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

ОТЧЕТ

по лабораторной работе № 3
по дисциплине «Алгоритмы и структуры данных»
Тема: «Комбинированные структуры данных и стандартная библиотека шаблонов»

Студентки гр. 2306 _____ Коробенко С.А. Платто А.А.

Преподаватель _____ Колинько П.Г.

Санкт-Петербург
2024

Цель работы.

Получение навыков работы со структурами данных и стандартными библиотеками шаблонов.

Задание (вариант 5).

Реализовать индивидуальное задание темы «Множества + последовательности» в виде программы, используя свой контейнер для заданной структуры данных (хеш-таблицы или одного из вариантов ДДП), и доработать его для поддержки операций с последовательностями.

Указания к заданию по работе со множествами согласно варианту:

1) Мощность множества = 10;
2) Необходимо вычислить: A ⊕ B / (C ∪ D) / E.

Для выполнения задания с последовательностями:

1) Базовая СД – ДДПм;
2) Реализовать функции: MERGE, CONCAT, SUBSTR.

Ход работы.

В ходе выполнения данной лабораторной работы была написана программа, которая вычисляет множество по заданной формуле, исходя из генерируемых множеств A, B, C, D, E. Все аналогичные вычисления также проводятся для последовательностей. Каждый этап работы выводится на экран.

Для хранения множеств был выбран контейнер подходящего типа, то есть дерево двоичного поиска. Также он был доработан для поддержки операций с последовательностями.

Для работы с последовательностями были реализованы следующие функции:

1) MERGE – слияние. Объединение двух упорядоченных последовательностей в третью с сохранением упорядоченности. От операции объединения множеств отличается только возможностью появления дубликатов ключей.
2) CONCAT - сцепление. Вторая последовательность подсоединяется к концу первой, образуя её продолжение.

3) CHANGE – включение. Вторая последовательность включается в первую с указанной позиции p. Операция похожа на конкатенацию. Сперва берётся начало первой последовательности до позиции p, затем идёт вторая последовательность, а за ней — остаток первой.

Этот контейнер представляет собой структуру данных двоичного дерева с хранением мощности поддерева (Binary Tree) с возможностью выполнения различных операций над ним.

Структура данных множество (set): Множество элементов представлено в виде дерева двоичного поиска с автоматической балансировкой. Реализовано с использованием стандартной библиотеки шаблонов set. Ключи элементов множества представляются в узлах дерева.

Массив итераторов (sA): Хранит указатели на значения элементов множества, реализован в виде контейнера вектора указателей.

