

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

**КУРСОВАЯ РАБОТА**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Бинарное дерево поиска с рандомизированной вставкой**

Студент гр. 9303

\_\_\_\_\_

Ефимов М.Ю.

Преподаватель

\_\_\_\_\_

Филатов Ар.Ю.

Санкт-Петербург

2020

## ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Ефимов М.Ю.

Группа 9303

Тема работы : Бинарное дерево поиска с рандомизированной вставкой  
( Демонстрация )

Исходные данные:

Создать бинарное дерево с использованием рандомизированной вставки.  
Максимально подробно демонстрировать и объяснять происходящие процессы.

Содержание пояснительной записки:

Аннотация

Введение

Основные теоретические положения.

Описание кода программы

Заключение

Список используемых источников

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 06.11.2020

Дата сдачи реферата: 25.12.2020

Дата защиты реферата: 25.12.2020

Студент

\_\_\_\_\_

Ефимов М.Ю.

Преподаватель

\_\_\_\_\_

Филатов Ар.Ю.

## АННОТАЦИЯ

Курсовая работа представляет собой программу ,предназначенную для демонстрации создания бинарного дерева с рандомизированной вставкой. На вход подается последовательность значений, а результатом работу программы является файл формата “.pdf” где максимально подробно показаны шаги создания дерева. Для создания визуализации графа использована утилита “Graphviz”, а для конвертации в один файл “ ImageMagick”. Код программы написан на языке C++, запуск подразумевается на операционных системах семейства Linux. Для проверки работоспособности программы проводилось тестирование. Исходный код, скриншоты, показывающие корректную работу программы, и результаты тестирования представлены в приложениях.

## СОДЕРЖАНИЕ

	Введение	4
1.	Основные теоретические положения	6
2.	Описание работы	8
2.1.	Описание структуры данных и функций	8
2.2.	Описание алгоритма	8
3.	Тестирование	10
	Заключение	11
	Список использованных источников	12
	Приложение А. Программный код	13
	Приложение Б. Тестирование	21
	Приложение В. Сравнение	23

## **ВВЕДЕНИЕ**

Формальная постановка задачи:” Случайные БДП с рандомизацией. Демонстрация”. Цель работы — разработка программы для демонстрации создания бинарного дерева с рандомизированной вставкой. Для реализации данной цели требуется

1. Изучить теоретический материал по данной структуре данных.
2. Научиться использовать инструменты для визуализации графов.
3. Написать программный код.
4. Тестирование программного кода.

## 1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

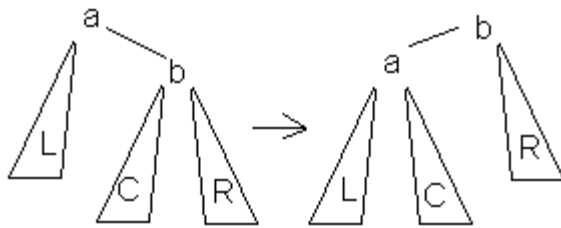
**Двоичное дерево поиска** (англ. binary search tree, BST) — это двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

1. Оба поддерева — левое и правое — являются двоичными деревьями поиска.
2. У всех узлов левого поддерева произвольного узла  $X$  значения ключей данных меньше либо равны, нежели значение ключа данных самого узла  $X$ .
3. У всех узлов правого поддерева произвольного узла  $X$  значения ключей данных больше либо равны, нежели значение ключа данных самого узла  $X$ .

Всегда желательно, чтобы все пути в дереве от корня до листьев имели примерно одинаковую длину, то есть чтобы глубина и левого, и правого поддеревьев была примерно одинакова в любом узле. В противном случае теряется производительность.

В вырожденном случае может оказаться, что всё левое дерево пусто на каждом уровне, есть только правые деревья, и в таком случае дерево вырождается в список (идущий вправо). Поиск (а значит, и удаление и добавление) в таком дереве по скорости равен поиску в списке и намного медленнее поиска в сбалансированном дереве.

Для балансировки дерева применяется операция «поворот дерева». Поворот налево выглядит так:



- было  $\text{Left}(A) = L$ ,  $\text{Right}(A) = B$ ,  $\text{Left}(B) = C$ ,  $\text{Right}(B) = R$
- поворот меняет местами A и B, получая  $\text{Left}(A) = L$ ,  $\text{Right}(A) = C$ ,  $\text{Left}(B) = A$ ,  $\text{Right}(B) = R$
- также меняется в узле  $\text{Parent}(A)$  ссылка, ранее указывавшая на A, после поворота она указывает на B.

