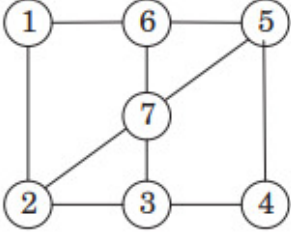


1. Цель работы

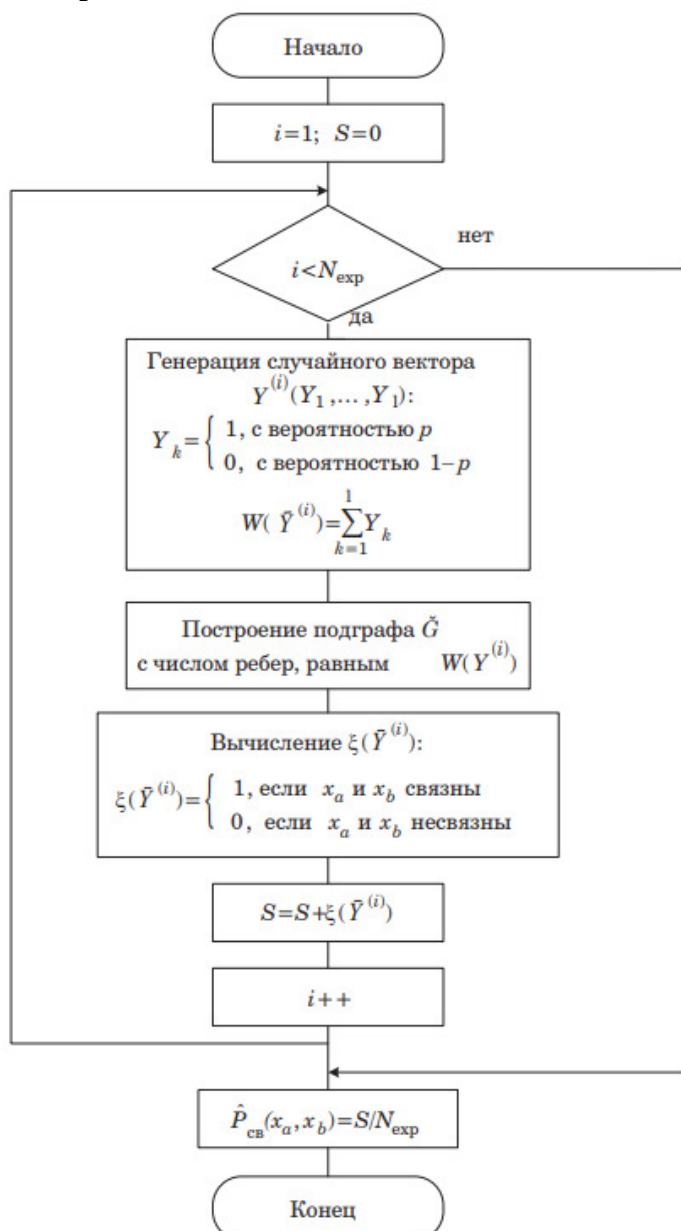
Получение практических навыков использования моделирования для оценки надежности вычислительных сетей.

