1) Цель работы

С помощью имитационного моделирования определить вероятность существования пути при различных вероятностях существования ребер, сравнить с результатами полного перебора, построить графики.

2) Вариант задания: **12**

Задан случайный граф ***G(X,Y,P)***, где $\boldsymbol{X = \{x_i\}}$ – множество вершин, $\boldsymbol{Y = \{(x_i, x_j)\}}$ – множество ребер, $\boldsymbol{P = \{p_i\}}$ – множество вероятностей существования ребер. Вероятности существования ребер равны между собой и равны ***p***.

В ходе выполнения лабораторной работы необходимо написать программу обычного и ускоренного имитационного моделирования и вычислить вероятность существования пути с заданной точностью $\boldsymbol{\varepsilon = 0.01}$.

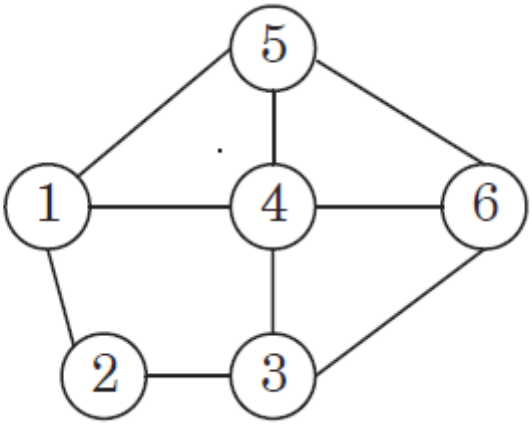

Рис. 1 – Топология исходного графа

3) Описание программы

Программа выполняет вычисление вероятности наличия искомого пути четырьмя способами: полным перебором для точного результата, методом имитационного моделирования, ускоренным методом имитационного моделирования, а также методом имитационного моделирования на j-ном слое.

Точность вычисления для всех пунктов $\boldsymbol{\varepsilon = 0.01}$. В качестве значений $\boldsymbol{l_{min}}$ и $\boldsymbol{l_{max}}$ для третьего варианта подсчета были приняты $\boldsymbol{l_{min} = 2}$, $\boldsymbol{l_{max} = 7}$.

Листинг исходного кода представлен в Приложении 1.

На *рисунках 2* выведены результаты работы нескольких запусков программы с одинаковыми параметрами, причем первая строка отражает вероятность существования ребра ***p*** для всех вычислений в данном столбце. В последующих строках выведены результаты подсчета по каждому из четырех алгоритмов.

На *рисунках 3* показаны распределения количества экспериментов в зависимости от слоя ***j*** и вероятности существования ребра ***p***.

0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.021853231294	0.092809740447	0.213831179093	0.373805145963	0.55078125	0.718078500733	0.852680511901	0.942293039493	0.987836011374	1
0	0.022311111111	0.093066666667	0.217466666667	0.378177777778	0.551822222222	0.713555555556	0.853155555556	0.943333333333	0.9876	1
0	0.023511111111	0.0936	0.219733333333	0.376133333333	0.554933333333	0.723688888889	0.852044444444	0.941644444444	0.989422222222	1
0	0.020844444444	0.092	0.215777777778	0.372044444444	0.548933333333	0.718844444444	0.852266666667	0.942844444444	0.987866666667	1

0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.021853231294	0.092809740447	0.213831179093	0.373805145963	0.55078125	0.718078500733	0.852680511901	0.942293039493	0.987836011374	1
0	0.021955555556	0.091111111111	0.2192	0.371955555556	0.555111111111	0.721688888889	0.8536	0.941377777778	0.988177777778	1
0	0.020888888889	0.093733333333	0.212222222222	0.373066666667	0.548622222222	0.715066666667	0.854755555556	0.945688888889	0.986844444444	1
0	0.021822222222	0.0928	0.210088888889	0.373955555556	0.549866666667	0.720711111111	0.854044444444	0.941066666667	0.987911111111	1

