Отчет по лабораторной работе №23 по курсу

Языки и методы программирования _____

Студент группы М8О-101Б-21 Постнов Александр Вячеславович, № по списку 17

Контакты www, e-mail: 61pav03@mail.ru

Работа выполнена: «» 202 2г.

Преподаватель: ___ каф. 806 _____Титов В.К._____

Входной контроль знаний с оценкой _____

Отчет сдан « » _____ 2022_ г., итоговая оценка _____

Подпись преподавателя _____

1. **Тема:** Динамические структуры данных. Обработка деревьев._____
   _____
2. **Цель работы:** Составить программу на языке Си для построения и обработки дерева общего вида или упорядоченного двоичного дерева._____
   _____
   _____
3. **Задание** (*вариант 17 + 8 = 25*)**:** Определить глубину двоичного дерева.
4. **Оборудование(лабораторное)**:
   ЭВМ -, процессор -, имя узла сети - с ОП - ГБ,
   НМД - ГБ, терминал- адрес -, принтер -
   Другие устройства -

   Оборудование ПЭВМ студента, если использовалось:
   Процессор AMD Ryzen 5 4500U, с ОП 8 ГБ
   Другие устройства -
5. **Программное обеспечение:**
   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 GNU/Linux, наименование Manjaro версия 5-13-12-1
   интерпретатор команд GNOME Terminal версия 3.38.2.
   Система программирования _____версия_____
   Редактор текстов emacs версия 3.27.20
   Утилиты операционной системы
   Прикладные системы и программы -
   Местонахождение и имена файлов программ и данных -
6. **Идея, метод, алгоритм** решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)

Двоичное дерево поиска (англ. binary search tree, BST) — двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

оба поддерева — левое и правое — являются двоичными деревьями поиска;

у всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X;

у всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны(в моей реализации строго больше), нежели значение ключа данных самого узла X.

Рассмотрим реализацию двоичного дерева поиска на языке Си:

1) Создадим структуру node(вершины дерева или узла дерева). Она содержит(вершина):

   1) поле data типа tdata(tdata можно любым типом, в данном случае int) - значение вершины

   2) указатель на левое поддерево - link left

   3) указатель на правое поддерево - link right

2) Рассмотрим основные функции для работы с двоичным деревом поиска:

   1) insert_tree(link &t, tdata v).(вставка значения v в дерево t) 1) Если вершина пустая, то создаем новую вершину: значение вершины = v, она терминальная, поэтому указатели указывают на 0; 2) Если значение больше значения текущей вершины, то запускаем функцию для правой вершины(рекурсия) 3) Если значение меньше значения текущей вершины, то запускаем функцию для левой вершины(insert_tree(t -> left, v). Рекурсия будет происходить до тех пор, пока не встретит пустую вершину.

   2) print_tree(link t) - вывод дерева на экран. Чтобы вывести дерево, заводим переменную, чтобы запоминать текущее значение глубины. Вызываем рекурсию правого поддерева, обрабатываем текущую вершину(слева дописываем пробелы в зависимости от значения глубины, и пишем само значение вершины), вызываем рекурсию левого поддерева. Также стоит отметить, что переменная, которая запоминает значение глубины статическая, т.е при вызове функции она сохраняет свое значение). Также при каждом вызове функции увеличиваем значение переменной глубины на единичку, а в конце отнимаем единицу(так как уже выходим из этого значения глубины наверх). Рекурсия вызывается до тех пор, пока не встретит пустую вершину.

   3) delete_tree(link &tree, tdata v) - удаление элемента в дереве.

   Идея удаления элемента делится на несколько случаев:

   у вершины нет дочерних вершин;

   у вершины есть левый дочерних узлов;

   у вершины есть правый дочерних узлов;

   у вершины есть оба ребёнка.

   В случае 1 просто удаляем узел, дополнительная работа не требуется.

   В случае 2 и 3 заменяем удаляемый узел на его потомка, на этом удаление заканчивается.