2. Вариант задания

Вариант 14		$x_i=2$ $x_j=5$
---------------	---	--------------------

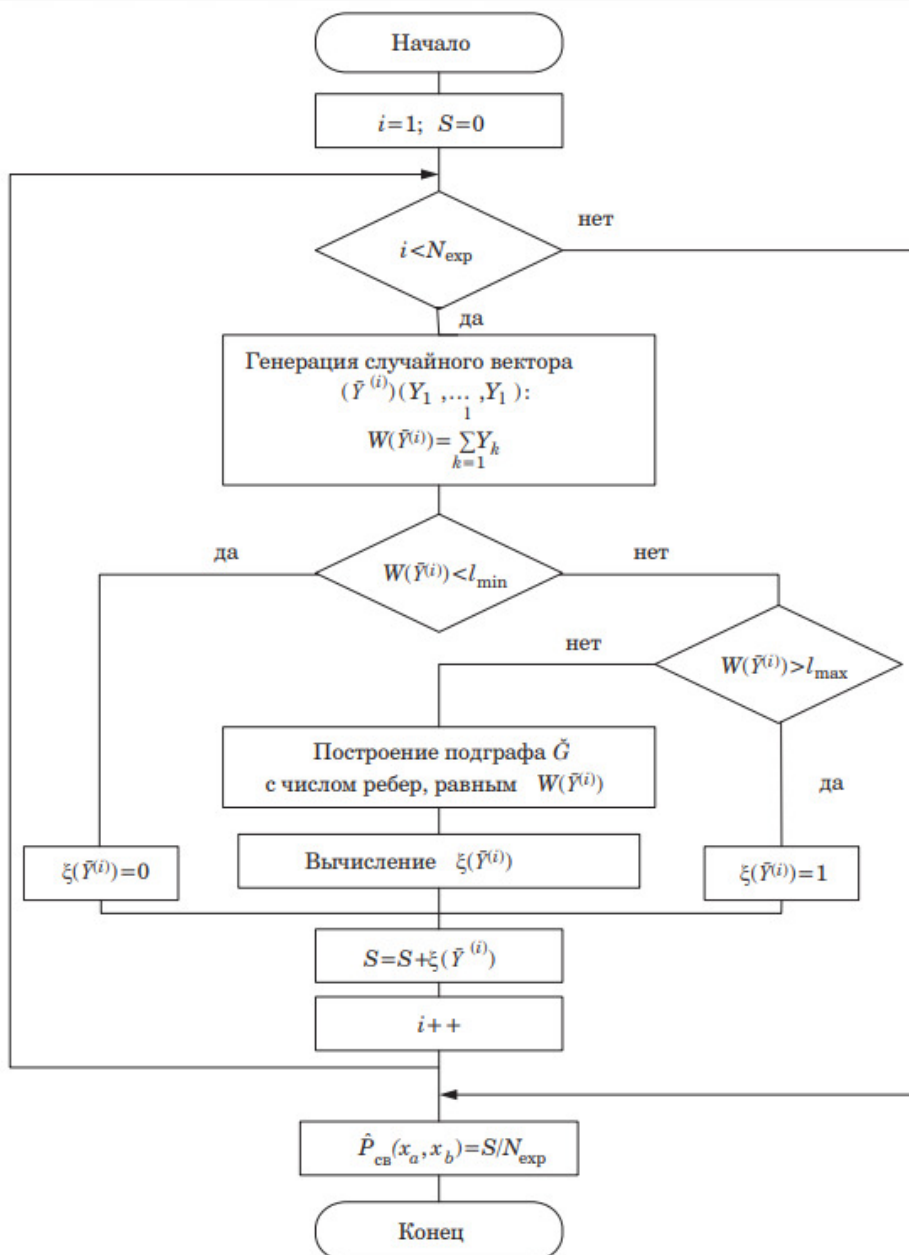
3. Простое и ускоренное имитационное моделирование.

Описание разработанной программы простого имитационного моделирования:



Описание разработанной программы ускоренного имитационного моделирования:

$l_{\min} = 2, l_{\max} = 7$



Результаты работы программ:

p=0
enumeration: 0
simulation modeling: 0
quick simulation modeling: 0

p=0.1
enumeration: 0.0139995
simulation modeling: 0.0136
quick simulation modeling: 0.0136444

p=0.2
enumeration: 0.0704112
simulation modeling: 0.0707556
quick simulation modeling: 0.0728

p=0.3
enumeration: 0.182481
simulation modeling: 0.183822
quick simulation modeling: 0.185733

p=0.4
enumeration: 0.346271
simulation modeling: 0.351378
quick simulation modeling: 0.3456

p=0.5
enumeration: 0.539063
simulation modeling: 0.537289
quick simulation modeling: 0.535556

p=0.6
enumeration: 0.725971
simulation modeling: 0.724711
quick simulation modeling: 0.724933

p=0.7
enumeration: 0.872984
simulation modeling: 0.871244
quick simulation modeling: 0.871733

p=0.8
enumeration: 0.961231
simulation modeling: 0.962311
quick simulation modeling: 0.9612

p=0.9
enumeration: 0.99538
simulation modeling: 0.995289
quick simulation modeling: 0.995689