0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.021853231294	0.092809740447	0.213831179093	0.373805145963	0.55078125	0.718078500733	0.852680511901	0.942293039493	0.987836011374	1
0	0.021955555556	0.091377777778	0.213733333333	0.375911111111	0.555422222222	0.722577777778	0.852266666667	0.941644444444	0.987555555556	1
0	0.020177777778	0.093422222222	0.217333333333	0.376088888889	0.547466666667	0.725822222222	0.849422222222	0.943377777778	0.989111111111	1
0	0.021066666667	0.093466666667	0.211022222222	0.372177777778	0.551288888889	0.715866666667	0.847288888889	0.942044444444	0.9888	1

Рис. 2 – Результат работы программы

```
P = 0
j = 1 n_j = 0
j = 2 n_j = 0
j = 3 n_j = 0
j = 4 n_j = 0
j = 5 n_j = 0
j = 6 n_j = 0
j = 7 n_j = 0
j = 8 n_j = 0
j = 9 n_j = 0

P = 0.1
j = 1 n_j = 8716
j = 2 n_j = 3874
j = 3 n_j = 1004
j = 4 n_j = 167
j = 5 n_j = 18
j = 6 n_j = 1
j = 7 n_j = 0
j = 8 n_j = 0
j = 9 n_j = 0

P = 0.2
j = 1 n_j = 6794
j = 2 n_j = 6794
j = 3 n_j = 3963
j = 4 n_j = 1486
j = 5 n_j = 371
j = 6 n_j = 61
j = 7 n_j = 6
j = 8 n_j = 0
j = 9 n_j = 0
```

```
P = 0.3
j = 1 n_j = 3502
j = 2 n_j = 6003
j = 3 n_j = 6003
j = 4 n_j = 3859
j = 5 n_j = 1654
j = 6 n_j = 472
j = 7 n_j = 86
j = 8 n_j = 9
j = 9 n_j = 0

P = 0.4
j = 1 n_j = 1360
j = 2 n_j = 3627
j = 3 n_j = 5643
j = 4 n_j = 5643
j = 5 n_j = 3762
j = 6 n_j = 1672
j = 7 n_j = 477
j = 8 n_j = 79
j = 9 n_j = 5

P = 0.5
j = 1 n_j = 395
j = 2 n_j = 1582
j = 3 n_j = 3691
j = 4 n_j = 5537
j = 5 n_j = 5537
j = 6 n_j = 3691
j = 7 n_j = 1582
j = 8 n_j = 395
j = 9 n_j = 43
```

```
P = 0.6
j = 1 n_j = 79
j = 2 n_j = 477
j = 3 n_j = 1672
j = 4 n_j = 3762
j = 5 n_j = 5643
j = 6 n_j = 5643
j = 7 n_j = 3627
j = 8 n_j = 1360
j = 9 n_j = 226

P = 0.7
j = 1 n_j = 9
j = 2 n_j = 86
j = 3 n_j = 472
j = 4 n_j = 1654
j = 5 n_j = 3859
j = 6 n_j = 6003
j = 7 n_j = 6003
j = 8 n_j = 3502
j = 9 n_j = 907

P = 0.8
j = 1 n_j = 0
j = 2 n_j = 6
j = 3 n_j = 61
j = 4 n_j = 371
j = 5 n_j = 1486
j = 6 n_j = 3963
j = 7 n_j = 6794
j = 8 n_j = 6794
j = 9 n_j = 3019
```

```
P = 0.9
j = 1 n_j = 0
j = 2 n_j = 0
j = 3 n_j = 1
j = 4 n_j = 18
j = 5 n_j = 167
j = 6 n_j = 1004
j = 7 n_j = 3874
j = 8 n_j = 8716
j = 9 n_j = 8716

P = 1
j = 1 n_j = 0
j = 2 n_j = 0
j = 3 n_j = 0
j = 4 n_j = 0
j = 5 n_j = 0
j = 6 n_j = 0
j = 7 n_j = 0
j = 8 n_j = 0
j = 9 n_j = 22500
```

