Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления
Кафедра — интеллектуальных информационных технологий

К защите допустить:

Заведующий кафедрой ИИТ
Д.В. Шункевич

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к лабораторной работе
на тему:

Нелинейные списки

Студент гр. 321701 А. С. Астахов
Руководитель С.И. Матюшкин

Минск 2024

Цель: изучить алгоритмы обработки данных с использованием нелинейных структур в виде дерева.
Задача: составить и отладить программу.

Индивидуальное задание

Разработать проект для работы с деревом поиска, содержащий следующие обработчики, которые должны:

– ввести информацию из компоненты *StringGrid* в массив. Каждый элемент массива должен содержать строку текста и целочисленный ключ (например, ФИО и номер паспорта);

– внести информацию из массива в дерево поиска;

– сбалансировать дерево поиска;

– добавить в дерево поиска новую запись;

– по заданному ключу найти информацию и отобразить ее;

– удалить из дерева поиска информацию с заданным ключом;

– распечатать информацию прямым, обратным обходом и в порядке возрастания ключа;

– решить одну из поставленных задач и решение оформить в виде блок-схемы.

1. Поменять местами информацию, содержащую максимальный и минимальный ключи.

```
#include <iostream>
#include <vector>
using namespace std;

struct Tree {
    int info;
    string value;
    Tree *left, *right;
} *root;

Tree *leave(int in, string value) {
    Tree *t = new Tree;

    t -> value = value;
    t -> info = in;
    t -> left = t -> right = NULL;

    return t;
}

void addList(Tree *root, int key, string value) {
    Tree *before = nullptr, *t;
    bool find = true;

    t = root;

    while (t && find) {
        before = t;
```

```
        if (key == t -> info) {
            find =  false;
            cout << "повторяющийся ключ" << endl;
            break;
        } else {
            if (key < t -> info)
                t = t -> left;
            else
                t = t -> right;
        }
    }

    if (find) {
        t = leave(key, value);

        if (key < before -> info) //?
            before -> left = t;
        else
            before -> right = t;
    }

}

void viewStraight(Tree *root, int level) {

    for (int i = 0; i < level; i++) {
        cout << " ";
    }
    cout << " >";
    cout << root -> info << endl;

    if (root -> left != NULL)
        viewStraight(root -> left, level + 1);

    if (root -> right != NULL)
        viewStraight(root -> right, level + 1);
}



void ascendingView(Tree *root, int level) {
    if (root -> right != NULL)
        ascendingView(root -> right, level + 1);

    for (int i = 0; i < level; i++) {
        cout << "   ";
    }
    cout << ">";
    cout << root -> info << endl;

    if (root -> left != NULL)
        ascendingView(root -> left, level + 1);
}



void viewReverse(Tree *root, int level) {
    if (root -> right != NULL)
        viewReverse(root -> right, level + 1);

    for (int i = 0; i < level; i++) {
        cout << "   ";
    }
    cout << ">";
    cout << root -> info << endl;
```

```
    if (root -> left != NULL)
        viewReverse(root -> left, level + 1);
}


Tree *getValues(Tree *root, vector<string> arr, vector<int> arrOfKeys) {

    for (int i = 0; i < arr.size(); i++) {
        addList(root, arrOfKeys.at(i), arr.at(i));
    }

    return root;
}

void findElement(Tree *root, int key) {
    Tree *t = root;

    while (t) {
        if (key == t -> info) {
            cout << t->value << " имеет номер паспорта: " << key << endl;
            break;
        } else {
            if (t -> info < key)
                t = t -> right;

            if (t -> info > key)
                t = t -> left;
        }
    }
}


Tree *delElement(Tree* root, int key) {
    Tree *del, *delBefore, *R, *RBefore = nullptr;

    del = root;
    delBefore = NULL;

    while (del != NULL && del -> info != key) {
        delBefore = del;

        if (del -> info > key)
            del = del -> left;
        else
            del = del -> right;
    }

    if (del == NULL) {
        cout << "Значение не найдено" << endl;
        return root;
    }

    if (del -> right == NULL)
        R = del -> left;
    else
        if (del -> left == NULL)
            R = del -> right;
    else {
        RBefore = del;
        R = del -> left;

        while (R -> right != NULL) {
            RBefore = R;
            R = R -> right;
        }
```

```
        if (RBefore == del)
            R -> right = del -> right;
        else {
            R -> right = del -> right;
            RBefore -> right = R -> left;
            R -> left = RBefore;
        }

    }

    if (del == root)
        root = R;
    else
        if (del -> info < delBefore -> info)
            delBefore -> left = R;
        else
            delBefore -> right = R;
