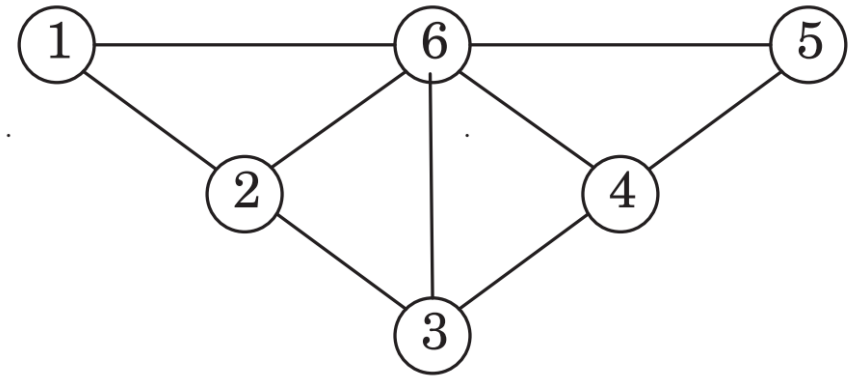
Цель работы: получение практических навыков оценки надежности вычислительных сетей.

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. Применение простого имитационного моделирования.
2. Уменьшение множества рассматриваемых при имитационном моделировании графов.



Описание программы

Программа выполняет вычисление вероятности наличия пути из 1 в 4 двумя способами: простым имитационным моделированием, при котором генерируются вектор с помощью распределения Бернулли и ускоренном варианте данного моделирования, при котором сразу определяются определённые варианты

Для $\varepsilon = 0.001$

Результаты работы программы:
Для формулы: 0, 0.013966183, 0.069577216, 0.177839829, 0.332941312, 0.513671875, 0.690835968, 0.837255601, 0.936484864, 0.986959647, 1
Для первого варианта: 0, 0.01351111111, 0.06968888889, 0.1766222222, 0.3352, 0.5112888889, 0.6891111111, 0.8364888889, 0.9363555556, 0.9860888889, 1
Для второго варианта: 0, 0.0136, 0.06742222222, 0.1737333333, 0.3312, 0.5176, 0.6909777778, 0.8337333333, 0.9367111111, 0.9861777778, 1

