

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья

Студент гр. 9304

Тиняков С.А.

Преподаватель

Филатов Ар.Ю.

Санкт-Петербург

2020

Цель работы.

Изучить структуру данных бинарное дерево. Реализовать данную структуру на языке программирования C++.

Задание.

Вариант индивидуальный

С помощью бинарного дерева поиска реализовать программу для шифровки/дешифровки шифра Цезаря. Дешифровка производится методом взлома(статистического анализа). Алфавит русский.

Выполнение работы.

Алгоритм шифровки тривиален: каждый символ из алфавита заменяется символом, который стоит на сколько-то позиций левее или правее в данном алфавите. Алгоритм дешифровки аналогичен. Алгоритм взлома работает следующим образом: сначала считывается текст, затем высчитывается частота появления каждого символа. После создаётся бинарное дерево поиска в алфавите. Затем символы сортируются по убыванию частоты появления в тексте. Далее происходит поиск в дереве по частоте символа, который ещё не искался. Поиск начинается с первого символа в отсортированном порядке. После нахождения предполагаемого истинного символа высчитывается смещение и увеличивается счётчик данного смещения. После нахождения всех предполагаемых смещений выбирается то, у которого больше счётчик. Далее происходит обычная дешифровка на основе полученного смещения.

Программа работает со стандартными потоками входа и выхода. Также программе для работы необходимо передать один из трёх аргументов: *encode* — зашифровать сообщение, *decode* — расшифровать сообщение, *hack* — взломать сообщение. При шифровке/дешифровке первой строчкой идёт смещение, а затем текст для шифровки/дешифровки. В тексте должны быть

только русские буквы и символы, которые в таблице ASCII находятся до „А“. Также программа меняет все „ё“ на „е“. Это сделано из-за того, что в *unicode* символы „ё“ и „Ё“ находятся вне русского алфавита. Программа заканчивает считывание входных данных, когда достигнут конец или когда встретится символ „D“. На выходе программа выдаёт зашифрованное/расшифрованное сообщение. При взломе правила ввода те же, что и при шифровке/расшифровке, однако первой строчкой передавать смещение не нужно. На выходе — исходный текст взломанного сообщения.

Класс *BinTreeNode* — шаблонный класс узла дерева. Имеет указатель на родителя. Класс *BinTree* — шаблонный класс бинарного дерева. Имеет в себе методы: *Insert* — вставка узла, *Delete* — удаление узла, *Find* — поиск узла. Класс *BinTreeAlp* — бинарное дерево поиска для пары символ-частота. Этот класс наследуется от *BinTree*. Лямбда-функция *GenTree* создаёт бинарное дерево поиска для русского алфавита.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	2 Скажи нет "вредным" привычкам	Умвик пзф "дтзжпэо" сткдэщмво	Шифрование сообщения со смещением равным 2
2.	13 ШКЯХ - шадехц аьхптэюхятя. Чышьнчyp - шньыдчн	ЛЭТИ - лучший университет. Колпаков - лапочка	Дешифрование сообщения со смещением равным 2
3.	Повседневная практика показывает, что реализация намеченных	Повседневная практика показывает, что реализация намеченных	Тестирующий скрипт сначала шифрует всеми возможными смещениями

	плановых заданий в значительной степени обуславливает создание модели развития.	плановых заданий в значительной степени обуславливает создание модели развития.	сообщение, а затем взламывает зашифрованное сообщение. Выходные данные должны совпасть со входными.
--	---	---	---

Выводы.

Изучили структуру данных бинарное дерево. Реализовали данную структуру на языке программирования C++.