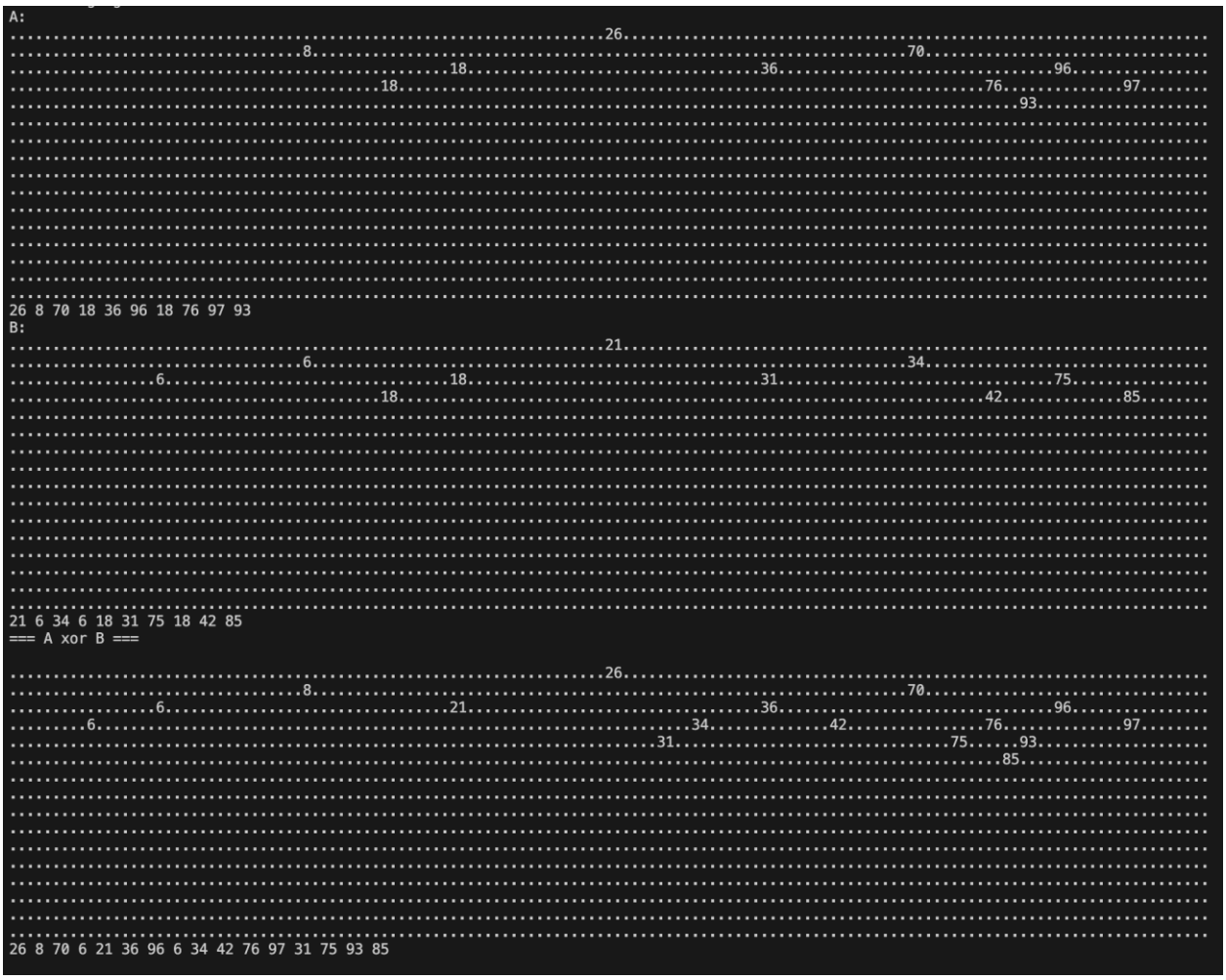
Основные операции: Вставка нового элемента в дерево (insert),Объединение двух деревьев (MERGE), Конкатенация двух деревьев (CONCAT), Изменение содержимого дерева (SUBSTR), Выполнение операции объединения, пересечения и разности деревьев (+, /, ^), Вывод дерева на экран в виде дерева (printTree, OutTree).

Итератор: Реализован внутренний класс Iterator, который позволяет проходить по элементам дерева в порядке обхода префикса (pre-order traversal).Итератор используется для итерации по элементам дерева в методах begin() и end().

Визуализация: Реализована функция OutTree(), которая выводит дерево на экран с помощью ASCII-графики..

