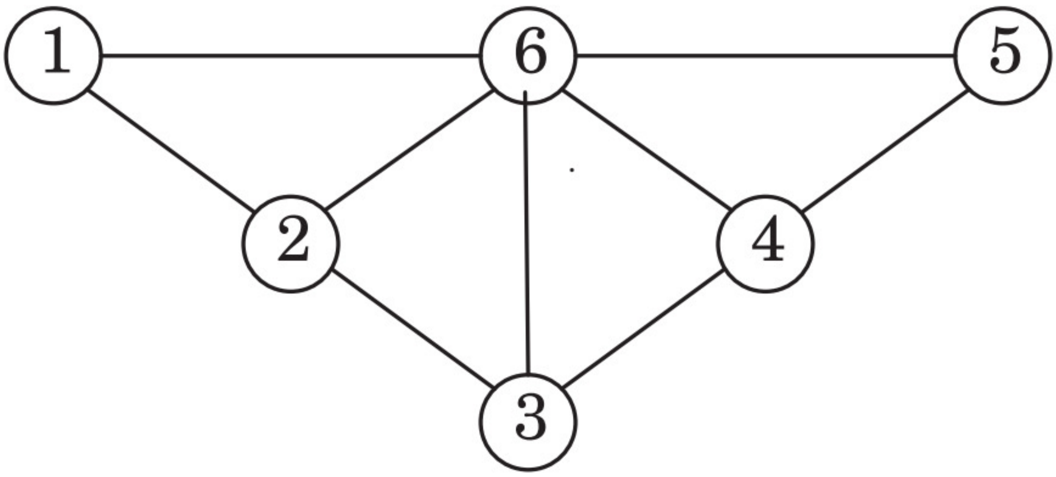
Цель работы: получение практических навыков оценки надежности вычислительных сетей.

Вариант задания: 19

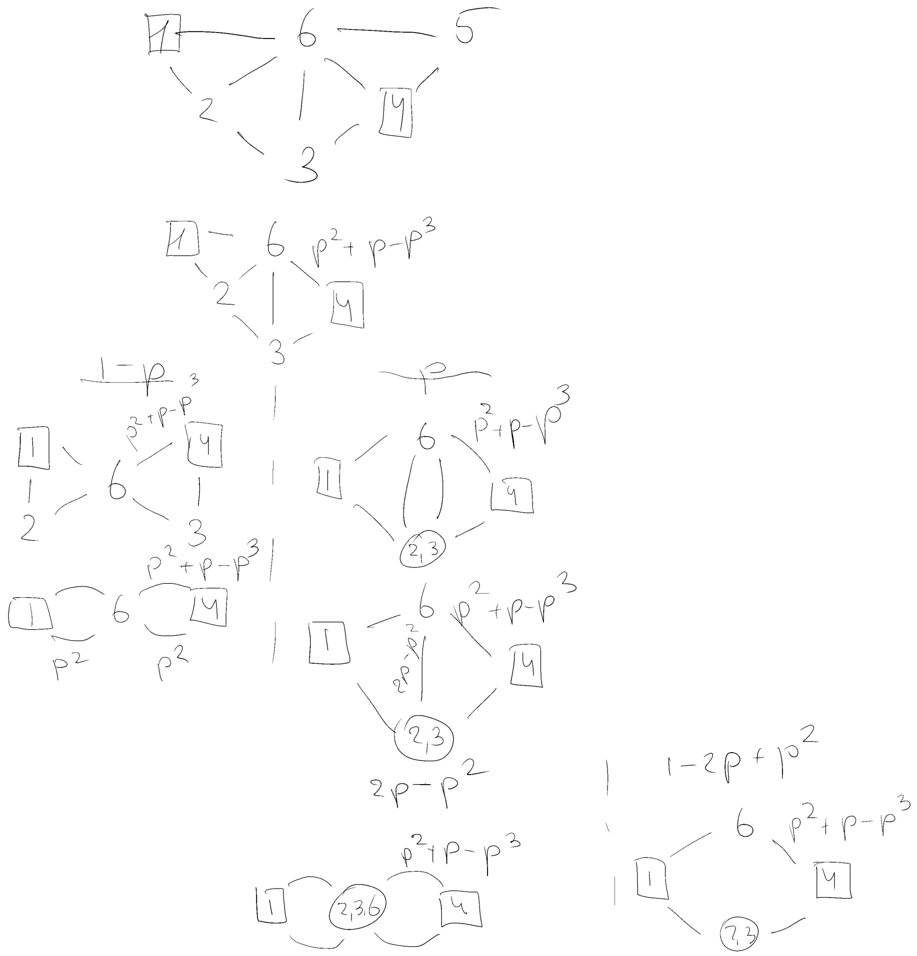
Задан случайный граф G(X,Y,P), где $X=\{x_i\}$ – множество вершин, $Y=\{(x_i, x_j)\}$ – множество ребер, $P=\{p_i\}$ – множество вероятностей существования ребер. Вероятности существования ребер равны между собой и равны p. В ходе выполнения лабораторной работы необходимо выполнить следующие действия.

