**Цель работы**: получение практических навыков использования моделирования для оценки надежности вычислительных сетей.

**Задание:**

Вычислить, используя имитационное моделирование, оценку вероятности связности пары вершин в случайном графе Ğ(X,Y,P), где X={xi} – множество вершин, Y={(xi,xj)} – множество ребер, P={pi} – множество вероятностей существования ребер. Вероятности существования ребер равны между собой и равны p.

**Исходные данные:** вариант 11, e =0.01, $l_{min} = 2,\ l_{max} = 6$.

Необходимы вычислить вероятность пути из вершины 3 в вершину 5 в графе, представленном на рисунке ниже (рисунок 1)

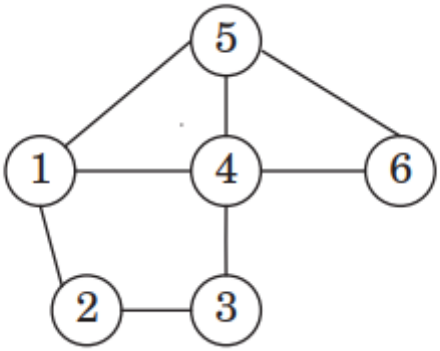


*Рисунок 1. Исходный граф.*

**Ход работы**

1) Необходимо написать программу обычного и ускоренного имитационного моделирования и вычислить вероятность существования пути с заданной точностью

Задача программы состоит в том, чтобы случайным образом создать ребра в графе и если в этом графе существует путь, то увеличить счетчик существования пути. После считается отношение «удачных» экспериментов к их общему числу

В программе граф задаётся в виде списка смежности. Перебор осуществляется по всем вероятностям существования рёбер от 0 до 1 с шагом

0.1 и всем подграфам, количество которых $2^L$, где L – количество рёбер. Для нахождения пути в подграфе используется алгоритм DFS.

Также создается подграф случайным образом и в нем происходит поиск нахождения пути с помощью алгоритма DFS

Результат работы программы приведен на рисунке ниже (рисунок 2).

```
p= 0.0
practical= 0,00000
Pr = 0,00000
Pr2 = 0,00000

p= 0.1
practical= 0,01286
Pr = 0,01298
Pr2 = 0,01298

p= 0.2
practical= 0,06111
Pr = 0,06009
Pr2 = 0,06009

p= 0.3
practical= 0,15341
Pr = 0,15142
Pr2 = 0,15142

p= 0.4
practical= 0,28823
Pr = 0,28667
Pr2 = 0,28667

p= 0.5
practical= 0,45312
Pr = 0,45040
Pr2 = 0,45040

p= 0.6
practical= 0,62708
Pr = 0,62729
Pr2 = 0,62729

p= 0.7
practical= 0,78545
Pr = 0,78391
Pr2 = 0,78391

p= 0.8
practical= 0,90657
Pr = 0,90347
Pr2 = 0,90347

p= 0.9
practical= 0,97805
Pr = 0,97938
Pr2 = 0,97938

p= 1.0
practical= 1,00000
Pr = 1,00000
Pr2 = 1,00000
```

*Рисунок 2. Результат работы программы*

На рисунках ниже представлены блок схемы алгоритмов имитационного моделирования: обычного и ускоренного графиков (рисунок 3,4).