Проблемы бинарных деревьев проявляется, когда дерево вырождается и фактически становится линейным списком. Максимальная скорость обращения к элементу в этом случае становится линейна. Одним из простых способов нивелировать шанс вырождения дерева является рандомизированная вставка в корень. Известно, что если заранее перемешать как следует все ключи и потом построить из них дерево (ключи вставляются по стандартной схеме в полученном после перемешивания порядке), то построенное дерево окажется неплохо сбалансированным (его высота будет порядка  $2\log_2 n$  против  $\log_2 n$  для идеально сбалансированного дерева). Заметим, что в этом случае корнем может с одинаковой вероятностью оказаться любой из исходных ключей. Что делать, если мы заранее не знаем, какие у нас будут ключи (например, они вводятся в систему в процессе использования дерева)? Раз любой ключ (в том числе и тот, который мы сейчас должны вставить в дерево) может оказаться корнем с вероятностью  $1/(n+1)$  ( $n$  — размер дерева до вставки), то мы выполняем с указанной вероятностью вставку в корень. Проверка на вставку происходит рекурсивно при переходе к новому узлу.

## 2. ОПИСАНИЕ РАБОТЫ

### 2.1. Описание структур данных и функций

В основе работы лежит шаблонная структура Tree  $\langle T \rangle$ , которая содержит в себе `key` — шаблонный ключ дерева, `size` — количество потомков, `left` и `right` — ссылка на потомка с меньшим и большим ключом соответственно. Создана функция `InsertRoot` для вставки в корень. Она рекурсивно просчитывает шанс вставки в зависимости от значения выполняет базовую вставку или вставку в корень. Для выполнения поворотов созданы функции `leftRotate` и `rightRotate`. Каждый шаг фиксируется функциями `Plot` и зависимой от нее `Wrote`, которые создают файл формата `dot` и в необходимой структуре записывают в него текущее дерево с необходимыми комментариями происходящего. С помощью функции “`system`” происходит компиляция и последующая сборка файлов. В конце работы программы создается файл формата “`pdf`”, в котором пошагово описано создание дерева.

### 2.2. Описание алгоритма

Ключевая идея случайных БДП с рандомизацией состоит в чередовании обычной вставки в дерево поиска и вставки в корень. Чередование происходит случайным (рандомизированным) образом с использованием компьютерного генератора псевдослучайных чисел. Цель такого чередования — сохранить хорошие свойства случайного БДП в среднем и исключить (сделать маловероятным) появление «худшего случая» (поддеревьев большой высоты). Вставка элемента со значением `key` в дерево `tree`. Рассмотрим операцию вставки в корень. Если дерево пусто, создаем новый узел со значением `key`, иначе, если `key(tree) > key` то выполняем вставку в корень в левом поддереве `tree` и выполняем правое вращение, иначе — вставку в корень в правом поддереве и левое вращение. Таким образом узел со значением `key` становится корнем дерева. Опишем теперь рандомизированную вставку значением `key` в дерево `tree`. Пусть в дереве имеется  $n$  узлов. Тогда будем считать, что после



добавления еще одного узла любой узел с равной вероятностью может быть корнем дерева. Тогда, с вероятностью  $1/(n + 1)$  осуществим вставку в корень, иначе рекурсивно используем рандомизированную вставку в левое или правое поддерево в зависимости от значения ключа `key`

### **3. ТЕСТИРОВАНИЕ**

Программа была протестированная на паре характерных примеров, где без использования рандомизированной ставки дерево вырождалось. Это значения от 1 до 15 ,и от 15 до 1 с шагом 1 и -1 соответственно. Главные результаты работы представлены в приложении Б. Для сравнения в приложении В представлено бинарное дерево без использования рандомизированной вставки.