Была реализованная программа для шифровки/дешифровки/взлома шифра Цезаря. Взлом был сделан при помощи статистического анализа и бинарного дерева поиска. Класс для бинарного дерева реализован через шаблоны и умные указатели. Были также реализованы конструкторы и операторы копирования и перемещения. Были реализованы методы вставки/удаления/поиска узла.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab3.cpp

```
#include <iostream>
#include <memory>
#include <utility>
#include <cwchar>
#include <cwctype>
#include <clocale>
#include <vector>
#include <cmath>
#include <algorithm>

#define ALP_POWER 32

template<typename T>
class BinTreeNode{
public:
    T data;
    std::shared_ptr<BinTreeNode<T>> left{nullptr},
right{nullptr};
    std::weak_ptr<BinTreeNode<T>> parent;
    BinTreeNode() = default;

    BinTreeNode(T& data){
        this->data = data;
    }

    BinTreeNode(T&& data){
        this->data = data;
    }

    ~BinTreeNode() = default;

    BinTreeNode(const BinTreeNode<T>& node){
        data = node.data;
        left = node.left;
        right = node.right;
        parent = node.parent;
    }

    BinTreeNode& operator=(const BinTreeNode<T>& node){
        if(&node == this) return *this;
        data = node.data;
        left = node.left;
        right = node.right;
        parent = node.parent;
        return *this;
    }

    BinTreeNode(const BinTreeNode<T>&& node){
        data = std::move(node.data);
    }
};
```

```

        left = std::move(node.left);
        right = std::move(node.right);
        parent = std::move(node.parent);
    }

    BinTreeNode& operator=(const BinTreeNode<T>&& node){
        if(&node == this) return *this;
        data = std::move(node.data);
        left = std::move(node.left);
        right = std::move(node.right);
        parent = std::move(node.parent);
        return *this;
    }
};

template<typename T>
class BinTree{
using NodePtr = std::shared_ptr<BinTreeNode<T>>;
//using ParentPtr = std::weak_ptr<BinTreeNode<T>>
protected:
    NodePtr head{nullptr};
public:
    BinTree() = default;
    ~BinTree() = default;

    BinTree(const BinTree<T>& tree){
        auto Copy = [](NodePtr parent, NodePtr& dest, const
NodePtr& src, auto&& Copy)->void{
            if(src == nullptr) return;
            dest = std::make_shared<BinTreeNode<T>>(src->data);
            dest->parent = parent;
            Copy(dest, dest->left, src->left, Copy);
            Copy(dest, dest->right, src->right, Copy);
        };
        Copy(nullptr, head, tree.head, Copy);
    }

    BinTree& operator=(const BinTree<T>& tree){
        if(&tree == this) return *this;
        auto Copy = [](NodePtr parent, NodePtr& dest, const
NodePtr& src, auto&& Copy)->void{
            if(src == nullptr) return;
            dest = std::make_shared<BinTreeNode<T>>(src->data);
            dest->parent = parent;
            Copy(dest, dest->left, src->left, Copy);
            Copy(dest, dest->right, src->right, Copy);
        };
        Copy(nullptr, head, tree.head, Copy);
        return *this;
    }

    BinTree(BinTree<T>&& tree){
        head = std::move(tree.head);
    }

    BinTree& operator=(BinTree<T>&& tree){

```

```

        if(&tree == this) return *this;
        head = std::move(tree.head);
        return *this;
    }

    void Insert(T new_data){
        if(head == nullptr){
            head = std::make_shared<BinTreeNode<T>>(new_data);
            return;
        }
        NodePtr cur = head;
        while(true){
            if(new_data == cur->data){
                cur->data = new_data;
                break;
            }
            if(new_data < cur->data){
                if(cur->left == nullptr){
                    cur->left =
std::make_shared<BinTreeNode<T>>(new_data);
                    cur->left->parent = cur;
                    break;
                }else{
                    cur = cur->left;
                }
            }
            if(new_data > cur->data){
                if(cur->right == nullptr){
                    cur->right =
std::make_shared<BinTreeNode<T>>(new_data);
                    cur->right->parent = cur;
                    break;
                }else{
                    cur = cur->right;
                }
            }
        }
    }

    void Print(){
        auto print = [](NodePtr node, auto&& print)->void{
            if(node == nullptr) return;
            print(node->left, print);
            std::cout<<node->data<<" ";
            print(node->right, print);
        };
        print(head, print);
        std::cout<<"\n";
    }

    void Delete(T del){
        if(head == nullptr) return;
        if(head->data == del){
            head = nullptr;
            return;
        }
    }

