МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Рекурсия. Графы. Деревья
Отчет по лабораторной работе №3
Вариант № 17

по дисциплине

Алгоритмы и структуры данных

РУКОВОДИТЕЛЬ:

_____ Капранов С.Н.

СТУДЕНТ:

_____ Сапожников В.О.

19-ИВТ-3

Работа защищена «___» _____

С оценкой _____

Нижний Новгород 2020

Текст задания

Дано N-дерево. Найти в дереве самый длинный путь без ветвлений.

Используемый язык программирования: Java

Замечание: интерфейс readAndSave содержит вспомогательные функции ввода данных. Его алгоритмы не относятся к выполнению задания.

Класс SinglyLinkedList взят из 1ой лабораторной работы.

Текст программы

Main.java

```
import java.util.Scanner;       //класс-оболочка потока ввода

/**
 * Класс, содержащий точку входа в программу - метод  main.
 * Язык: java
 *
 * Реализация третьей лабораторной работы по дисциплине:
 * Алгоритмы и структуры данных. Вариант№17.
 *
 * Текст задания: Дано N-дерево. Найти в дереве самый длинный
 *                путь без ветвлений.
 *
 * @release: 14.10.20
 * @last_update: 14.10.20
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 */
public class Main {
    public static void main(String[] args)
    {
        System.out.println("\t\t\u001B[33mНачало +
                           + работы...\u001B[0m");

       //открытие потока ввода
        Scanner scanner = new Scanner(System.in);

        int[][] initMatrix;     //создание ссылки на двумерный
                                //целочисленный массив

       //создание ссылки на список, содержащий строки
```

```
        SinglyLinkedList<String> vertexNameList = new
SinglyLinkedList<>();



   System.out.println("\t   \u001B[33mСоздание n +
                      +   дерева\u001B[0m");

        System.out.println("Для выхода введите '--q'");
        System.out.print("Введите количество вершин в дереве);

       //заполнение списка вершин
        readAndSave.fillVertexList(vertexNameList, scanner);
        System.out.println();

        System.out.print("\t   \u001B[33mЗаполнение матрицы +
                         + смежности\u001B[0m");

       //заполнение матрицы смежности
        initMatrix = readAndSave.fillAdjacencyMatrix
                     (vertexNameList, scanner);

        System.out.println();
        System.out.println("\t\u001B[33mЗаполненая матрица +
                         + смежности \u001B[0m");

        //вывод заполненной матрицы смежности
        readAndSave.printAdjacencyMatrix(vertexNameList,
                                         initMatrix);
        System.out.println();

        //создание объекта - дерево
        Tree<String> tree = new Tree<>(vertexNameList,
                                       initMatrix);

        //вывод дерева в виде списков смежности
        System.out.println("\t\u001B[33mПредставление дерева +
                           + списками смежности\u001B[0m");
        tree.printAdjacencyList();

        //поиск кратчайшего пути без ветвлений
        System.out.println("\t\u001B[33m\n\tПоиск длиннейшего+
                         + пути без ветвлений\u001B[0m");
```

```
            tree.searchLongWay();
        }
    }
```