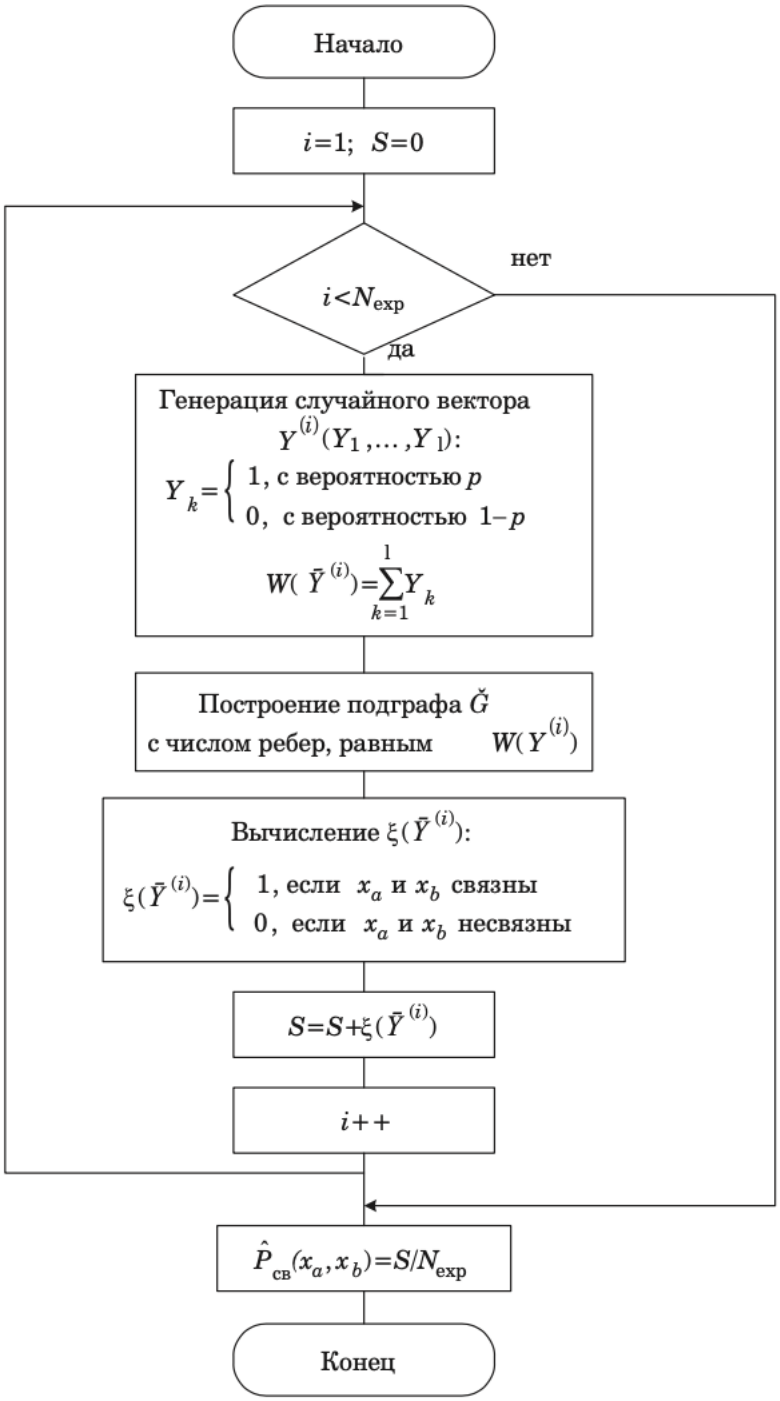


*Рисунок 3. Блок-схема имитационного моделирования*

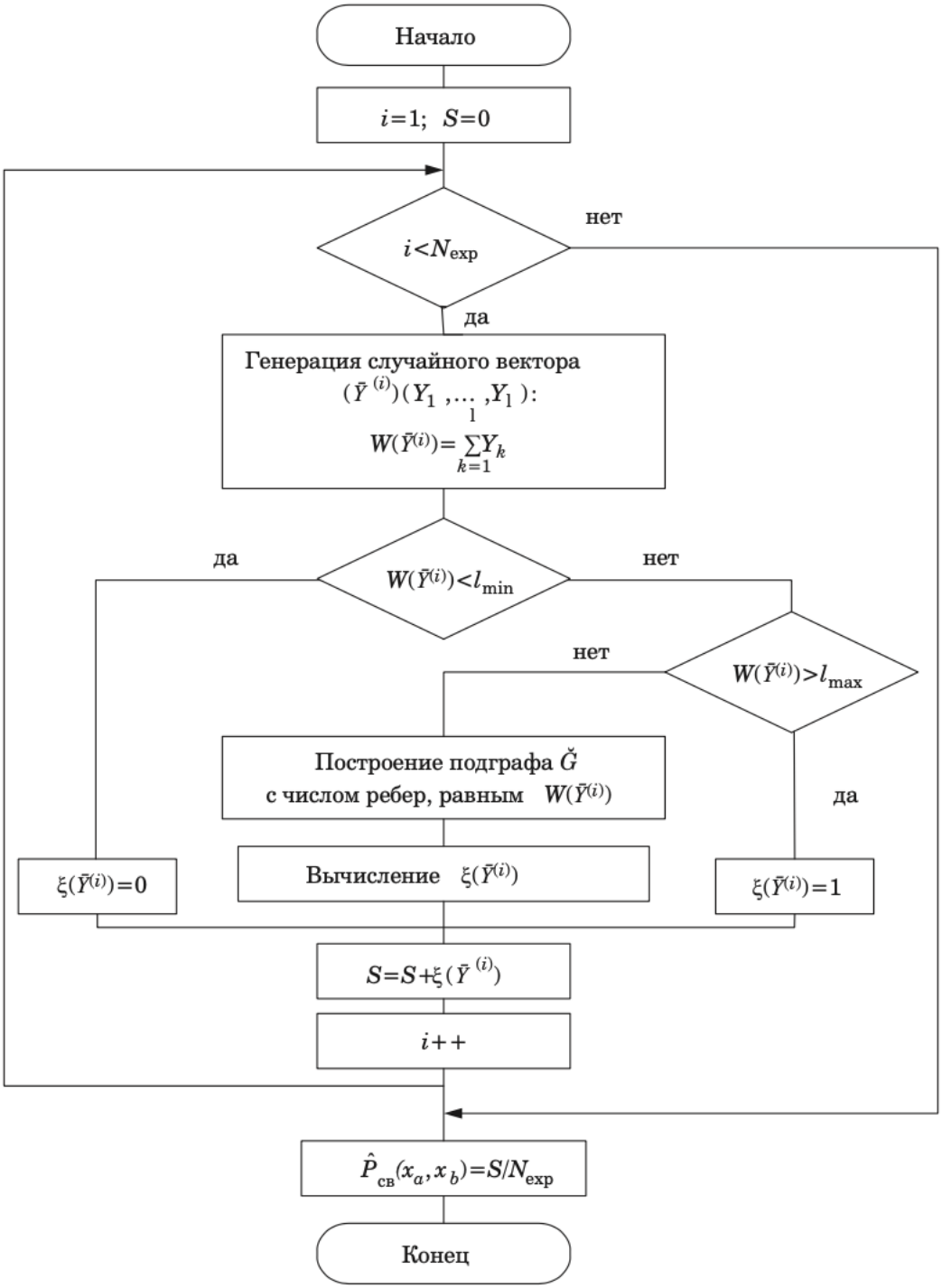


*Рисунок 4. Блок-схема ускоренного имитационного моделирования*

Графики с результатами работы программы представлены ниже (рисунок 5,6,7).

