ГУАП

КАФЕДРА № 25

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Доцент, канд. техн. наук		Н.В. Марковская
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ
ИСПОЛЬЗОВАНИЕ МЕТОДА ПОЛНОГО ПЕРЕБОРА ДЛЯ ОЦЕНКИ НАДЕЖНОСТИ СЕТЕЙ

по курсу: Надежность инфокоммуникационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	3031		И.А.Пастушок
		подпись, дата	инициалы, фамилия

г. Санкт-Петербург
2023 г.

Цель работы: получение практических навыков оценки надежности вычислительных сетей.

Вариант задания: 16

Задан случайный граф G (X, Y, P), где X={x_i} – множество вершин, Y={(x_i, xj)} – 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.

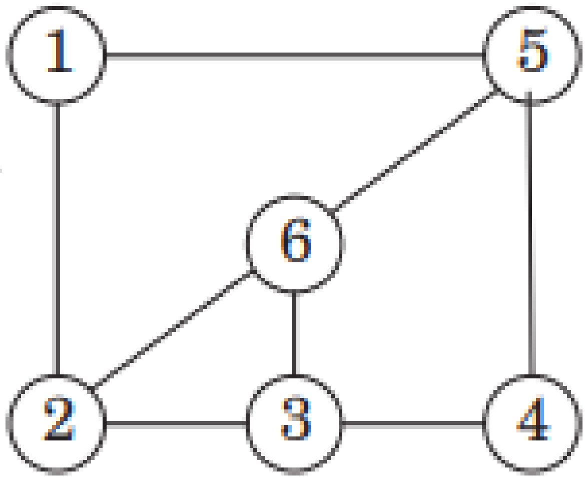
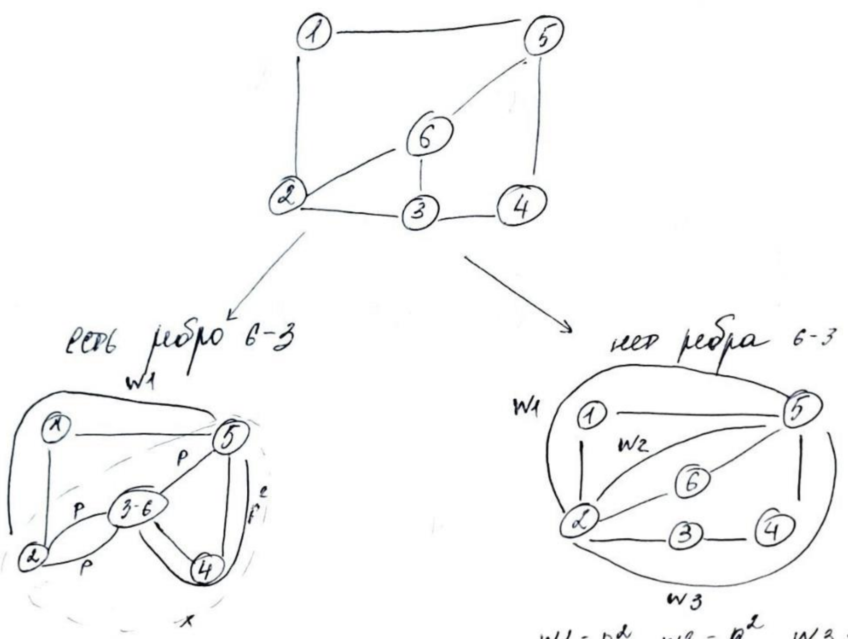


Рисунок 1: исходный график

Упрощение графа (метод декомпозиции):



$x = (p+p-p^2)\cdot(p+p^2-p^3)$

$W1 = p^2$

$res3 = x + p^2 - x\cdot p^2$

$res4 = res3\cdot p$

$RES = rs4 + res2$

$W1 = p^2 \quad W2 = p^2 \quad W3 = p^3$

$res1 = p^2+p^2+p^3-p^2\cdot p^2-p^2\cdot p^3-$
$-p^2\cdot p^3$

$res2 = res1\cdot(1-p)$

Описание программы.