p=1
enumeration: 1
simulation modeling: 1
quick simulation modeling: 1|

4. Графики зависимости $p_{ij}(p)$

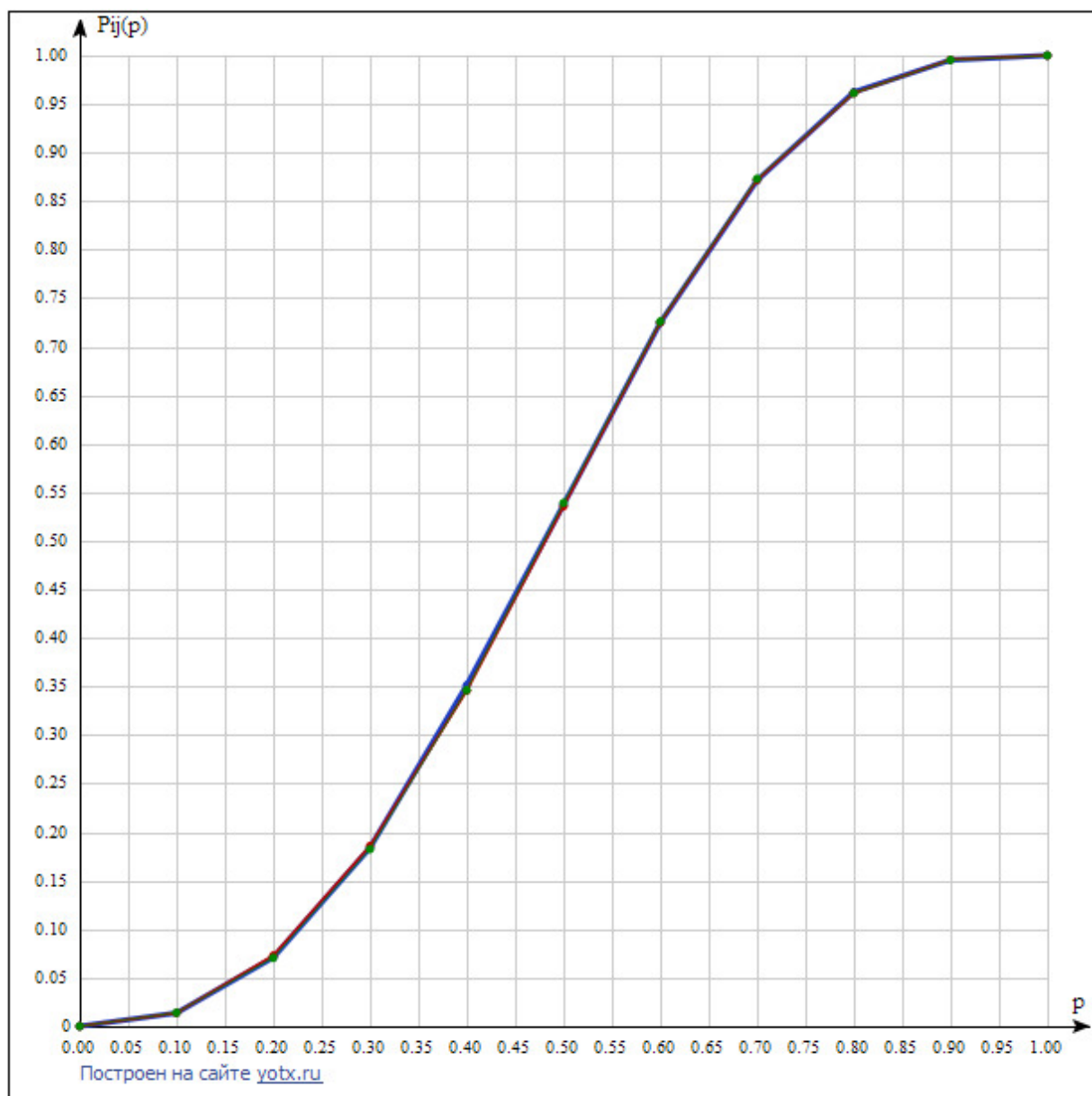


Рисунок 1 - Графики зависимостей $P_{ij}(p)$. Синяя линия – простое имитационное моделирование, красная – ускоренное имитационное моделирование, зеленая – по формуле