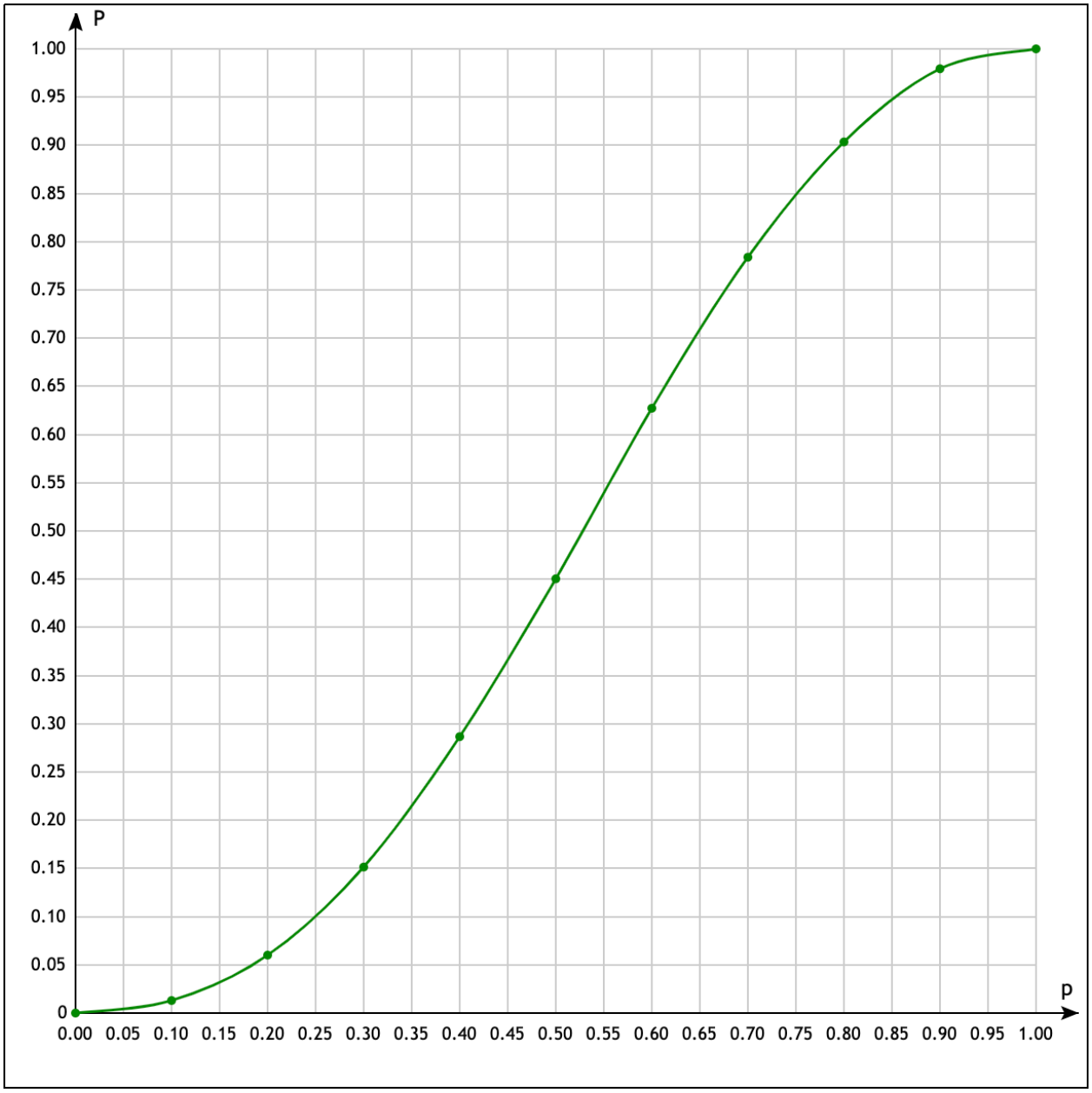


*Рисунок 5 График вероятности существования пути при имитационном моделировании.*