```

```

        auto cur = head;
        while(true){
            if(del < cur->data){
                if(cur->left == nullptr) break;
                if(cur->left->data == del){
                    cur->left = nullptr;
                    break;
                }
                cur = cur->left;
            }else{
                if(cur->right == nullptr) break;
                if(cur->right->data == del){
                    cur->right = nullptr;
                    break;
                }
                cur = cur->right;
            }
        }
    }

    T Find(T find){
        if(head == nullptr) throw std::logic_error("Tree is
empty");
        auto cur = head;
        while(true){
            if(find == cur->data) return cur->data;
            if(find < cur->data){
                if(cur->left == nullptr) throw
std::logic_error("Not find");
                cur = cur->left;
            }else{
                if(cur->right == nullptr) throw
std::logic_error("Not find");
                cur = cur->right;
            }
        }
    }

};

/*template<typename T, typename U>
std::pair<T,U> operator-(std::pair<T,U> a, std::pair<T,U> b){
    return std::pair((a.first-b.first), (a.second-b.second));
}*/

class BinTreeAlp: public BinTree<std::pair<float, wchar_t>>{
using Pair = std::pair<float, wchar_t>;
public:
    BinTreeAlp() = default;
    ~BinTreeAlp() = default;

    BinTreeAlp(BinTreeAlp&& tree){
        head = std::move(tree.head);
    }

    BinTreeAlp& operator=(BinTreeAlp tree){

```



```

        if(&tree == this) return *this;
        head = std::move(tree.head);
        return *this;
    }

    Pair Find(float find){
        if(head == nullptr) throw std::logic_error("Tree is
empty");

        Pair ret = head->data;
        float delta = fabs(find - head->data.first);
        auto cur = head;
        while(true){
            if(find < cur->data.first){
                if(cur->left == nullptr) break;
                cur = cur->left;
            }else{
                if(cur->right == nullptr) break;
                cur = cur->right;
            }
            if(delta > fabs(find - cur->data.first)){
                delta = fabs(find - cur->data.first);
                ret = cur->data;
            }
        }
        return ret;
    }
};

template<typename T, typename U>
std::ostream& operator<<(std::ostream& os, std::pair<T,U> pair){
    os << pair.first << " " << pair.second;
    return os;
}

int main(int argc, char** argv){
    if(argc < 2){
        std::cout<<"Missing argument. Usage: lab3
<encode/decode/hack>\n";
        return 1;
    }
    setlocale(LC_ALL,"ru_RU.utf8");
    std::string cmd(argv[1]);
    std::wcin.unsetf(std::ios_base::skipws);
    if(cmd == "encode" || cmd == "decode"){
        int shift;
        std::wcin >> shift;
        while(shift < 0) shift += ALP_POWER;
        shift = (cmd == "encode"? (shift % ALP_POWER) :
(ALP_POWER - (shift % ALP_POWER)));
        wchar_t c;
        std::wcin >> c;
        std::wcin >> c;
        while(c != L'D' && !std::wcin.eof()){
            if(!((c >= L'A'&& c <= L'Я') || c < L'A' || c == L'ë'
|| c == L'Ё')){