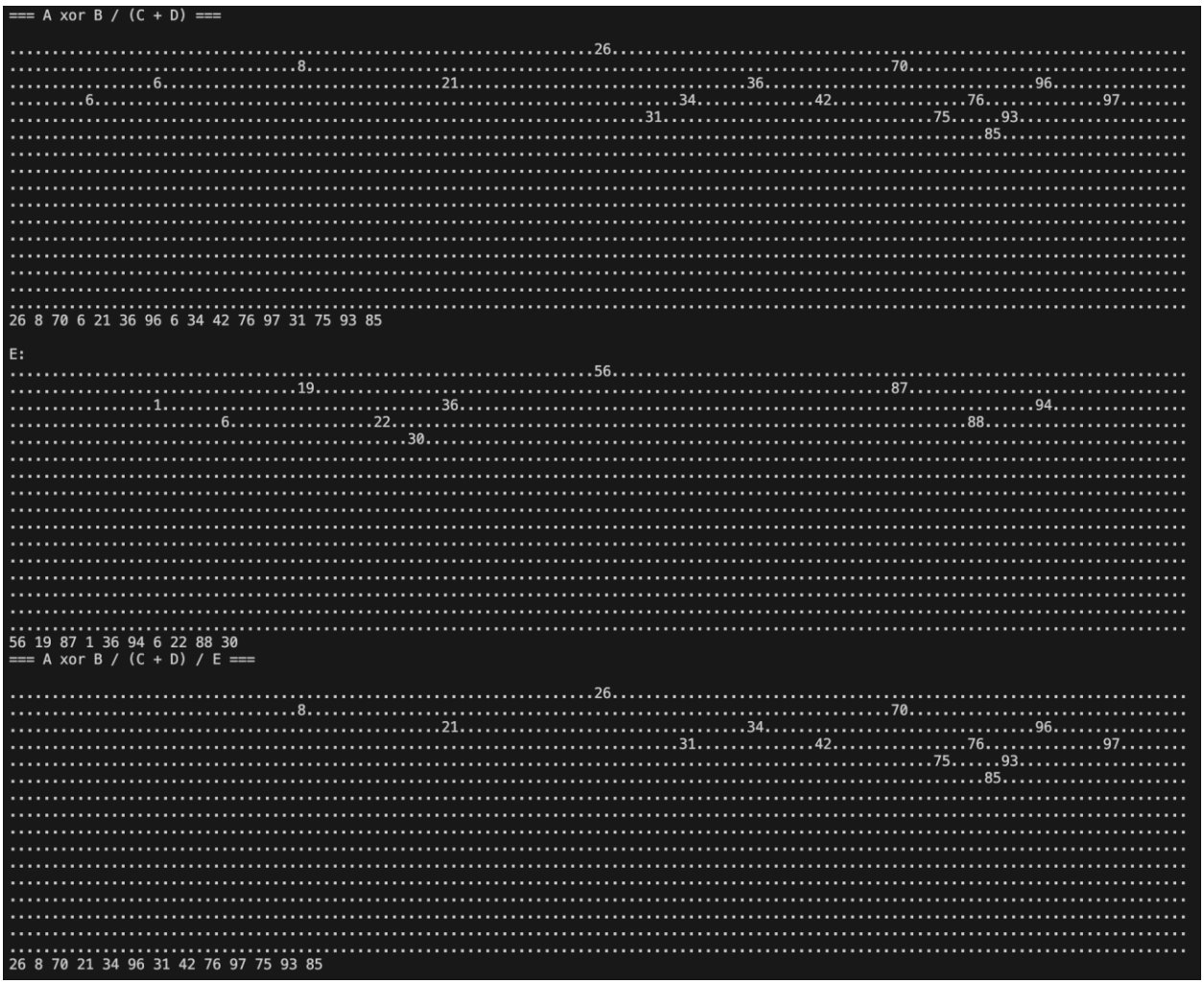
Контрольные тесты.