Программа выполняет задание 2-мя способами: полным перебором (идет рассмотрение всевозможных комбинаций ребер и вычисление вероятности существования комбинаций, при которых существует пусть из 2 в 5) и вычисление выведенной выше формулы.

Результат работы программы для вероятности существования ребра 0.125:

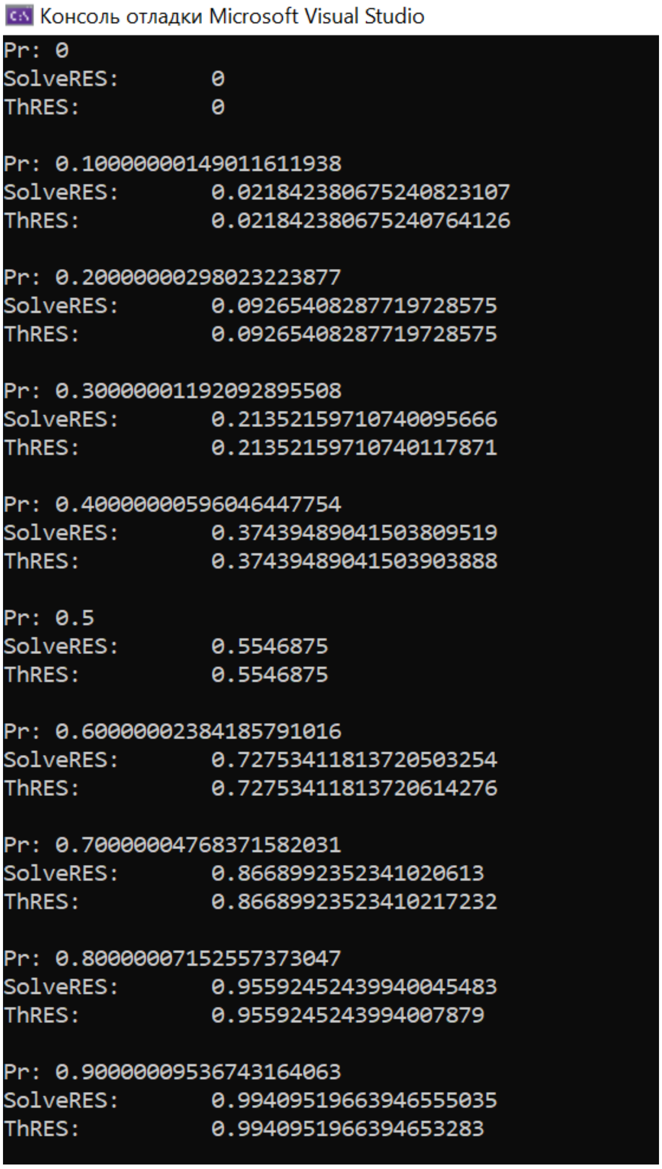
Pr: 0.00014311075210571289	Maska: 00010111
Pr: 0.00701242685317993164	Maska: 00011000
Pr: 0.00100177526473999023	Maska: 00011001
Pr: 0.00100177526473999023	Maska: 00011010
Pr: 0.00014311075210571289	Maska: 00011011
Pr: 0.00100177526473999023	Maska: 00011100
Pr: 0.00014311075210571289	Maska: 00011101
Pr: 0.00014311075210571289	Maska: 00011110
Pr: 0.00002044439315795898	Maska: 00011111
Pr: 0.00100177526473999023	Maska: 00100011
Pr: 0.00014311075210571289	Maska: 00100111
Pr: 0.00014311075210571289	Maska: 00101011
Pr: 0.00100177526473999023	Maska: 00101100
Pr: 0.00014311075210571289	Maska: 00101101