Tree.java

Класс с реализацией структуры данных – дерево

```
/**
 * Tree - класс, реализующий n-арное дерево.
 *
 * @author  Vladislav Sapozhnikov
 * @param <E> - тип, содержащийся в данной коллекции (узлах
 *              дерева)
 * */
public class Tree<E>
{
    //список 'имен' узлов дерева
    private final SinglyLinkedList<Node<E>> nodeList;

    /**
     * Класс - узел/вершина дерева.
     * Стандартный класс для большинства динамических структур.
     *
     * @param <E> тип, содержащийся в данном узле
     * */
    private static class Node<E>
    {
        //Данное задание не требует наличия ссылки на родителя
        //вершины

        //'имя' данного узла
        private final String                    name;

        //список потомков данного узла
        private final SinglyLinkedList<Node<E>> child;

        /**
         * Конструктор с параметром.
         *
         * @param name - 'имя' для данной вершины.
         * */
        Node(String name)
        {
            this.name = name;
            child     = new SinglyLinkedList<>();
        }
    }
```

```
/**
 * Воспитательный метод.
 * Создает список узлов с именами переданными
 * в списке строк.
 *
 * @param list      - список строк - имена вершин.
 *
 * @return nodeList - список вершин, с именами, полученными
 *                    из списка.
 * */
private SinglyLinkedList<Node<E>> makeNodeList
                          (SinglyLinkedList<String> list)
{
    SinglyLinkedList<Node<E>> nodeList = new
                                        SinglyLinkedList<>();

    for (int i = 0; i < list.getSize(); i++)
    {
        //создание узла с именем из списка
        Node<E> buffNode = new Node<>(list.get(i));

        //добавление узла в список узлов
        nodeList.push_back(buffNode);
    }

    return nodeList;
}

/**
 * Вспомогательный метод.
 * Сортировка массива пузырьком.
 * Помогает отсортировать самый длинный путь.
 * Используется в вспомогательном методе pathBuilder.
 *
 * @param arr - целочисленный массив, который необходимо
 *              отсортировать.
 * */
private void sortArr(int[] arr)
{
    int buff;  //вспомогательная переменная для обмена

    //Алгоритм сортировки пузырьком.
    for (int i = 0; i < arr.length-1; i++)
    {
        if (arr[i] > arr[i+1])        //Если текущий элемент
```

```
                                        //больше следящего, то
        {                               //сохраняем в
                                        //переменную значение
            buff       = arr[i];        //текущего элемента.
            arr[i]     = arr[i + 1];    //В текущий элемент
                                        //записываем следующий
            arr[i + 1] = buff;       //значение следующего
                                     //меняем на значение из
                                     //переменной
        }                            //
    }                                //Максимальный элемент
                                    //становится крайним
                                    //правым элементом массива
}                                  //элементом массива

/**
 * Вспомогательный метод.
 * Создает одномерный массив, содержащий длины списков из
 * переданного двумерного списка.
 *
 * @param list     - двумерный список, длины списков
 *                   которого требуется получить.
 *
 * @return sizeArr - получившийся массив размерностей
 *                  списков.
 * */
private int[] getSizeArr
          (SinglyLinkedList<SinglyLinkedList<Node<E>>> list)
{
    int size      = list.getSize();      //создание массива
                                         //размерностью
    int[] sizeArr = new int[size];       //равной кол-ву
                                         //списков в
                                         //двумерном списке

    for (int i = 0; i < size; i++)
    {
        sizeArr[i] = list.get(i).getSize();  //запись в
                                             //элементы
                                             //массива длин
                                             //списков
    }

    return sizeArr;
}



/**
 * Вспомогательный метод.
 * По переданному отсортированному массиву размерностей
```

```
     * списка находим самый длинный путь.
     *
     * @param arr  - отсортированный массив размерностей списка.
     *
     * @param list - двумерный список путей.
     * */
    private void foundLongWay(int[] arr,
                SinglyLinkedList<SinglyLinkedList<Node<E>>> list)
    {
        System.out.println();
        if (arr[arr.length-1] == 1)
        {
            System.out.println("Все пути в дереве содержат +
                                + ветвления.");
            return;
        }
        //т.к. самый большой элемент массива
        //является последним, то сравниваем
        //длину списков списка с последним элементом массива
        for (int i = 0; i < arr.length; i++)
        {
            if (list.get(i).getSize() == arr[arr.length-1])
            {
                //вывод самого длинного пути в консоль.
                System.out.print("Длинейший путь: ");
                printNodeList(list.get(i));
            }
        }
    }


    /**
     * Вспомогательный метод.
     * Заполняет список вершинами пути, от которых не имеет
     * ветвлений.
     *
     * @param nodeList - список вершин без ветвлений.
     *
     * @param wayList  - список для заполнения путей.
     * */
    private void pathBuilder (SinglyLinkedList<Node<E>>
nodeList, SinglyLinkedList<SinglyLinkedList<Node<E>>> wayList)
    {
        //на каждую вершину списка вершин
        //начинаем построение собственного пути
        for (int i = 0; i < nodeList.getSize(); i++) {
            wayList.push_back(new SinglyLinkedList<>());

            //при помощи вспомогательной переменной – узла
            //производим обход по потомкам
            Node<E> buffNode = nodeList.get(i);
```

```
                wayList.get(i).push_back(buffNode);




                //и записываем путь в список вплоть до
                //конечной вершины данного пути
                while (buffNode.child.getSize() != 0)
                {
                    wayList.get(i).push_back(buffNode.child.get(0));
                    buffNode = buffNode.child.get(0);
                }
            }
        }

        /**
         * Вспомогательный метод.
         * Выводит элементы одномерного списка.
         * */
        private void printNodeList(SinglyLinkedList<Node<E>> list)
        {
            for (int i = 0; i < list.getSize(); i++)
            {
                System.out.print(list.get(i).name + " ");
            }
            System.out.println();
        }

        /**
         * Вспомогательный метод.
         * Заполняет список вершинами, от которых не идет ветвлений.
         *
         * @param list  - список для заполнения.
         * */
        private void
    makeVertexWOBranchingList(SinglyLinkedList<Node<E>> list)
        {
            for (int i = 0; i < nodeList.getSize(); i++)
            {
                //если у вершины нет потомков или только 1 потомок,
                //то заносим её в список
                if (nodeList.get(i).child.getSize() == 0 ||
                    nodeList.get(i).child.getSize() == 1)
                {
                    list.push_back(nodeList.get(i));
                }
            }
        }
```

```
/**
 * Конструктор дерева с параметрами.
 *  @param initList   - список с именами вершин, которые
 *                      будут хранится в дереве.
 *
 * @param initMatrix – введённая матрица смежности данного
 *                     дерева. Матрица не сохраняется, а
 *                     используется только в момент создания
 *                     дерева.
 **/
Tree(SinglyLinkedList<String> initList, int[][] initMatrix)
{
    //заполнение списка вершин данного дерева
    nodeList = makeNodeList(initList);
    int size     = initMatrix.length;

    //заполнение списков смежности вершин
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (initMatrix[i][j] == 1)
            {
                 nodeList.get(i).child.push_back
                 (nodeList.get(j));
            }
        }
    }
}

/**
 * Метод, находящий самый длинный путь без ветвлений.
 * */
public void searchLongWay()
{
    //создание ссылки на двумерный список, в котором будут
    //хранится все пути без ветвлений
    SinglyLinkedList<SinglyLinkedList<Node<E>>> wayList =
                                new SinglyLinkedList<>();

    //создание ссылки на список вершин, в котором будут
    //хранится все вершины, пути от которых не имеют
    //ветвлений.

    SinglyLinkedList<Node<E>> vertexWithoutBranching = new
                                     SinglyLinkedList<>();

    //заполнение списка вершинами, пути от которых не
    //имеет ветвлений
    makeVertexWOBranchingList(vertexWithoutBranching);
```

```
        //заполнение списка всеми возможными путями без
        //ветвлений
        pathBuilder(vertexWithoutBranching, wayList);

        //создание массива с размерностями путей
        int[] sizeArr = getSizeArr(wayList);

        //сортировка массива
        sortArr(sizeArr);

        //поиск самого длинного пути без ветвлений на основе
        //списка путей и размерностей.
        foundLongWay(sizeArr, wayList);
    }

    /**
     * Вывод списков смежности для каждой вершины.
     * Аналог матрицы смежностей.
     * */
    public void printAdjacencyList()
    {
        for (int i = 0; i < nodeList.getSize(); i++)
        {
            //Вывод имени вершины список смежности которой
            //выводится
            System.out.print("\u001B[34m" + nodeList.get(i).name
                             + "\u001B[0m: ");
            if (nodeList.get(i).child.getSize() != 0)
            {
                for(int j=0; j<nodeList.get(i).child.getSize();
                                                        j++)
                {
                    //вывод списка смежности вершины
                    System.out.print(nodeList.get(i).child.get(j)
                    .name + " ");
                }
            }
            //если вершина не имеет дочерних вершин, то выводим
            //'прочерк'
            else
            {
                System.out.print("-");
            }
            System.out.println();
        }
    }
}
```

