Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Томский государственный университет систем
управления и радиоэлектроники»

Факультет систем управления (ФСУ)

Кафедра автоматизированных систем управления (АСУ)

Деревья

отчет по лабораторной работе №4 по дисциплине
«Структуры и алгоритмы обработки данных в ЭВМ»

Обучающийся гр. 431-3
_____ Сергиевский Д.В.
«_22_» _ноября____ 2022 г.

Проверил: доцент каф. АСУ, д.т.н.
_____Горитов А. Н.
«_22_» _ноября____ 2022 г.

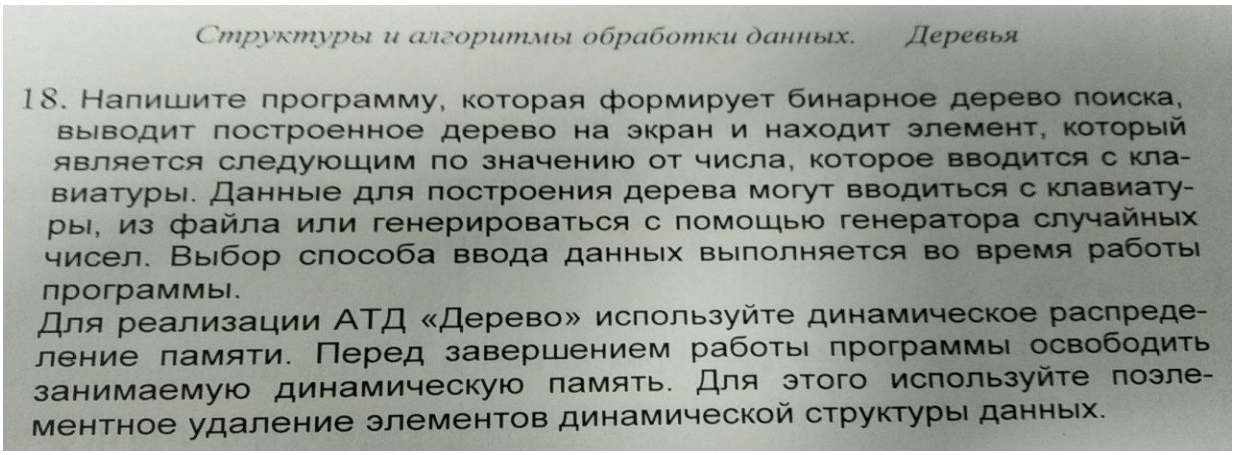
Томск 2022

Содержание

Введение

В рамках данной лабораторной работы необходимо решить небольшую задачу с применением бинарного дерева поиска для закрепления теоретического материала.

Задание на лабораторную работу представлено на Рисунке 1.



Структуры и алгоритмы обработки данных. Деревья

18. Напишите программу, которая формирует бинарное дерево поиска, выводит построенное дерево на экран и находит элемент, который является следующим по значению от числа, которое вводится с клавиатуры. 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.
Для реализации АТД «Дерево» используйте динамическое распределение памяти. Перед завершением работы программы освободить занимаемую динамическую память. Для этого используйте поэлементное удаление элементов динамической структуры данных.

Рисунок 1 - задание

1 Ход работы

1.1 Алгоритм решения

Поскольку, в соответствии с условием задания, пользователю необходимо было предоставить несколько вариантов заполнения дерева, было решено организовать взаимодействие с пользователем через систему команд.

Также для решения данного задания бинарное дерево поиска, помимо базовых операций, должно иметь возможность вывода в консоль и получения узла со значением, ближайшим к данному.

1.2 Реализация решения

Для решения данной задачи потребовалось написать небольшую программу, выполняющую поставленные задачи, а также бинарное дерево поиска на основе динамического распределения памяти в качестве вспомогательного компонента.

Листинг программы приведен в Приложении А.

1.3 Пример решения

Пример работы программы представлен на Рисунке 1.1.

