МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Липецкий Государственный Технический Университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

Лабораторная работа

по технологиям программирования №4

“ Программирование алгоритмов реализации и обработки древовидных

структур неспециального вида ”

Студент \_\_\_\_\_\_\_\_\_\_\_\_ Станиславчук С.М.

(подпись, дата)

Группа АС-21-1

Руководитель

Доцент, кандидат наук \_\_\_\_\_\_\_\_\_\_\_\_ Хромов А. М.

(подпись, дата)

Липецк 2022 г.

**Содержание:**

1. Титульный лист.

2. Задание кафедры, соответствующее варианту, номер варианта.

3. Цель работы.

4. Блок-схема алгоритма разработанной программы

5. Текст программы

6. Контрольный пример

7. Выводы по работе.

## 2. Задание кафедры

Осуществить программную реализацию одного из деревьев, выбранных из табл.1 приложения, в соответствии с заданным номером варианта. Программой должны выполнятся функции по работе с деревом. Следует предусмотреть удобный пользовательский интерфейс, позволяющий создавать дерево и осуществлять его обработку. По результатам работы программы необходимо оценить зависимость среднего времени операции произвольного доступа к элементу от размера дерева (количества уровней и количества узлов). В программе следует предусмотреть генерацию дерева заданной степени, предоставляя пользователю возможность указывать среднюю степень узла и дисперсию. Например, если для некоторого дерева среднее = 6, а дисперсия = 2, то в среднем по дереву у каждого узла будет примерно от 4 до 8 потомков (необходимо учитывать также максимальную степень дерева). Для генерации количества узлов, следует использовать закон распределения, указанный в табл.1 для выбранного варианта задания. Функции, помимо основных: просмотр дерева, просмотр последовательности обхода дерева.