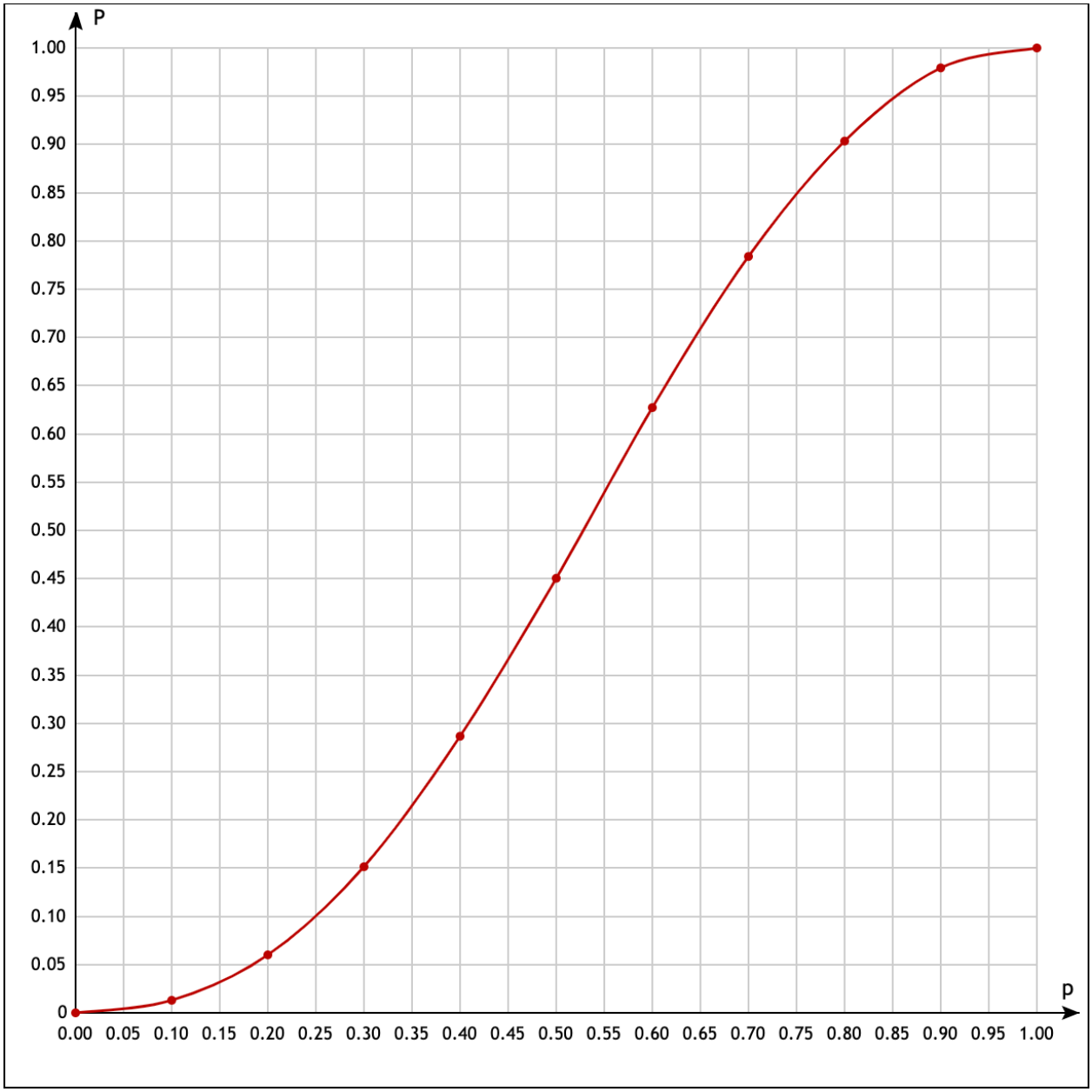


*Рисунок 6 График вероятности существования пути при ускоренном алгоритме имитационного моделирования.*

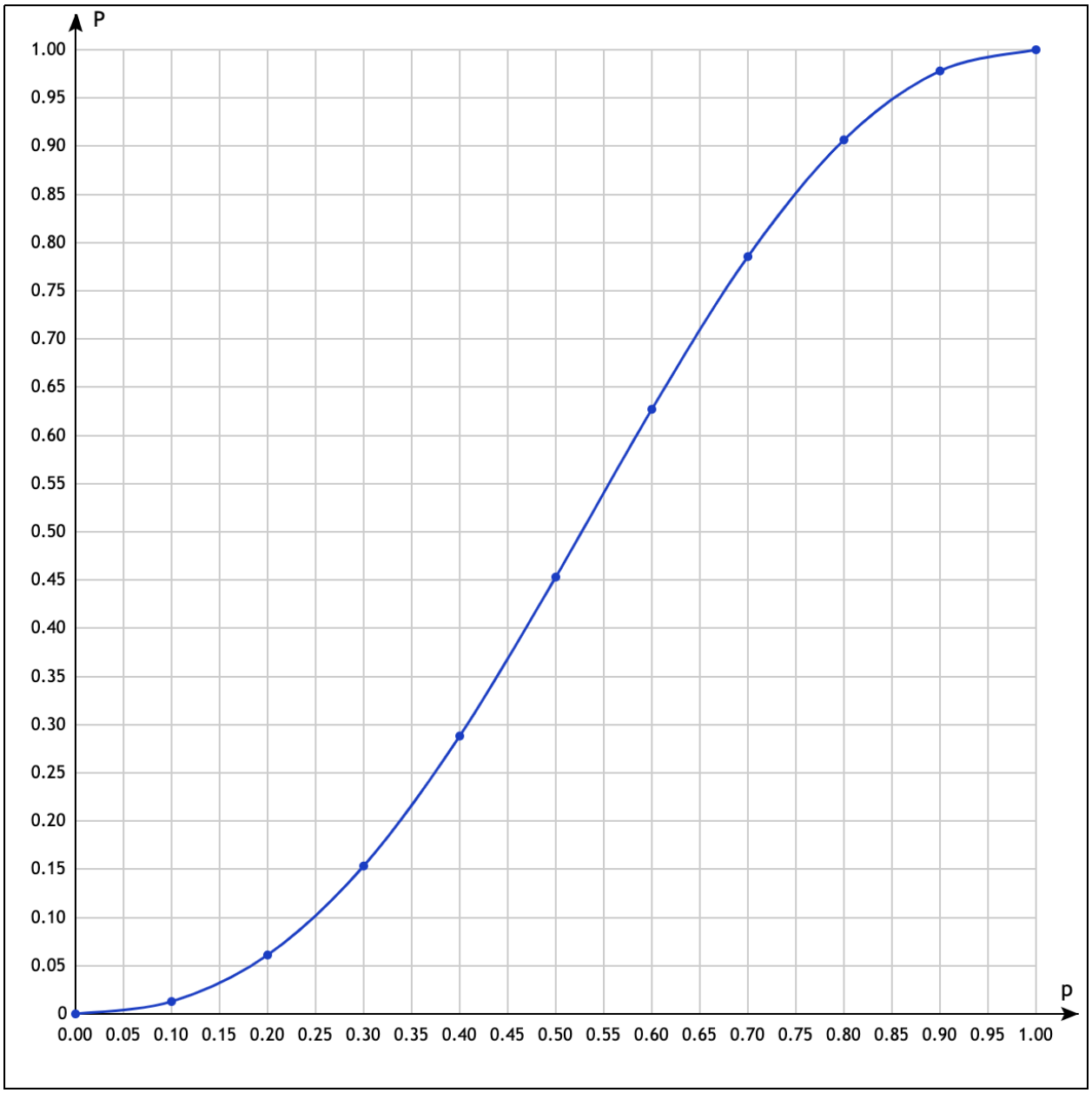


*Рисунок 7 График вероятности существования пути при полном переборе.*

Также был получен график времени работы от вероятности существования ребра при имитационном моделировании и ускоренном алгоритме имитационного моделирования (рисунок 8).