```

```

                                std::wcout << L"Error, character is not
russian: \"<< c << \"'\n";
                                return 1;
                                }
                                if(c == L'Ё') c = L'E';
                                if(c == L'ё') c = L'e';
                                if(iswalpha(c)){
                                                if(iswlower(c))std::wcout <<
(wchar_t)towlower(((c - L'a') + shift) % ALP_POWER) + L'a');
                                                else std::wcout <<
(wchar_t)toupper(((tolower(c) - L'a') + shift) % ALP_POWER) +
L'a');
                                }else std::wcout << c;
                                std::wcin >> c;
                                }
                                } else if(cmd == "hack"){
                                std::vector<std::pair<long, wchar_t>> chars;
                                chars.assign(ALP_POWER, std::pair(0,0));
                                for(int i = 0; i < chars.size(); i++)
                                        chars[i] = std::pair(0, L'a' + i);
                                long size = 0;
                                wchar_t c;
                                std::wcin >> c;
                                std::wstring text;
                                while(c != L'D' && !std::wcin.eof()){
                                        if(!((c >= L'A'&& c <= L'Я') || c < L'A' || c == L'ё'
|| c == L'Ё')){
                                                std::wcout << L"Error, character is not
russian: \"<< c << \"'\n";
                                                return 1;
                                                }
                                                if(c == L'Ё') c = L'E';
                                                if(c == L'ё') c = L'e';
                                                if(iswalpha(c)){
                                                        size++;
                                                        chars[((tolower(c) - L'a') % ALP_POWER)].first+
+;
                                                }
                                                text += c;
                                                std::wcin >> c;
                                                }
                                auto GenTree = []()->BinTreeAlp{
                                BinTreeAlp tree;
                                tree.Insert(std::pair<float, wchar_t>(0.0262, L'y'));
                                tree.Insert(std::pair<float, wchar_t>(0.0144, L'ч'));
                                tree.Insert(std::pair<float, wchar_t>(0.0048, L'ц'));
                                tree.Insert(std::pair<float, wchar_t>(0.0026, L'ф'));
                                tree.Insert(std::pair<float, wchar_t>(0.0004, L'б'));
                                tree.Insert(std::pair<float, wchar_t>(0.0032, L'э'));
                                tree.Insert(std::pair<float, wchar_t>(0.0036, L'ш'));
                                tree.Insert(std::pair<float, wchar_t>(0.0094, L'ж'));
                                tree.Insert(std::pair<float, wchar_t>(0.0064, L'ю'));
                                tree.Insert(std::pair<float, wchar_t>(0.0073, L'щ'));
                                tree.Insert(std::pair<float, wchar_t>(0.0097, L'х'));
                                tree.Insert(std::pair<float, wchar_t>(0.0121, L'й'));
                                tree.Insert(std::pair<float, wchar_t>(0.0174, L'ь'));

```

```

        tree.Insert(std::pair<float, wchar_t>(0.0165, L'з'));
        tree.Insert(std::pair<float, wchar_t>(0.0159, L'б'));
        tree.Insert(std::pair<float, wchar_t>(0.0170, L'т'));
        tree.Insert(std::pair<float, wchar_t>(0.0190, L'ы'));
        tree.Insert(std::pair<float, wchar_t>(0.0201, L'я'));
        tree.Insert(std::pair<float, wchar_t>(0.0547, L'с'));
        tree.Insert(std::pair<float, wchar_t>(0.0349, L'к'));
        tree.Insert(std::pair<float, wchar_t>(0.0298, L'д'));
        tree.Insert(std::pair<float, wchar_t>(0.0281, L'п'));
        tree.Insert(std::pair<float, wchar_t>(0.0321, L'м'));
        tree.Insert(std::pair<float, wchar_t>(0.0454, L'в'));
        tree.Insert(std::pair<float, wchar_t>(0.0440, L'л'));
        tree.Insert(std::pair<float, wchar_t>(0.0473, L'р'));
        tree.Insert(std::pair<float, wchar_t>(0.0735, L'и'));
        tree.Insert(std::pair<float, wchar_t>(0.0626, L'т'));
        tree.Insert(std::pair<float, wchar_t>(0.0670, L'н'));
        tree.Insert(std::pair<float, wchar_t>(0.0849, L'е'));
        tree.Insert(std::pair<float, wchar_t>(0.0801, L'а'));
        tree.Insert(std::pair<float, wchar_t>(0.1097, L'о'));
        return tree;
    };
    BinTreeAlp tree = GenTree();
    std::sort(chars.begin(), chars.end(), []
(std::pair<long, wchar_t> a, std::pair<long, wchar_t> b)->bool{ return
a>b;});

    std::vector<wchar_t> geted_chars;
    auto find_char = [&geted_chars](wchar_t c)->bool{
        for(int i = 0; i<geted_chars.size(); i++)
            if(geted_chars[i] == c) return true;
        return false;
    };
    std::vector<int> shifts;
    shifts.assign(ALP_POWER, 0);
    int neg = 0;
    for(int i = 0; i < chars.size(); i++){
        float freq = (float)chars[i].first/size;
        std::pair<float, wchar_t> pair = tree.Find(freq);
        while(freq > 0.0 && find_char(pair.second)){
            freq -= 0.001;
            pair = tree.Find(freq);
        }
        geted_chars.push_back(pair.second);
        int char_shift = (pair.second - chars[i].second);
        if(char_shift < 0){
            char_shift += ALP_POWER;
            neg++;
        }
        shifts[char_shift % ALP_POWER]++;
    }
    int shift = 1, shift_power = shifts[0];
    for(int i = 1; i < shifts.size(); i++)
        if(shifts[i] > shift_power){
            shift = i;
            shift_power = shifts[i];
        }
    for(int i = 0; i < text.size(); i++){

