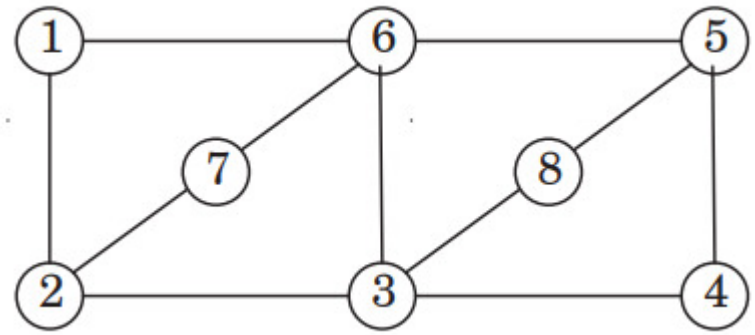
Цель работы: получение практических навыков оценки надежности вычислительных сетей.

Вариант задания: 17

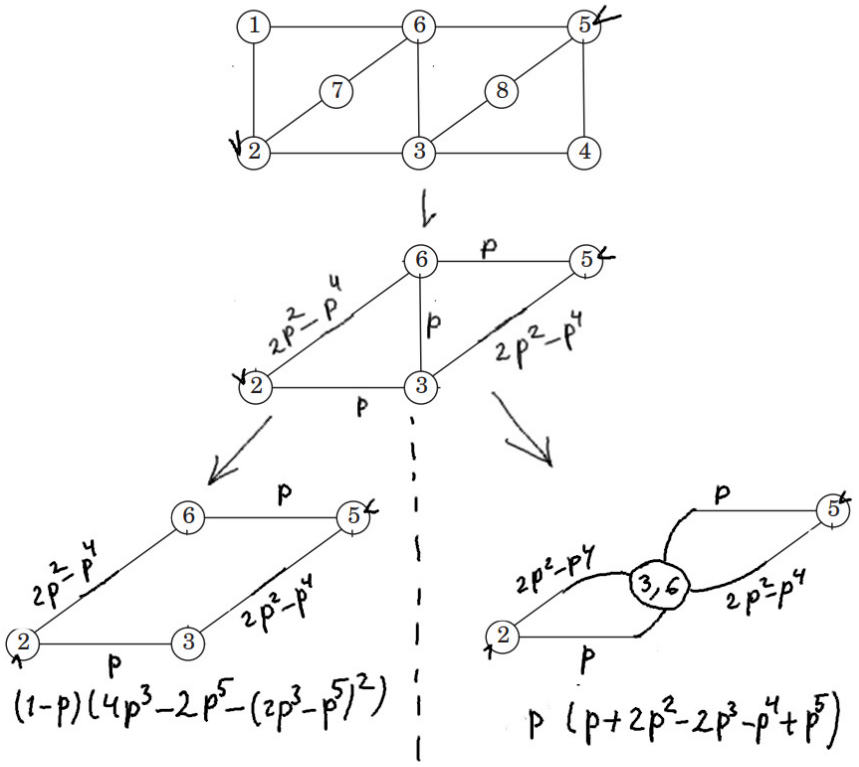
Задан случайный граф G(X,Y,P), где $X=\{x_i\}$ – множество вершин, $Y=\{(x_i ,x_j)\}$ – множество ребер, $P=\{p_i\}$ – множество вероятностей существования ребер. Вероятности существования ребер равны между собой и равны p. В ходе выполнения лабораторной работы необходимо выполнить следующие действия.

1. Вычислить вероятность существования пути между заданной парой вершин $x_i=2,x_j=5$ в графе G.

2. Построить зависимость вероятности существования пути в случайном графе от вероятности существования ребра..



Вывод формулы



Описание программы

Программа выполняет вычисление вероятности наличия пути из 2 в 5 двумя способами: перебором возможных графов с путём (поиск пути происходит через полный обход графа) и суммированием вероятности каждого отдельного такого графа, а также вычисления по выведенной формуле

Результаты работы программы:
Для полного перебора: 0 , 0.00496871272000002 , 0.0386956697600003 , 0.123339567239999 , 0.26599702528 , 0.453125 , 0.652695736320002 , 0.825302631159994 , 0.94062116864 , 0.991758368880001 , 1
Для формулы: 0 , 0.00496871272 , 0.03869566976 , 0.12333956724 , 0.26599702528 , 0.453125 , 0.65269573632 , 0.82530263116 , 0.9406211 6864 , 0.99175836888 , 1