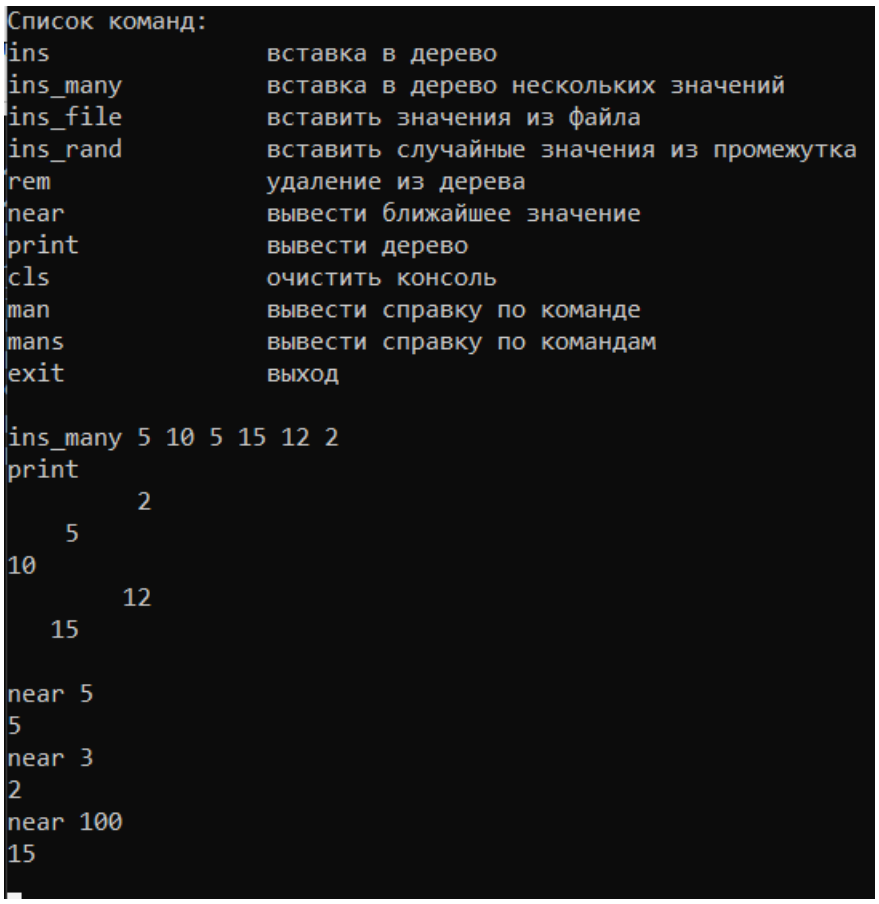


Рисунок 1.1 - пример работы программы

В данном примере в дерево были вставлены вручную введенные значения (10, 5, 15, 12, 2) командой ins_many, после чего заполненное дерево было выведено на экран командой print. Затем были получены ближайшие из содержащихся в дереве значения к числам 5, 3 и 100 с помощью команды near.

Вывод

В результате данной лабораторной работы были подкреплены теоретические знания по теме «Бинарные деревья, бинарное дерево поиска» реализацией соответствующей структуры и её применением для решения небольшой задачи.