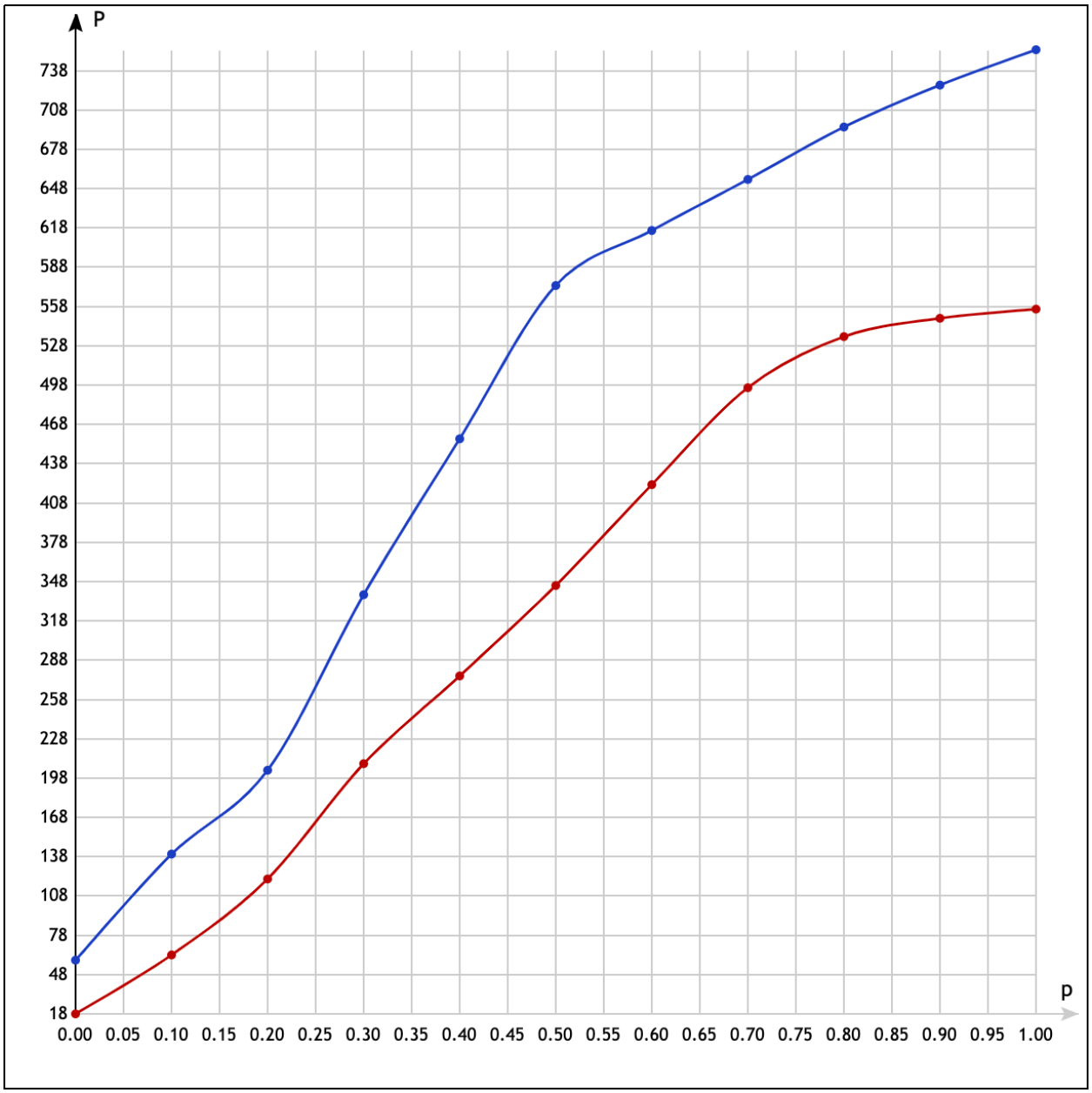


*Рисунок 8 График времени работы имитационного алгоритма (синий) и ускоренного имитационного алгоритма (красный)*

Из графика можно сделать вывод, что ускоренный алгоритм работает быстрее обычного имитационного алгоритма.