Pr: 0.00014311075210571289 Maska: 00101110
Pr: 0.000020444393157958984 Maska: 00101111
Pr: 0.00014311075210571289 Maska: 00110011
Pr: 0.000020444393157958984 Maska: 00110111
Pr: 0.0010017752647399902 Maska: 00111000
Pr: 0.00014311075210571289 Maska: 00111001
Pr: 0.00014311075210571289 Maska: 00111010
Pr: 0.000020444393157958984 Maska: 00111011
Pr: 0.00014311075210571289 Maska: 00111100
Pr: 0.000020444393157958984 Maska: 00111101
Pr: 0.000020444393157958984 Maska: 00111110
Pr: 0.0000029206275939941406 Maska: 00111111
Pr: 0.000020444393157958984 Maska: 01010111
Pr: 0.0010017752647399902 Maska: 01011000
Pr: 0.00014311075210571289 Maska: 01011001
Pr: 0.00014311075210571289 Maska: 01011010
Pr: 0.000020444393157958984 Maska: 01011011
Pr: 0.00014311075210571289 Maska: 01011100
Pr: 0.000020444393157958984 Maska: 01011101
Pr: 0.000020444393157958984 Maska: 01011110
Pr: 0.0000029206275939941406 Maska: 01011111
Pr: 0.00014311075210571289 Maska: 01100011
Pr: 0.000020444393157958984 Maska: 01100111
Pr: 0.000020444393157958984 Maska: 01101011
Pr: 0.00014311075210571289 Maska: 01101100
Pr: 0.000020444393157958984 Maska: 01101101
Pr: 0.000020444393157958984 Maska: 01101110
Pr: 0.0000029206275939941406 Maska: 01101111
Pr: 0.000020444393157958984 Maska: 01110011
Pr: 0.0000029206275939941406 Maska: 01110111
Pr: 0.00014311075210571289 Maska: 01111000
Pr: 0.000020444393157958984 Maska: 01111001
Pr: 0.000020444393157958984 Maska: 01111010
Pr: 0.0000029206275939941406 Maska: 01111011
Pr: 0.000020444393157958984 Maska: 01111100
Pr: 0.0000029206275939941406 Maska: 01111101
Pr: 0.0000029206275939941406 Maska: 01111110
Pr: 0.0000004172325134277344 Maska: 01111111
Pr: 0.000020444393157958984 Maska: 10010111
Pr: 0.0010017752647399902 Maska: 10011000
Pr: 0.00014311075210571289 Maska: 10011001
Pr: 0.00014311075210571289 Maska: 10011010
Pr: 0.000020444393157958984 Maska: 10011011
Pr: 0.00014311075210571289 Maska: 10011100
Pr: 0.000020444393157958984 Maska: 10011101
Pr: 0.000020444393157958984 Maska: 10011110
Pr: 0.0000029206275939941406 Maska: 10011111
Pr: 0.00014311075210571289 Maska: 10100011
Pr: 0.000020444393157958984 Maska: 10100111
Pr: 0.000020444393157958984 Maska: 10101011
Pr: 0.00014311075210571289 Maska: 10101100
Pr: 0.000020444393157958984 Maska: 10101101
Pr: 0.000020444393157958984 Maska: 10101110
Pr: 0.0000029206275939941406 Maska: 10101111
Pr: 0.000020444393157958984 Maska: 10110011
Pr: 0.0000029206275939941406 Maska: 10110111
Pr: 0.00014311075210571289 Maska: 10111000