SinglyLinkedList.java

```
/**
 * Урезанная реализация односвязного списка (Данный класс взят
 * из Лабораторной работы №1).
 *
 * @author  Vladislav Sapozhnikov
 * @param <E> тип, содержащийся в данной коллекции
 */
public class SinglyLinkedList<E>
{
    /**
     * Класс - узел/ячейка списка.
     * Стандартный класс для большинства динамических
     * структур.
     *
     * @param <E> тип, содержащийся в данном узле
     * */
    private static class Node<E>
    {
        E value;                //поле, хранящее данные
        Node<E> next;           //указатель на след. узел

        Node(Node<E> next, E value)
        {
            this.value = value;
            this.next  = next;
        }
    }

    private Node<E> head;       //указатель на начало списка
    private int size;           //поле для хранения
                                //размерности списка
                                //Является необходимым для
                                //работы данной программы.
```

```
/**
 * private метод для получения узла по заданному списку.
 *
 * @param index                          - индекс, по которому
 *                                         необходимо
 *                                         получить узел.
 *
 * @throws IndexOutOfBoundsException -исключение,
 *                                    сообщающее о выход е
 *                                    за пределы списка.
 *                                    Данное исключение не
 *                                    обрабатывается
 *                                    внутри функции, а
 *                                    делегируется
 *                                    методу, из которого
 *                                    вызвана данная
 *                                    функция.
 *
 * @return Node<E>                      - возвращает узел
 *                                         списка.
 * */
private Node<E> getByIndex(int index)
{
    if (index < 0)
    {
        throw new IndexOutOfBoundsException("Index must be
                                    + a positive number.");
    }

    Node<E> buffNode = head;
    for(int i = 0; i < index; i++)
    {
        if (buffNode.next == null)
        {
            throw new IndexOutOfBoundsException("Index ["
                            + index + "] out of list.");
        }
        buffNode = buffNode.next;
    }
    return buffNode;
}
```

```
/**
 * Конструктор по умолчанию.
 **/
public SinglyLinkedList()
{
    head = null;
    size = 0;
}

/**
 * Метод, добавляющий новый узел в конец списка.
 *
 * @param value - данные, которые следует добавить в новый
 *                узел списка.
 * */
public void push_back(E value)
{
    if (head == null)
    {
        head = new Node<>(null, value);
    }
    else
    {
        Node<E> buffNode = head;
        while(buffNode.next != null)
        {
            buffNode = buffNode.next;
        }
        buffNode.next = new Node<>(null, value);
    }

    size++;
}
```

```
    /**
     * Перегруженный метод от класса Object.
     * Отвечает за вывод объекта в консоль.
     *
     * @return String - строка, содержащая строковое
     *                  представление объекта.
     * */
    @Override
    public String toString()
    {
        StringBuilder result = new StringBuilder("[");

        Node<E> buffNode = head;

        while(buffNode.next != null)
        {
            result.append(buffNode.value).append(", ");
            buffNode = buffNode.next;
        }
        result.append(buffNode.value).append("]");

        return new String(result);
    }

    /**
     * Метод, возвращающий значение из узла по переданному
     * индексу.
     * @param index - индекс узла, содержимое которого
     *                необходимо получить.
     * */
    public E get(int index)
    {
        Node <E> buffNode = getByIndex(index);
        return buffNode.value;
    }
    /**
     * Метод, возвращающий размерность данного списка.
     * @return size - целочисленное значение равное
     *                размерности текущего списка.
     * */
    public int getSize()
    { return size;}
}
```

```
import java.util.Scanner;

/**
 * Интерфейс, содержащий вспомогательные функции ввода.
 * */
public interface readAndSave
{
    /**
     * Вспомогательный метод, проверяющий желает ли
     * пользователь досрочно завершить выполнение программы.
     *
     * @param str - строка введённая пользователем,
     *              которая передается для проверки.
     * */
    private static void checkForExit(String str)
    {
        if (str.equals("--q"))    //если пользователь ввел
        {                         //'--q' - досрочный выход

            System.err.println("Досрочный выход из +
                               + программы...");
            System.exit(-1);
        }
    }

    /**
     * Вспомогательный метод, проверяющий корректность ввода
     * кол-во вершин в дереве.
     *
     * @param scanner - класс - оболочка потока ввода.
     *
     * @return Integer.parseInt(str) - введенное пользователь
     *                                 значение, переведенное
     *                                 в целочисленный формат.
     * */
    private static Integer inputVertexQuantity(Scanner
scanner)
    {
        String str;
        int value;
```

```
    while (true)          //цикл ввода и проверки значений
    {
        str = scanner.nextLine();

        checkForExit(str);

        if (str.matches("[0-9]+")){//проверка содержимого
                                   //введенной строки на
                                   //содержание только
                                   //чисел


            value = Integer.parseInt(str);

             if (value == 0)
            {
                System.err.println("Кол-во вершин не может
                                    + быть равно нулю.");
            }
            else
            {
                return value;
            }
        }
        else
        {
            System.err.println("Ошибка ввода. Повторите +
                                + попытку.");
        }
    }
}

/**
 * Вспомогательный метод, проверяющий корректность ввода
 * имени вершин дерева.
 *
 * @param scanner - класс - оболочка потока ввода.
 *
 * @return str    - строка - имя вершины.
 * */
private static String inputVertexName(Scanner scanner)
{
    String str;
```

```
    while (true) {        //цикл ввода и проверки значений
        str = scanner.nextLine();

        checkForExit(str);

        //проверка на содержание строки только чисел и
        //букв латинского алфавита
        if (str.matches("\\w+|[0-9, \\w]"))
        {
            return str;
        }
        else
        {
            System.err.println("Ошибка ввода. Повторите +
                                + попытку.");
        }
    }
}

/**
 * Вспомогательный метод, проверяющий корректность ввода
 * пути между вершинами
 *
 * @param scanner - класс - оболочка потока ввода.
 *
 * @param count                  - счетчик, указывающий
 *                                 сдвиг по строке
 *                                 матрицы смежности
 *
 * @return Integer.parseInt(str) - введенное пользователь
 *                                 значение переведенное в
 *                                 целочисленный формат.
 * */
private static Integer hasWay(Scanner scanner, int[]
                                                  count) {
    String str;

    while (true)          //цикл ввода и проверки значений
    {
        str = scanner.nextLine();
        checkForExit(str);

        if (str.matches("[0-1]")) //проверка на ввод 0 | 1
```

