Министерство науки и высшего образования
Пензенский государственный университет
Кафедра “Вычислительная техника”

Отчет

по лабораторной работе №4
по курсу “ Логика и основы алгоритмизации в инженерных задачах”
на тему “Оценка времени выполнения программ”

Выполнили
студенты группы 22ВВП2:
Гавин В.Н.
Дулатов Д.А.
Приняли
Акифьев И.В.
Юрова О.В.

Пенза 2023

Задание:

1. Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.
2. Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.
3. *Изменить функцию добавления элементов для исключения добавления одинаковых символов.
4. *Оценить сложность процедуры поиска по значению в бинарном дереве.

Решение заданий

Задание 1:

В программу была добавлена функция Search, которая выполняет поиск заданного значения в дереве. После построения дерева пользователь может ввести значение для поиска, и программа сообщит, найдено ли оно в дереве или нет.

```
struct Node *Search(struct Node *root, int target) {
    if (root == NULL || root->data == target)
        return root;

    if (target < root->data)
        return Search(root->left, target);
    else
        return Search(root->right, target);
}
```

В функцию main было добавлено:

```
   struct Node *result = Search(root, D);
   if (result != NULL) {
        printf("Значение %d найдено в дереве.\n", D);
    } else {
        printf("Значение %d не найдено в дереве.\n", D);
    }
```

Задание 2:

В программу была добавлена функция CountOccurrences, которая рекурсивно подсчитывает количество вхождений заданного значения в

дерево. После построения дерева пользователь может ввести значение для подсчета, и программа выведет количество его вхождений в дереве.

```
int CountOccurrences(struct Node *root, int target) {
    if (root == NULL)
        return 0;

    if (root->data == target)
        return 1 + CountOccurrences(root->right, target);
    else if (target < root->data)
        return CountOccurrences(root->left, target);
    else
        return CountOccurrences(root->right, target);
}
```

В функцию main было добавлено:

```
   printf("Введите значение для подсчета вхождений: ");
    scanf("%d", &D);

    int count = CountOccurrences(root, D);

         printf("Значение %d встречается %d раз(а) в
дереве.\n", D, count);
```

Задание 3:

В этой версии программы была модифицирована функция CreateTree так, чтобы она не добавляла дублирующиеся значения.

```
struct Node *CreateTree(struct Node *root, int data) {
    if (root == NULL) {
            root = (struct Node *)malloc(sizeof(struct
Node));
        if (root == NULL) {
            printf("Ошибка выделения памяти");
            exit(0);
        }
        root->left = NULL;
        root->right = NULL;
        root->data = data;
        return root;
    }

    if (data < root->data) {
        root->left = CreateTree(root->left, data);
    } else if (data > root->data) {
        root->right = CreateTree(root->right, data);
```

```
    } // Игнорируем дублирующиеся значения

    return root;
}
```

Задание 4:

Сложность процедуры поиска Search в бинарном дереве поиска зависит от высоты дерева и может быть оценена как O(h), где h - высота дерева.

В лучшем случае, когда дерево сбалансировано, высота дерева будет O(log n), где n - количество узлов в дереве. В этом случае сложность поиска будет O(log n).

В наихудшем случае, когда дерево является вырожденным (все узлы идут в одну из ветвей), высота дерева будет равна n, и сложность поиска составит O(n).

Листинг

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

struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

struct Node *CreateTree(struct Node *root, int data) {
    if (root == NULL) {
            root = (struct Node *)malloc(sizeof(struct
Node));
        if (root == NULL) {
            printf("Ошибка выделения памяти");
            exit(0);
        }
        root->left = NULL;
        root->right = NULL;
        root->data = data;
        return root;
```

```
        }

        if (data < root->data) {
            root->left = CreateTree(root->left, data);
        } else if (data > root->data) {
            root->right = CreateTree(root->right, data);
        } // Игнорируем дублирующиеся значения

        return root;
    }

    int CountOccurrences(struct Node *root, int target) {
        if (root == NULL)
            return 0;

        if (root->data == target)
             return 1 + CountOccurrences(root->left, target) +
CountOccurrences(root->right, target);
        else if (target < root->data)
            return CountOccurrences(root->left, target);
        else
            return CountOccurrences(root->right, target);
    }

    void print_tree(struct Node *r, int l) {
        if (r == NULL)
            return;

        print_tree(r->right, l + 1);
        for (int i = 0; i < l; i++) {
            printf(" ");
        }
        printf("%d\n", r->data);
        print_tree(r->left, l + 1);
    }

    int main() {
        setlocale(LC_ALL, "");
        int D, start = 1;

        struct Node *root = NULL;

        printf("-1 - окончание построения дерева\n");
        while (start) {
```

```
            printf("Введите число: ");
            scanf("%d", &D);
            if (D == -1) {
                printf("Построение дерева окончено\n\n");
                start = 0;
            } else {
                root = CreateTree(root, D);
            }
        }

        print_tree(root, 0);

        printf("Введите значение для подсчета вхождений: ");
        scanf("%d", &D);

        int count = CountOccurrences(root, D);

             printf("Значение %d встречается %d раз(а) в
дереве.\n", D, count);

        return 0;
    }
```

Результаты работы программы

```
Вв◆дите число: 20
Введите число: 1
Введите число: 2
Введите число: 2
Введите число: 2
Введите число: 2
Введите число: 2
Введите число: 2
Введите число: 2
Введите число: 2
Введите число: 23
Введите число: 3
Введите число: 4
Введите число: 5
Введите число: 6
Введите число: 7
Введите число: 7
Введите число: 21
Введите число: 34
Введите число: 35
Введите число: 36
Введите число: 37
Введите число: 38
Введите число: -1
Построение дерев◆окончено

      38
     37
    36
   35
  34
 23
  21
20
       7
      6
     5
    4
   3
  2
 1
Введите значение для подсчет◆вхождений: 3
Значение 3 встреч◆ется 1 раз(а) в дереве.
```

Вывод

В результате выполнения лабораторной работы были успешно реализованы алгоритмы поиска, подсчёта вхождений и добавления элементов в бинарное дерево поиска, а также оценена сложность процедуры поиска.