Вариант 13

Ключ: **int**

Удаляемый узел: **заменяется самым левым дочерним узлом**

Распределение: **нормальное**

Указатели: **указатели**

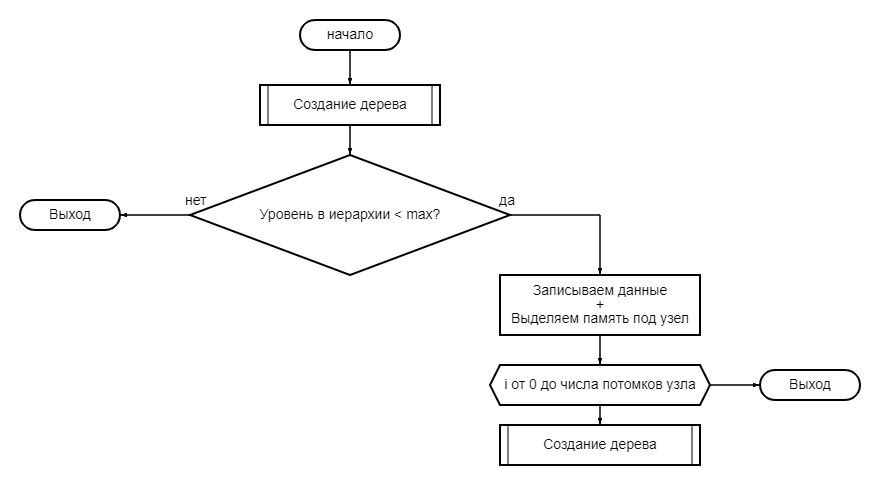
Степень дерева: **4**

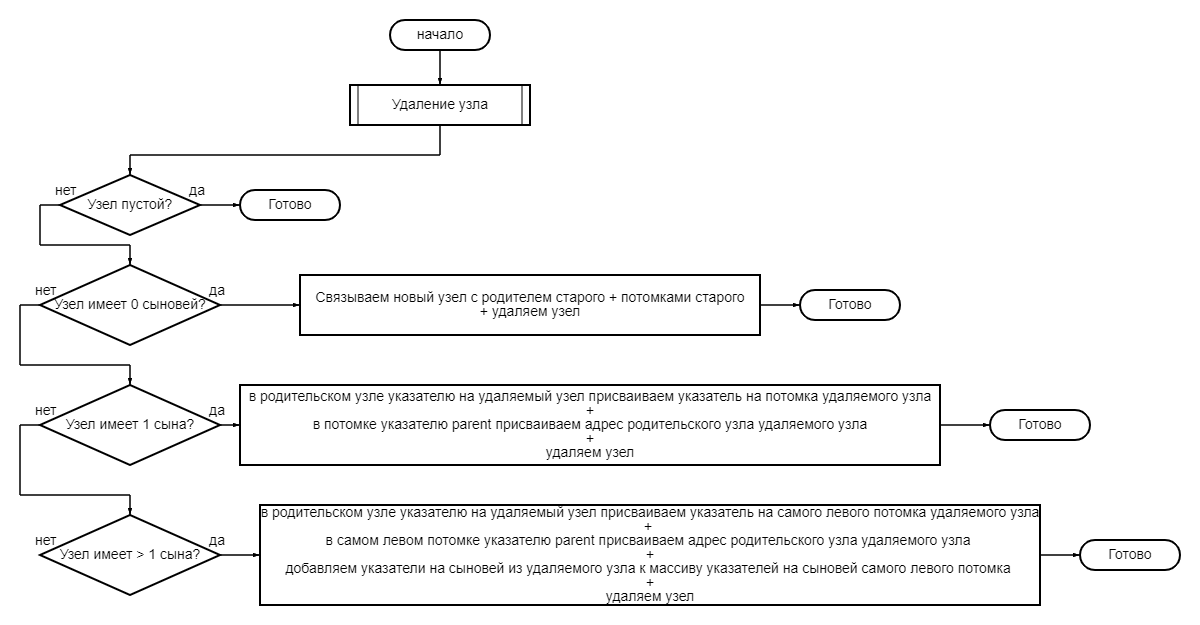
Метод обхода: **Прямой**

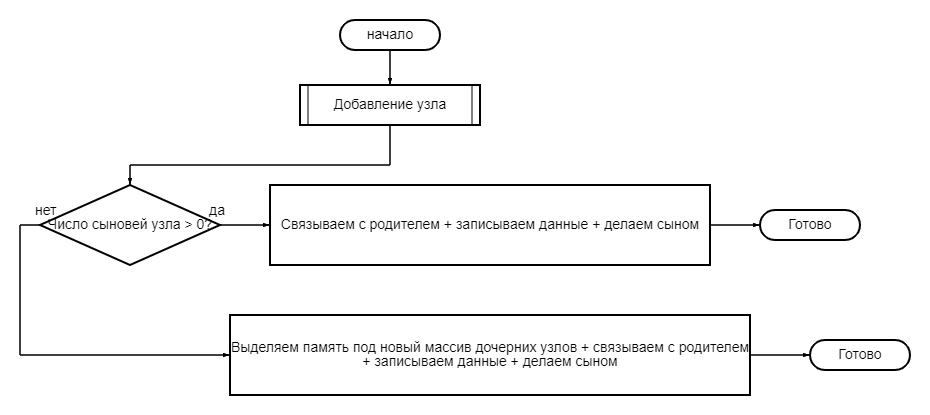
## 3. Цель работы

Получение навыков реализации и использования деревьев неэффективного вида, анализ эффективности работы с ним.

**4. Блок-схема**







**4. Код программы**

#include <iostream>

using namespace std;

struct Node

{

int key = 0; //ключ

int id = 0; //индекс

int level = 0; //уровень в иерархии

Node\* parent = nullptr; //указатель на родителя

Node\*\* son = nullptr; //массив указателей на сыновей

int count\_son = 0; //количество сыновей

};

Node\* Root(Node\* node)

{

return node;

}

Node\* createTree(Node\* node, int id, int level, int dispersion, int maxlvl)

{

if (node->level < maxlvl) {

node->id = id;

node->level = level;

node->key = rand() % 100;

node->count\_son = rand() % ((4 + dispersion) - (4 - dispersion)+1)+(4-dispersion); //dispersion can't be > than 4 [b-a+1] + a

node->son = new Node \* [node->count\_son];

level++;

int iddd = 0;

for (int i = 0; i < node->count\_son; i++)