На рисунке ниже представлен график выигрыша алгоритма ускоренного имитационного моделирования в зависимости от вероятности существования ребра (рисунок 9).

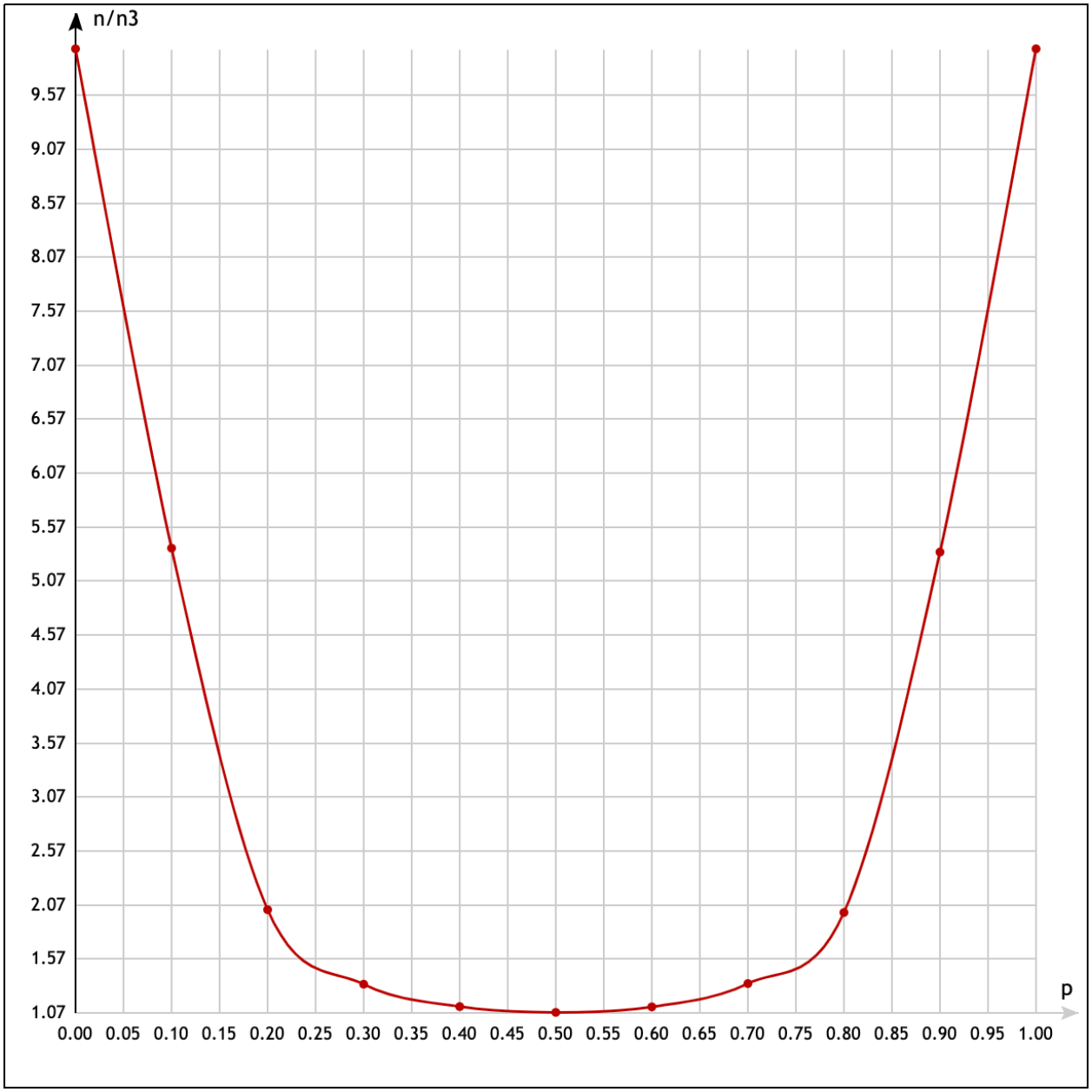


*Рисунок 9 График зависимости выигрыша ускоренного имитационного алгоритма от вероятности существования ребра.*

**Выводы:**

В ходе выполнения лабораторной работы, были получены практические навыки использования моделирования для оценки надежности вычислительных сетей, была вычислена вероятность существования пути в случайном графе, как функции от p, построена программа для полного перебора графа и для случайно созданного подграфа.