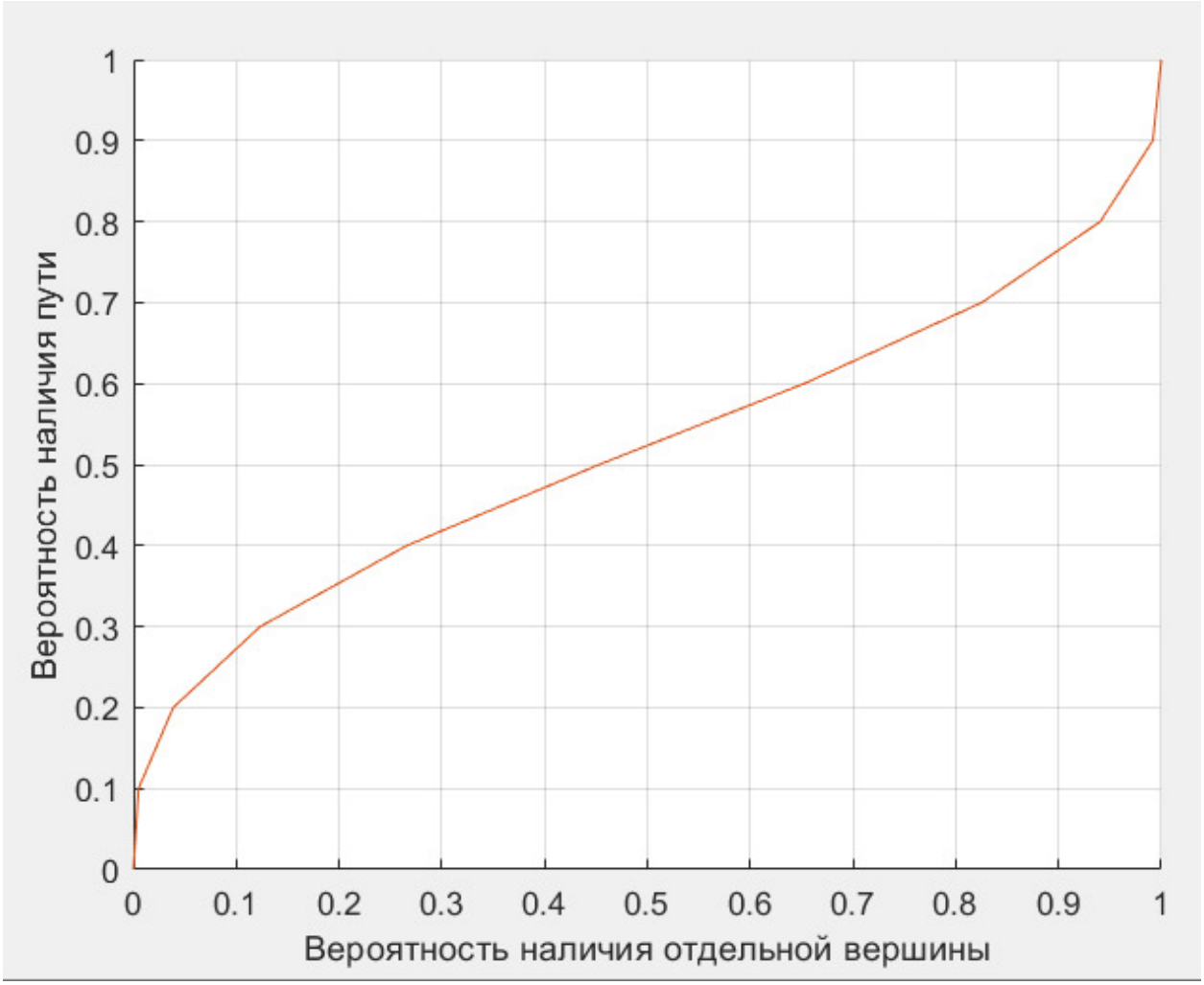


Рисунок 1 График вероятности для полного перебора

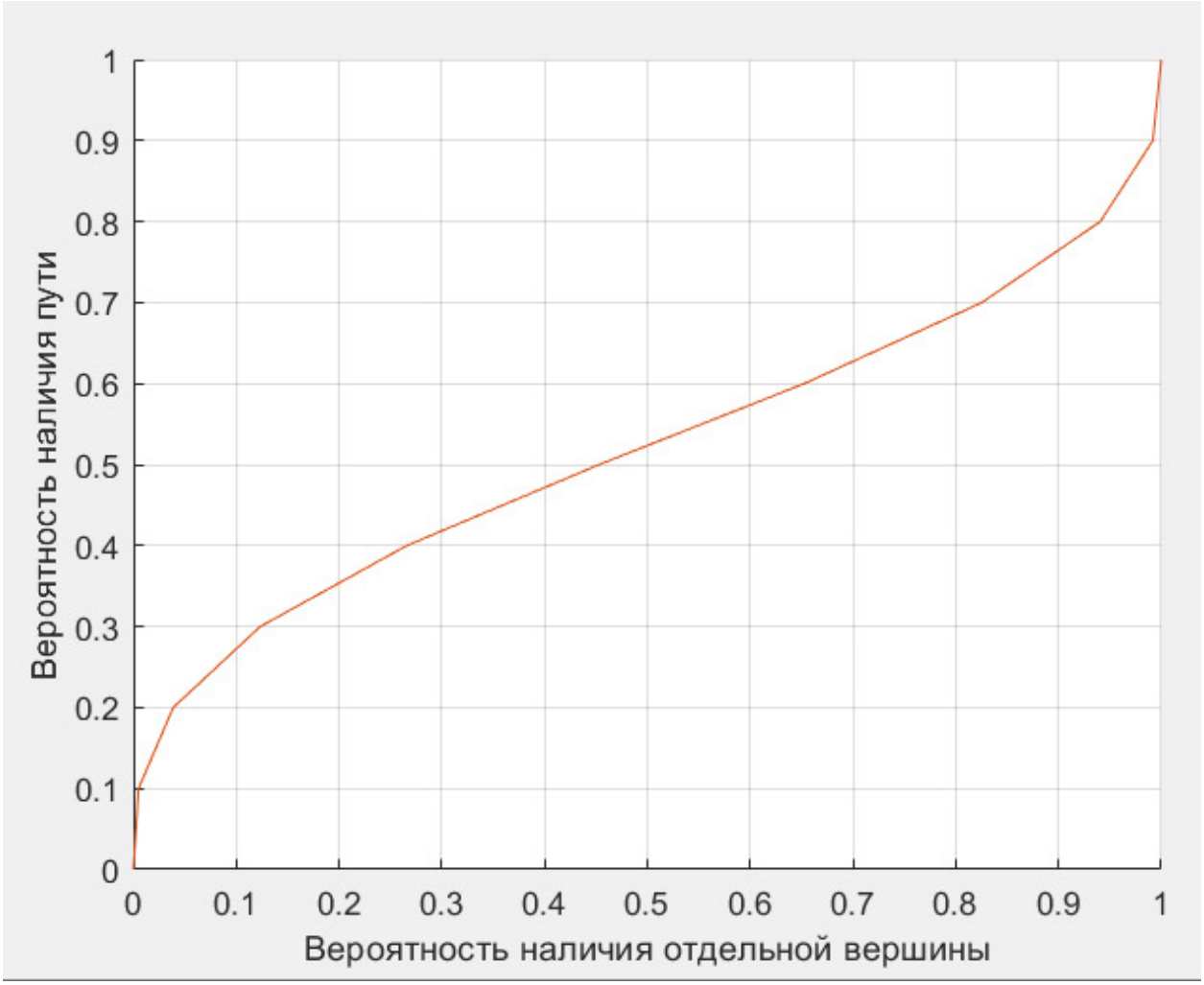


Рисунок 2 График вероятности формулы

Выводы

Результаты полного перебора и формулы сошлись с точностью до 6 знака после запятой, что свидетельствует о правильности расчётов и высокой точности вычисления при использовании полного перебора

Текст программы

```
#include <iostream>
#include <vector>
#include <cmath>
#include <queue>
#include <bitset>
#include <iomanip>

struct edge {
    int a;
    int b;
};

bool bfs(std::vector<edge> &forcer);

int main() {

    std::vector<double> v0;
    std::vector<double> v1;
    std::vector<bool> flags(11, false);
    std::vector<edge> forcer;
    std::vector<edge> edges{
            {0, 1},
            {0, 5},
            {1, 2},
            {1, 6},
            {2, 3},
            {2, 5},
            {2, 7},
            {3, 4},
            {4, 5},
            {4, 7},
            {5, 6}
    };
    double p = 0;
    for (; p <= 1; p += 0.1) {
        double calc = 0;
        for (int i = 0; i < 2048; i++) {
            for (int j = 0; j < 11; j++) {
                (i >> j) % 2 ? forcer.push_back(edges[j]) : void();
            }
            int p_pow = std::bitset<11>(i).count();