{

node->son[i] = new Node;

node->son[i]->parent = node;

if (node->level!=0)

node->son[i]->id = i; //i

node->son[i]->level = level;

node->son[i]->key = rand() % 100;

}

for (int i = 0; i < node->count\_son; i++)

createTree(node->son[i], i, level, dispersion, maxlvl);

return node;

}

else node->count\_son = 0;

return node;

}

void printTree(Node\* node)

{

if (node != nullptr) {

for (int i = 0; i < node->level; i++) cout << "| ";

if (node->son != NULL) cout << "[+]";

cout << "(" << node->level << "-" << node->key << ")" << endl;

if (node->count\_son > 0)

for (int i = 0; i < node->count\_son; i++)

printTree(node->son[i]);

}

}

Node\* Find(Node\* node, int key) // Ищет узел с указанным ключом

{

if (node->key == key)

return node;

if (node->count\_son > 0)

{

for (int i = 0; i < node->count\_son; i++)

{

auto found = Find(node->son[i], key);

if (found)

return found;

}

}

return nullptr;

}

Node\* FindNonKey(Node\* node, int level, int id) // Ищет узел с указанным ключом

{

if (node->level == level && node->id == id)

return node;

if (node->count\_son > 0)

{

for (int i = 0; i < node->count\_son; i++)

{

auto found = FindNonKey(node->son[i], level, id);

if (found)

return found;

}

}

return nullptr;

}

int editKey(Node\* node, int key, int keykey)

{

auto found = Find(node, key);

if (found) {

found->key = keykey;

return found->key;

}

return 0;

}

Node\* parentNode(Node\* node, int key)//возвращает предка

{

auto Node = Find(node, key);

auto ParentNode = Node->parent;

return ParentNode;

}

bool nodeKill(Node\* root, int key)

{

Node\* DelNode = Find(root, key); Node\* ParentNode = DelNode->parent; Node\* NewNode;

//0-ой случай, когда дерево пустое

if (DelNode == NULL)

return true;

if (DelNode->count\_son == 0)

{

ParentNode->son[DelNode->id] = nullptr;

DelNode->parent = nullptr;

free(DelNode);

return true;

}

//1-ый случай, если сын один

if (DelNode->count\_son == 1)

{

NewNode = DelNode->son[0];

NewNode->parent = ParentNode;

ParentNode->son[ParentNode->count\_son - 1] = NewNode;

NewNode->level = DelNode->level;

NewNode->id = DelNode->id;

delete DelNode;

return true;

}

//2-ой случай, когда дерево имеет несколько потомков

if (DelNode->count\_son > 1)

{

NewNode = DelNode->son[0];

NewNode->level = DelNode->level;

NewNode->id = DelNode->id;

for (int i = 0; i < ParentNode->count\_son; i++)

{

ParentNode->son[i] = NewNode;

break;

}

NewNode->parent = ParentNode;// указателю родителя дочернего узла присваиваем адрес родительского узла удаляемого элемента теперь удаляемый элемент исключен из дерева и как бы висит в воздухе вместе с потомками, кроме самого левого

// нужно оставшихся потомков перепривязать к новому узлу

Node\*\* tmp = new Node \* [NewNode->count\_son + DelNode->count\_son - 1];

for (int j = 0; j < NewNode->count\_son; j++)

tmp[j] = NewNode->son[j];

for (int j = 1; j < DelNode->count\_son; j++) // копируем все указатели на потомков из удаляемого узла в новый массив нового узла

tmp[NewNode->count\_son + j - 1] = DelNode->son[j];

delete[] NewNode->son; // удаляем старый массив указателей на потомков

NewNode->son = tmp; // запоминаем новый массив указателей

NewNode->count\_son = DelNode->count\_son - 1; // сохраняем новое количество потомков

for (int i = 0; i < NewNode->count\_son; i++)

NewNode->son[i]->level = NewNode->level+1;

// Теперь удаляем старый узел

delete[] DelNode->son; // сначала удаляем массив указателей на потомков

delete DelNode;

return true;

}

return false;