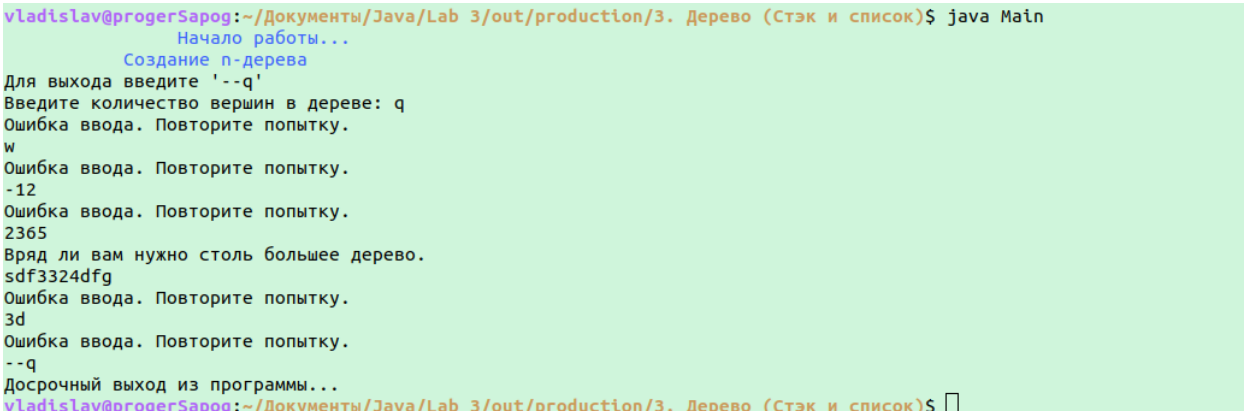
```
        {
            if (str.equals("1"))
            {
                count[0]++;
            }
            return Integer.parseInt(str);
        }
        else
        {
            System.err.println("Ошибка ввода. Повторите +
                                + попытку.");
        }
    }
}

/**
 * Метод, выводящий матрицу смежности.
 *
 * @param list   - списко с именем вершин дерева.
 *
 * @param matrix - матрица смежности дерева.
 * */
static void printAdjacencyMatrix(SinglyLinkedList<String>
                                 list, int[][] matrix)
{
    int size = list.getSize();

    System.out.print("\t");
    for (int i = 0 ; i < size; i++)
    {
         System.out.print(list.get(i) + "\t");
    }
    System.out.println();
    for (int i = 0; i < size; i++)
    {
        System.out.print(list.get(i) + "\t");
        for (int j = 0; j < size; j++)
        {
            System.out.print(matrix[i][j] + "\t");
        }
        System.out.println();
    }
}
```

```
    /**
     * Метод для заполнения списка имен вершин дерева.
     *
     * @param scanner - класс - оболочка потока ввода.
     *
     * @param list    - список для заполнения.
     *  */
    static void fillVertexList(SinglyLinkedList<String> list,
                               Scanner scanner)
    {
        int size = inputVertexQuantity(scanner);

        System.out.println();
        System.out.println("Имена вершин могут содержать +
                            +только буквы латинского алфавита+
                            +и целые числа.");
        System.out.println("Заполняйте имена вершин аккуратно+
                           + в противном случае необходим +
                           + перезапуск программы.");
        System.out.println();

        for (int i = 1; i <= size; i++)
        {
            System.out.print("Введите имя вершины " + i+": ");
            list.push_back(inputVertexName(scanner));
        }
    }

    /**
     * Метод для заполнения матрицы смежности.
     *
     * @param scanner - класс - оболочка потока ввода.
     *
     * @param list    - список с именами вершин.
     *
     * @return matrix - заполненая матрицы смежности.
     * */
    static int[][]
fillAdjacencyMatrix(SinglyLinkedList<String> list,
                    Scanner scanner)
    {
```

```
        int size = list.getSize();
        int[][] matrix = new int[size][size];

        System.out.println();
        System.out.println("'1' - есть путь между вершинами");
        System.out.println("'0' - пути между вершинами нет");

        int[] count = {1};
        for (int i = 0; (i < size-1) && (count[0] != size);
i++)
        {
            System.out.println();
            System.out.println("Заполнение пути для вершины "
                                + list.get(i));
           for (int j = count[0]; j < size; j++)
           {
               System.out.print("Путь " + list.get(i) + " - "
+ list.get(j) + ": ");
               matrix[i][j] = hasWay(scanner, count);
           }
        }
        return matrix;
    }
}
```