## **ЗАКЛЮЧЕНИЕ**

Была создана необходимая структура данных и алгоритм рандомизированной вставки в корень. По ходу программы каждый шаг демонстрируется и объясняется. Исходный код программы смотрите в приложении А.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Методические указания к лабораторным работам, практическим занятиям и курсовой работе по дисциплине «Алгоритмы и структуры данных»»: учеб.-метод. пособие / сост.: С.А Ивановский , Т.Г. Фомичева , О.М. Шолохова.. СПб. 2017. 88 с
2. Роберт Седжвик - Фундаментальные алгоритмы на C++

## ПРИЛОЖЕНИЕ А

### ПРОГРАММНЫЙ КОД

```
#include "tree.h"
#include "tree.cpp"
#include <iostream>
#include <fstream>
int COUNT(0);
int COUTN_NULL(0);

template <typename T>
Tree<T>* START(nullptr);
template <typename T>
Tree <T>* Create(T& key, Tree<T>left, Tree<T>right)
{
    Tree <T>* New = new Tree<T> (key, left, right);
    return New;
}

template <typename T>
Tree <T>* Create(T& key, Tree<T>left, Tree<T>right)
{
    Tree <T>* New = new Tree<T> (key, left, right);
    return New;
}

template <typename T>
unsigned int Tree_Height(Tree<T>* tr){
    unsigned int l, r;
    if(tr != NULL){
        l = (tr->left != NULL) ? Tree_Height(tr->left) : 0;
        r = (tr->right != NULL) ? Tree_Height(tr->right) : 0;
        return ((l > r) ? l : r) + 1;
    }
    return 0;
}

template <typename T>
void write(Tree<T>* head, std::ofstream& myGraph){
    if(head!=nullptr){
        if(head->left!=nullptr){
            myGraph<<" "<<std::to_string(head->key)<<" "<<"-
>"<<" "<<std::to_string(head->left->key)<<" "<<std::endl;
            write(head->left, myGraph);
        }
        else{
            myGraph<<" "<<std::to_string(head->key)<<" "<<"-
>"<<" "<<"N"<<std::to_string(COUTN_NULL++)<<" "<<std::endl;
        }
    }
}
```

```

        }
        if(head->right!=nullptr) {
            myGraph<<"'"<<std::to_string(head->key)<<"'"<<"-
>"<<"'"<<std::to_string(head->right->key)<<"'"<<std::endl;
            write(head->right,myGraph);
        }
        else{
            myGraph<<"'"<<std::to_string(head->key)<<"'"<<"-
>"<<"'"<<"N"<<std::to_string(COUTN_NULL++)<<"'"<<std::endl;
        }
    }

}

template <typename T>
void Plot(Tree<T>* head,std::string* message){
    ::COUNT++;
    std::string count = std::string(3 -
std::to_string(COUNT).length(), '0') + std::to_string(COUNT);
    std::string name = count +std::string(".gv");
    std::ofstream myGraph(name);
    myGraph<<"digraph G"<<"{\n";
    myGraph<<"graph [ordering=\"out\"]\n";
    write(head,myGraph);
    if(message!=nullptr)
        myGraph<<"label="<<"'"<<*message<<"'"<<std::endl;
    myGraph<<"}\n";
    myGraph.close();
    COUTN_NULL = 0;
    std::string command = std::string("dot -Tpng ");
    command+=name;
    command+=std::string(" -o");
    command+=count;
    command+=std::string(".jpg");
    std::cout<<command;
    system(command.c_str());
}

```

```

template <typename T>
int getsize(Tree<T>* tree)
{
    if( !tree ) return 0;
    return tree->size;
}

template <typename T>
void fixsize(Tree<T>* tree)
{
    tree->size = getsize(tree->left)+getsize(tree->right)+1;
}

```

```

}

template <typename T>
void  rotateright(Tree<T>*  &tree)
{
    if(tree==nullptr) return;
    std::string message = "Поворот вправо: значение
"+std::to_string(tree->left->key);
    message+=std::string(" поднимается вверх\n");
    message+=std::string("Значение ");
    message+=std::to_string(tree->key);
    message+=std::string(" опускается вправо\n");

    Tree<T>* tmp = tree->left;
    tree->left = tmp->right;

    if(tmp->right!=nullptr){
        message+=std::string("значение ");
        message+=std::to_string(tmp->right->key);
        message+=std::string(" становится левым <<ребенком>> значения
");
        message+=std::to_string(tree->key);
    }

    tmp->right =  tree;
    tmp->size = tree->size;
    if(START<int> == tree){START<int> = tmp;}
    tree = tmp;
    fixsize(tree);
    Plot(START<int>,&message);
}

template <typename T>
void  rotateleft(Tree<T>*  &tree)
{
    if(tree==nullptr) return;
    std::string message = "Поворот влево: значение
"+std::to_string(tree->right->key);
    message+=std::string(" поднимается вверх\n");
    message+=std::string("Значение ");
    message+=std::to_string(tree->key);
    message+=std::string(" опускается влево\n");

    Tree <T>* tmp = tree->right;
    tree->right = tmp->left;

    if(tmp->left!=nullptr){
        message+=std::string("значение ");