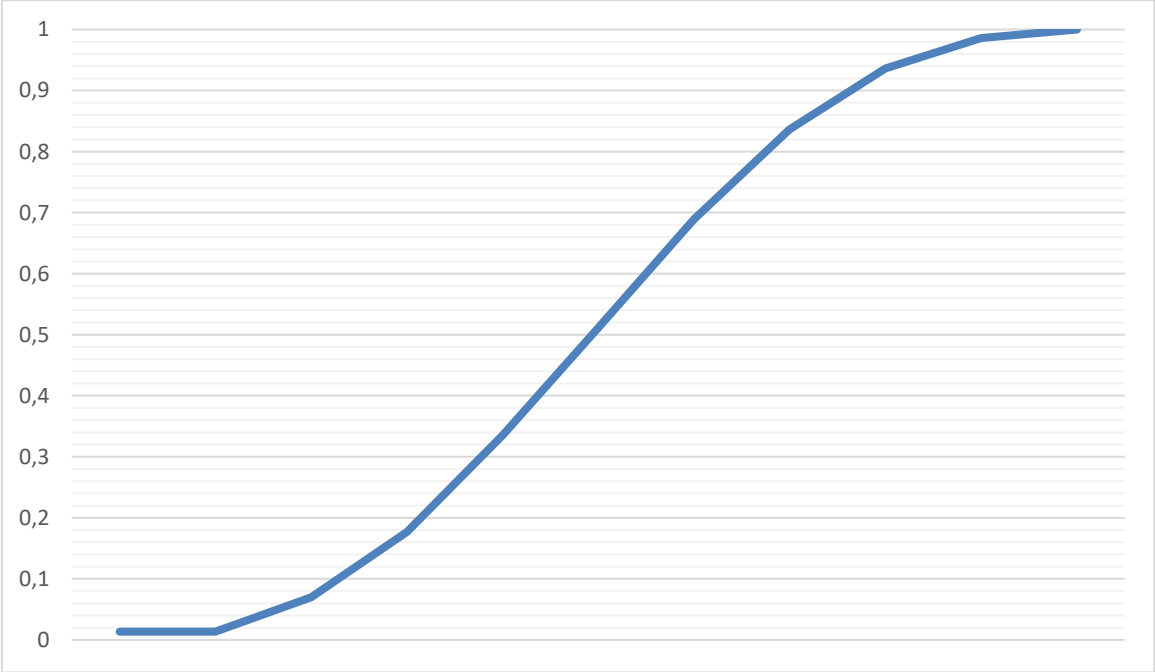


Рисунок 1 График вероятности для первого варианта

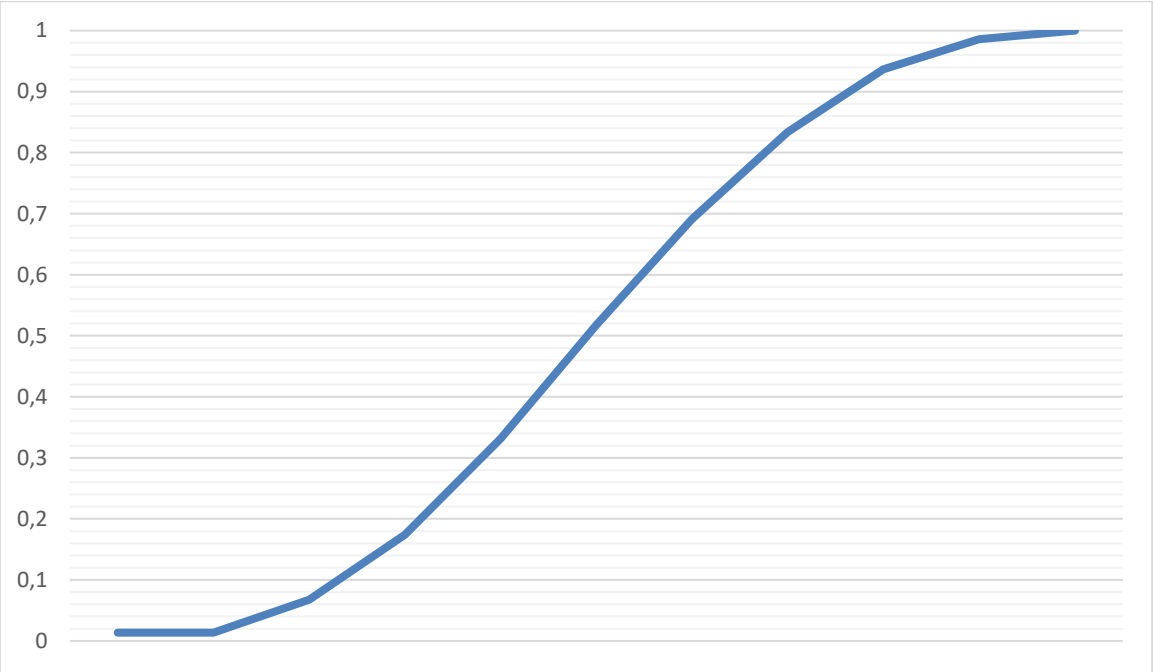


Рисунок 2 График вероятности для второго варианта

Выводы

Результаты первого, второго варианта и прошлой лабораторной работы сошлись с точностью ε, что свидетельствует о правильности расчётов и точности вычисления при использовании имитационного моделирования