Приложение А

Файл s3_sad_lab3.cpp. Точка входа в программу и решение задачи.

```
// s3_sad_lab4.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.
//


#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
#include <sstream>
#include <functional>
using namespace std;


using data_type = int;


enum class TraverseOrder { PreOrder, InOrder, PostOrder };
enum class SearchDirection { Left, End, Right };


template <typename T>
struct Maybe
{
    bool has_data;
    T data;
```

```
};


class TreeNode
{
public:
    friend class Tree;


private:
    using ChildSlot = TreeNode*;


    data_type data;
    TreeNode* left;
    TreeNode* right;


    TreeNode(data_type data) : data(data) { /* * cout << " &" << data; /* */ }
    ~TreeNode() { /* * cout << " ~" << data; /* */ }


    void add_child(TreeNode* child, ChildSlot* slot = nullptr)
    {
        if (data == child->data)
            throw "attempt to insert a node with a value that is already contained in the tree";


        if (slot == nullptr)
            slot = child->data < data ? &left : &right;
```

```
    if (*slot != nullptr)
        throw "attempt to insert a node into an occupied slot";

    *slot = child;
}

TreeNode* rem_child(TreeNode* child)
{
    if (child == nullptr)
        throw "attempt to remove nullptr node";
    if (child != left && child != right)
        throw "attempt to remove someone else's child";
    return rem_child(child == left ? &left : &right);
}

TreeNode* rem_child(ChildSlot* slot)
{
    if (slot == nullptr)
        throw "attempt to remove from nullptr slot";

    auto child = *slot;

    if (child == nullptr)
```

```
    throw "attempt to remove a node from an empty slot";


switch (child->child_count())
{
case 0:
    return exchange(*slot, nullptr);


case 1:
    return exchange(*slot, child->left ? child->left : child->right);


case 2:


    TreeNode* replacement_parent;
    if (!child->right->left)
        replacement_parent = child;
    else
        replacement_parent = child->right->search([&](TreeNode* node) {
        return node->left->left ? SearchDirection::Left : SearchDirection::End;
    }).data;


    auto replacement = replacement_parent->rem_child(replacement_parent->left);


    replacement->left = exchange(child->left, nullptr);
    replacement->right = exchange(child->right, nullptr);
```

```
        return exchange(*slot, replacement);
    }
}

bool traverse(TraverseOrder order, function<bool(TreeNode*, unsigned int)> callback, unsigned int depth = 0)
{
    auto traverse_child = [&](TreeNode* node) {return node ? node->traverse(order, callback, depth+1) : true; };

    switch (order)
    {
    case TraverseOrder::PreOrder:
        return callback(this, depth) && traverse_child(left) && traverse_child(right);

    case TraverseOrder::InOrder:
        return traverse_child(left) && callback(this, depth) && traverse_child(right);

    case TraverseOrder::PostOrder:
        return traverse_child(left) && traverse_child(right) && callback(this, depth);
    }
}

bool traverse(TraverseOrder order, function<bool(TreeNode*)> callback)
```

```
{
    return traverse(order, [&](TreeNode* node, unsigned int depth) {return callback(node); });
}

Maybe<TreeNode*> search(function<SearchDirection(TreeNode*)> callback)
{
    TreeNode* node = this;
    while (node)
    {
        switch (callback(node))
        {
        case SearchDirection::Left:
            node = node->left;
            break;
        case SearchDirection::Right:
            node = node->right;
            break;
        case SearchDirection::End:
            return Maybe<TreeNode*>{true, node};
        }
    }
    return Maybe<TreeNode*>{false, nullptr};
}
```

```
    int child_count()
    {
        return (left != nullptr) + (right != nullptr);
    }

    friend ostream& operator<<(ostream& output, TreeNode* node)
    {
        if (node == nullptr)
            return output << "tree is empty";

        node->traverse(TraverseOrder::InOrder, [&](TreeNode* node, unsigned int depth) {
            output << setw(5*depth) << std::right << node->data << endl;
            return true;
        });
        return output;
    }
};

class Tree
{
    TreeNode* root;

public:
```

```
Tree() : root(nullptr) {}


~Tree()
{
    if (root)
        root->traverse(TraverseOrder::PostOrder, [&](auto node) {delete node; return true; });
}


void insert(data_type data)
{
    TreeNode* node = new TreeNode(data);
    if (node == nullptr)
        throw "new returned nullptr";


    if (root == nullptr)
    {
        root = node;
        return;
    }


    try
    {
        root->search([&](auto node) {return
            (node->left && data < node->data) ? SearchDirection::Left :
```

```
            (node->right && node->data < data) ? SearchDirection::Right :
            SearchDirection::End;
            }).data->add_child(node);
    }
    catch(const char* error)
    {
        if (error == "attempt to insert a node with a value that is already contained in the tree")
            throw "attempt to insert a value already containing in the tree";
        else
            throw error;
    }
}

void remove(data_type data)
{
    TreeNode* parent;

    if (!root)
        throw "attempt to remove a node from an empty tree";

    if (root->data == data)
    {
        delete root->rem_child(&root);
        return;
```

```
    }

    auto maybe_child = root->search([&](auto node) {
        if (node->data == data)
            return SearchDirection::End;
        parent = node;
        return data < node->data ? SearchDirection::Left : SearchDirection::Right;
        });

    if (!maybe_child.has_data)
        throw "attempt to remove a non-existent node";

    delete parent->rem_child(maybe_child.data);
}

void traverse(TraverseOrder order, function<bool(data_type)> callback)
{
    if (root)