```

```

        message+=std::to_string(tmp->left->key);
        message+=std::string( " становится правым <<ребенком>>
значения ");
        message+=std::to_string(tree->key);
    }

```

```

        tmp->left = tree;
        tmp->size = tree->size;
        if(START<int> == tree){START<int> = tmp;}
        tree = tmp;
        fixsize(tree);

```

```

        Plot(START<int>,&message);

```

```

    }

```

```

template <typename T>
void PritTree(Tree<T>* tree, int n, std::ofstream &fout)
{

```

```

    if (tree->right != nullptr)
    {
        PritTree(tree->right, n + 1, fout);
    }

```

```

    for (int i = 0; i < n; i++)
    {
        std::cout << "\t";
        fout << "      ";
    }

```

```

    std::cout << tree->key<<'('<<tree->size<<')'<<"\n";
    fout << tree->key<<'('<<tree->size<<')'<<"\n";
    if (tree->left != nullptr)
    {
        PritTree(tree->left, n + 1, fout);
    }

```

```

}

```

```

template <typename T>

```



```

void PrintTree(Tree<T>* tree, int r){
    r++;
    if(tree->right!=nullptr)
        PrintTree(tree->right,r+1);
    for (int i=0;i<(4*r);i++)
        std::cout << " ";
    std::cout<<tree->key<<' ('<<tree->size<<' )'<<std::endl;
    if(tree->left!=nullptr)
        PrintTree(tree->left,++r);
    --r;
}

template <typename T>
Tree <T>* Create(const T& key,Tree<T>* left, Tree<T>* right)
{
    Tree<T>* result = new Tree <T>();
    result->key=key;
    result->left = left;
    result->right = right;
    return result;
}

template <typename T>
void insertroot(Tree<T>* &tree, T& value,std::string* message){
    if(tree==nullptr)
    {
        tree = Create<T>(value,nullptr,nullptr);
        *message+="\nСейчас значение не ходит на своем базовом
месте,но так как произошло выпадения вставки в корень, будут "
"\nсовершаться повороты обратные спуску,пока
новое значение не поднимется до места выпадения вставки в корень";
        Plot(START<int>,message);
        return;
    }
    if(tree->key>value){
        insertroot(tree->left,value, message);
        rotateright(tree);
    }
    else{
        insertroot(tree->right,value,message);
        rotateleft(tree);
    }
}

template <typename T>
void insert(Tree<T>* &tree, T& value,std::ofstream &fout){
    if(tree==nullptr)

```

```

    {
        tree = Create<T>(value,nullptr,nullptr);
        std::string message = "Шанс выпадения в корень не
сработал, произошла базовая вставка значения "+std::to_string(tree-
>key);
        Plot(START<int>,&message);
        return;

    }
    srand( time(0) );
    if( rand()%(tree->size+1)==0 ){
        std::string message ="Проходя вершину  "+ std::to_string(tree-
>key);
        message+=std::string(" выпала вставка в корень значения ");
        message+=std::to_string(value);
        message+=std::string(". Шанс выпадения был 1:");
        message+=std::to_string(tree->size+1);
        std::cout<<" Вставка в корень на уровне "<<tree->size<< "
значения "<< value<<std::endl;
        fout<<"Вставка в корень на уровне "<<tree->size<< " значения
"<< value<<std::endl;
        insertroot(tree,value,&message);
        // START<int> = tree;
        return;
    }
    if(tree->key>value){
        insert(tree->left,value,fout);

    }
    else{
        insert(tree->right,value,fout);
    }
    fixsize(tree);
}

template <typename T>
void simple_insert(Tree<T>* &tree, T& value){
    if(tree==nullptr)
    {
        tree = Create<T>(value,nullptr,nullptr);
        return;
    }
    if(tree->key>value){
        simple_insert(tree->left,value);
    }
    else{
        simple_insert(tree->right,value);
    }
    fixsize(tree);
}