1. Вычислить вероятность существования пути между заданной парой вершин $x_i=1$, $x_j=4$ в графе G.

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



Вывод формулы



$$P=(p^2+p-p^3)(p^2+p^2+p-p^3-(p^4+p^3-p^5))(1-p)+((2p-p^2)(p^2+p-p^3+$$
$$+p-p(p^2+p-p^3))(2p-p^2)+(p^2+(p^3+p^2-p^4)-p^2(p^3+p^2-p^4))(1-$$
$$-2p+p^2))p$$

Описание программы

Программа выполняет вычисление вероятности наличия пути из 1 в 4 двумя способами: перебором возможных графов с путём и суммированием вероятности каждого отдельного такого графа, а также вычисления по выведенной формуле

Результаты работы программы:

Для полного перебора:
0, 0.013966183, 0.069577216, 0.177839829, 0.332941312, 0.513671875, 0.690835968, 0.837255601, 0.936484864, 0.986959647, 1

Для формулы:
0, 0.013966183, 0.069577216, 0.177839829, 0.332941312, 0.513671875, 0.690835968, 0.837255601, 0.936484864, 0.986959647, 1

Рисунок 1- График вероятности для полного перебора

Рисунок 2 - График вероятности формулы

Выводы

Результаты полного перебора и формулы сошлись с точностью до 16 знака после запятой, что свидетельствует о правильности расчётов и высокой точности вычисления при использовании полного перебора

Текст программы

```cpp
#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<edge> forcer;
    std::vector<edge> edges{
            {0, 1},
            {0, 5},
            {1, 2},
            {1, 5},
            {2, 3},
            {2, 5},
            {3, 4},
            {3, 5},
            {4, 5},
    };
    int edges_count = 9;



    double p = 0;
    for (; p <= 1; p += 0.1) {
        double calc = 0;
        for (int i = 0; i < pow(2, edges_count); i++) {
            for (int j = 0; j < edges_count; j++) {
                (i >> j) % 2 ? forcer.push_back(edges[j]) : void();
            }
            int p_pow = std::bitset<9>(i).count();
            calc += bfs(forcer) * pow(p, p_pow) * pow(1 - p, edges_count - p_pow);
            forcer.clear();
        }
        v0.push_back(calc);
        double ff = (pow(p, 2) + p - pow(p, 3)) * ((pow(p, 5) + 2 * pow(p, 2) + p - 2 * pow(p, 3) - pow(p, 4)) * (1 - p));
        double sf = pow((2 * p - pow(p, 2)), 2) * (pow(p, 4) + 2 * p - 2 * pow(p, 3));
        double ss = (pow(p, 6) + pow(p, 3) + 2 * pow(p, 2) - 2 * pow(p, 4) - pow(p, 5)) * (1 - 2 * p + pow(p, 2));

        double a = ff + (sf + ss) * p;
        v1.push_back(a);
    }
```

```cpp
    std::cout << std::setprecision(14);
    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) {
    int vertex_count = 6;
    std::vector<std::vector<bool>> matrix;
    std::vector<bool> used(vertex_count, false);
    for (int i = 0; i < vertex_count; i++) {
        matrix.emplace_back(vertex_count, false);
    }
    for (edge e : forcer) {
        matrix[e.a][e.b] = true;
        matrix[e.b][e.a] = true;
    }
    std::queue<int> queue;
    queue.push(0);
    while (!queue.empty()) {
        int v = queue.front();
        queue.pop();
        for (int i = 0; i < vertex_count; i++) {
            if (matrix[v][i] && !used[i]) {
                used[i] = true;
                queue.push(i);
            }
        }
    }
    return used[3];
}
```