   В случае 4 находим в левом поддереве максимальный элемент и перемещаем его на место удаляемого.

   4) void count(link t) - подсчитываем кол-во вершин. Просто совершаем стандартный рекурсивный обход и считаем кол-во вершин

   5) void add_tree(link &t, int n) - создание дерева с n вершинами. Оно будет создать с помощью генератора рандомных чисел и добавление в дерево с помощью функции insert_tree(описана выше)

3) Опишу дополнительную функцию, которую мне нужно сделать.

   void depth_tree(link t) - подсчитывает глубину двоичного дерева. Используем ту же идею, что и для print_tree, считаем для каждой вершины текущую глубину, но записываем в переменную max_depth максимальную из них. Вызываем рекурсивный обход.

4) В функции main будет проводиться работа с деревом с помощью меню, в котором описаны функции, которые реализованы.

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

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

```
#include <stdlib.h>
#include <time.h>
#define N 20

typedef int tdata;

int i, s, max_depth = 0;

struct node;

typedef node *link;

int max(int a, int b) {
   if (a > b) {
      return a;
   }
   return b;
}

struct node {
  tdata data;
  link left;
  link right;
} * tree;

void print_tree(link t) {
  static int l = 0;
  l++;
  if (t) {
    print_tree(t->right);
    for (i = 0; i < l; i++)
      printf("    ");
    printf("\\__%d\n", t->data);
    print_tree(t->left);
  }
  l--;
}

void insert_tree(link &t, tdata v)
{
  if (!t) {
    t = new node;
    t->data = v;
    t->left = 0;
    t->right = 0;
  } else {
    if (v < t->data)
      insert_tree(t->left, v);
    else if (v > t->data)
      insert_tree(t->right, v);
  }
}

link q;
```

```
void del(link &);

void delete_tree(link &tree, tdata v) {
 if (tree)
   if (v < tree->data)
     delete_tree(tree->left, v);
   else if (v > tree->data)
     delete_tree(tree->right, v);
   else if (!(tree->right))
     tree = tree->left;
   else if (!(tree->left))
     tree = tree->right;
   else {
     q = tree;
     del(q->left);
   }
}

void del(link &t) {
 if (t->right)
   del(t->right);
 else {
   q->data = t->data;
   q = t;
   t = t->left;
 }
}

void add_tree(link &t, int n) {
 for (i = 0; i < n; i++) {
   int v = rand() % 120 + 1;
   insert_tree(tree, v);
 }
}

void count(link t) {
 if (t) {
   count(t->right);
   s++;
   count(t->left);
 }
}

void depth_tree(link t) {
 static int l = 0;
 max_depth = max(l, max_depth);
 l++;
 if (t) {
   depth_tree(t->right);
   depth_tree(t->left);
 }
 l--;
}
```

```
int main() {
  time_t t;
  srand(time(&t));
  int k = 1, n;
  tree = 0;
  tdata v;
  while (k) {
    printf("\n    MENU\n 0 - exit\n 1 - add random tree"
           "\n 2 - print tree\n 3 - insert item"
           "\n 4 - delete item\n 5 - number of nodes"
           "\n 6 - depth(особое действие по варианту)"
           "\n 7 - clear tree"
           "\n 8 - exit\n ==>");
    scanf("%d", &k);
    if (!k)
      break;
    if (k == 1) {
      printf("\nInput number of items: ==>");
      scanf("%d", &n);
      add_tree(tree, n);
    }
    if (k == 2)
      if (tree)
        print_tree(tree);
      else
        printf("\nTree is empty.\n");
    if (k == 3) {
      printf("For insert Input v=");
      scanf("%d", &v);
      insert_tree(tree, v);
    }
    if (k == 4) {
      printf("For delete Input v=");
      scanf("%d", &v);
      delete_tree(tree, v);
    }
    if (k == 5) {
      s = 0;
      count(tree);
      printf("\nNumber of nodes =%d\n", s);
    }
    if (k == 7)
      tree = 0;
    if (k == 6) {
       max_depth = 0;
       depth_tree(tree);
       printf("\n Depth = %d\n", max_depth);
    }
    if (k == 8) {
       return 0;
    }
  }
  return 0;
```

}

Тестирование:

Буду создавать случайное дерево и определять глубину. Буду добавлять элементы, удалять. С помощью ручного подсчета буду сравнивать ответ, который будет давать программа.