```

```

template <typename T>
int contain(Tree<T>* &tree, T& value){
    if(!tree) return 0;
    int count= 0;
    if(tree->key == value) count++;
    if( value < tree->key )
        count+=contain(tree->left,value);
    else
        count+=contain(tree->right,value);
    return count;
}

void start(int* arr,int n,std::ofstream &fout){
    Tree<int >* New = Create<int >(arr[0],nullptr,nullptr);
    START<int> = New;
    int count = 0;
    for(int i = 1;i<n;i++){
        count = contain<int>(New,arr[i]);
        if(!count){
            insert<int >(New,arr[i],fout);

            std::cout<<"Вставка значения "<<arr[i]<<std::endl;
            fout<<"Вставка значения "<<arr[i]<<std::endl;

            //Plot(New,nullptr); дублирует
            PritTree(New,0,fout);

            std::cout<<std::endl<<"\n\n\n\n\n";
            fout<<std::endl<<"\n\n\n\n\n";
        }
        else{
            std::cout<<"Элемен:"<<arr[i]<<"уже в дереве"<<std::endl;
            fout<<"Элемен:"<<arr[i]<<"уже в дереве"<<std::endl;
        }
    }
    PritTree(New,0,fout);
    std::cout<<std::endl<<"\n\n";
    fout<<std::endl<<"\n\n";
    Tree<int>* New1 = Create<int>(arr[0],nullptr,nullptr);
    for(int i = 1;i<n;i++){
        count = contain<int>(New1,arr[i]);
        if(!count)
            simple_insert<int>(New1,arr[i]);
        else{
            std::cout<<"Элемен:"<<arr[i]<<"уже в дереве"<<std::endl;
            fout<<"Элемен:"<<arr[i]<<"уже в дереве"<<std::endl;
        }
    }
    std::string new_message ="Для сравнения. Максимальная высота
    "+std::to_string(Tree_Height(New));
    Plot(New,&new_message);
}

```

```

new_message = std::string("Максимальная высота при базовом
построении ");
new_message+=std::to_string(Tree_Height(New1));
Plot(New1,&new_message);
PritTree(New1,0,fout);
}
int main(int argc, char const *argv[]){

    std::ifstream fin;
    std::ofstream fout;
    if( argc > 1 ){

        fin.open(argv[1]);
        if( argc > 2 ) {
            fout.open(argv[2], std::ios_base::app);
        }
        else {
            fout.open("result.txt");
        }
        int m;
        fin >> m;
        int *arr = new int[m];
        for (int j = 0; j < m; j++)
        {
            fin >> arr[j];
        }

        start(arr,m,fout);
        system("convert *.jpg my.pdf");
        system("rm *.gv");
        system("rm *.jpg");
        delete[] arr;
        fin.close();
        fout.close();
    }
    else{
        fout.open("result.txt");
        int m;
        std::cin >> m;
        int *arr = new int[m];
        for (int j = 0; j < m; j++)
        {
            std::cin >> arr[j];
        }
        start(arr,m,fout);
        delete[] arr;
    }

    return 0;
}