Скриншоты

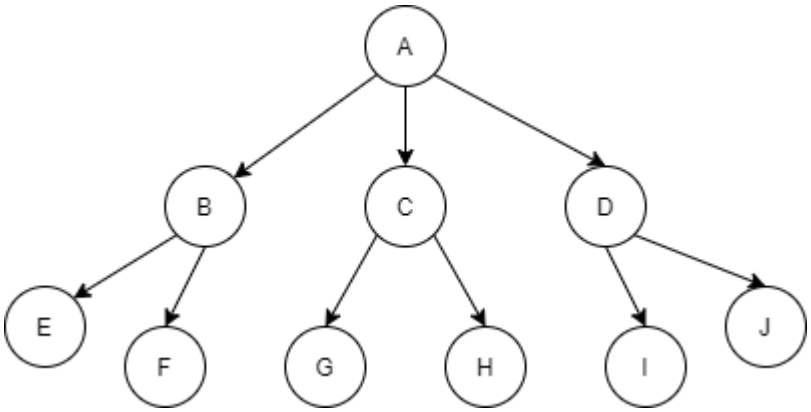
1. Неверно веденные данные.



Замечание: в любой момент ввода пользователь может досрочно завершить работу программы введя последовательность ‘--q’

2. Введено дерево, все пути в котором содержат ветвления.

