

Цель работы: получение практических навыков оценки надежности вычислительных сетей.

Вариант задания: 1

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  $x_i=1, x_j=5$  в графе  $G$ .
2. Построить зависимость вероятности существования пути в случайном графе от вероятности существования ребра.

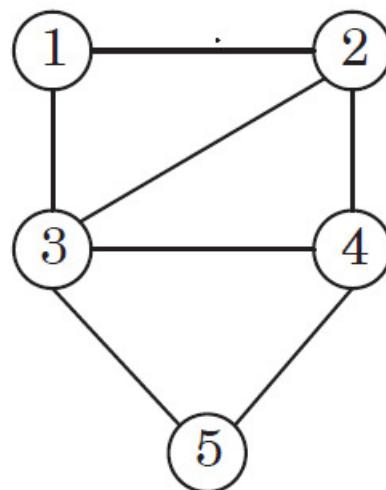


Рисунок 1 – исходный граф.



## Вывод формулы

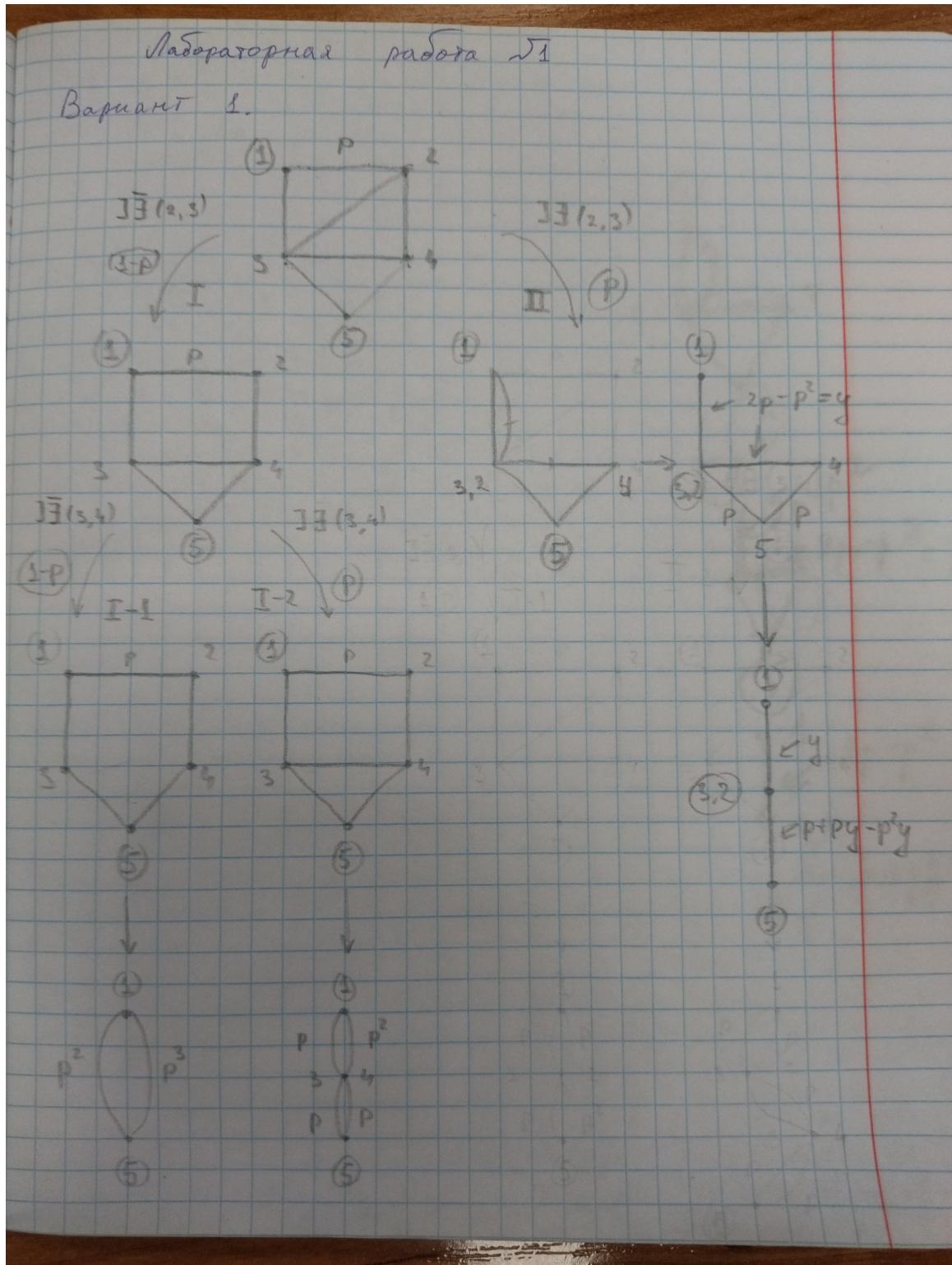


Рисунок 2 – преобразование исходного графа

$$\text{I. } (p^2 + p^3 - p^5)$$

$$\text{II. } (p + p^2 - p^3)(2p - p^2)$$

$$\text{III. } (2p - p^2)(p + p(2p - p^2) - p^2(2p - p^2)) = (2p - p^2)(p + 2p^2 - 3p^3 + p^4)$$

$$\Pr\{\text{Путь 1, 5}\} = (\text{I} * (1 - p) + \text{II} * p)(1 - p) + \text{III} * p$$

$$\Pr\{\text{Путь 1, 5}\} = (1 - p)((1 - p)(p^2 + p^3 - p^5) + p(p + p^2 - p^3)(2p - p^2)) + p(2p - p^2)(p + p(2p - p^2) - p^2(2p - p^2)) = (2p - p^2)(p + 2p^2 - 3p^3 + p^4)$$



## Описание программы

Программа выполняет вычисление вероятности наличия пути из 1 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, 0.0129907, 0.0624256, 0.1573029, 0.2948608, 0.4609375, 0.6337152, 0.7893361, 0.9078784, 0.9781803, 1.

Для формулы:

0, 0.0129907, 0.0624256, 0.1573029, 0.2948608, 0.4609375, 0.6337152, 0.7893361, 0.9078784, 0.9781803, 1.