Pr: 0.00002044439315795898 Maska: 10111001
Pr: 0.00002044439315795898 Maska: 10111010
Pr: 0.00000292062759399414 Maska: 10111011
Pr: 0.00002044439315795898 Maska: 10111100
Pr: 0.00000292062759399414 Maska: 10111101
Pr: 0.00000292062759399414 Maska: 10111110
Pr: 0.00000041723251342773 Maska: 10111111
Pr: 0.00701242685317993164 Maska: 11000000
Pr: 0.00100177526473999023 Maska: 11000001
Pr: 0.00100177526473999023 Maska: 11000010
Pr: 0.00014311075210571289 Maska: 11000011
Pr: 0.00100177526473999023 Maska: 11000100
Pr: 0.00014311075210571289 Maska: 11000101
Pr: 0.00014311075210571289 Maska: 11000110
Pr: 0.00002044439315795898 Maska: 11000111
Pr: 0.00100177526473999023 Maska: 11001000
Pr: 0.00014311075210571289 Maska: 11001001
Pr: 0.00014311075210571289 Maska: 11001010
Pr: 0.00002044439315795898 Maska: 11001011
Pr: 0.00014311075210571289 Maska: 11001100
Pr: 0.00002044439315795898 Maska: 11001101
Pr: 0.00002044439315795898 Maska: 11001110
Pr: 0.00000292062759399414 Maska: 11001111
Pr: 0.00100177526473999023 Maska: 11010000
Pr: 0.00014311075210571289 Maska: 11010001
Pr: 0.00014311075210571289 Maska: 11010010
Pr: 0.00002044439315795898 Maska: 11010011
Pr: 0.00014311075210571289 Maska: 11010100
Pr: 0.00002044439315795898 Maska: 11010101
Pr: 0.00002044439315795898 Maska: 11010110
Pr: 0.00000292062759399414 Maska: 11010111
Pr: 0.00014311075210571289 Maska: 11011000
Pr: 0.00002044439315795898 Maska: 11011001
Pr: 0.00002044439315795898 Maska: 11011010
Pr: 0.00000292062759399414 Maska: 11011011
Pr: 0.00002044439315795898 Maska: 11011100
Pr: 0.00000292062759399414 Maska: 11011101
Pr: 0.00000292062759399414 Maska: 11011110
Pr: 0.00000041723251342773 Maska: 11011111
Pr: 0.00100177526473999023 Maska: 11100000
Pr: 0.00014311075210571289 Maska: 11100001
Pr: 0.00014311075210571289 Maska: 11100010
Pr: 0.00002044439315795898 Maska: 11100011
Pr: 0.00014311075210571289 Maska: 11100100
Pr: 0.00002044439315795898 Maska: 11100101
Pr: 0.00002044439315795898 Maska: 11100110
Pr: 0.00000292062759399414 Maska: 11100111
Pr: 0.00014311075210571289 Maska: 11101000
Pr: 0.00002044439315795898 Maska: 11101001
Pr: 0.00002044439315795898 Maska: 11101010
Pr: 0.00000292062759399414 Maska: 11101011
Pr: 0.00002044439315795898 Maska: 11101100
Pr: 0.00000292062759399414 Maska: 11101101
Pr: 0.00000292062759399414 Maska: 11101110
Pr: 0.00000041723251342773 Maska: 11101111
Pr: 0.00014311075210571289 Maska: 11110000
Pr: 0.00002044439315795898 Maska: 11110001

Pr: 0.00002044439315795898 Maska: 11110010
Pr: 0.000002920627593994l4 Maska: 11110011
Pr: 0.00002044439315795898 Maska: 11110100
Pr: 0.00000292062759399414 Maska: 11110101
Pr: 0.00000292062759399414 Maska: 11110110
Pr: 0.00000041723251342773 Maska: 11110111
Pr: 0.00002044439315795898 Maska: 11111000
Pr: 0.00000292062759399414 Maska: 11111001
Pr: 0.00000292062759399414 Maska: 11111010
Pr: 0.00000041723251342773 Maska: 11111011
Pr: 0.00000292062759399414 Maska: 11111100
Pr: 0.00000041723251342773 Maska: 11111101
Pr: 0.00000041723251342773 Maska: 11111110
Pr: 0.00000005960464477539 Maska: 11111111
SolveRES: 0.03473842144012451172
ThRES: 0.03473842144012451172

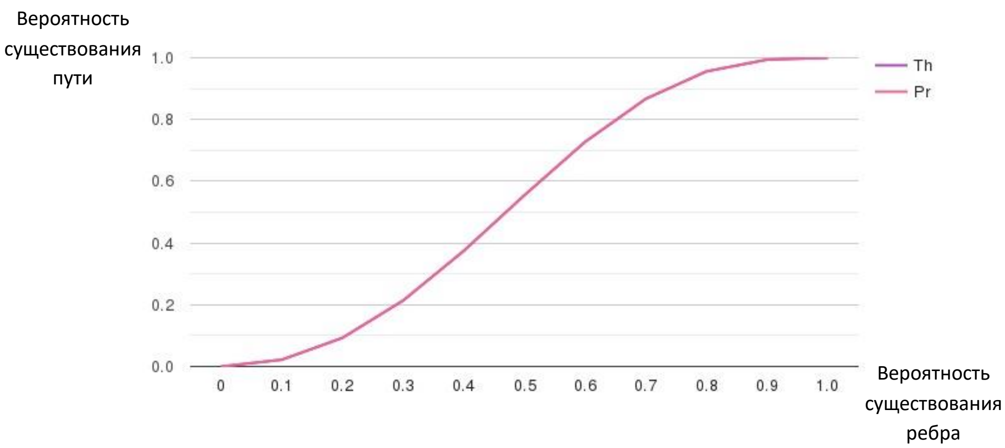
Результат работы программы для разных вероятностей существования ребра:


