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

Отчёт

по лабораторной работе №4
по курсу “Логика и основы алгоритмизации в инженерных задачах”
на тему “Бинарное дерево поиска”

Выполнил студент гр. 22ВВВ3:
Кулахметов С.И.

Приняли:
к.т.н., доцент Юрова О.В.
к.э.н., доцент Акифьев И.В.

Пенза 2023

Цель работы

Ознакомиться с принципами работы алгоритма упорядоченной структуры данных «Бинарное дерево». Выполнить лабораторное задание.

Лабораторное 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.

Пояснительный текст к прораммам

Лабораторное задание выполнено в виде двух программ.

В первой программе выполняются задания 1 и 2. Значения в бинарном дереве генерируются псевдорандомом.

Во второй программе выполняется задание 3 и результаты вводятся вручную для более показательной демонстрации обработки варианта недопустимости ввода уже существующих значений.

Программы базируются на следующей структуре.

```
struct Node
{
    int data;    // Данные
    struct Node* left;  // Указатель на левого потомка
    struct Node* right; // Указатель на правого потомка
};
```

Результаты работы программ

Данные генерируются псевдорадномом в следующем порядке:

1) -2
2) 4
3) -6
4) -5
5) -3
6) -8

7) -9

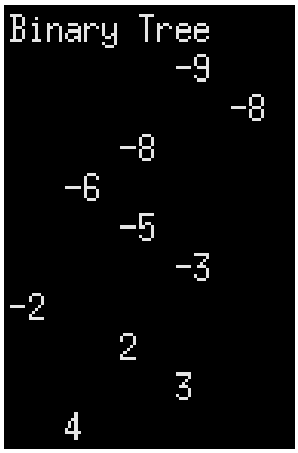
8) -8

9) 2

10) 3

(см. Приложение Б рис. 1)

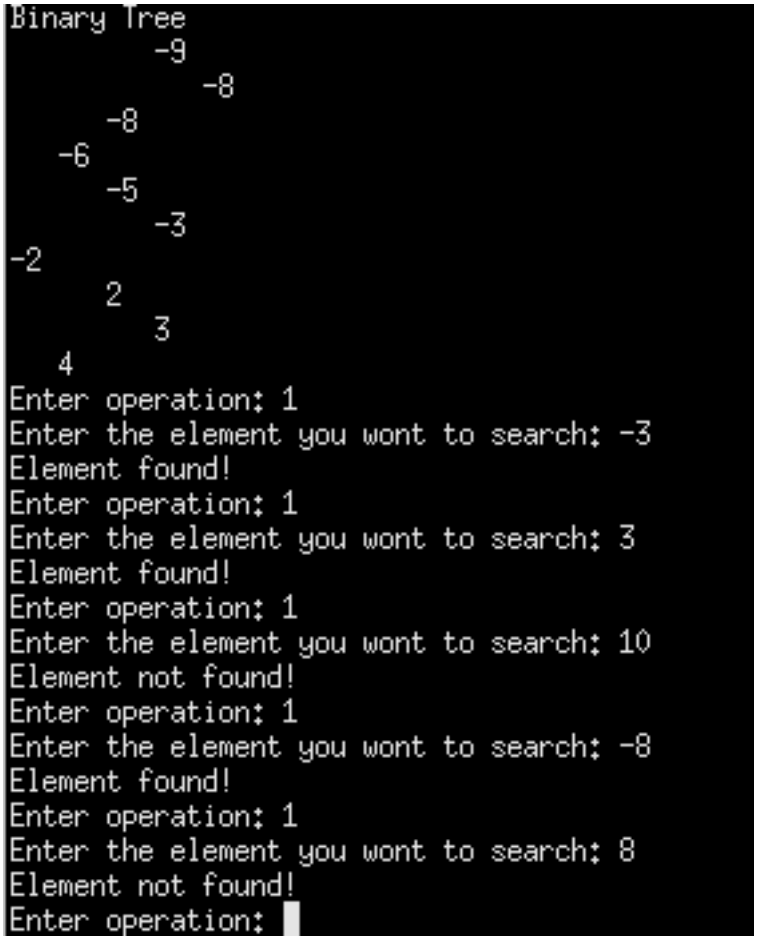
Результат построения бинарного дерева.



Стоит также учесть, что для простоты вывода дерево повёрнуто на 90°.