Рис. 3 – Количество опытов при методе имитационного моделирования в j-м слое

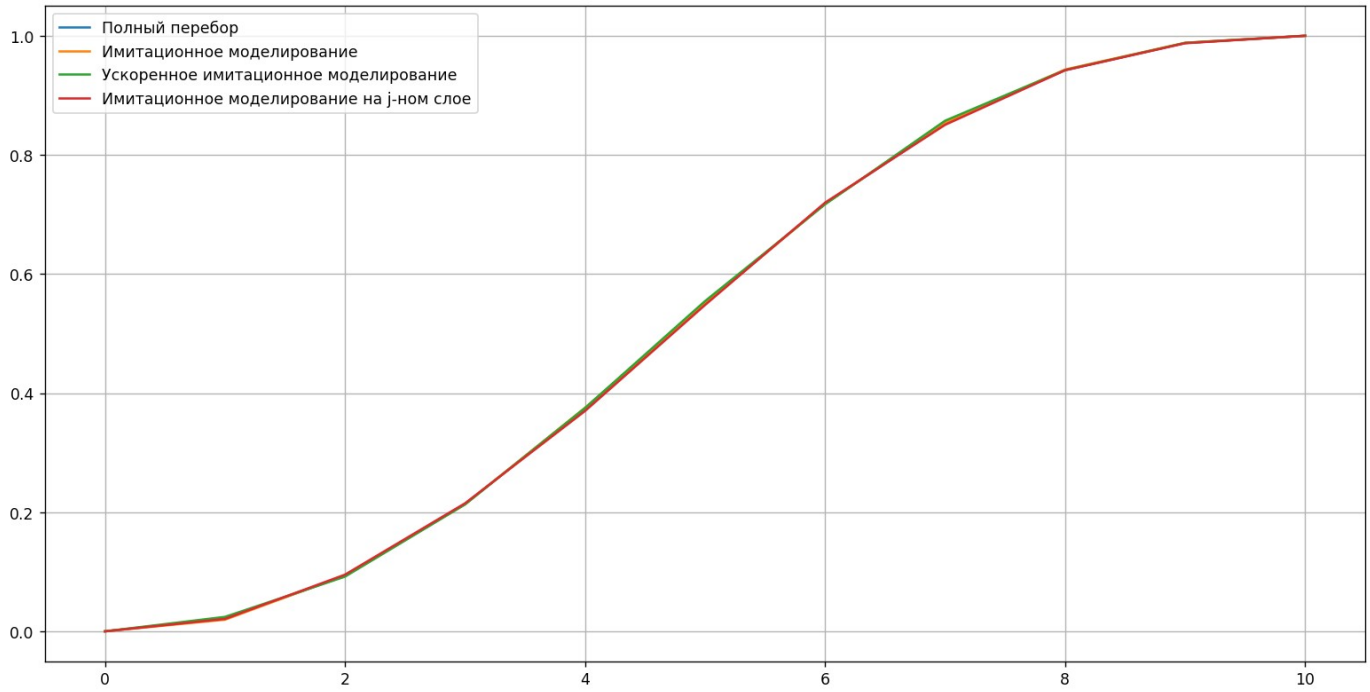

Рис. 4 – Графики зависимости наличия искомого пути от вероятности ***p*** при различных алгоритмах

5) Вывод

В ходе лабораторной работы были вычислены вероятности существования пути методом имитационного моделирования, ускоренным методом имитационного моделирования, а также методом имитационного моделирования на j-ном слое. Вероятности существования путей, как правило, совпадают с заданной точностью, однако результат имитационное моделирование может иметь несущественное отклонение от заданной точноси.

