ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

ОТЧЕТ

Лабораторная работа №4

по дисциплине “Моделирование”

Выполнили
студенты Брескун И. М., Селиванов В. В.
Ф.И.О.

Группы ИВ-022

Работу принял _____ ст. преп. кафедры ПМиК Бублей Д. А.
подпись

Защищена _____ Оценка _____

Новосибирск – 2024

СОДЕРЖАНИЕ

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая генерирует случайное дерево со случайными значениями максимальной ширины и глубины. Провести анализ полученных результатов

Ход работы

1.1 Описание реализованного алгоритма построения случайного дерева

Алгоритм работы программы для построения случайного дерева:

Имеется исходный набор (множество) нумерованных вершин. На первом шаге случайным образом выбирается исходная вершина, от которой будет строиться дерево. Затем на основании выбранной вершины генерируется множество возможных связей (рёбер) с каждой вершиной (петли исключаются). Случайным образом выбирается элемент из множество потенциальных связей. Этот элемент также добавляется во множество исключений, в котором находятся связи, которые уже построены, или те, которые в процессе построения дерева не могут быть построены (из-за образования циклов). Вновь генерируется уже подмножество потенциальных связей, из них выбирается новая связь, часть добавляется в исключение. Так происходит до тех пор, пока множество потенциальных связей не опустеет

1.2 Результат работы программы

В результате работы программы построено случайное дерево, визуальное представление которого изображено на рисунке 1:

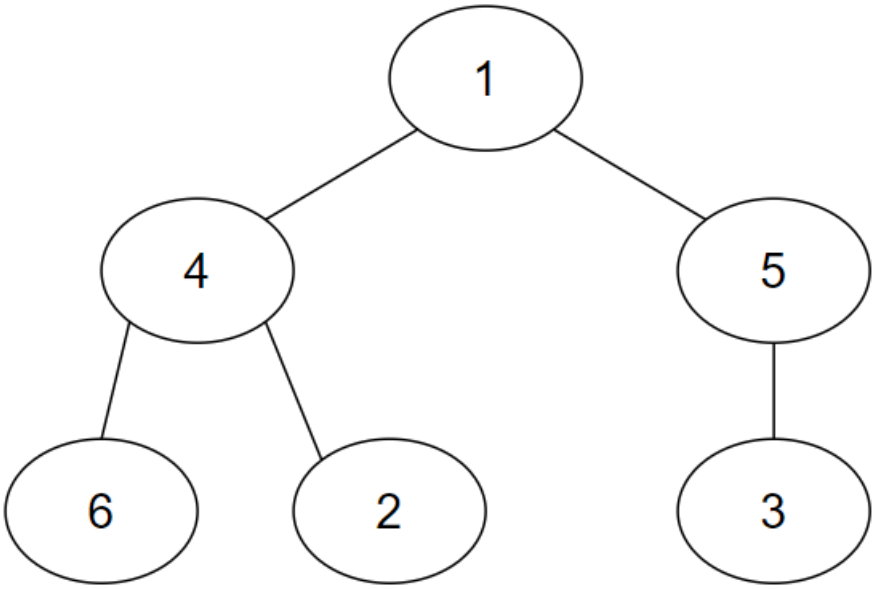

Рисунок 1 – Случайно построенное дерево из 6 элементов

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы изучен и реализован алгоритм построения случайного дерева методом допустимого выбора с ограничением на степень вершины и глубину дерева