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).

```
[alex@fedora 23(?)]$ cat head.txt
-----------------------------------------------------
|       Лабораторная работа №23                     |
|       Обработка деревьев                  |
|       Выполнил:  студент группы М8О-101Б-21       |
|       Постнов Александр Вячеславович             |
-----------------------------------------------------
[alex@fedora 23(?)]$ cat main.cpp
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 20

typedef int tdata;

int i, s, max_depth = 0;

struct node;

typedef node *link;

int max(int a, int b) {
  if (a > b) {
    return a;
```

```
    }
    return b;
}

struct node {
  tdata data;
  link left;
  link right;
} * tree;

void print_tree(link t) {
  static int l = 0;
  l++;
  if (t) {
    print_tree(t->right);
    for (i = 0; i < l; i++)
      printf("    ");
    printf("\\__%d\n", t->data);
    print_tree(t->left);
  }
  l--;
}

void insert_tree(link &t, tdata v)
{
  if (!t) {
    t = new node;
    t->data = v;
    t->left = 0;
    t->right = 0;
  } else {
    if (v < t->data)
      insert_tree(t->left, v);
    else if (v > t->data)
      insert_tree(t->right, v);
  }
}

link q;

void del(link &);

void delete_tree(link &tree, tdata v) {
  if (tree)
    if (v < tree->data)
      delete_tree(tree->left, v);
    else if (v > tree->data)
      delete_tree(tree->right, v);
    else if (!(tree->right))
      tree = tree->left;
    else if (!(tree->left))
      tree = tree->right;
    else {
      q = tree;
```

```
    del(q->left);
   }
}

void del(link &t) {
 if (t->right)
   del(t->right);
 else {
   q->data = t->data;
   q = t;
   t = t->left;
 }
}

void add_tree(link &t, int n) {
 for (i = 0; i < n; i++) {
   int v = rand() % 120 + 1;
   insert_tree(tree, v);
 }
}

void count(link t) {
 if (t) {
   count(t->right);
   s++;
   count(t->left);
 }
}

void depth_tree(link t) {
 static int l = 0;
 max_depth = max(l, max_depth);
 l++;
 if (t) {
   depth_tree(t->right);
   depth_tree(t->left);
 }
 l--;
}

int main() {
 time_t t;
 srand(time(&t));
 int k = 1, n;
 tree = 0;
 tdata v;
 while (k) {
   printf("\n    MENU\n 0 - exit\n 1 - add random tree"
        "\n 2 - print tree\n 3 - insert item"
        "\n 4 - delete item\n 5 - number of nodes"
        "\n 6 - depth(особое действие по варианту)"
        "\n 7 - clear tree"
        "\n 8 - exit\n ==>");
   scanf("%d", &k);
```

```
   if (!k)
     break;
   if (k == 1) {
     printf("\nInput number of items: ==>");
     scanf("%d", &n);
     add_tree(tree, n);
   }
   if (k == 2)
     if (tree)
       print_tree(tree);
     else
       printf("\nTree is empty.\n");
   if (k == 3) {
     printf("For insert Input v=");
     scanf("%d", &v);
     insert_tree(tree, v);
   }
   if (k == 4) {
     printf("For delete Input v=");
     scanf("%d", &v);
     delete_tree(tree, v);
   }
   if (k == 5) {
     s = 0;
     count(tree);
     printf("\nNumber of nodes =%d\n", s);
   }
   if (k == 7)
     tree = 0;
   if (k == 6) {
       max_depth = 0;
       depth_tree(tree);
       printf("\n Depth = %d\n", max_depth);
   }
   if (k == 8) {
       return 0;
   }
 }
 return 0;
}
[alex@fedora 23(?)]$ g++ main.cpp
[alex@fedora 23(?)]$ ./a.out

   MENU
 0 - exit
 1 - add random tree
 2 - print tree
 3 - insert item
 4 - delete item
 5 - number of nodes
 6 - depth(особое действие по варианту)
 7 - clear tree
 8 - exit
 ==>1
```

```
Input number of items: ==>3

   MENU
 0 - exit
 1 - add random tree
 2 - print tree
 3 - insert item
 4 - delete item
 5 - number of nodes
 6 - depth(особое действие по варианту)
 7 - clear tree
 8 - exit
 ==>2
      \__81
   \__80
      \__53

   MENU
 0 - exit
 1 - add random tree
 2 - print tree
 3 - insert item
 4 - delete item
 5 - number of nodes
 6 - depth(особое действие по варианту)
 7 - clear tree
 8 - exit
 ==>6

 Depth = 2

   MENU
 0 - exit
 1 - add random tree
 2 - print tree
 3 - insert item
 4 - delete item
 5 - number of nodes
 6 - depth(особое действие по варианту)
 7 - clear tree
 8 - exit
 ==>4
For delete Input v=80

   MENU
 0 - exit
 1 - add random tree
 2 - print tree
 3 - insert item
 4 - delete item
 5 - number of nodes
 6 - depth(особое действие по варианту)
 7 - clear tree
```