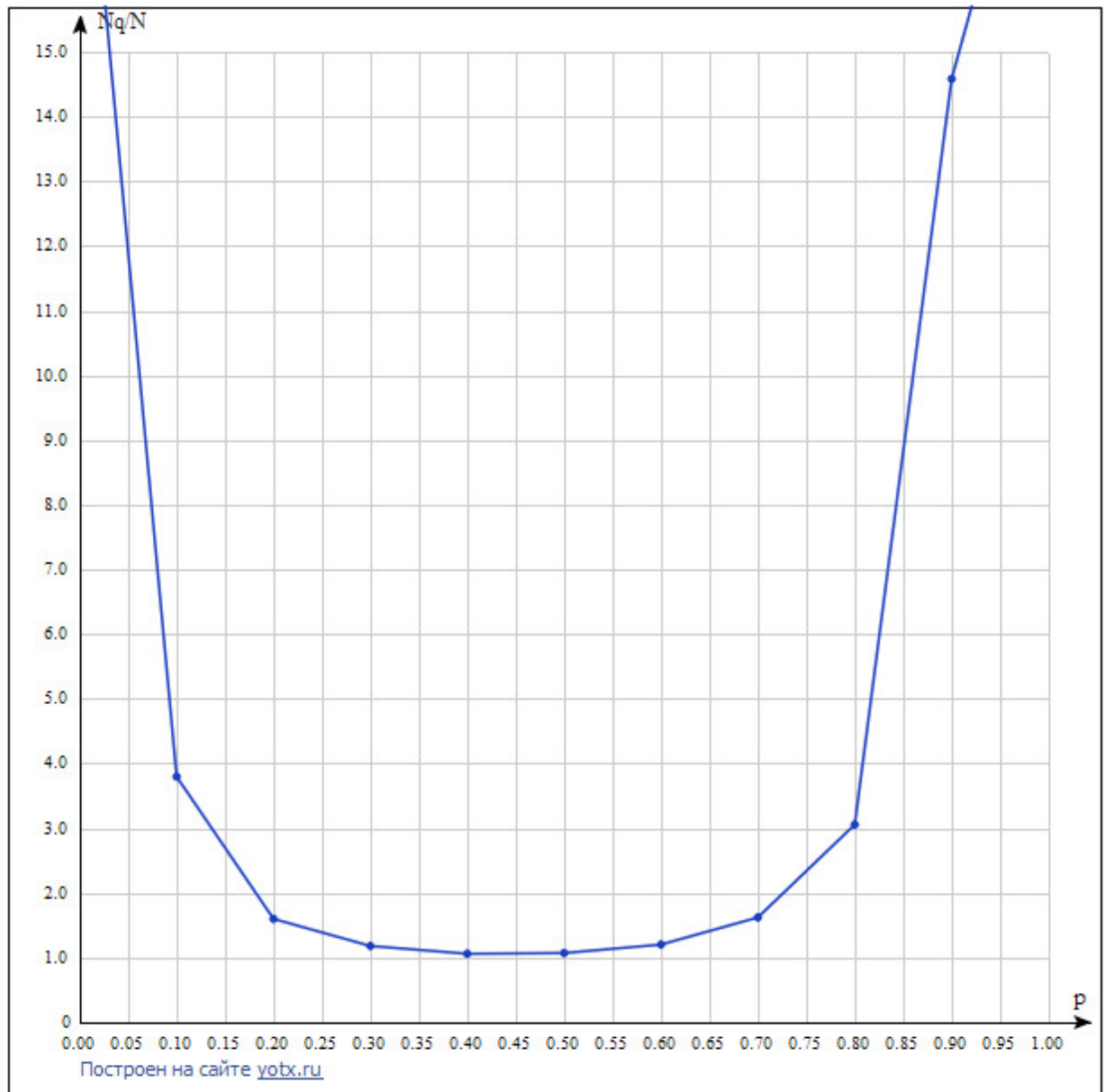


Рисунок 2 - Зависимость выигрыша при ускоренном имитационном моделировании от вероятности

5. Выводы

В ходе выполнения лабораторной работы, были получены практические навыки оценки надежности вычислительных сетей, была вычислена вероятность существования пути в случайном графе, как функции от p , построена программа для оценки вероятности связности пары вершин при помощи имитационного моделирования.