Дерево



Матрица смежности дерева

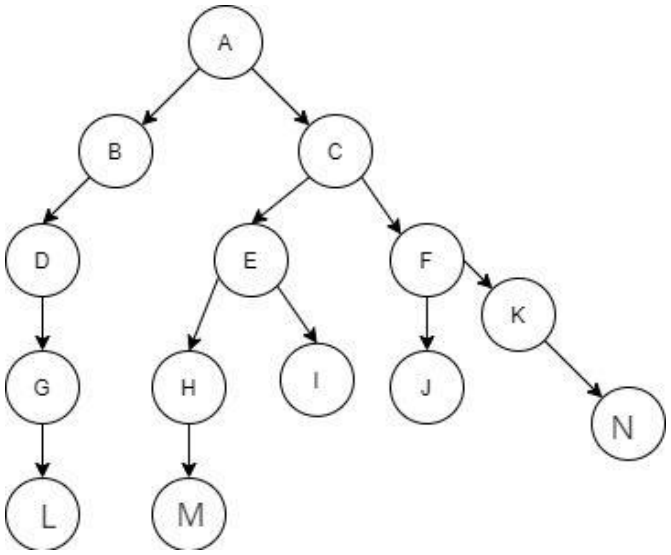
	A	B	C	D	E	F	G	H	I	J
A	0	1	1	1	0	0	0	0	0	0
B	0	0	0	0	1	1	0	0	0	0
C	0	0	0	0	0	0	1	1	0	0
D	0	0	0	0	0	0	0	0	1	1
E	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0

Работа программы

```
vladislav@progerSapog:~/Документы/Java/Lab 3/out/production/Lab 3$ java Main
                Начало работы...
            Создание n-дерева
Для выхода введите '--q'
Введите количество вершин в дереве: 10

Имена вершин могут содержать только буквы латинского алфавита и положительные целые числа.
Заполняйте имена вершин аккуратно, в противном случае необходим перезапуск программы.

Введите имя вершины 1: A
Введите имя вершины 2: B
Введите имя вершины 3: C
Введите имя вершины 4: D
Введите имя вершины 5: E
Введите имя вершины 6: F
Введите имя вершины 7: G
Введите имя вершины 8: H
Введите имя вершины 9: I
Введите имя вершины 10: J

            Заполнение матрицы смежности
'1' - есть путь между вершинами
'0' - пути между вершинами нет

Заполнение пути для вершины A
Путь A - B: 1
Путь A - C: 1
Путь A - D: 1
Путь A - E: 0
Путь A - F: 0
Путь A - G: 0
Путь A - H: 0
Путь A - I: 0
Путь A - J: 0

Заполнение пути для вершины B
Путь B - E: 1
Путь B - F: 1
Путь B - G: 0
Путь B - H: 0
Путь B - I: 0
Путь B - J: 0

Заполнение пути для вершины C
Путь C - G: 1
Путь C - H: 1
Путь C - I: 0
Путь C - J: 0

Заполнение пути для вершины D
Путь D - I: 1
Путь D - J: 1

          Заполненая матрица смежности
          A       B       C       D       E       F       G       H       I       J
A         0       1       1       1       0       0       0       0       0       0
B         0       0       0       0       1       1       0       0       0       0
C         0       0       0       0       0       0       1       1       0       0
D         0       0       0       0       0       0       0       0       1       1
E         0       0       0       0       0       0       0       0       0       0
F         0       0       0       0       0       0       0       0       0       0
G         0       0       0       0       0       0       0       0       0       0
H         0       0       0       0       0       0       0       0       0       0
I         0       0       0       0       0       0       0       0       0       0
J         0       0       0       0       0       0       0       0       0       0

          Представление дерева списками смежности
A: B C D
B: E F
C: G H
D: I J
E: -
F: -
G: -
H: -
I: -
J: -

          Поиск длиннейшего пути без ветвлений
Вычисление вершин, участвующий в путях без ветвлений
Построение всех возможных путей без ветвлений
Вычисление длин путей и их сравнение

Все пути в дереве содержат ветвления.
vladislav@progerSapog:~/Документы/Java/Lab 3/out/production/Lab 3$
```

3. Произвольное дерево.

Дерево



Матрица смежности дерева

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	0	1	1	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	1	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	1	1	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	1	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	1	1	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	1	1	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0	1	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	1
L	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0