1. Вычисление по

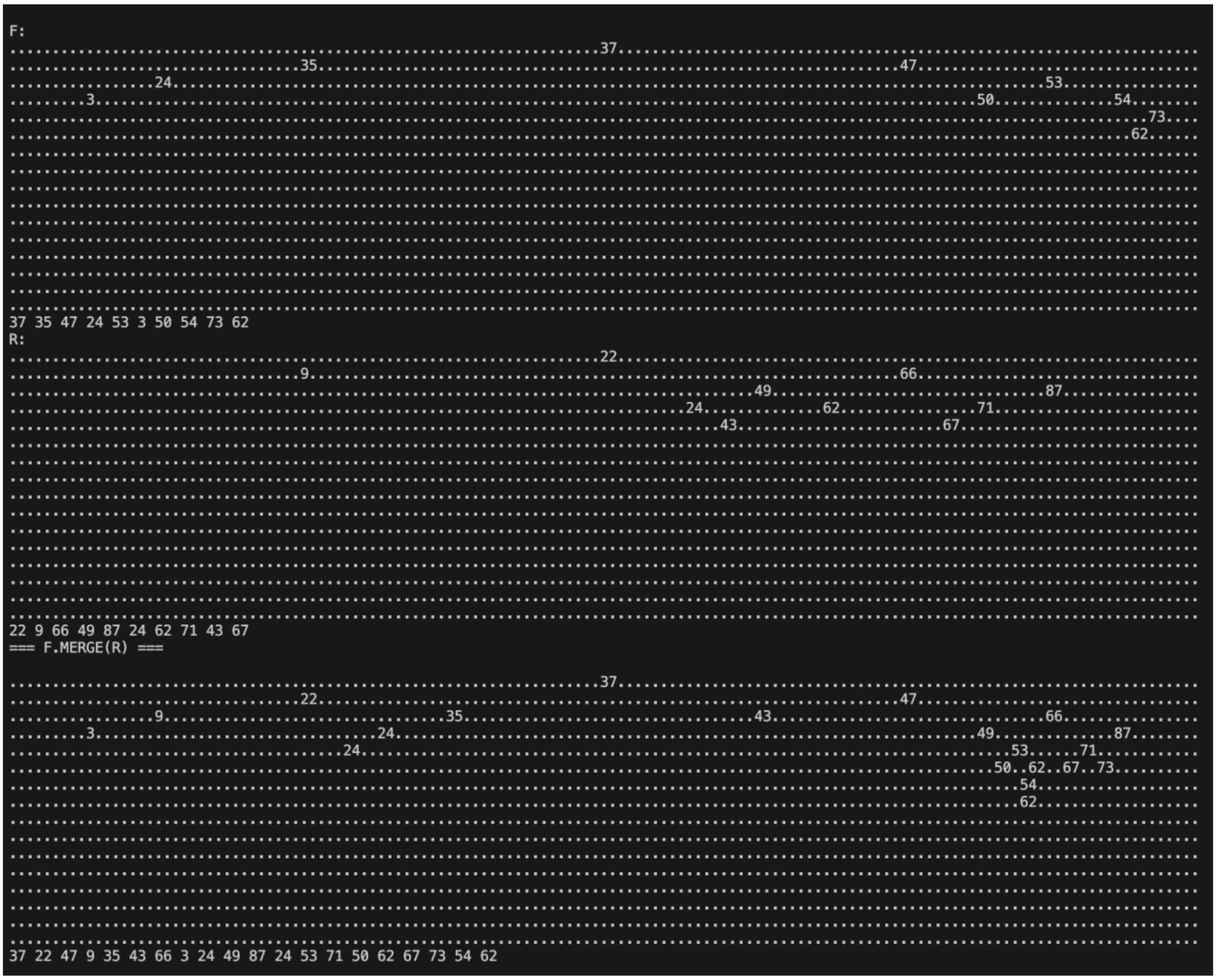

A:
26
8
70
18
36
96
18
76
97
93
26 8 70 18 36 96 18 76 97 93
B:
21
6
34
6
18
31
75
18
42
85
21 6 34 6 18 31 75 18 42 85
=== A xor B ===
26
8
70
6
21
36
96
6
34
42
76
97
31
75
93
85
26 8 70 6 21 36 96 6 34 42 76 97 31 75 93 85

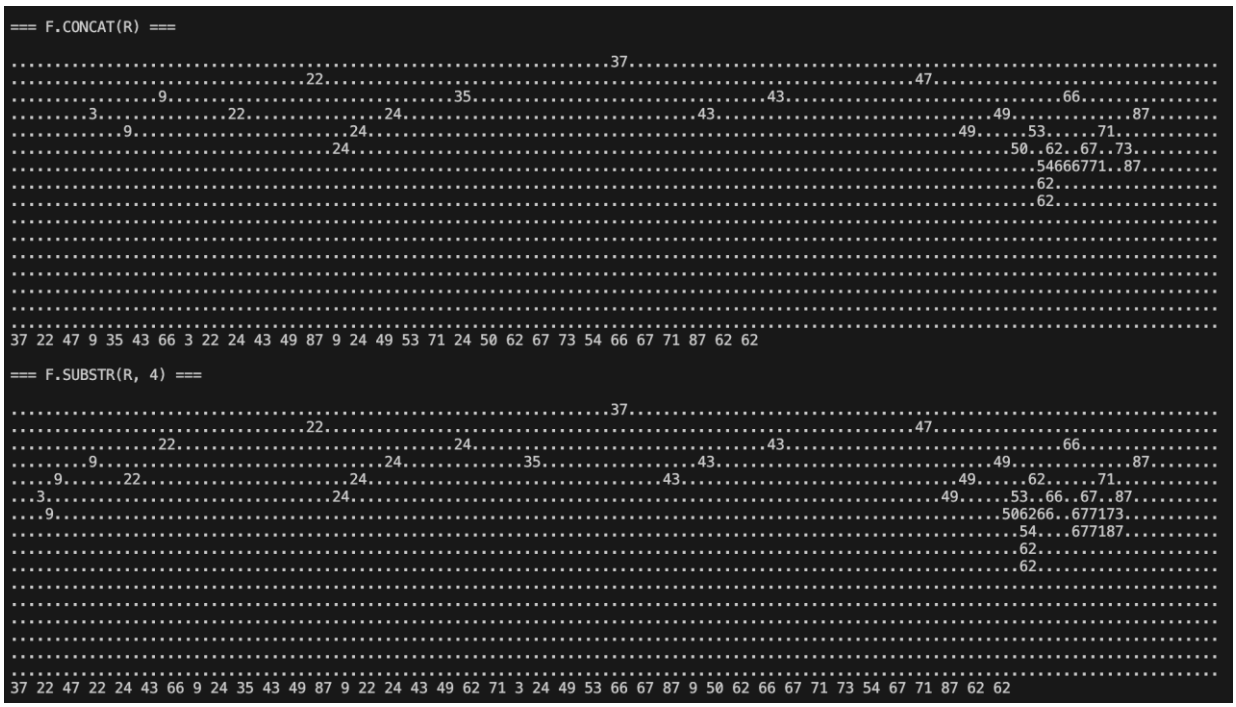
```
C:
.....................................................................69.....................................................................
..................................23....................................................................83..................................
.................15................................38..................................78...................................................
.........................19................29..............59...............................................................................
.......................................................51...................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
69 23 83 15 38 78 19 29 59 51
D:
.....................................................................38.....................................................................
..................................7.....................................................................87..................................
...................................................9...................................62...................................................
...............................................................................52..............84...........................................
...........................................................................39..............82...............................................
.........................................................................................63.................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
38 7 87 9 62 52 84 39 82 63
=== (C + D) ===

.....................................................................69.....................................................................
..................................23....................................................................83..................................
.................15................................38..................................78................................87.................
.........7...............19................29..............59..................................82................84.........................
.............9.................................38......51......62...........................................................................
.....................................................39..52......63.........................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
............................................................................................................................................
69 23 83 15 38 78 87 7 19 29 59 82 84 9 38 51 62 39 52 63
```



