Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

ДЕРЕВЬЯ

Отчёт по лабораторной работе №4

По дисциплине

«Структуры и алгоритмы обработки данных в ЭВМ»

Выполнил: студент гр. 439-3

_____ Мазовец А. С.

«___»_____ 2020 г.

Проверил: ассистент каф. АСУ

_____ Яблонский Я. В.

«___»_____ 2020 г.

Томск 2020

1 Задание на лабораторную работу

Вариант11.

Написать программу, которая формирует бинарное дерево поиска, выводит построенное дерево на экран.

Данные для построения дерева могут вводиться с клавиатуры, из файла или генерироваться с помощью генератора случайных чисел. Выбор способа ввода данных выполняется во время работы программы. В построенном дереве необходимо подсчитать число элементов, которые попадают в заданный интервал. Числа, определяющие интервал вводятся с клавиатуры.

Для реализации АТД «Дерево» использовать динамическое распределение памяти.

Перед завершением работы программы освободить занимаемую динамическую память.

2 Алгоритм решения задачи

1. Создать выбранным способом бинарное дерево поиска;
2. Установить временный указатель на корень дерева;
3. Пока i-й ключ временного указателя больше нижней границы, совершать переход к предыдущему по величине элементу (сдвигать временный указатель в левую сторону, если для указателя существует левое поддерево, переходить к минимальному элементу этого поддерева);
4. Пока i-й ключ временного указателя меньше высшей границы, совершать переход к следующему по величине элементу;
5. Достигнув таким образом нижней границы диапазона с помощью переходов к следующему к величине элементу до верхней границы подсчитывать переходы.