}

int countSon(Node\* node, int key)

{

auto found = Find(node, key);

return found->count\_son;

}

Node\* nodeAdd(Node\* node, int key, int keykey)

{

auto found = Find(node, key);

Node\* NewNode = new Node;

if (found->count\_son > 0)

{

NewNode->parent = found;

NewNode->son = NULL;

NewNode->key = keykey;

NewNode->level = found->level + 1;

found->count\_son++;

found->son[found->count\_son - 1] = NewNode;

}

else

{

found->son = new Node\*[1];

found->son[0] = NewNode;

NewNode->parent = found;

NewNode->son = NULL;

NewNode->key = keykey;

NewNode->level = found->level + 1;

NewNode->id = 0;

found->count\_son++;

}

return NewNode;

}

Node\* leftSonNode(Node\* node, int key)//возвращает самого левого сына

{

auto Node = Find(node, key);

if (Node->count\_son > 0) {

auto SonNode = Node->son[0];

return SonNode;

}

return NULL;

}

Node\* rightSiblingNode(Node\* node, int key)//возвращает самого правого брата

{

auto found = Find(node, key);

auto SiblingNode = found->parent->son[found->parent->count\_son-1];

return SiblingNode;

}

void clearTree(Node\* node)

{

if (node == NULL) return;

for (int i = 0; i < node->count\_son; i++)

clearTree(node->son[i]);

delete node;

}

int main()

{

setlocale(LC\_ALL, "");

int key, level, id, keykey, dispersion; Node\* root; size\_t N; int const maxlvl = 2;

root = new Node; root->parent = nullptr;

cout << "Введите значение дисперсии: "; cin >> dispersion;

if (dispersion >= 4)

{

cout << "Дисперсия не может быть >= 4";

return 0;

}

root->parent = root; // Это для 7

root = createTree(root, 0, 0, dispersion, maxlvl);

cout << "\n-----Получившееся дерево-----\n";

printTree(root);

while (1) {

cout << "\n---Главное меню---";

cout << "\n1. Вернуть корень.";

cout << "\n2. Вернуть ключ указанного узла.";

cout << "\n3. Изменить ключ узла.";

cout << "\n4. Поиск в дереве по ключу.";

cout << "\n5. Добавить узел как дочернюю вершину.";

cout << "\n6. Удалить узел по ключу.";

cout << "\n7. Вернуть родителя узла.";

cout << "\n8. Вернуть самого левого сына узла.";

cout << "\n9. Вернуть правого соседа (левого, если !правый).";

cout << "\n10. Сделать дерево пустым.";

cout << "\n11. Вывести дерево.";

cout << "\n0. Выход.\n";

cin >> N;

if (N == 1)

{

auto rootc = Root(root);

cout << rootc->key;

}

if (N == 2)

{

cout << "\nВведите уровень и индекс узла, ключ которого хотите найти > ";

cin >> level >> id;

auto found = FindNonKey(root, level, id);

cout << found->key << "\n";

}

if (N == 3)

{

cout << "\nВведите ключ узла, который хотите изменить > ";

cin >> key;

cout << "\nВведите новый ключ > ";

cin >> keykey;

if (editKey(root, key, keykey)) cout << "\nКлюч изменён\n";

else cout << "\nОшибка в изменении ключа\n";

}

if (N == 4)

{

cout << "\nВведите ключ узла, который хотите найти > "; cin >> key;

auto found = Find(root, key);

cout << found->level << "-" << found->id << "\n";

}

if (N == 5)

{

cout << "\nВведите ключ узла, которому хотите добавить сына > "; cin >> key;

cout << "\nВведите ключ узла, который хотите присвоить новому сыну > "; cin >> keykey;

if (nodeAdd(root, key, keykey)) cout << "\nДочерний узел добавлен\n";

else cout << "\nДобавить узел не удалось\n";

}

if (N == 6)

{

cout << "\nВведите ключ узла, которого хотите убить > "; cin >> key;

if (nodeKill(root, key)) cout << "\nУзел удалён\n";

else cout << "\nОшибка в удалении узла\n";

}

if (N == 7)