2. Реализация функций MERGE, CONCAT, SUBSTR для последовательностей


F:
37 35 47 24 53 3 50 54 73 62
R:
22 9 66 49 87 24 62 71 43 67
=== F.MERGE(R) ===
37 22 47 9 35 43 66 3 24 49 87 24 53 71 50 62 67 73 54 62



Вывод.

В ходе выполнения лабораторной работы был изучен принцип работы с комбинированными структурами данных и стандартной библиотекой шаблонов. Также было изучено и использовано на практике применение контейнеров для хранения данных. Был реализован алгоритм для множеств и последовательностей, который вычисляет по заданной формуле множество или последовательность, соответственно. В рамках изучения работы с последовательностями в структуре данных для множеств были реализованы функции MERGE (слияние), CONCAT (сцепление), SUBSTR (замена).

Приложение.

main.cpp

```
#include <iostream>
#include "BinaryTree.h"
int N = 10;

int main() {
    srand(time(0));
    BinaryTree<int> A(N);
    std::cout << "A: ";
    A.OutTree();
    A.printTree();
    BinaryTree<int> B(N);
    std::cout << "B: ";
    B.OutTree();
    B.printTree();
    std::cout << "=== A xor B ===" << "\n";
    (A^B).OutTree();
    (A^B).printTree();
    std::cout << "\n";
    BinaryTree<int> C(N);
    std::cout << "C: ";
    C.OutTree();
    C.printTree();
    BinaryTree<int> D(N);
    std::cout << "D: ";
    D.OutTree();
    D.printTree();
    std::cout << "=== (C + D) ===" << "\n";
    (C + D).OutTree();
    (C + D).printTree();
    std::cout << "\n";
    std::cout << "=== A xor B / (C + D) ===" << "\n";
    BinaryTree<int> L;
    L = (C + D);
    ((A ^ B) / L).OutTree();
    ((A ^ B) / L).printTree();
    std::cout << "\n";
    BinaryTree<int> E(N);
    std::cout << "E: ";
    E.OutTree();
    E.printTree();
    std::cout << "=== A xor B / (C + D) / E ===" << "\n";
    ((A ^ B) / L / E).OutTree();
    ((A ^ B) / L / E).printTree();
    std::cout << "\n";

    BinaryTree<int> F(N);
    std::cout << "F: ";
    F.OutTree();
    F.printTree();
```

```
    BinaryTree<int> R(N);
    std::cout << "R: ";
    R.OutTree();
    R.printTree();

    std::cout << "=== F.MERGE(R) ===" << "\n";
    F.MERGE(R);
    F.OutTree();
    F.printTree();
    std::cout << "\n";

    std::cout << "=== F.CONCAT(R) ===" << "\n";
    F.CONCAT(R);
    F.OutTree();
    F.printTree();
    std::cout << "\n";

    std::cout << "=== F.SUBSTR(R, 4) ===" << "\n";
    F.SUBSTR(R, 4);
    F.OutTree();
    F.printTree();
    std::cout << "\n";

    return 0;
}
```