ПРИЛОЖЕНИЕ

Код программы

```
use rand::Rng;
use std::collections::{HashMap, HashSet};

fn generate_pull(vec_nums: &Vec<u64>) -> HashSet<(u64, u64)> {
    let mut set: HashSet<(u64, u64)> = HashSet::new();

    // весь возможный пулл
    for i in vec_nums.iter() {
        for j in vec_nums.iter() {
            if j != i {
                set.insert((*i, *j));
            }
        }
    }
    let mut ban_vec = HashSet::new();

    // вычисляем двойников
    for (i, j) in set.iter() {
        if !ban_vec.contains(&(*i, *j)) {
            ban_vec.insert((*j, *i));
        }
    }

    // удаляем двойников
    for (i, j) in ban_vec {
        set.remove(&(i, j));
    }

    return set;
}

// убирает пересечения
fn clear_pull(
    mut input_pull: HashSet<(u64, u64)>,
    ban_pull: HashSet<(u64, u64)>,
) -> HashSet<(u64, u64)> {
    for (i, j) in ban_pull {
        input_pull.remove(&(i, j));
```

```
        input_pull.remove(&(j, i));
    }

    return input_pull;
}

fn get_usefull_pull(
    vec_nums: &Vec<u64>,
    full_pull: HashSet<(u64, u64)>,
    // параметры глубины и ширины
    all_nodes: &Vec<u64>,
    nodes_params: HashMap<u64, (u64, u64)>,
    max_width: u64,
    max_depth: u64,
) -> HashSet<(u64, u64)> {
    let mut result_pull: HashSet<(u64, u64)> = HashSet::new();
    for node in vec_nums.iter() {
        for (i, j) in full_pull.iter() {
            if *i == *node || *j == *node {
                result_pull.insert((*i, *j));
            }
        }
    }

    for (i, (width, depth)) in nodes_params {
        if width >= max_width || depth >= max_depth {
            // выясняем запрещенные теперь связи
            let mut ban_vec = HashSet::new();

            // вычисляем связи, которые нельзя переиспользовать
            for j in all_nodes.iter() {
                if result_pull.contains(&(i, *j)) {
                    ban_vec.insert((i, *j));
                }

                if result_pull.contains(&(*j, i)) {
                    ban_vec.insert((*j, i));
                }
            }

            for k in ban_vec {
```

```
                result_pull.remove(&k);
            }
        }
    }

    return result_pull;
}

pub fn main() {
    let mut rng = rand::thread_rng();

    let all_vec = &vec![1, 2, 3, 4, 5, 6];
    let mut use_nodes: Vec<u64> = vec![1];
    let full_pull: HashSet<(u64, u64)> = generate_pull(&all_vec);

    // let max_w = rng.gen_range(2..6);
    // let max_d = rng.gen_range(2..6);

    // по номеру узла понимаем его глубину и кол-во потомков
    let mut nodes_params: HashMap<u64, (u64, u64)> = HashMap::new();

    nodes_params.insert(*use_nodes.get(0).unwrap(), (0, 0));

    loop {
        let ban_pull = generate_pull(&use_nodes);

        let pull_not_ban = clear_pull(full_pull.clone(), ban_pull);

        let usefull_vec = get_usefull_pull(
            &use_nodes,
            pull_not_ban,
            all_vec,
            nodes_params.clone(),
            10,
            10,
        )
        .iter()
        .map(|p: &(u64, u64)| *p)
        .collect::<Vec<(u64, u64)>>();

        if usefull_vec.is_empty() {
```

```
            break;
        }

        // выбираем номер связи и ноду, с которой связали

        let num = {
            if usefull_vec.len() == 1 {
                0
            } else {
                rng.gen_range(0..usefull_vec.len() - 1)
            }
        };

        let (i, j) = usefull_vec.get(num).unwrap();

        let mut flag_i = true;
        let mut flag_j = true;

        for k in use_nodes.iter() {
            if *i == *k {
                flag_i = false
            }
            if *j == *k {
                flag_j = false
            }
        }

        if flag_i {
            // i - новая нода
            use_nodes.push(*i);

            let wight = {
                let (wight, childes) = nodes_params.get_mut(j).unwrap();

                *childes += 1;
                *wight
            };

            nodes_params.insert(*i, (wight + 1, 0));
        }
        if flag_j {
```

```
            use_nodes.push(*j);

            let wight = {
                let (wight, childes) = nodes_params.get_mut(i).unwrap();

                *childes += 1;
                *wight
            };

            nodes_params.insert(*j, (wight + 1, 0));
        }

        println!("{} -> {}", i, j);
    }

    println!("params: {:#?}", nodes_params);
}
```