3 Листинг программы

Файл tree.hpp

```
#ifndef TREE_HPP
#define TREE_HPP

#include <iostream>
#include <stdexcept>
#include <iomanip>

namespace ads {
    template <typename T>
    class tree {
        private:
            T data;
            tree* left;
            tree* right;
            tree* parent;
            tree* insert(tree* node);

        public:
           tree(T data);
            ~tree();

            T get_data() const;
            bool is_empty() const;
            tree* find(T data);
            void dir_order() const; // direct order
            void rev_order() const; // reverse order
            void sym_order() const; // symmetric order
            tree* insert(T data);
            tree& remove(T data);
            void print();
            tree* next();
            tree* prev();
            tree* min();
            tree* max();
    };

    template <typename T>
    tree<T>* tree<T>::min() {
        if (left)
            return left->min();
        return this;
    }

    template <typename T>
    tree<T>* tree<T>::max() {
        if (right)
            return right->max();
        return this;
    }

    template <typename T>
    tree<T>* tree<T>::next() {
        if (right)
            return right->min();

        tree<T>* tmp = this;
        while (tmp && tmp->parent && tmp == tmp->parent->right)
            tmp = tmp->parent;

        return tmp->parent;
```

```
}

template <typename T>
tree<T>* tree<T>::prev() {
    if (left)
        return left->max();

    tree<T>* tmp = this;
    while (tmp && tmp->parent && tmp == tmp->parent->left)
        tmp = tmp->parent;

    return tmp->parent;
}

template <typename T>
tree<T>::tree(T data)
: data(data), left(nullptr), right(nullptr), parent(nullptr) {}

template <typename T>
tree<T>::~tree() {
    if (left)
        delete left;
    if (right)
        delete right;
}

template <typename T>
T tree<T>::get_data() const {
    return this->data;
}

template <typename T>
bool tree<T>::is_empty() const {
    return !(left || right);
}

template <typename T>
tree<T>* tree<T>::find(T data) {
    if (data == this->data)
        return this;
    if (left && data < this->data)
        return left->find(data);
    if (right && data >= this->data)
        return right->find(data);
    return nullptr;
}

template <typename T>
void tree<T>::dir_order() const {
    std::cout << data << " ";
    if (left)
        left->dir_order();
    if (right)
        right->dir_order();
}

template <typename T>
void tree<T>::rev_order() const {
    if (left)
        left->rev_order();
    if (right)
        right->rev_order();
    std::cout << data << " ";
}

template <typename T>
```

```
void tree<T>::sym_order() const {
    if (left)
        left->sym_order();
    std::cout << data << " ";
    if (right)
        right->sym_order();
}

template <typename T>
tree<T>* tree<T>::insert(T data) {
    if (data < this->data) {
        if (left)
            left->insert(data);
        else {
            left = new tree<T> (data);
            left->parent = this;
        }
    } else {
        if (right)
            right->insert(data);
        else {
            right = new tree<T> (data);
            right->parent = this;
        }
    }
    return this;
}

template <typename T>
tree<T>* tree<T>::insert(tree<T>* node) {
    if (node->data < this->data) {
        if (left)
            left->insert(node);
        else {
            left = node;
            node->parent = this->parent;
        }
    } else {
        if (right)
            right->insert(node);
        else {
            right = node;
            node->parent = this->parent;
        }
    }
    return this;
}

template <typename T>
tree<T>& tree<T>::remove(T data) {
    if (tree<T>* node = find(data)) {
        if (node->left && node->right) {
            tree<T>* min = node->right->min();
            node->data = min->data;
            node->right-> remove (min->data);
        } else if (node->right) {
            tree<T>* tmp = node->right;
            node->left = tmp->left;
            node->right = tmp->right;
            node->data = tmp->data;
            tmp->left = nullptr;
            tmp->right = nullptr;
            delete tmp;
        } else if (node->left) {
            tree<T>* tmp = node->left;
            node->left = tmp->left;
```

```
                node->right = tmp->right;
                node->data = tmp->data;
                tmp->left = nullptr;
                tmp->right = nullptr;
                delete tmp;
            } else {
                node->data = 0;
                if (node->parent) {
                    if (node == node->parent->left)
                        node->parent->left = nullptr;
                    if (node == node->parent->right)
                        node->parent->right = nullptr;
                    delete node;
                }
            }
        }
        return *this;
    }

    template <typename T>
    void tree<T>::print() {
        static int h = 0;
        if (parent)
                 std::cout << "-"   << "\x1b[1;36m[" << std::setfill('.') <<
std::setw(4) << data << "]" << "\x1b[0m" ;
        else
                  std::cout << "  " << "\x1b[1;36m[" << std::setfill('.') <<
std::setw(4) << data << "]" << "\x1b[0m" ;
        if (right) {
            h ++;
            right->print();
            h --;
        }
        if (left) {
            std::cout << std::endl;
            tree<T>* tmp = this;
            while (tmp->parent)
                tmp = tmp->parent;
            while (tmp != this) {
                if (tmp->data < this->data) {
                    if (tmp->left)
                        std::cout << "    |  ";
                    else
                        std::cout << "       ";
                    tmp = tmp->right;
                } else {
                    std::cout << "       ";
                    tmp = tmp->left;
                }
            }
            std::cout << "    `--";
            h ++;
            left->print();
            h --;
        }
    }
}

#endif
```