8 - exit
==>2
    __81
  __53

  MENU
0 - exit
1 - add random tree
2 - print tree
3 - insert item
4 - delete item
5 - number of nodes
6 - depth(особое действие по варианту)
7 - clear tree
8 - exit
==>6

Depth = 2

  MENU
0 - exit
1 - add random tree
2 - print tree
3 - insert item
4 - delete item
5 - number of nodes
6 - depth(особое действие по варианту)
7 - clear tree
8 - exit
==>4
For delete Input v=53

  MENU
0 - exit
1 - add random tree
2 - print tree
3 - insert item
4 - delete item
5 - number of nodes
6 - depth(особое действие по варианту)
7 - clear tree
8 - exit
==>2
  __81

  MENU
0 - exit
1 - add random tree
2 - print tree
3 - insert item
4 - delete item
5 - number of nodes
6 - depth(особое действие по варианту)
7 - clear tree

```
8 - exit
==>6

Depth = 1

  MENU
0 - exit
1 - add random tree
2 - print tree
3 - insert item
4 - delete item
5 - number of nodes
6 - depth(особое действие по варианту)
7 - clear tree
8 - exit
==>7

  MENU
0 - exit
1 - add random tree
2 - print tree
3 - insert item
4 - delete item
5 - number of nodes
6 - depth(особое действие по варианту)
7 - clear tree
8 - exit
==>1

Input number of items: ==>10

  MENU
0 - exit
1 - add random tree
2 - print tree
3 - insert item
4 - delete item
5 - number of nodes
6 - depth(особое действие по варианту)
7 - clear tree
8 - exit
==>2
            \__103
          \__93
            \__86
        \__84
    \__74
            \__72
              \__46
          \__45
        \__18
  \__8

  MENU
```

0 - exit
1 - add random tree
2 - print tree
3 - insert item
4 - delete item
5 - number of nodes
6 - depth(особое действие по варианту)
7 - clear tree
8 - exit
==>6

Depth = 6

MENU
0 - exit
1 - add random tree
2 - print tree
3 - insert item
4 - delete item
5 - number of nodes
6 - depth(особое действие по варианту)
7 - clear tree
8 - exit
==>8

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
1	дом	23.04	2:00	Не считалась правильно глубина дерева второй и более разы, так как я забыл обнулять эту переменную перед применением функции	обнулял эту переменную	

10. Замечания автора

11.**Выводы**

В ходе лабораторной работы я познакомился со структурой данных, деревьями, и понял, что это сильная структура данных, которая помогает размещать данные в удобном порядке

Недочёты при выполнении задания могут быть устранены следующим образом:

Подпись студента ____Постнов_____