Консоль отладки Microsoft Visual Studio

```
Pr: 0
SolveRES:       0
ThRES:          0

Pr: 0.10000000149011611938
SolveRES:       0.02184238067524082310 7
ThRES:          0.02184238067524076412 6

Pr: 0.20000000298023223877
SolveRES:       0.09265408287719728575
ThRES:          0.09265408287719728575

Pr: 0.30000001192092895508
SolveRES:       0.21352159710740095666
ThRES:          0.21352159710740117871

Pr: 0.40000000596046447754
SolveRES:       0.37439489041503809519
ThRES:          0.37439489041503903888

Pr: 0.5
SolveRES:       0.5546875
ThRES:          0.5546875

Pr: 0.60000002384185791016
SolveRES:       0.72753411813720503254
ThRES:          0.72753411813720614276

Pr: 0.70000004768371582031
SolveRES:       0.8668992352341020613
ThRES:          0.86689923523410217232

Pr: 0.80000007152557373047
SolveRES:       0.95592452439940045483
ThRES:          0.9559245243994007879

Pr: 0.90000009536743164063
SolveRES:       0.99409519663946555035
ThRES:          0.9940951966394653283
```

Рисунок 2: результат

График сопоставления теоретического расчета вероятности существования пути и практического:



Вывод:

Результаты полного перебора и формулы сошлись с точностью до 20 знака после запятой минимум, что свидетельствует о правильности расчётов и высокой точности вычисления при использовании полного перебора