```

## ПРИЛОЖЕНИЕ Б

### ТЕСТИРОВАНИЕ

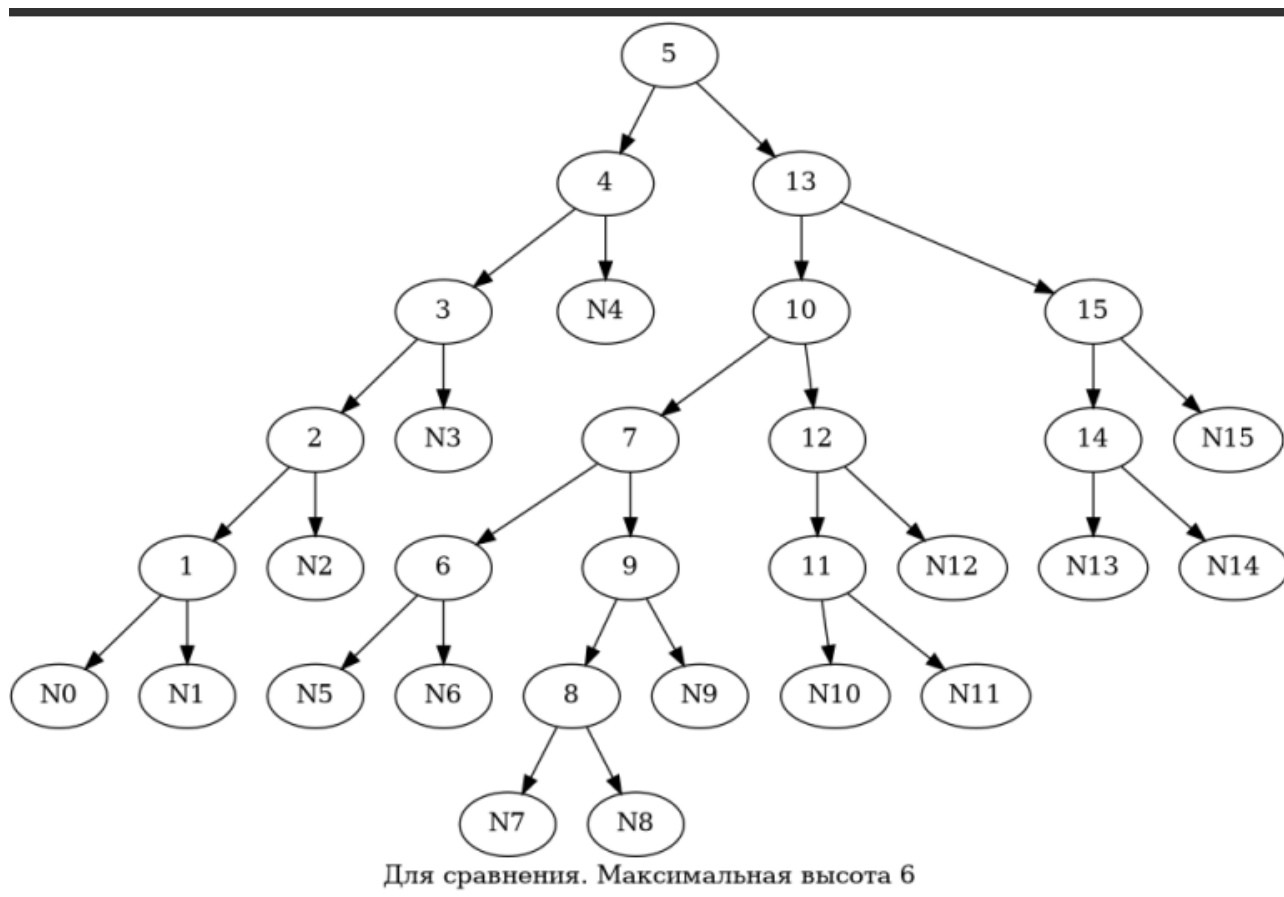


Рисунок 1 – результат работы алгоритма с данными 1– 15

