);

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

printf("Write down %d name: ", i + 1);

scanf("%s", name);

addElem(q, name);

}

printf("\nTap some button to return to main menu...");

getchar();

system("cls");

displayMenu(q);

break;

case(2):

displayElems(q);

printf("\nTap some button to return to main menu...");

getchar();

getchar();

system("cls");

displayMenu(q);

break;

case(3):

deleteElem(q);

printf("\nThe element deleted.\nTap some button to return to main menu...");

getchar();

system("cls");

displayMenu(q);

break;

}

}

int main() {

Stack\* q = (Stack\*)malloc(sizeof(Stack));

q->head = NULL;

q->tail = NULL;

displayMenu(q);

return 0;

}

**Вывод**

В ходе работы выполнения лабораторной работы была рассчитана сложность программы (О-символикой). Так же было выполнено измерение времени выполнения перемножения массивов и разных видов сортировки.