{

cout << "\nВведите ключ узла, если хотите познакомиться с его родителями > "; cin >> key;

auto found = parentNode(root, key);

if (found) cout << found->key;

else cout << "\nВернуть отца не удалось...\n";

}

if (N == 8)

{

cout << "\nВведите ключ узла, если хотите узнать его левого сына поближе > "; cin >> key;

auto found = leftSonNode(root, key);

if (found) cout << found->key;

else cout << "\nВернуть сына не удалось...\n";

}

if (N == 9)

{

cout << "\nВведите ключ узла, если хотите узнать его правого брата поближе > "; cin >> key;

auto found = rightSiblingNode(root, key);

if (found) cout << found->key;

else cout << "\nВернуть брата не удалось...\n";

}

if (N == 10) {

clearTree(root);

cout << "\nДерево очищено";

}

if (N == 11)

{

printTree(root);

}

/\*if (N == 12) {

cout << "Введите ключ: "; cin >> key;

cout << countSon(root, key);

}\*/

if (N == 0) {

clearTree(root); //очищаем память + выходим

break;

}

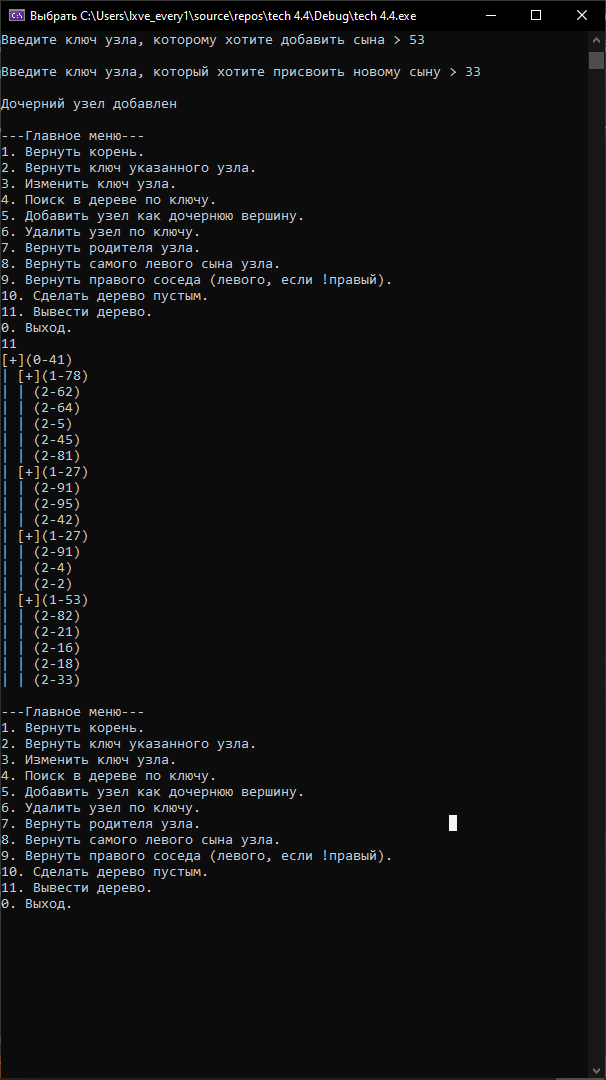
}

return 0;