BinaryTree.h

```
#include <iostream>
#include <stack>
#include <string>
#include <cstring>
int MAXNUM = 200;
template <typename T>

class BinaryTree {
private:
    struct Node {
        T key;
        int size; // мощность поддерева
        Node* left;
        Node* right;

        Node(const T k) {
            key = k;
            size = 1;
            left = nullptr;
            right = nullptr;
        }

    };
    Node* root;
    int len;
    char** SCREEN;
    int maxrow, offset, maxcol;
    void clrscr() {
        for(int i = 0; i < maxrow; i++) {
```

```
            memset(SCREEN[i], '.', maxcol);
        }
    }
    int Order(int num)
    {
        int order = 0;

        while(num){
            num /= 10;

            ++order;
        }

        return order;
    }

    void ItoA(int num, char* buffer, size_t buffer_size)
    {
        int num_order = Order(num);


        buffer[num_order] = '\0';

        for(int i = num_order; i > 0; --i){
            buffer[i - 1] = '0' + num % 10;

            num /= 10;
        }

    }
public:
    BinaryTree() {
        len = 0;
        maxrow = 16;
        maxcol = 140;
        offset = 70;
        SCREEN = new char * [maxrow];
        for(int i = 0; i < maxrow; ++i) {
            SCREEN[i] = new char [maxcol];
        }
        root = nullptr;
    }

    BinaryTree(int n) {
        len = 0;
        root = nullptr;
        maxrow = 16;
        maxcol = 140;
        offset = 70;
        SCREEN = new char * [maxrow];
        for(int i = 0; i < maxrow; ++i) {
            SCREEN[i] = new char [maxcol];
        }
        while (len < n) {
            insert(rand() % (MAXNUM - 1 + 1) + 1);
        }
    }
```

```
int Power() {
    return len;
}

void insert(int key) {
    if (root == nullptr) {
        root = insertRec(root, key);
    } else {
        insertRec(root, key);
    }
}

Node* insertRec(Node* root, int key) {
    if (root == nullptr) {
        len += 1;
        return new Node(key);
    }

    if (key <= root->key) {
        root->left = insertRec(root->left, key);
    } else if (key > root->key) {
        root->right = insertRec(root->right, key);
    }

    // обновляем размер поддерева
    root->size = 1;
    if (root->left != nullptr) {
        root->size += root->left->size;
    }
    if (root->right != nullptr) {
        root->size += root->right->size;
    }

    return root;
}

int getSize() {
    if (root == nullptr) {
        return 0;
    }
    return root->size;
}

int getSize(Node* node) {
    if (node == nullptr) {
        return 0;
    }
    return node->size;
}

int subtreeSize(int data) {
    Node* currentNode = root;
    int count = 0;

    while (currentNode != nullptr) {
        if (data == currentNode->key) {
            count = getSize(currentNode);
            break;
```

```
            } else if (data < currentNode->key) {
                currentNode = currentNode->left;
            } else {
                currentNode = currentNode->right;
            }
        }

        return count;
    }

    class Iterator {
    private:
        std::stack<Node *> stack;
        Node *current;
    public:
        Iterator(Node *root) {
            preOrderTravers(root);
            stack = foo(stack);
            if (!stack.empty()) {
                current = stack.top();
                stack.pop();
            } else current = nullptr;
        }
        void preOrderTravers(Node* root) {
            if (root != nullptr) {
                stack.push(root);
                preOrderTravers(root->left);
                preOrderTravers(root->right);
            }
        }

        Iterator &operator++() {
            if (!stack.empty()) {
                current = stack.top();
                stack.pop();
            } else current = nullptr;
            return *this;
        }
        Iterator &operator++(int) {
            auto res = *this;
            ++*this;
            return res;
        }

        bool operator!=(const Iterator& other) const {
            return (stack.size() != other.stack.size()) || (current !=
other.current);
        }
        bool operator==(const Iterator& other) const {
            return (stack.size() == other.stack.size()) && (current ==
other.current);
        }

        int& operator*() {
            return current->key;
        }
        std::stack<Node *> foo(std::stack<Node *>& in){
            std::stack<Node *> out;
```

```
        while(! in.empty() ){