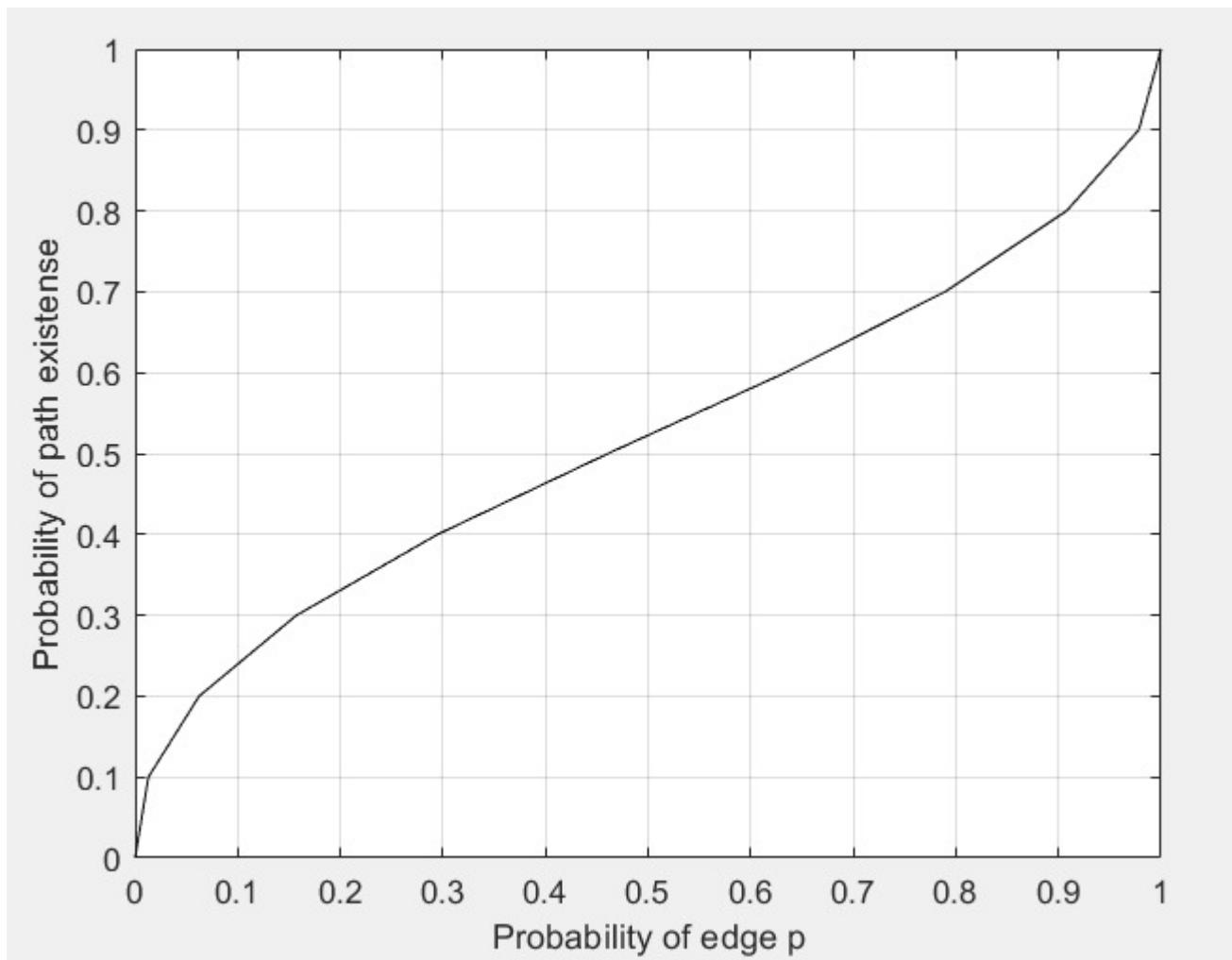


Рисунок 2 – график зависимости вероятности существования пути от вероятности существования ребра.

## 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.

## Листинг программы

```
#include <iostream>
#include <vector>
#include <cmath>
#include <queue>
#include <bitset>
#include <iomanip>
#define N 5
#define L 7
#define IN 0
#define OUT 4
//Node numeration starts from zero

struct edge {
    int a;
    int b;
};

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

int main0 {

    std::vector<double> v0;
    std::vector<double> v1;
    std::vector<edge> forcer;
    std::vector<edge> edges{
        {0, 1},
        {0, 2},
        {1, 2},
        {1, 3},
        {2, 3},
        {2, 4},
        {3, 4}
    };
    double p = 0;
    for (; p <= 1; p += 0.1) {
        double calc = 0;
        for (int i = 0; i < (int)pow(2, L); i++) {
            for (int j = 0; j < L; j++) {
                (i >> j) % 2 ? forcer.push_back(edges[j]) : void();
            }
            int p_pow = std::bitset<L>(i).count();
            calc += bfs(forcer) * pow(p, p_pow) * pow(1 - p, L - p_pow);
            forcer.clear();
        }
        v0.push_back(calc);
        double a = (1 - p) * ((pow(p, 2) + pow(p, 3) - pow(p, 5)) * (1 - p) + (2 * p - pow(p, 2)) * (p + pow(p, 2) -
        pow(p, 3)) * p
            + p * (2 * p - pow(p, 2)) * ((p + 2 * pow(p, 2) - 3 * pow(p, 3) + pow(p, 4)));
        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(N, false);
    for (int i = 0; i < N; i++) {
        matrix.emplace_back(N, false);
    }
    for (edge e : forcer) {
        matrix[e.a][e.b] = true;
        matrix[e.b][e.a] = true;
    }
    std::queue<int> queue;
    queue.push(IN);
    while (!queue.empty()) {
        int v = queue.front();
        queue.pop();
        for (int i = 0; i < N; i++) {
            if (matrix[v][i] && !used[i]) {
                used[i] = true;
                queue.push(i);
            }
        }
    }
    return used[OUT];
}

```