Задание 1

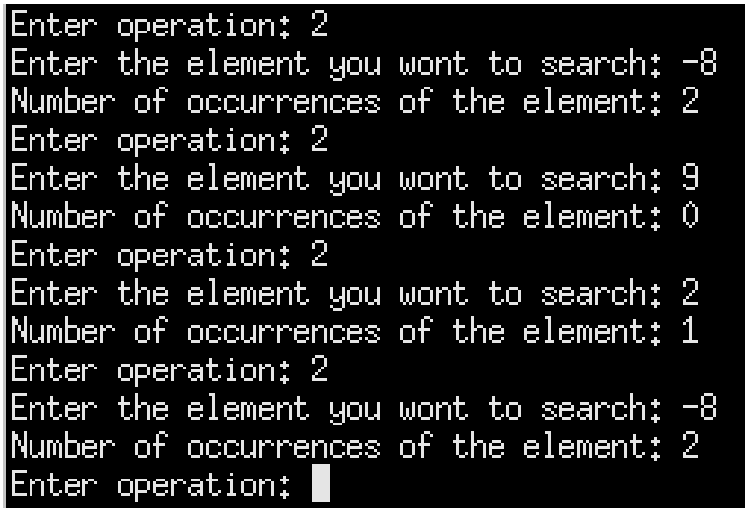
Поиск вводимого с клавиатуры значения в уже существующем дереве.



Если введённый элемент есть в бинарном дереве, то программа выводит «Element found!», в ином случает - «Element not found!».

Задание 2

Подсчёт числа вхождений числа в структуру данных.



В результате работы алгоритма показывается число вхождений элемента в структуру данных. Если же данный элемент в структуру не входит, то его число вхождений = 0.

Задание 3

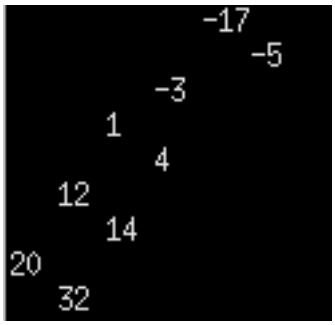
Данные заносятся в бинарное дерево в следующем порядке:

1) 20
2) 12
3) 1
4) 4
5) -3
6) 20
7) 1
8) -17
9) 14
10) 32
11) 4
12) -5

(см. Приложение Б рис. 2)

При вводе одинаковых элементов программа выводит предупреждение «Element already exists» и попросту не добавляет его в структуру.

Результат построения бинарного дерева с учётом исключения повтора элементов.



Задание 4

Сложность процедуры поиска в бинарном дереве по значению O(2N), где N - число рекурсивных вызовов, так как операция сравнения происходит столько же раз, сколько и вызывается функция. Умножене на 2 происходит из-за вызова рекурсии внутрии самой себя дважды.

```
void SearchElement(struct Node *pointer, int element)
{
    if (pointer == NULL)
    {
        return;
    }

    SearchElement(pointer->right, element);

    if(pointer -> data == element) finding = 1;

    SearchElement(pointer->left, element);
}
```

Вывод

В ходе выполнения данной лабораторной работы были получены навыки реализации упорядоченной структуры данных «Бинарное дерево» на языке программирования Си, и реализации рекурсивных алгоритмов, в целом.

Ссылка на *GitHub* репозиторий с лабораторной работой

https://github.com/KulakhmetovS/Lab4

Приложение А

Листинг программ

Файл BinaryTree/main.c

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

struct Node;    // Структура, отвечающая за бинарное дерево
struct Node *CreateTree(struct Node *, struct Node *, int); // Создание и добавление элемента
void PrintTree(struct Node *, int); // Вывод дерева в консоль
void SearchElement(struct Node *, int); // Поиск введённого элемента
void Occurrence(struct Node *, int);    // Поиск числа вхождений элементов

int finding, entry = 0; // переменные наличия элемента и числа вхождений


int main()
{
    srand(time(NULL));
    struct Node *root = NULL;
    int D, amount = 0, operation = 0, element = 0;

    printf("\t# Binary Tree #\n\n");
     printf(" 0 - Quit\n 1 - Search the element\n 2 - Occurrence of element\n\n");
    printf("Enter amount of elements: ");
    scanf("%d", &amount);

 for(int i = 0; i < amount; i++)
 {
       D = rand() % 21 - 10;
         root = CreateTree(root, root, D);
         printf("Element: %d\n", D);
 }
    printf("Binary Tree\n");
 PrintTree(root,0);

    label:
 printf("Enter operation: ");
 scanf("%d", &operation);

 switch(operation)
 {
 case 0:
        return 0;
 break;

 case 1: // Первая задача: поиск элемента, введённого с клавиатуры
        finding = 0;
        printf("Enter the element you wont to search: ");
        scanf("%d", &element);
        SearchElement(root, element);
        if(finding == 1) printf("Element found!\n");
        else if(finding == 0) printf("Element not found!\n");
 break;

 case 2: // Вторая задача: число вхождений элемента
        printf("Enter the element you wont to search: ");
        scanf("%d", &element);
        Occurrence(root, element);
        printf("Number of occurrences of the element: %d\n", entry);
        entry = 0;
 break;
```

