МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра информатики и программирования

АНАЛИЗ ДВОИЧНОГО ДЕРЕВА ПОИСКА

ОТЧЕТ

студентки 2 курса 211 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета компьютерных наук и информационных технологий
Никитенко Яны Валерьевны

Саратов 2025

СОДЕРЖАНИЕ

1 Текст программы

```
//
struct node {
 int key;
 node* left;
 node* right;
 node(int k) : key(k), left(nullptr), right(nullptr) {}
};

//

HANDLE outp = GetStdHandle(STD_OUTPUT_HANDLE);
CONSOLE_SCREEN_BUFFER_INFO csbInfo;
node* root = nullptr;

void max_height(node* x, short& max, short deepness = 1)
{ // требует проверки на существование корня
 if (deepness > max) max = deepness;
 if (x->left) max_height(x->left, max, deepness + 1);
 if (x->right) max_height(x->right, max, deepness + 1);
}
bool isSizeOfConsoleCorrect(const short& width, const short& height) {
 GetConsoleScreenBufferInfo(outp, &csbInfo);
 COORD szOfConsole = csbInfo.dwSize;
 if (szOfConsole.X < width && szOfConsole.Y < height)
    cout << "Please increase the height and width of the terminal. ";
 else if (szOfConsole.X < width) cout << "Please increase the width of the terminal. ";
 else if (szOfConsole.Y < height) cout << "Please increase the height of the terminal. ";
 if (szOfConsole.X < width || szOfConsole.Y < height) {
 cout << "Size of your terminal now: " << szOfConsole.X << ' ' << szOfConsole.Y
   << ". Minimum required: " << width << ' ' << height << ".\n";
  return false;
 }
```

```
  return true;
}

void print_helper(node* x, const COORD pos, const short offset) {
 SetConsoleCursorPosition(outp, pos);
 cout << setw(offset + 1) << x->key;
 if (x->left) print_helper(x->left, { pos.X, short(pos.Y + 1) }, offset >> 1);
 if (x->right) print_helper(x->right, { short(pos.X + offset), short(pos.Y + 1) }, offset >>
}

void print() {
 if (root) {
  short max = 1;
  max_height(root, max);
  short width = 1 << max + 1, max_w = 128; // вычисляем ширину вывода
  if (width > max_w) width = max_w;
  while (!isSizeOfConsoleCorrect(width, max)) system("pause");
 for (short i = 0; i < max; ++i) cout << '\n'; // резервируем место для вывода
  GetConsoleScreenBufferInfo(outp, &csbInfo); // получаем данные
  COORD endPos = csbInfo.dwCursorPosition;
  print_helper(root, { 0, short(endPos.Y - max) }, width >> 1);
  SetConsoleCursorPosition(outp, endPos);
  SetConsoleTextAttribute(outp, 7);
 }
}



// Добавить узел
void insert(node*& root, int key) {
    if (!root) {
        root = new node(key);
    }
    else if (key < root->key) {
```

```
        insert(root->left, key);
    }
    else {
        insert(root->right, key);
    }
}
//

// Найти
node* findMin(node* root) {
    while (root && root->left) {
        root = root->left;
    }
    return root;
}
//

// Удалить узел
node* deleteNode(node* root, int key) {
    if (!root) return root;
    if (key < root->key) {
        root->left = deleteNode(root->left, key);
    }
    else if (key > root->key) {
        root->right = deleteNode(root->right, key);
    }
    else {
        if (!root->left) {
            node* temp = root->right;
            delete root;
            return temp;
        }
        else if (!root->right) {
            node* temp = root->left;
```

```
            delete root;
            return temp;
        }
        node* temp = findMin(root->right);
        root->key = temp->key;
        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}
//

// Поиск
node* search(node* root, int key) {
    if (!root || root->key == key) return root;
    if (key < root->key) return search(root->left, key);
    return search(root->right, key);
}
//

// Обход в прямом порядке
void inorder(node* root) {
    if (root) {
        inorder(root->left);
        cout << root->key << " ";
        inorder(root->right);

    }
}
//

// Обход в симметричном порядке
void preorder(node* root) {
    if (root) {
        cout << root->key << " ";
```

```
        preorder(root->left);
        preorder(root->right);
    }
}
//

// Обход в обратном порядке
void postorder(node* root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        cout << root->key << " ";
    }
}
//

// Меню
void menu() {
    cout << "1. Добавить узел\n";
    cout << "2. Удалить узел\n";
    cout << "3. Поиск узла\n";
    cout << "4. Обход в прямом порядке\n";
    cout << "5. Обход в симметричном порядке\n";
    cout << "6. Обход в обратном порядке\n";
    cout << "7. Вывести дерево\n";
    cout << "0. Выход\n";
}
//

int main() {
    setlocale(LC_ALL, "Russian");
    int choice, key;
    while (true) {
        menu();
```

```
cout << "Выберите действие: ";
cin >> choice;

switch (choice) {
case 1:
    while (true) {
        cout << "-1 - выход из цикла" << endl;
        cin >> key;
        if (key == -1) break; // Выход из цикла добавления
        insert(root, key);
    }
    break;

case 2:
    cout << "ключ для удаления: ";
    cin >> key;
    root = deleteNode(root, key);
    break;
case 3:
    cout << "ключ для поиска: ";
    cin >> key;
    if (search(root, key)) {
        cout << "узел с ключом " << key << " найден.\n";
    }
    else {
        cout << "узел с ключом " << key << " не найден.\n";
    }
    break;
case 4:
    cout << "обход в прямом порядке: ";
    preorder(root);
    cout << endl;
    break;
case 5:
```

```
            cout << "обход в симметричном порядке: ";
            inorder(root);
            cout << endl;
            break;
        case 6:
            cout << "обход в обратном порядке: ";
            postorder(root);
            cout << endl;
            break;
        case 7:
            print();
            break;
        case 0:
            return 0;
        default:
            cout << "\n";
        }
    }
    return 0;
}
```