Листинг кода:

```
#include <stdio.h>
#include <iostream>
#include <vector>
#include <iostream>
#include <cmath>

#define graphVertices 7
#define edges 8
#define start 2
#define finish 5
#define combinations 256

using namespace std;

bool haveWay = false;
vector<double> probabilities;
int visited[graphVertices] = {0};
long double solveRES = 0;
double p = 0.125;

vector<vector<int>> createMtx(int mtx[edges][2], vector<int> maska)
{
    vector<vector<int>> res;
    int step = 0;
    for (vector<int>::iterator iter = maska.begin() ; iter != maska.end(); iter++,
step++)
    {
        if (*iter == 1)
        {
            vector<int> tmpE;

            tmpE.push_back(mtx[step][0]);
            tmpE.push_back(mtx[step][1]);

            res.push_back(tmpE);
        }
    }

    return res;
}

int** vectorToMtx(vector<vector<int>> vec)
{
    int** res = (int**)malloc(sizeof(int*) * vec.size());

    for (int i = 0; i < vec.size(); i++)
    {
        res[i] = (int*)malloc(sizeof(int) * 2);

        int tmp = vec[i][0];
        res[i][0] = tmp;
        tmp = vec[i][1];
        res[i][1] = tmp;
    }

    return res;
}

void dfs(int cur, vector<vector<int>> mtx)
{
    if (cur == finish)
    {
        haveWay = true;
```

```
    }

    if (haveWay == true)
    {
        return;
    }

    visited[cur] = 1;

    for (int i = 0; i < mtx.size(); i++)
    {
        if (haveWay == true) { return; }
        if (mtx[i][0] == cur && visited[mtx[i][1]] == 0)
        {
            dfs(mtx[i][1], mtx);
        }
        else if (mtx[i][1] == cur && visited[mtx[i][0]] == 0)
        {
            dfs(mtx[i][0], mtx);
        }
    }
}

void myDecToBin(int number, vector<int>* res)
{
    res->clear();

    while (number > 0)
    {
        vector<int>::iterator it = res->begin();
        res->insert(it, number % 2);
        number = number / 2;
    }

    while (res->size() < edges)
    {
        vector<int>::iterator it = res->begin();
        res->insert(it, 0);
    }
}

void zeriongVisited()
{
    for (int i = 0; i < graphVertices; i++)
    {
        visited[i] = 0;
    }
}

void countProbability(vector<vector<int>> mtx)
{
    double res;

    // Count uints ============================================================

    double tmpRes1 = pow(p, mtx.size());
    int nullTmp = edges - mtx.size();

    // Count zeros ============================================================

    double tmpRes2 = pow((1 - p), nullTmp);
    res = tmpRes1 * tmpRes2;

    probabilities.push_back(res);
}
```

```
void solve(int mtx[edges][2])
{
    for (int mask = 0; mask < combinations; mask++)
    {
        // Create params ======================================================

        vector<int> binMask;
        myDecToBin(mask, &binMask);
        vector<vector<int>> tmpMtx = createMtx(mtx, binMask);

        // Call DFS ==========================================================

        zeriongVisited();
        haveWay = false;
        dfs(start, tmpMtx);

        // Check result ======================================================

        if (haveWay == true)
        {
            countProbability(tmpMtx);
            cout << "\nPr: " << fixed << probabilities[probabilities.size() - 1];
            printf("\tMaska: ");
            for (int i = 0; i < binMask.size(); i++)
            {
                printf("%d", binMask[i]);
            }
        }
    }

    // Cout results ==========================================================

    for (int i = 0; i < probabilities.size(); i++)
    {
        solveRES += probabilities[i];
    }
    cout << "\nSolveRES:\t" << solveRES << endl;
}

int main()
{
    cout.precision(10);
    // Create Matrix =========================================================

    int mtx[edges][2];
    mtx[0][0] = 1;
    mtx[0][1] = 2;
    mtx[1][0] = 1;
    mtx[1][1] = 5;
    mtx[2][0] = 2;
    mtx[2][1] = 3;
    mtx[3][0] = 2;
    mtx[3][1] = 6;
    mtx[4][0] = 6;
    mtx[4][1] = 5;
    mtx[5][0] = 6;
    mtx[5][1] = 3;
    mtx[6][0] = 3;
    mtx[6][1] = 4;
    mtx[7][0] = 4;
    mtx[7][1] = 5;

    // Solving ===============================================================
```

```
    solve(mtx);

    // Theoretical Solution ================================================

    long double x1 = pow(p, 2) + pow(p, 2) - pow(p, 4);
    long double without = pow(p, 3) + x1 - x1 * pow(p, 3);
    long double x2 = (p + p - p * p) * (p + pow(p, 2) - pow(p, 3));
    long double with = pow(p, 2) + x2 - pow(p, 2) * x2;
    long double thRES = ((1 - p) * without) + (p * with);

    cout << "ThRES:\t\t" << thRES << endl;

    // Different probabilities =============================================

    /*for (long double q = 0.0; q <= 1.0; q = q + 0.1)
    {
        p = q;
        cout << "Pr: " << p;
        int mtx[edges][2];
        mtx[0][0] = 1;
        mtx[0][1] = 2;
        mtx[1][0] = 1;
        mtx[1][1] = 5;
        mtx[2][0] = 2;
        mtx[2][1] = 3;
        mtx[3][0] = 2;
        mtx[3][1] = 6;
        mtx[4][0] = 6;
        mtx[4][1] = 5;
        mtx[5][0] = 6;
        mtx[5][1] = 3;
        mtx[6][0] = 3;
        mtx[6][1] = 4;
        mtx[7][0] = 4;
        mtx[7][1] = 5;

        solve(mtx);

        long double r1 = pow(p, 2) + pow(p, 2) - pow(p, 4);
        long double r2 = pow(p, 3) + r1 - r1 * pow(p, 3);
        long double x = (p + p - p * p) * (p + pow(p, 2) - pow(p, 3));
        long double with = pow(p, 2) + x - pow(p, 2) * x;
        long double thRES = ((1 - p) * r2) + (p * with);
        cout  << "ThRES:\t\t" << thRES << "\n" <<endl;

        haveWay = false;
        probabilities.clear();
        zeriongVisited();
        solveRES = 0;
    }*/

    return 0;
}
```