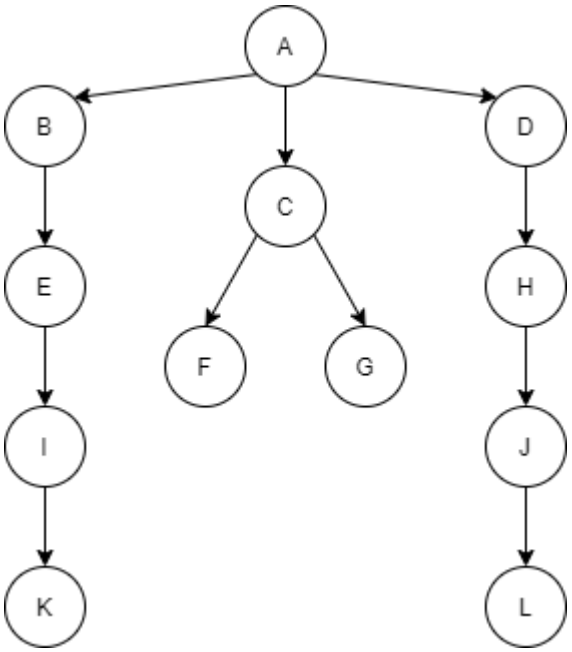
Работа программы

```
vladislav@progerSapog:~/Документы/Java/Lab 3/out/production/Lab 3$ java Main
                Начало работы...
           Создание n-дерева
Для выхода введите '--q'
Введите количество вершин в дереве: 14

Имена вершин могут содержать только буквы латинского алфавита и положительные целые числа.
Заполняйте имена вершин аккуратно, в противном случае необходим перезапуск программы.

Введите имя вершины 1: A
Введите имя вершины 2: B
Введите имя вершины 3: C
Введите имя вершины 4: D
Введите имя вершины 5: E
Введите имя вершины 6: F
Введите имя вершины 7: G
Введите имя вершины 8: H
Введите имя вершины 9: I
Введите имя вершины 10: J
Введите имя вершины 11: K
Введите имя вершины 12: L
Введите имя вершины 13: M
Введите имя вершины 14: N

           Заполнение матрицы смежности
'1' - есть путь между вершинами
'0' - пути между вершинами нет

Заполнение пути для вершины A
Путь A - B: 1
Путь A - C: 1
Путь A - D: 0
Путь A - E: 0
Путь A - F: 0
Путь A - G: 0
Путь A - H: 0
Путь A - I: 0
Путь A - J: 0
Путь A - K: 0
Путь A - L: 0
Путь A - M: 0
Путь A - N: 0

Заполнение пути для вершины B
Путь B - D: 1
Путь B - E: 0
Путь B - F: 0
Путь B - G: 0
Путь B - H: 0
Путь B - I: 0
Путь B - J: 0
Путь B - K: 0
Путь B - L: 0
Путь B - M: 0
Путь B - N: 0

Заполнение пути для вершины C
Путь C - E: 1
Путь C - F: 1
Путь C - G: 0
Путь C - H: 0
Путь C - I: 0
Путь C - J: 0
Путь C - K: 0
Путь C - L: 0
Путь C - M: 0
Путь C - N: 0

Заполнение пути для вершины D
Путь D - G: 1
Путь D - H: 0
Путь D - I: 0
```

```
Путь D - I: 0
Путь D - J: 0
Путь D - K: 0
Путь D - L: 0
Путь D - M: 0
Путь D - N: 0

Заполнение пути для вершины E
Путь E - H: 1
Путь E - I: 1
Путь E - J: 0
Путь E - K: 0
Путь E - L: 0
Путь E - M: 0
Путь E - N: 0

Заполнение пути для вершины F
Путь F - J: 1
Путь F - K: 1
Путь F - L: 0
Путь F - M: 0
Путь F - N: 0

Заполнение пути для вершины G
Путь G - L: 1
Путь G - M: 0
Путь G - N: 0

Заполнение пути для вершины H
Путь H - M: 1
Путь H - N: 0

Заполнение пути для вершины I
Путь I - N: 0

Заполнение пути для вершины J
Путь J - N: 0

Заполнение пути для вершины K
Путь K - N: 1

        Заполненая матрица смежности
        A       B       C       D       E       F       G       H       I       J       K       L       M       N
A       0       1       1       0       0       0       0       0       0       0       0       0       0       0
B       0       0       0       1       0       0       0       0       0       0       0       0       0       0
C       0       0       0       0       1       1       0       0       0       0       0       0       0       0
D       0       0       0       0       0       0       1       0       0       0       0       0       0       0
E       0       0       0       0       0       0       0       1       1       0       0       0       0       0
F       0       0       0       0       0       0       0       0       0       1       1       0       0       0
G       0       0       0       0       0       0       0       0       0       0       0       1       0       0
H       0       0       0       0       0       0       0       0       0       0       0       0       1       0
I       0       0       0       0       0       0       0       0       0       0       0       0       0       0
J       0       0       0       0       0       0       0       0       0       0       0       0       0       0
K       0       0       0       0       0       0       0       0       0       0       0       0       0       1
L       0       0       0       0       0       0       0       0       0       0       0       0       0       0
M       0       0       0       0       0       0       0       0       0       0       0       0       0       0
N       0       0       0       0       0       0       0       0       0       0       0       0       0       0

        Представление дерева списками смежности
A: B C
B: D
C: E F
D: G
E: H I
F: J K
G: L
H: M
I: -
J: -
K: N

        Представление дерева списками смежности
A: B C
B: D
C: E F
D: G
E: H I
F: J K
G: L
H: M
I: -
J: -
K: N
L: -
M: -
N: -

        Поиск длиннейшего пути без ветвлений
Вычисление вершин, участвующий в путях без ветвлений
Построение всех возможных путей без ветвлений
Вычисление длин путей и их сравнение

Длинейший путь: B D G L
vladislav@progerSapog:~/Документы/Java/Lab 3/out/production/Lab 3$
```

4. Дерево с несколькими одинаково длинными путями без ветвлений.