2 Анализ случаев

Лучший случай

Идеально сбалансированное дерево. Высота такого дерева равна $\lceil \log_2(n+1) \rceil$ или $O(\log n)$, где n — количество элементов в дереве.

Худший случай

Вырожденное дерево (вытянутое в цепочку). Высота такого дерева равна n, где n — количество элементов.

Средний случай

Для случайно построенного дерева ожидаемая высота составляет $O(\log n)$.

3 Вставка в двоичном дереве

Лучший случай

$O(\log n)$ — вставка в сбалансированное дерево.

Средний случай

$O(\log n)$ — для случайных данных.

Худший случай

$O(n)$ — вставка в вырожденное дерево.

Математически сложность вставки определяется высотой дерева, так как алгоритм проходит путь от корня до места вставки.

4 Удаление в двоичном дереве

Лучший случай

$O(\log n)$ — удаление листа в сбалансированном дереве.

Средний случай

$O(\log n)$.

Худший случай

$O(n)$ — удаление в вырожденном дереве.

При удалении узла с двумя потомками требуется найти минимальный элемент в правом поддереве (операция за $O(h)$), что не меняет асимптотическую сложность.

5 Поиск в двоичном дереве

Лучший случай

$O(1)$ — если искомый элемент находится в корне.

Средний случай

$O(\log n)$ — для сбалансированного дерева.

Худший случай

$O(n)$ — для вырожденного дерева.

6 Обходы дерева

Все три типа обхода (прямой, симметричный, обратный) имеют сложность:

$$O(n)$$

где n — количество узлов в дереве, так как каждый узел посещается ровно один раз.

7 Расход памяти

Память, потребляемая двоичным деревом поиска:

- Каждый узел содержит: значение (4 байта для int) + 2 указателя (8 байт каждый на 64-битной системе)
- Общий размер узла: $4 + 8 + 8 = 20$ байт
- Общая память для n узлов: $O(n)$

Дополнительная память для рекурсивных операций (в худшем случае): $O(h)$, где h — высота дерева.