Файл main.cpp

```
#include <iostream>
#include <ctime>
```

```
#include <fstream>
#include "tree.hpp"

using namespace ads;

void task (tree<int> &t) {
    t.print();
    std::cout << std::endl << std::endl;

    int A;
    std::cout << "Enter min diapazon" << std::endl;
    std::cin >> A;
    int B = A - 1;
    std::cout << "Enter max diapazon" << std::endl;
    while (B < A) {
        std::cin >> B;
        if (B < A)
                std::cout << "     Should be bigger than min. Try again" <<
std::endl;
    }
    std::cout << std::endl;

    int count = 0;

    tree<int>* tmp = &t;
    while (tmp->get_data() >= A)  // начало слева
        if (tmp->prev())
            tmp = tmp->prev();
        else
            break;
    while (tmp->get_data() < A)  // начало справа
        if (tmp->next())
            tmp = tmp->next();
        else
            break;

    std::cout << "symmetric: ";
    t.sym_order();
    std::cout << std::endl;
    std::cout << "current  : ";
    if (tmp->get_data() < A || tmp->get_data() > B) {
    } else {
        while (tmp && tmp->get_data() <= B) {
            std::cout << tmp->get_data() << " ";
            tmp = tmp->next();
            count++;
        }
    }
    std::cout << std::endl;
    std::cout << "count    : " << count << std::endl;
}

void settings() {
    srand (std::time(0));

    int mode;
    std::cout << "0 - manual\n1 - random\n2 - file" << std::endl;
    do {
        std::cin >> mode;
        if (mode != 0 && mode != 1 && mode != 2)
            std::cout << "    Enter 0, 1 or 2 to continue" << std::endl;
    } while (mode != 0 && mode != 1 && mode != 2);
    std::cout << std::endl;

    if (mode == 0) {
        std::cout << "Enter size of tree" << std::endl;
```

```
        int tsize = 0;
        while (tsize <= 0) {
            std::cin >> tsize;
            if (tsize <= 0)
                      std::cout << "     Should be positive. Try again" <<
std::endl;
        }
        std::cout << std::endl;

        int data;
        std::cout << "Enter nodes # one string - one node" << std::endl;
        std::cin >> data;
        tree<int> b(data);
        for (int i = 0; i < tsize-1; i++) {
            std::cin >> data;
            b.insert(data);
        }
        task (b);
    } else if (mode == 1) {
        int min;

        std::cout << "Enter size of tree" << std::endl;
        int tsize = 0;
        while (tsize <= 0) {
            std::cin >> tsize;
            if (tsize <= 0)
                      std::cout << "     Should be positive. Try again" <<
std::endl;
        }
        std::cout << std::endl;

        std::cout << "Enter min possible random" << std::endl;
        std::cin >> min;
        int max = min - 1;
        std::cout << "Enter max possible random" << std::endl;
        while (max < min) {
            std::cin >> max;
            if (max < min)
                  std::cout << "    Should be bigger than min. Try again" <<
std::endl;
        }
        std::cout << std::endl;

        tree<int> t(std::rand() % (max - min + 1) + min);
        for (int i = 0; i < tsize-1; i++)
            t.insert(std::rand() % (max - min + 1) + min);
        task (t);
    } else {
        int i;
        std::fstream file;
        char path[255];
        std::cout << "Enter file path" << std::endl;
        std::cin >> path;
        file.open(path);
        file >> i;
        tree<int> t(i);
        while (file >> i)
            t.insert (i);
        file.close();
        task (t);
    }
}
int main() {
    settings ();
    return 0;
}
```

4 Пример решения

Тест 1. Входные данные: заданное генератором псевдослучайных чисел дерево двоичного поиска (см. Рисунок 4.1)

Пример работы программы:

```
0 - manual
1 - random
2 - file
1

Enter size of tree
50

Enter min possible random
-20
Enter max possible random
20

 [..-4]-[..18]-[..20]-[..20]
    |      `---[..-2]-[..17]-[..17]-[..17]
    |             |      `---[...9]-[..10]-[..16]-[..16]
    |             |             |             `---[..10]-[..15]
    |             |             |                           `---[..11]
    |             |             `---[..-2]-[...1]-[...4]-[...4]-[...8]
    |             |                           |      |             `---[...5]-[...7]
    |             |                           |      `---[...2]-[...2]-[...2]
    |             |                           |             `---[...1]
    |             |                           `---[..-1]-[..-1]
    |             `---[..-3]
    `---[.-18]-[.-13]-[.-12]-[.-10]-[..-9]
           |      |             |      `---[.-10]
           |      |             `---[.-12]-[.-11]
           |      `---[.-17]-[.-17]-[.-17]-[.-17]-[.-14]-[.-14]
           |             |                           `---[.-16]
           |             `---[.-18]
           `---[.-20]-[.-19]-[.-19]-[.-19]-[.-19]

Enter min diapazon
0
Enter max diapazon
5

symmetric: -20 -19 -19 -19 -19 -18 -18 -17 -17 -17 -17 -16 -14 -14 -13 -12 -12 -11 -1
current  : 1 1 2 2 2 4 4 5
count    : 8
```

Рисунок 4.1 — Тест 1.

5 Вывод

Была изучена АТД бинарное дерево поиска с динамически выделяемой памятью. Был реализован класс tree, реализующий функционал и структуры данных изученной АТД. Решена поставленная задача.