            out.push(in.top());
            in.pop();
        }
        return out;
    }
};

Iterator begin() {
    return Iterator(root);
}
Iterator end() const {
    return Iterator(nullptr);
}


void printTree() {
    for (auto it = begin(); it != end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << "\n";
}

void MERGE(BinaryTree& other) {
    BinaryTree C;
    auto it1 = begin();
    auto it2 = other.begin();
    while (it1 != end() && it2 != other.end()) {
        C.insert(*it1);
        C.insert(*it2);
        it1++;
        it2++;
    }
    if (it1 == end()) {
        while (it2 != other.end()) {
            C.insert(*it2);
            it2++;
        }
    } else if (it2 == other.end()) {
        while (it1 != other.end()) {
            C.insert(*it1);
            it1++;
        }
    }
    root = C.root;
    len = C.len;
}

void CONCAT(BinaryTree& other) {
    for (auto it = other.begin(); it != other.end(); ++it) {
        insert(*it);
    }
}

void SUBSTR(BinaryTree& other, int n) {
    BinaryTree C;
    auto it = begin();
    for (int i = 0; i < n-1; i++) {
```

```
            C.insert(*it);
            ++it;
        }
        for (auto it = other.begin(); it != other.end(); ++it) {
            C.insert(*it);
        }
        while (it != end()) {
            C.insert(*it);
            ++it;
        }
        root = C.root;
        len = C.len;
    }


    BinaryTree operator+(BinaryTree& other) {
        BinaryTree merge;
        for (auto it = begin(); it != end(); ++it) {
            merge.insert(*it);
        }
        for (auto it = other.begin(); it != other.end(); ++it) {
            merge.insert(*it);
        }
        return merge;
    }
    void operator|=(BinaryTree& other) {
        root = (*this+other).root;
        len = (*this+other).len;
    }
    BinaryTree operator*(BinaryTree& other) {
        BinaryTree merge;
        for (auto it = begin(); it != end(); ++it) {
            for (auto it1 = other.begin(); it1 != other.end(); ++it1)
{
                if (*it1 == *it) {
                    merge.insert(*it);
                }
            }
        }
        return merge;
    }
    void operator&=(BinaryTree& other) {
        root = (*this*other).root;
        len = (*this*other).len;
    }
    BinaryTree operator^(BinaryTree& other) {
        BinaryTree merge;
        bool key;
        for (auto it = begin(); it != end(); ++it) {
            key = true;
            for (auto it1 = other.begin(); it1 != other.end(); ++it1)
{
                if (*it1 == *it) {
                    key = false;
                }
            }
            if (key) {
                merge.insert(*it);
```

```
            }
        }
        for (auto it = other.begin(); it != other.end(); ++it) {
            key = true;
            for (auto it1 = begin(); it1 != end(); ++it1) {
                if (*it1 == *it) {
                    key = false;
                }
            }
            if (key) {
                merge.insert(*it);
            }
        }
        return merge;
    }
    void operator^=(BinaryTree& other) {
        root = (*this^other).root;
        len = (*this^other).len;
    }
    BinaryTree operator/(BinaryTree& other) {
        BinaryTree C;
        bool key;
        for (auto it = begin(); it != end(); ++it) {
            key = true;
            for (auto it1 = other.begin(); it1 != other.end(); ++it1)
{
                if (*it1 == *it) {
                    key = false;
                }
            }
            if (key) {
                C.insert(*it);
            }
        }
        return C;
    }

    void OutNodes(Node* v, int r, int c) {
        int num = v->key;
        int num_order = Order(num);

        char* str = new char[num_order + 1];

        ItoA(num, str, num_order + 1);
        if (r && c && (c<maxcol)) SCREEN[ r - 1 ][ c - 1 ] = str[0];
        if (r && c && (c<maxcol) && str[1]) SCREEN[ r - 1 ][ c ] =
str[1];
        ; // вывод метки
        if (r < maxrow) {
            if (v->left) OutNodes(v->left, r + 1, c - (offset >> r));
//левый сын
            if (v->right) OutNodes(v->right, r + 1, c + (offset >>
r)); //правый сын
        }

    }
    void OutTree() {
        clrscr();
```

```
        OutNodes(root, 1, offset);
        for (int i = 0; i < maxrow; i++) {
            SCREEN[i][maxcol-1] = 0;
            std::cout << '\n' << SCREEN[i];
        }
        std::cout << '\n';
    }
};
```