Текст программы

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

#define graphVertices 7
#define edges 9
#define start 1
#define finish 4
#define combinations 512

using namespace std;

bool haveWay = false;
vector<double> probabilities;
int visited[graphVertices] = { 0 };
long double solveRES = 0;
double p = 0.111;
double eps = 0.01;
int N = (9 * 0.25) / (eps * eps);
int Nhw = 0;
double resLab1;
double resLab2Part1;
double resLab2Part2;
int lenMax = edges - 2;
int lenMin = 2;
int NPart2 = 0;

vector<int> generateMask()
{
	vector<int> res;
	for (int i = 0; i < edges; i++)
	{
		double g = (double)(rand()) / RAND_MAX;
		if (g > p)
		{
			res.push_back(0);
		}
		else
		{
			res.push_back(1);
		}
	}

	return res;
}

int weightMask(vector<int> mask)
{
	int res = 0;

	for (auto iter = mask.begin(); iter != mask.end(); iter++)
	{
		if (*iter == 1) res++;
	}

	return res;
}

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 solveLab1(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;
}

void solveLab2Part1(int mtx[edges][2])
{
	Nhw = 0;
	for (int i = 0; i < N; i++)
```

```
	{

		// Генерация маски
================================================================================
======

		vector<int> binMask = generateMask();

		// Создание списка ребер
================================================================================

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

		// Определение наличия пути в подграфе
========================================================================

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

		if (haveWay == true)
		{
			Nhw++;
		}
	}

	resLab2Part1 = (double)Nhw / N;

}

void solveLab2Part2(int mtx[edges][2])
{
	Nhw = 0;
	NPart2 = 0;
	for (int i = 0; i < N; i++)
	{

		// Генерация маски
================================================================================

		vector<int> binMask = generateMask();

		// Определение веса
================================================================================

		int w = weightMask(binMask);

		// Оптимизация
================================================================================
===

		if (w > lenMax)
		{
			Nhw++;
			continue;
		}
		else if (w < lenMin)
		{
			continue;
		}
		else
		{
			NPart2++;
			// Создание списка ребер
========================================================================
```

```
                        vector<vector<int>> tmpMtx = createMtx(mtx, binMask);

                        // Определение наличия пути в подграфе
=============================================================

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

                        if (haveWay == true)
                        {
                                Nhw++;
                        }
                }
        }

        resLab2Part2 = (double)Nhw / N;
}

int main()
{
        // Init random
=================================================================================
==========

        srand((unsigned int)time(0));
        cout.precision(10);
        cout << "All N:   " << N << endl;

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

        for (p = 0.0; p <= 1.0; p = p + 0.1)
        {

                int mtx1[edges][2];
                mtx1[0][0] = 1;
                mtx1[0][1] = 2;
                mtx1[1][0] = 1;
                mtx1[1][1] = 6;
                mtx1[2][0] = 2;
                mtx1[2][1] = 6;
                mtx1[3][0] = 6;
                mtx1[3][1] = 5;
                mtx1[4][0] = 2;
                mtx1[4][1] = 3;
                mtx1[5][0] = 6;
                mtx1[5][1] = 3;
                mtx1[6][0] = 6;
                mtx1[6][1] = 4;
                mtx1[7][0] = 3;
                mtx1[7][1] = 4;
                mtx1[8][0] = 4;
                mtx1[8][1] = 5;

                solveLab2Part1(mtx1); //Имитационное моделирование
                solveLab2Part2(mtx1); //Ускорение имитационного моделирования

                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;

            cout << "Pr: " << p << endl;
            cout << "Lab1:     \t\t" << a << endl;
            cout << "Lab2Part1:\t\t" << resLab2Part1 << endl;
            cout << "Lab2Part2:\t\t" << resLab2Part2 << endl;
            cout << "N:        \t\t" << NPart2 << endl;
            cout << "win:      \t\t" << (double)N / NPart2 << endl;
            haveWay = false;
            NPart2 = 0;
            Nhw = 0;
            probabilities.clear();
            zeriongVisited();
            solveRES = 0;
        }

        return 0;
}
```