            calc += bfs(forcer) * pow(p, p_pow) * pow(1 - p, 11 - p_pow);
            forcer.clear();
        }
        v0.push_back(calc);
        double a = (1 - p) * (4 * p * p * p - 2 * pow(p, 5) - pow((2 * p * p
* p - pow(p, 5)), 2))
                   + p * pow(p + 2 * p * p - 2 * p * p * p - pow(p, 4) +
pow(p, 5), 2);
        v1.push_back(a);
    }
    std::cout << std::setprecision(15);
```

```
    for (double v: v0) {
        std::cout << v << " , ";
    }
    std::cout << std::endl;
    for (double v: v1) {
        std::cout << v << " , ";
    }
    std::cout << std::endl;
    return 0;
}

bool bfs(std::vector<edge> &forcer) {
    std::vector<std::vector<bool>> matrix;
    std::vector<bool> used(8, false);
    for (int i = 0; i < 8; i++) {
        matrix.emplace_back(8, false);
    }
    for (edge e: forcer) {
        matrix[e.a][e.b] = true;
        matrix[e.b][e.a] = true;
    }
    std::queue<int> queue;
    queue.push(1);
    while (!queue.empty()) {
        int v = queue.front();
        queue.pop();
        for (int i = 0; i < 8; i++) {
            if (matrix[v][i] && !used[i]) {
                used[i] = true;
                queue.push(i);
            }
        }
    }
    return used[4];
}
```