        root->traverse(order, [&](auto node) {return callback(node->data); });
}

Maybe<data_type> search(function<SearchDirection(data_type)> callback)
{
    if (!root)
```

```
        return Maybe<data_type>{false, 0};


    auto maybe_answer = root->search([&](auto node) {return callback(node->data); });
                return Maybe<data_type>{maybe_answer.has_data, maybe_answer.has_data ? maybe_answer.data->data : 0};
  }


  Maybe<data_type> nearest(data_type data)
  {
    if (!root)
      return Maybe<data_type>{false, 0};


    auto nearest_node = root->search([&](auto node) {return
      (node->left && data < node->data) ? SearchDirection::Left :
      (node->right && node->data < data) ? SearchDirection::Right :
      SearchDirection::End;
      }).data;


    return Maybe<data_type>{ true, nearest_node->data};
  }


  friend ostream& operator<<(ostream& output, const Tree& tree)
  {
    return output << tree.root;
  }
```

```
    bool contains(data_type data)
    {
        auto maybe = nearest(data);
        return maybe.has_data && maybe.data == data;
    }
};

struct Command
{
    const char* codeword;
    const char* parameters_description;
    const char* description;
    function<void(Tree*)> handler;
};

void do_task()
{
    bool exit = false;
    Tree tree;

    Command commands[11] = {
    Command
    {
```

```
    "ins",
    "значение",
    "вставка в дерево",
    [](Tree* tree) {
        data_type data;
        cin >> data;
        tree->insert(data);
    }
},
Command
{
    "ins_many",
    "количество значений, значение...",
    "вставка в дерево нескольких значений",
    [](Tree* tree) {
        unsigned int n;
        data_type data;
        cin >> n;
        for (unsigned int i = 0; i < n; ++i)
        {
            cin >> data;
            tree->insert(data);
        }
    }
```

```
},
Command
{
   "ins_file",
   "",
   "вставить значения из файла",
   [](Tree* tree) {
      fstream input("input.txt", fstream::in);
      if (!input.is_open())
      {
         cout << "input.txt didn`t open";
         return;
      }

      for (data_type data; input >> data; )
            tree->insert(data);
   }
},
Command
{
   "ins_rand",
   "количество генерируемых чисел, минимум, максимум",
   "вставить случайные значения из промежутка",
   [](Tree* tree) {
```

```
        int n;
        data_type data, a, b;


        cin >> n >> a >> b;
        for (int i = 0; i < n; ++i)
        {
            data = rand() % (b - a + 1) + a;
            if (!tree->contains(data))
                tree->insert(data);
            else
                --i;
        }
    }
},
Command
{
    "rem",
    "значение",
    "удаление из дерева",
    [](Tree* tree) {
        data_type data;
        cin >> data;
        tree->remove(data);
    }
```

```
},
Command
{
    "near",
    "значение",
    "вывести ближайшее значение",
    [](Tree* tree) {
        data_type data;
        cin >> data;
        auto maybe = tree->nearest(data);


        if (maybe.has_data)
            cout << maybe.data << endl;
        else
            cout << "nothing" << endl;
    }
},
Command
{
    "print",
    "",
    "вывести дерево",
    [](Tree* tree) {
        cout << *tree << endl;
```

```
    }
},
Command
{
    "cls",
    "",
    "очистить консоль",
    [](Tree* tree) {
        system("cls");
    }
},
Command
{
    "man",
    "команда",
    "вывести справку по команде",
    [&](Tree* tree) {
        string codeword;
        cin >> codeword;

        bool ok = false;
        for (Command command : commands)
        {
            if (codeword != command.codeword)
```

```
                continue;


                    cout << codeword << " <" << command.parameters_description << "> - " <<
command.description << endl;


            ok = true;
            break;
        }
        if (!ok)
            cout << "Непонятная команда" << endl;
    }
  },
  Command
  {
    "mans",
    "",
    "вывести справку по командам",
    [&](Tree* tree) {
        cout << "Список команд: " << endl;
        for (Command c : commands)
        {
            cout << left
                << setw(18) << left << c.codeword
                << setw(1) << c.description
                << endl;
```

```
        }
        cout << endl;
    }
},
Command
{
    "exit",
    "",
    "выход",
    [&](Tree* tree) {
        exit = true;
    }
}
};

bool ok;
string codeword = "mans";
while(true)
{
    ok = false;
    for (auto command : commands)
    {
        if (codeword != command.codeword)
            continue;
```

```
        try
        {
            command.handler(&tree);
        }
        catch (const char* error)
        {
            cout << error << endl;
        }

        ok = true;
        break;
    }

    if (!ok)
        cout << "Непонятная команда" << endl;

    if (exit)
        break;

    cin >> codeword;
  }

}
```

```
int main()
{
    setlocale(LC_ALL, "Russian");

    try
    {
        do_task();
    }
    catch (const char* message)
    {
        cout << "error: " << message;
        getchar();
    }
}

// Запуск программы: CTRL+F5 или меню "Отладка" > "Запуск без отладки"
// Отладка программы: F5 или меню "Отладка" > "Запустить отладку"

// Советы по началу работы
//   1. В окне обозревателя решений можно добавлять файлы и управлять ими.
//   2. В окне Team Explorer можно подключиться к системе управления версиями.
//   3. В окне "Выходные данные" можно просматривать выходные данные сборки и другие сообщения.
//   4. В окне "Список ошибок" можно просматривать ошибки.
```

```
//    5. Последовательно выберите пункты меню "Проект" > "Добавить новый элемент", чтобы создать файлы кода, или "Проект" > "Добавить существующий элемент", чтобы добавить в проект существующие файлы кода.

//    6. Чтобы снова открыть этот проект позже, выберите пункты меню "Файл" > "Открыть" > "Проект" и выберите SLN-файл.
```