## Листинг

```
import java.text.DecimalFormat;
import java.util.LinkedList;

public class Main {
public static void main(String args[]) {
        double p;
int per = 1;
int reb = 8;
int bin_code[] = new int[100];
int bin_code1[] = new int[100];
int flag ;
double sum ;
double [] arrP = new double[11];
        arrP[0]=0.0;
        arrP[1]=0.1;
        arrP[2]=0.2;
        arrP[3]=0.3;
        arrP[4]=0.4;
        arrP[5]=0.5;
        arrP[6]=0.6;
        arrP[7]=0.7;
        arrP[8]=0.8;
        arrP[9]=0.9;
        arrP[10]=1;


        int s = 2, d = 4;
        Graph g = new Graph(6);

        g.addEdge(0, 1);
        g.addEdge(0, 3);
        g.addEdge(0, 4);

        g.addEdge(1, 0);
        g.addEdge(1, 2);

        g.addEdge(2, 1);
        g.addEdge(2, 3);



        g.addEdge(3, 0);
        g.addEdge(3, 2);
        g.addEdge(3, 4);
        g.addEdge(3, 5);

        g.addEdge(4, 0);
        g.addEdge(4, 3);
        g.addEdge(4, 5);


        g.addEdge(5, 3);
        g.addEdge(5, 4);

        int nemM [][] = g.getMatrix();
        int n = 0;
        int N = 0;
        int N2 = 0;
        int min = 2  ;
        int max = 6 ;
        double e = 0.01; // ввод точности;
        n =  (int)(2.25/ Math.pow(e,2));
```

```
		long start = System.currentTimeMillis();


		for (int k = 0; k <= 10 ; k++) {
			N = 0;
			N2 = 0;

			p = arrP[k];
			sum = 0.0;
			flag = 0;
			for (int i = 0; i < reb; i++) {
				bin_code1[i] = 0;
			}


		       // int nemM [][] = g.getMatrix();
			while (flag != Math.pow(2, reb) - 1) {

				per = 1;
				for (int i = reb - 1; i >= 0; i--) {
					if (per == 1) {
						if (bin_code1[i] == 1) {
							bin_code1[i] = 0;
						} else {
							bin_code1[i] = 1;
							per = 0;
						}
					}
				}
				flag++;


				int countNew = 0;
				Graph tmp = g.getTmpGraph(nemM, bin_code1);
				if (tmp.DFS(s, d) == 1) {
					countNew = 0;
					for (int i = reb - 1; i >= 0; i--) {
						if (bin_code1[i] == 1) {
							countNew++;
						}
					}
					sum += Math.pow(p, countNew) * Math.pow(1 -
p, 8 - countNew);
				}
			}

			for (int m = 0 ;m < n; m++) {

				for (int j = 0; j <= (reb - 1); j++) {
					double ver = Math.random();
					if (ver <= p) {
						bin_code[j] = 1;
					} else if (ver > p) {
						bin_code[j] = 0;
					}
				}

				int count = 0;

				for (int i = (reb - 1); i >= 0; i--) {
					if (bin_code[i] == 1) {
						count++;
					}
```

```
                        }

                        if( count > max){
                               N2++;
                        }
                        else if (count >= min && count <=max){
                                Graph tmp = g.getTmpGraph(nemM, bin_code);
                                if (tmp.DFS(s, d) == 1) {
                                        N2++;
                                }
                        }


                        Graph tmp = g.getTmpGraph(nemM, bin_code);
                        if (tmp.DFS(s, d) == 1) {
                                N++;
                        }
                }



                double Pr = (double) (N)/(double) n;
                double Pr2 = (double) (N2)/(double) n;
                long finish = System.currentTimeMillis();
                long elapsed = finish - start;

                double tmp2 = -2 * Math.pow(p, 8) + 5 * Math.pow(p, 7) -
Math.pow(p, 6) - 4 * Math.pow(p, 5) - Math.pow(p, 4) + 3 * Math.pow(p, 3) +
Math.pow(p, 2);
                String formattedDouble = new
DecimalFormat("#0.0000").format(p);
                String formattedDouble2 = new
DecimalFormat("#0.00000").format(tmp2);
                System.out.println("p= "+p);

                //  System.out.println("theoretical = " + formattedDouble2);
                String formattedDouble3 = new
DecimalFormat("#0.00000").format(sum);
                System.out.println("practical= "+formattedDouble3);

                String formattedDouble4 = new
DecimalFormat("#0.00000").format(Pr);
                String formattedDouble5 = new
DecimalFormat("#0.00000").format(Pr2);
                System.out.println("Pr = "+formattedDouble4);
                System.out.println("Pr2 = "+formattedDouble5);
                System.out.println();

               // System.out.println("time = "+elapsed);



        }
    }
}
```