6. Листинг кода

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <vector>
#include <bitset>
#include <fstream>

#define EDGES 10
#define VERTEX 7
```

```

// 10 7
using namespace std;
class Graph {
public:
    double probability = 0;
    bool* visited;
    bool isWay = false;
    vector<int> o;
    int v1 = 1;
    int v2 = 4;
    int l_min = 2;
    int l_max = 7;
    double probSim;
    double probSimQuick;

    int mtx[7][7] = {
        {0,1,0,0,0,1,0},
        {1,0,1,0,0,0,1},
        {0,1,0,1,0,0,1},
        {0,0,1,0,1,0,0},
        {0,0,0,1,0,1,1},
        {1,0,0,0,1,0,1},
        {0,1,1,0,1,1,0},
    };
    Graph() {
        visited = new bool[VERTEX];
        for (int i = 0; i < VERTEX; i++) {
            visited[i] = 0;
        }
        for (int i = 0; i < VERTEX - 1; i++) {
            for (int j = i + 1; j < VERTEX; j++) {
                o.push_back(mtx[i][j]);
            }
        }
    }
    void fill() {
        for (int i = 0; i < VERTEX; i++) {
            for (int j = 0; j < VERTEX; j++) {
                cin >> mtx[i][j];
            }
        }
    }
    void dfs(int st, int en) {
        visited[st] = true;
        if(st == en) {
            isWay = true;
            return;
        }
        for (int i = 0; i < VERTEX; i++) {
            if ((mtx[st][i] == 1) && (visited[i]==false)) {
                dfs(i, en);
            }
        }
    }
    void track(int v1, int v2, double p) {
        probability = 0;
        for (int k = 0; k < pow(2, EDGES); k++) {
            bitset<EDGES> bs(k);
            double prop = 1.0;
            int idx = 0;
            int oidx = 0;
            for (int i = 0; i < VERTEX - 1; i++) {
                for (int j = i + 1; j < VERTEX; j++) {
                    if (o[oidx] == 1) {
                        mtx[i][j] = bs[idx];
                        mtx[j][i] = bs[idx];
                    }
                }
            }
        }
    }
};

```

```

        prop *= bs[idx] ? p : (1.0 - p);
        idx++;
    }
    else {
        mtx[i][j] = 0;
        mtx[j][i] = 0;
    }
    oidx++;
}
}
isWay = false;
dfs(v1, v2);
if (isWay) {
    probability += prop;
}
for (int i = 0; i < VERTEX; i++) {
    visited[i] = 0;
}
}
}
int factorial(int i) {
    if (i == 0) return 1;
    else return i * factorial(i - 1);
}
double compinations(int n, int k) {
    return factorial(n) / (factorial(k) * factorial(n - k));
}
double disp(double p, double e) {
    int N = 9 / (4 * e * e);
    int jj[EDGES];
    int Nj[EDGES];
    double Pp[EDGES];
    double Pj[EDGES];
    for (int i = 0; i < EDGES; i++) {
        Nj[i] = (N / EDGES) * (i + 1);
        jj[i] = i + 1;
        Pp[i] = compinations(EDGES, i)*pow(p,i)*pow((p-1),(EDGES-i));
    }
    //for (int l = 0; l < N; l++) {
    for (int j = 0; j < EDGES; j++) {
        int Sj = 0;
        for (int l = j * N / EDGES; l < (j + 1) * N / EDGES; j++) {
            cout << l;
            vector<bool> y;
            int w = 0;
            for (int i = 0; i < EDGES; i++) {
                double rnd = ((double)rand() / (RAND_MAX));
                if (rnd <= p) {
                    y.push_back(1);
                    w++;
                }
                else
                    y.push_back(0);
            }
            if (w == jj[j]) {
                for (int k = i + 1; k < EDGES; k++) {
                    y.push_back(0);
                }
                break;
            }
        }
    }
    while (w < jj[j]) {
        int rnd = rand() % 10;
        if (y[rnd] == 0) {
            y[rnd] = 1;
            w++;
        }
    }
}

```