```
      default:
      printf("Invalid operation!\n");
      }
      goto label;

          return 0;
      }

      struct Node
      {
          int data;   // Данные
          struct Node* left;  // Указатель на левого потомка
          struct Node* right; // Указатель на правого потомка
      };

      struct Node *CreateTree(struct Node *root, struct Node *pointer, int data)
      {
          if (pointer == NULL)
       {
              pointer = (struct Node *)malloc(sizeof(struct Node));    // Выделение
памяти

              pointer->left = NULL;
              pointer->right = NULL;
              pointer->data = data;
              if (root == NULL) return pointer;

              if (data > root->data) root->left = pointer;    // Если данные больше
данных корня, запись идёт в левый потомок
              else root->right = pointer; // Иначе, запись производится в правый
потомок
              return pointer;
       }

       if (data > pointer->data)     //Если данные больше родителя, они будут
записываться в левый потомок
              CreateTree(pointer, pointer->left, data);
       else    // Иначе, запись производится в правый потомок
              CreateTree(pointer, pointer->right, data);

       return root;
      }

      void PrintTree(struct Node *pointer, int l)
      {
          if (pointer == NULL)    // Если родитель указывает на NULL, то return
       {
              return;
       }

       PrintTree(pointer->right, l + 1);     // В качестве родителя передаётся
указатель на правого потомка
       for(int i = 0; i < l; i++)
       {
              printf("   ");
       }

       printf("%d\n", pointer->data);
       PrintTree(pointer->left, l+1); // В качестве родителя передаётся указатель на
левого потомка
      }

      void SearchElement(struct Node *pointer, int element)
      {
          if (pointer == NULL)
       {
              return;
       }
```

```
 SearchElement(pointer->right, element);

 if(pointer -> data == element) finding = 1;

 SearchElement(pointer->left, element);
}

void Occurrence(struct Node *pointer, int element)
{
     if (pointer == NULL)
 {
       return;
 }

 Occurrence(pointer->right, element);

 if(pointer -> data == element) entry++;

 Occurrence(pointer->left, element);
}
```

Файл BinaryTree1/main.c

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

struct Node;    // Структура, отвечающая за бинарное дерево
struct Node *CreateTree(struct Node *, struct Node *, int); // Создание
и добавление элемента
void PrintTree(struct Node *, int); // Вывод дерева в консоль
void CheckRepeat(struct Node *, int);   // Проверка на совпадение
элементов

int entry = 0;

int main()
{
    struct Node *root = NULL;
    int D;
    float cont = 1;

    printf("\t# Binary Tree #\n\n");
 while(1)
 {
       printf("Enter integer: ");
       scanf("%d", &D);
       CheckRepeat(root, D);
       if(entry == 0)
       {
             root = CreateTree(root, root, D);
             printf("Continue entering(any integer) or not(0): ");
             scanf("%f", &cont);
         }
         else printf("Element already exists!\n");
         entry = 0;
         if(cont == 0) break;
 }
```

```
     PrintTree(root,0);

         return 0;
     }

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

     struct Node *CreateTree(struct Node *root, struct Node *pointer, int
data)
     {
         if (pointer == NULL)
      {
           pointer = (struct Node *)malloc(sizeof(struct Node));

           pointer->left = NULL;
           pointer->right = NULL;
           pointer->data = data;
           if (root == NULL) return pointer;

           if (data > root->data)     root->left = pointer;
           else root->right = pointer;
           return pointer;
      }

      if (data > pointer->data)
           CreateTree(pointer, pointer->left, data);
      else
           CreateTree(pointer, pointer->right, data);

      return root;
     }

     void PrintTree(struct Node *pointer, int l)
     {
         if (pointer == NULL)
      {
           return;
      }

      PrintTree(pointer->right, l + 1);
      for(int i = 0; i < l; i++)
      {
           printf("   ");
      }

      printf("%d\n", pointer->data);
      PrintTree(pointer->left,  l+1);
     }

     void CheckRepeat(struct Node *pointer, int element)
     {
         if (pointer == NULL)
      {
           return;
      }

      CheckRepeat(pointer->right, element);
```

```
 if(pointer -> data == element) entry++;

 CheckRepeat(pointer->left, element);
}
```

Приложение Б

Внесение данных в бинарное дерево

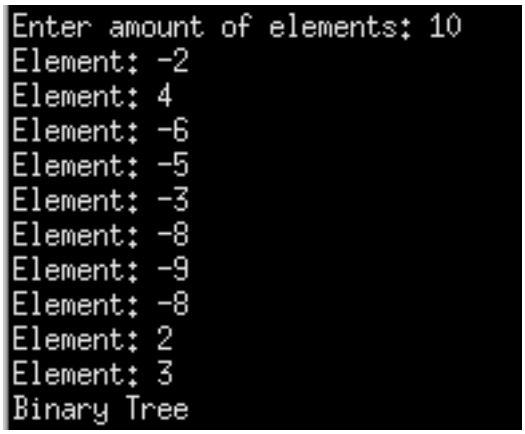


Рисунок 1 — Генерация данных для бинарного дерева (задания 1 и 2)

Рисунок 2 — Внесение данных в бинарное дерево (задание 3)