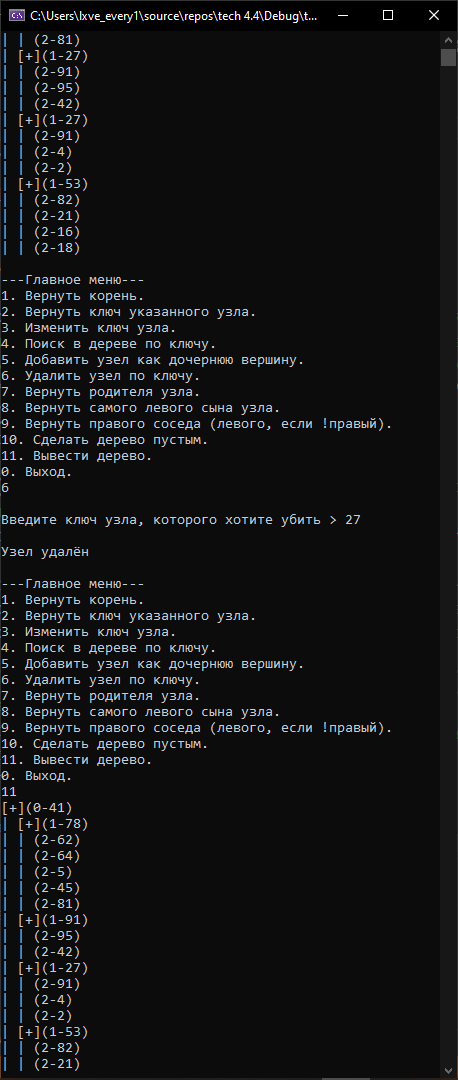
}

**6. Тест программы.**

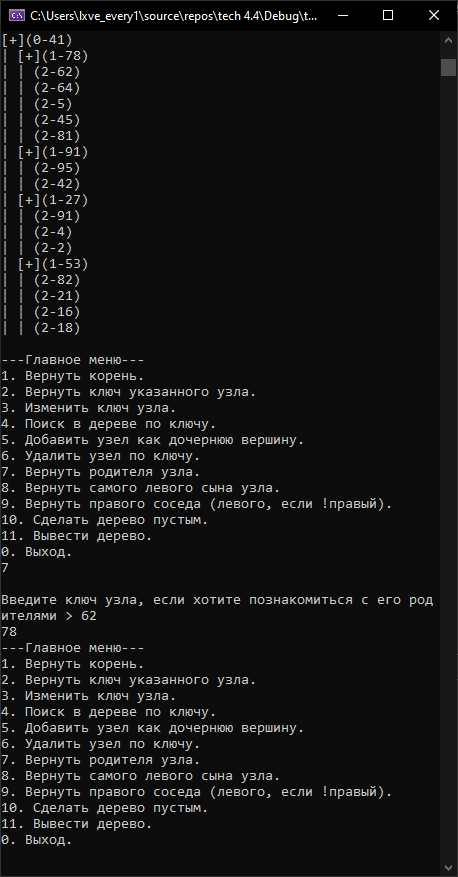
Добавление узла:



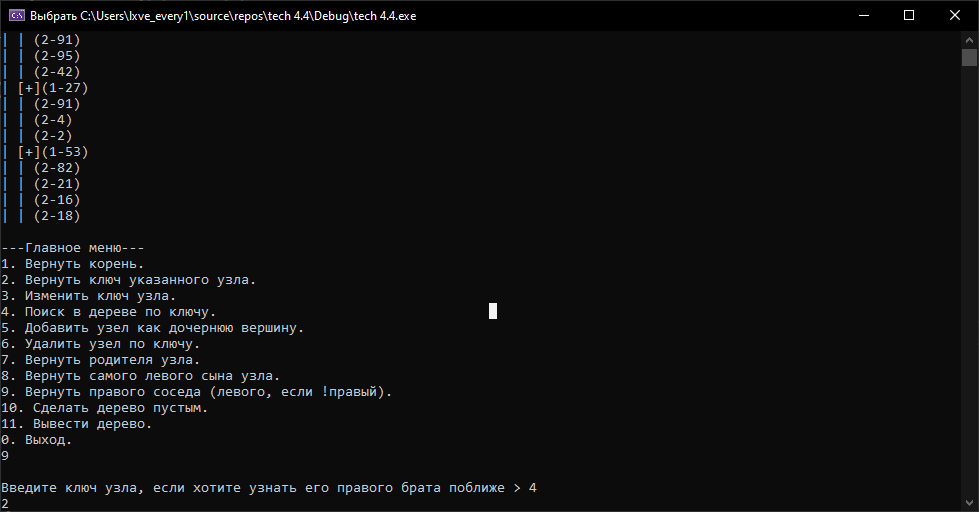
Удаление узла: удаляемый узел заменился на самый левый дочерний



Узнать родителя узла:



Узнать правого соседа:



Вывод: написал программу, реализующую Generic Tree.