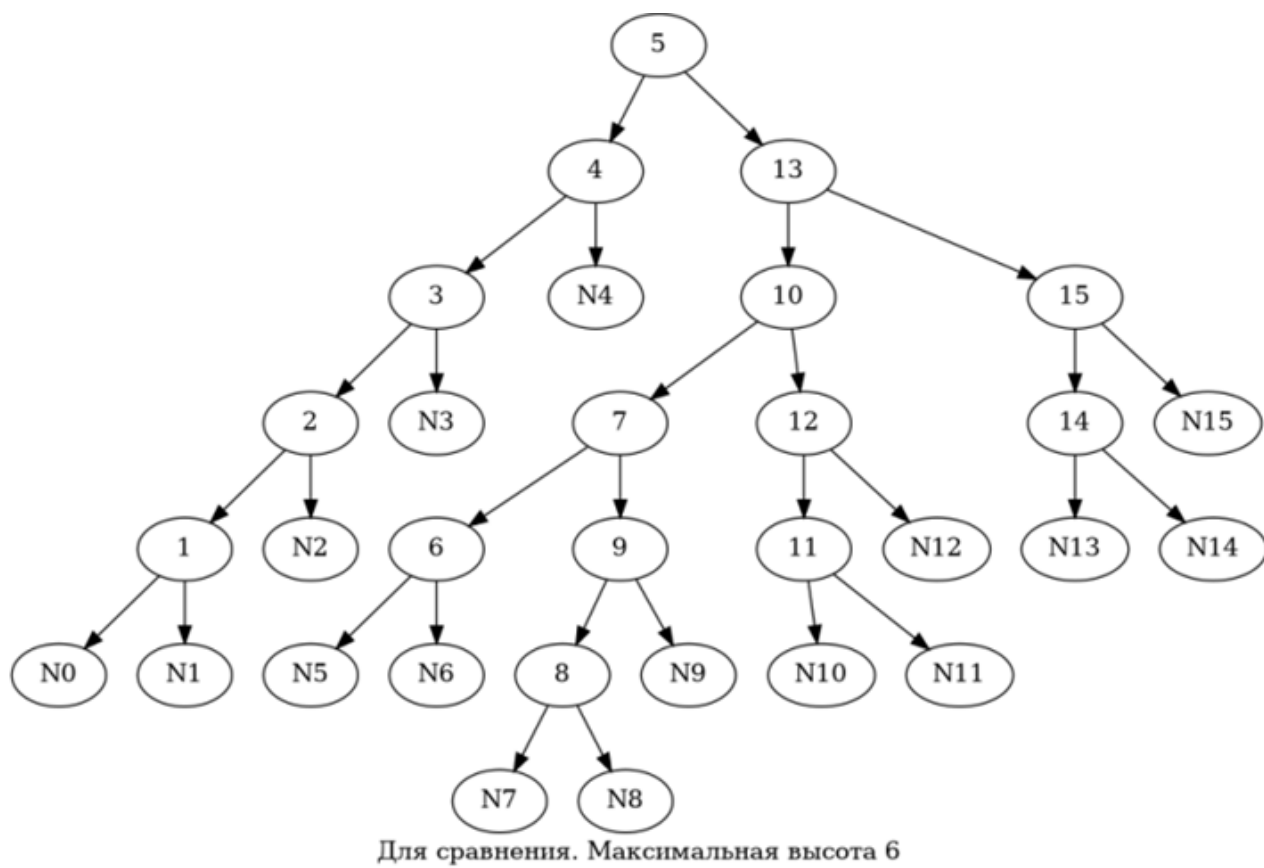


Рисунок 2 – результат работы алгоритма с данными 15 – 1

## ПРИЛОЖЕНИЕ А

### СРАВНЕНИЕ

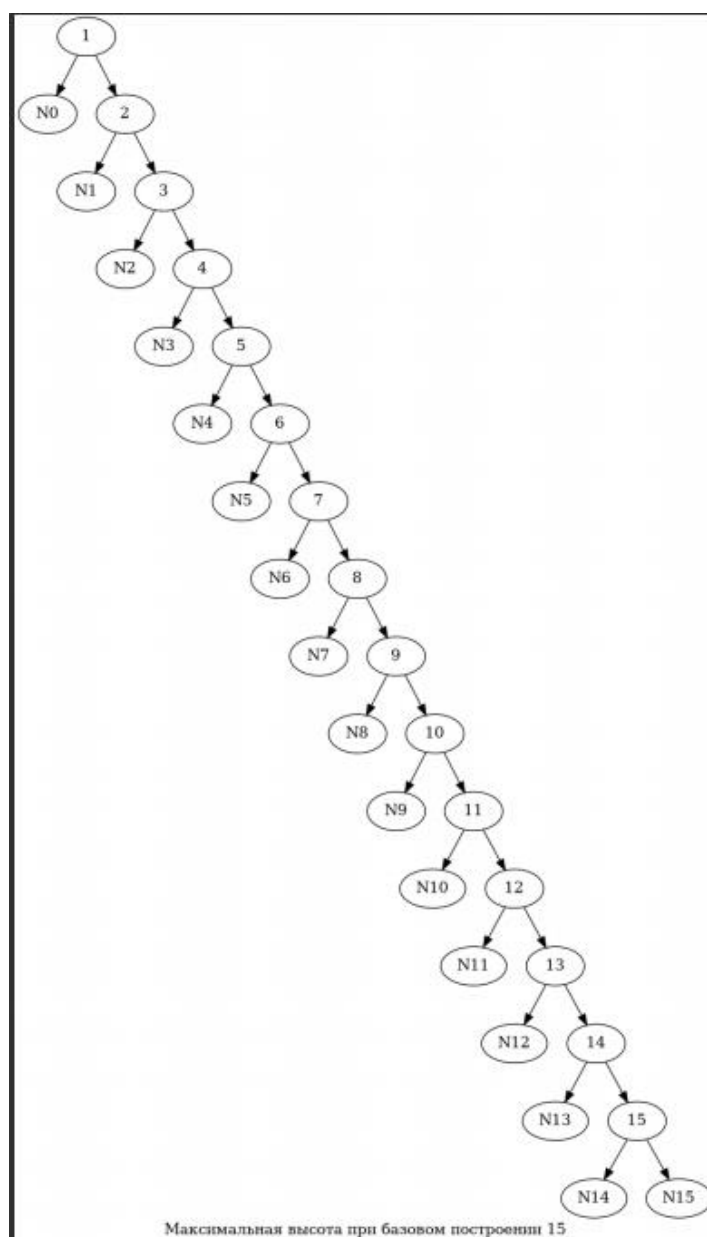