        delete del;
        return root;

}

void Balans(Tree **p, int n, int k, vector<int> a, vector<string> arr) {
    if (n == k) {
        *p = NULL;
        return;

    } else {
        int m = (n + k) / 2;
        *p = new Tree;
        (*p) -> info = a.at(m);
        (*p) -> value = arr.at(m);
        Balans(&(*p) -> left, n, m, a, arr);
        Balans(&(*p) -> right, m + 1, k, a, arr);
    }

}

void Sort(vector<int> *arrOfKeys, vector<string> *arr) {
    int i = 0;
    if (i == (*arr).size() - 2)
        return;
    while (true) {
        if ((*arrOfKeys).at(i) > (*arrOfKeys).at(i + 1)) {
            int temp;
            string temp1;

            temp = (*arrOfKeys).at(i);
            temp1 = (*arr).at(i);

            (*arrOfKeys).at(i) = (*arrOfKeys).at(i + 1);
            (*arrOfKeys).at(i + 1) = temp;

            (*arr).at(i) = (*arr).at(i + 1);
            (*arr).at(i + 1) = temp1;



        } else {
            i++;
        }

        if (i == (*arr).size() - 1)
            return;

    }
```

```
}


Tree *replaceMinMax(Tree *root) {
    Tree *min = root, *max = root, *t = root;

    if (t -> left == NULL && t -> right == NULL) {
        cout << "Имеется всего один элемент" << endl;
        return root;
    }

    if (t -> left == NULL) {

        while (t -> right != NULL) {
            t = t -> right;
            if (max -> info < t -> info)
                max = t;
        }

        min = root;

        string temp = min -> value;

        min -> value = max -> value;

        max -> value = temp;

            cout << "После заменты " << min -> value << " имеет минимальный
номер " << min ->info << endl;
            cout << "После заменты " << max -> value << " имеет максимальный
номер " << max ->info << endl;
        return root;

    } else if (t -> right == NULL) {

        while (t -> left != NULL) {
            t = t -> left;
            if (min -> info > t -> info)
                min = t;
        }

        max = root;

        string temp = min -> value;

        min -> value = max -> value;

        max -> value = temp;

            cout << "После заменты " << min -> value << " имеет минимальный
номер " << min ->info << endl;
            cout << "После заменты " << max -> value << " имеет максимальный
номер " << max ->info << endl;

        return root;
    }

    while (t -> left != NULL) {
        t = t -> left;
        if (min -> info > t -> info)
            min = t;

    }
    t = root;
    t = t -> right;
```

```
    while (t -> left != NULL) {
        t = t -> left;
        if (min -> info > t -> info)
            min = t;

    }

    t = root;
    while (t -> right != NULL) {
        t = t -> right;
        if (max -> info < t -> info)
            max = t;
    }
    t = root;
    t = t -> left;

    while (t -> right != NULL) {
        t = t -> right;
        if (max -> info < t -> info)
            max = t;

    }

        cout << "До заменты мест " << min -> value << " имеет минимальный номер
" << min ->info << endl;
        cout << "До заменты мест " << max -> value << " имеет максимальный номер
" << max ->info << endl;

    string temp = min -> value;

    min -> value = max -> value;

    max -> value = temp;

        cout << "После заменты " << min -> value << " имеет минимальный номер "
<< min ->info << endl;
        cout << "После заменты " << max -> value << " имеет максимальный номер "
<< max ->info << endl;
    return root;
}


int main(int argc, const char * argv[]) {
    root = leave(6, "Артём");


    cout << "Сколько записей вы хотите сделать?" << endl;
    int count;
    cin >> count;

    vector<string> arr(count);
    vector<int> arrOfKeys(count);

    for (int i = 0; i < count; i++) {
        cout << "\nВведите ФИО" << endl;
        fflush(stdin);
        getline(cin, arr.at(i));

        cout << "\nВведите номер паспрта" << endl;
        cin >> arrOfKeys.at(i);
    }
    if (count > 1)
        Sort(&arrOfKeys, &arr);
    Balans(&root, 0, count, arrOfKeys, arr);
    viewStraight(root, 0);
    cout << "___________" << endl;
```

```
viewReverse(root, 0);
cout << "___________" << endl;
ascendingView(root, 0);
cout << "___________" << endl;
root = replaceMinMax(root);
return 0;
```

```
Введите номер паспрта
31233

Введите ФИО
Мельник

Введите номер паспрта
23454

Введите ФИО
Иванов

Введите номер паспрта
55555
 >31233
  >23454
  >55555

-----------
  >55555
>31233
  >23454

-----------
  >55555
>31233
  >23454

-----------
До заменты мест Мельник имеет минимальный номер 23454
До заменты мест Иванов имеет максимальный номер 55555
После заменты Иванов имеет минимальный номер 23454
После заменты Мельник имеет максимальный номер 55555
Program ended with exit code: 0
```

Вывод: изучил алгоритмы обработки данных с использованием нелинейных структур в виде дерева, составил и отладил программу.