Дерево



Матрица смежности дерева

	A	B	C	D	E	F	G	H	I	J	K	L
A	0	1	1	1	0	0	0	0	0	0	0	0
B	0	0	0	0	1	0	0	0	0	0	0	0
C	0	0	0	0	0	1	1	0	0	0	0	0
D	0	0	0	0	0	0	0	1	0	0	0	0
E	0	0	0	0	0	0	0	0	1			
F	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	1	0	0
I	0	0	0	0	0	0	0	0	0	0	1	0
J	0	0	0	0	0	0	0	0	0	0	0	1
K	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0

Работа программы

```
vladislav@progerSapog:~/Документы/Java/Lab 3/out/production/Lab 3$ java Main
                Начало работы...
            Создание n-дерева
Для выхода введите '--q'
Введите количество вершин в дереве: 12

Имена вершин могут содержать только буквы латинского алфавита и положительные целые числа.
Заполняйте имена вершин аккуратно, в противном случае необходим перезапуск программы.

Введите имя вершины 1: A
Введите имя вершины 2: B
Введите имя вершины 3: C
Введите имя вершины 4: D
Введите имя вершины 5: E
Введите имя вершины 6: F
Введите имя вершины 7: G
Введите имя вершины 8: H
Введите имя вершины 9: I
Введите имя вершины 10: J
Введите имя вершины 11: K
Введите имя вершины 12: L

            Заполнение матрицы смежности
'1' - есть путь между вершинами
'0' - пути между вершинами нет

Заполнение пути для вершины A
Путь A - B: 1
Путь A - C: 1
Путь A - D: 1
Путь A - E: 0
Путь A - F: 0
Путь A - G: 0
Путь A - H: 0
Путь A - I: 0
Путь A - J: 0
Путь A - K: 0
Путь A - L: 0

Заполнение пути для вершины B
Путь B - E: 1
Путь B - F: 0
Путь B - G: 0
Путь B - H: 0
Путь B - I: 0
Путь B - J: 0
Путь B - K: 0
Путь B - L: 0

Заполнение пути для вершины C
Путь C - F: 1
Путь C - G: 1
Путь C - H: 0
Путь C - I: 0
Путь C - J: 0
Путь C - K: 0
Путь C - L: 0

Заполнение пути для вершины D
Путь D - H: 1
Путь D - I: 0
Путь D - J: 0
Путь D - K: 0
Путь D - L: 0

Заполнение пути для вершины E
Путь E - I: 1
Путь E - J: 0
Путь E - K: 0
Путь E - L: 0

Заполнение пути для вершины F
```

```
Путь F - J: 0
Путь F - K: 0
Путь F - L: 0

Заполнение пути для вершины G
Путь G - J: 0
Путь G - K: 0
Путь G - L: 0

Заполнение пути для вершины H
Путь H - J: 1
Путь H - K: 0
Путь H - L: 0

Заполнение пути для вершины I
Путь I - K: 1
Путь I - L: 0

Заполнение пути для вершины J
Путь J - L: 1

        Заполненая матрица смежности
        A       B       C       D       E       F       G       H       I       J       K       L
A       0       1       1       1       0       0       0       0       0       0       0       0
B       0       0       0       0       1       0       0       0       0       0       0       0
C       0       0       0       0       0       1       1       0       0       0       0       0
D       0       0       0       0       0       0       0       1       0       0       0       0
E       0       0       0       0       0       0       0       0       1       0       0       0
F       0       0       0       0       0       0       0       0       0       0       0       0
G       0       0       0       0       0       0       0       0       0       0       0       0
H       0       0       0       0       0       0       0       0       0       1       0       0
I       0       0       0       0       0       0       0       0       0       0       1       0
J       0       0       0       0       0       0       0       0       0       0       0       1
K       0       0       0       0       0       0       0       0       0       0       0       0
L       0       0       0       0       0       0       0       0       0       0       0       0
```

```
        Представление дерева списками смежности
A: B C D
B: E
C: F G
D: H
E: I
F: -
G: -
H: J
I: K
J: L
K: -
L: -

        Поиск длиннейшего пути без ветвлений
Вычисление вершин, участвующий в путях без ветвлений
Построение всех возможных путей без ветвлений
Вычисление длин путей и их сравнение

Длинейший путь: B E I K
Длинейший путь: D H J L
vladislav@progerSapog:~/Документы/Java/Lab 3/out/production/Lab 3$
```