```

    }

    int oidx = 0;
    int idx = 0;
    for (int i = 0; i < VERTEX - 1; i++) {
        for (int j = i + 1; j < VERTEX; j++) {
            if (o[oidx] == 1) {
                mtx[i][j] = y[idx];
                mtx[j][i] = y[idx];
                idx++;
            }
            else {
                mtx[i][j] = 0;
                mtx[j][i] = 0;
            }
            oidx++;
        }
    }
    isWay = false;
    for (int i = 0; i < VERTEX; i++) {
        visited[i] = 0;
    }
    dfs(v1, v2);
    if (isWay) {
        Sj += 1;
    }
}
Pj[j] = Sj / Nj[j];
}
double Pr = 0;
for (int i = 0; i < EDGES; i++) {
    Pr += Pj[i] * Pp[i];
}
return Pr;
//}
}

double probQuickSimModeling(double p, double e) {
    int N = 9 / (4 * e * e);
    int Sq = 0;
    for (int l = 0; l < N; l++) {
        vector<bool> y;
        int w = 0;
        for (int i = 0; i < EDGES; i++) {
            double rnd = ((double)rand() / (RAND_MAX));
            if (rnd <= p) {
                y.push_back(1);
                w++;
            }
            else
                y.push_back(0);
        }
        int oidx = 0;
        int idx = 0;
        for (int i = 0; i < VERTEX - 1; i++) {
            for (int j = i + 1; j < VERTEX; j++) {
                if (o[oidx] == 1) {
                    mtx[i][j] = y[idx];
                    mtx[j][i] = y[idx];
                    idx++;
                }
                else {
                    mtx[i][j] = 0;
                    mtx[j][i] = 0;
                }
                oidx++;
            }
        }
    }
}

```



```

    }
    if ((w >= l_min)) {
        if (w > l_max) {
            Sq += 1;
        }
        else {
            isWay = false;
            for (int i = 0; i < VERTEX; i++) {
                visited[i] = 0;
            }
            dfs(v1, v2);
            if (isWay) {
                Sq += 1;
            }
        }
    }
}

return (double)Sq / N;
}

double probSimModeling(double p, double e) {
    int N = 9 / (4 * e * e);
    int S = 0;
    int Sq = 0;
    int l = 0;
    for (int l = 0; l < N; l++) {
        vector<bool> y;
        int w = 0;
        for (int i = 0; i < EDGES; i++) {
            double rnd = ((double)rand() / (RAND_MAX));
            if (rnd <= p) {
                y.push_back(1);
                w++;
            }
            else
                y.push_back(0);
        }
        int oidx = 0;
        int idx = 0;
        for (int i = 0; i < VERTEX - 1; i++) {
            for (int j = i + 1; j < VERTEX; j++) {
                if (o[oidx] == 1) {
                    mtx[i][j] = y[idx];
                    mtx[j][i] = y[idx];
                    idx++;
                }
                else {
                    mtx[i][j] = 0;
                    mtx[j][i] = 0;
                }
                oidx++;
            }
        }
        isWay = false;
        for (int i = 0; i < VERTEX; i++) {
            visited[i] = 0;
        }
        dfs(v1, v2);
        if (isWay) {
            S += 1;
        }
        if ((w >= l_min)) {
            if (w > l_max) {
                Sq += 1;
            }
            else {

```

```

        if (isWay) {
            Sq += 1;
        }
    }
}

}
probSim = (double)S / N;
probSimQuick = (double)Sq / N;
return (double)S / N;
}
};

double formula(double p) {
    return 4 * pow(p, 10) - 14 * pow(p, 9) + 12 * pow(p, 8) + 4 * pow(p, 7) - pow(p, 6) - 10 * pow(p, 5) + pow(p,
4) + 4 * pow(p, 3) + p * p;
}

double formuladan(double p) {
    return 3 * pow(p, 9) - 13 * pow(p, 8) + 17 * pow(p, 7) - pow(p, 6) - 11 * pow(p, 5) + 2 * pow(p, 4) + 3 *
pow(p, 3) + p * p;
}

int main()
{
    Graph g = Graph();
    srand(time(NULL));

    ofstream out;
    double e = 0.01;
    out.open("out.txt");
    cout << "epsilon=" << e << endl;
    for (double p = 0; p <= 1; p += 0.1) {
        cout << "p=" << p << endl;
        cout << "enumeration: " << formula(p) << endl;
        cout << "simulation modeling: " << g.probSimModeling(p,e) << endl;
        cout << "quick simulation modeling: " << g.probQuickSimModeling(p,e) << endl;
        cout << endl;
    }
    out.close();

}

```