Рисунок 3 – пример вырождения дерева с данными 1 – 15

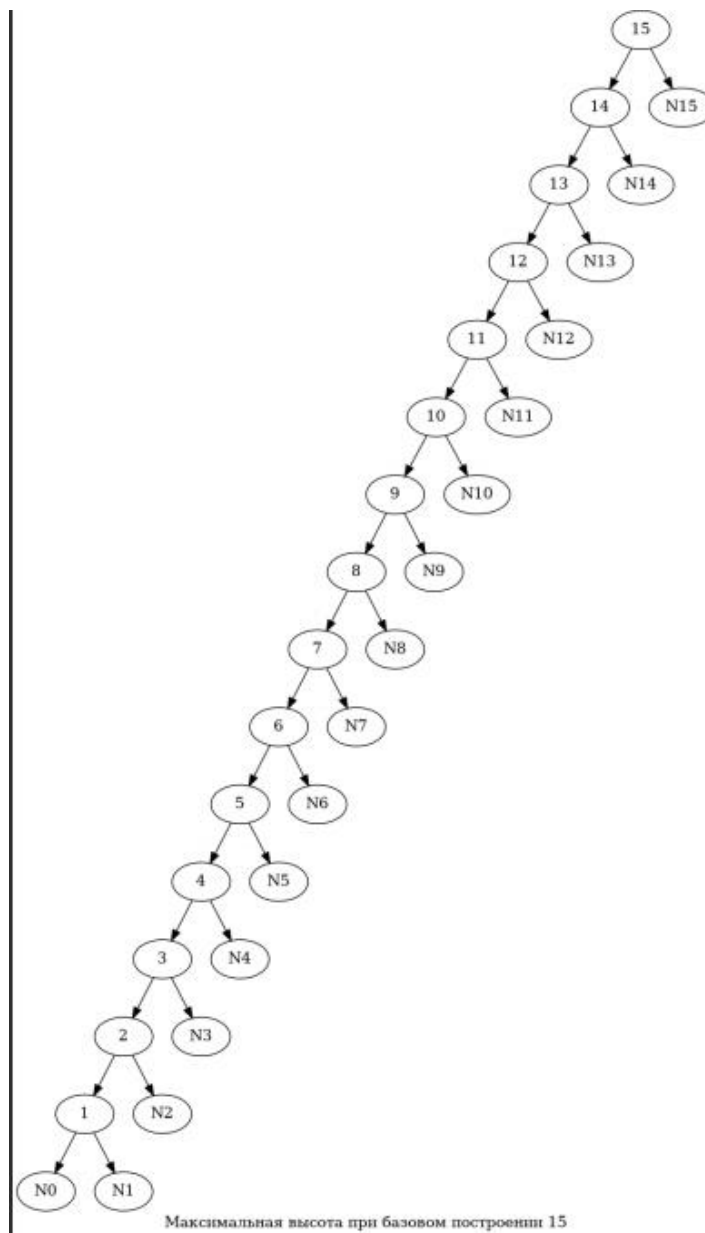


Рисунок 3 – пример вырождения дерева с данными 15 – 1