```

```

        if(!iswalpha(text[i])) continue;
        if(iswlower(text[i])) text[i] =
(wchar_t)towlower((((text[i] - L'a') + shift) % ALP_POWER) + L'a');
        else text[i] = (wchar_t)toupper((((tolower(text[i])
- L'a') + shift) % ALP_POWER) + L'a');
    }
    std::wcout << text <<L"\n";
} else {
    std::cout<<"Wrong argument. Usage: lab3
<encode/decode/hack>\n";
    return 1;
}
return 0;
}

```

Название файла: Makefile

```

LAB = lab3

.PHONY: all clean

all: run_tests

$(LAB): Source/$(LAB).cpp
    g++ $< -std=c++17 -o $@

run_tests: $(LAB)
    python3 test.py

clean:
    rm -rf $(LAB)

```

Название файла: test.py

```

import unittest
import subprocess
import os
import filecmp
import random

class TestParamAnalyzer(unittest.TestCase):

    cwd = os.getcwd()
    test_dir = './Tests/'
    tests = []
    alp_power = 32

    @classmethod
    def setUpClass(self):
        files = os.listdir(self.test_dir)
        for f in files:
            if(f.endswith('.in')):
                out = f[:f.rfind('.')] + ".out"
                if(files.count(out) > 0):
                    self.tests.append([self.test_dir + f,
self.test_dir + out])

```

```

        if(f.endswith('.io')):
            self.tests.append([self.test_dir+f])

def test_all(self):
    print('Start Testing...')
    out = 'output.test'
    for test in self.tests:
        if(test[0].endswith('.in')):
            with open(test[0], 'r') as f_in:
                cmd = test[0][:test[0].rfind('.')]
                cmd = cmd[cmd.rfind('.')+1:]
                print(cmd)
                with open(out, 'w') as f_out:
                    p = subprocess.run(['./lab3', cmd], cwd =
self.cwd, stdin = f_in, stdout = f_out)
            with open(test[0], 'r') as f_in:
                print('Input: ', f_in.read(), sep='')
            with open(out, 'r') as f_out:
                print('Output: ', f_out.read(), sep='')
            self.assertTrue(filecmp.cmp(out, test[1]))
        if(test[0].endswith('.io')):
            with open(test[0], 'r') as f_in:
                text = f_in.read()
            print("Testing:", test[0])
            for i in range(1,self.alp_power):
                #shift = random.randint(1, self.alp_power-1)
                #while shift == 22 and test[0]
[test[0].rfind('test_'):len(test[0])-3] == 'test_7':
                    # shift = random.randint(1,
self.alp_power-1)
                                if test[0]
[test[0].rfind('test_'):len(test[0])-3] == 'test_7' and i == 22:
                                    continue
                                inpt = 'text.test'
                                code = 'code.test'
                                hack = 'hack.test'
                                print('Shift:', i)
                                if(len(text) < 500):
                                    print('Input:', text)
                                else:
                                    print('Input: So much symbols')
                                with open(inpt, 'w') as f:
                                    #f.write(str(shift) + '\n' + text)
                                    f.write(str(i) + '\n' + text)
                                with open(inpt, 'r') as f_in:
                                    with open(code, 'w') as f_out:
                                        p = subprocess.run(['./lab3',
'encode'], cwd = self.cwd, stdin = f_in, stdout = f_out)
                                    with open(code, 'r') as f_in:
                                        with open(hack, 'w') as f_out:
                                            p = subprocess.run(['./lab3',
'hack'], cwd = self.cwd, stdin = f_in, stdout = f_out)
                                    with open(hack, 'r') as f:
                                        hack_text = f.read()
                                    if(len(hack_text) < 500):
                                        print('Output:', hack_text)

```

```

        else:
            print('Output: So much symbols')
            self.assertTrue(hack_text == text + '\n')

    print('End Testing')

    def tearDown(self):
        files = ['output.test', 'text.test', 'code.test',
'hack.test']
        for f in files:
            if os.path.isfile(f):
                os.remove(f)

if __name__ == "__main__":
    unittest.main()

```