Листинг исходного кода на языке

```
#import "Basic";
#import "PCG";
#import "Math";

#load "../../../Some staff/Handy.jai";
#load "../../../Some staff/Math.jai";
#load "../../../Some staff/Arrays.jai";

Graph :: struct {
 Edge :: struct { to : int; p : float = .5; fake := false; }

 edges : [..][..]Edge;
 edge_count := 0;

 make :: (in : [][]Edge, mask : s64 = -1) -> Graph {
        g : Graph;

        array_reserve(*g.edges, in.count);
        for i: 0..in.count-1 {
                array_add(*g.edges, new_d_array(in[i]));
                for ii : 0..g.edges[i].count-1
                        if g.edges[i][ii].to > i g.edge_count += 1;
        }

        if mask != -1 set_fakes(*g, mask);
        return g;
}

 copy :: (in : Graph) -> Graph {
        g : Graph;

        array_reserve(*g.edges, in.edges.count);
        for in.edges {
                t : [..]Edge;
                array_add(*g.edges, t);
                array_copy(*g.edges[it_index], it);
        }
        return g;
}
```

```
set_fakes :: (in : *Graph, mask : s64) {
	for in.edges for *it it.fake = false;

	m := 1;
	i := 0; ii := 0;
	for 1..in.edge_count {
		if !(m & mask) {
			in.edges[i][ii].fake = true;
			for *in.edges[in.edges[i][ii].to] if it.to == i { it.fake = true; break; }
		}
		m <<= 1; ii += 1;
		if ii >= in.edges[i].count {
			i += 1; ii = 0;
			while ii < in.edges[i].count && in.edges[i][ii].to <= i ii += 1;
		}
	}
}

has_way :: (g : Graph, cur : int, to : int, vis : []bool) -> bool {
	if cur == to return true;

	vis[cur] = true; defer vis[cur] = false;
	out := false;
	for g.edges[cur]
		if !it.fake && !vis[it.to] out |= #this(g, it.to, to, vis);

	return out;
}
}

operator == :: inline (a : Graph.Edge, b : Graph.Edge) -> bool {
return a.to == b.to && a.p == b.p;
}

v12 :: (mask := -1) -> Graph {
return Graph.make(.[
	.[.{1}, .{3}, .{4}],
	.[.{0}, .{2}],
	.[.{1}, .{3}, .{5}],
	.[.{0}, .{2}, .{4}, .{5}],
	.[.{0}, .{3}, .{5}],
	.[.{2}, .{3}, .{4}]
], mask);
}
```

```
print_with_tabs :: inline (ar : []float64) {
 for ar {
		w := 12;
		ff : FormatFloat;
		ff.value = it;
		ff.width = w;
		ff.trailing_width = w;
		//ff.zero_removal = .NO;
		print("%\t", ff);
 }
 print("\n");
}

part_0 :: (vn : (mask := -1)->(Graph), from : int, to : int) {
 g := vn();
 vis := lf(NewArray(g.edges.count, bool));
 p_out := NewArray(11, float64);

 max := 1 << g.edge_count;
 n := 0; while n < max { defer n += 1;
		Graph.set_fakes(*g, n);

		if Graph.has_way(g, from, to, vis) {
			p : float = .0; while p < 1.1 { defer p += .1;
				pt : float64 = 1.;
				m := 1;for 1..g.edge_count { defer m <<= 1;
					if m & n pt *= p;
					else pt *= 1. - p;
				}
				p_out[cast(s64)(p * 10)] += pt;
			}
		}
 }
 //print("%\n", p_out);
 print_with_tabs(p_out);
}

part_1 :: (vn : (mask := -1)->(Graph), from : int, to : int, n_exp : int) {
 g := vn();
 vis := lf(NewArray(g.edges.count, bool));

 p_out := lf(NewArray(11, float64));
```

```
	p := .0; while p < 1.1{ defer p += .1;
		for 1..n_exp {
			mask := 0;
			for 1..g.edge_count mask = (mask << 1) | xx(random_get_within_range(.0, 1.) <= p);

			Graph.set_fakes(*g, mask);
			if Graph.has_way(g, 1, 3, vis) p_out[cast(s64)(p * 10)] += 1;
		}
		p_out[cast(s64)(p * 10)] /= n_exp;
	}
	//print("%\n", p_out);
	print_with_tabs(p_out);
	}

	part_2 :: (vn : (mask := -1)->(Graph), from : int, to : int, n_exp : int, L_min : int, L_max : int) {
	g := vn();
	vis := lf(NewArray(g.edges.count, bool));

	p_out := lf(NewArray(11, float64));

	p := .0; while p < 1.1{ defer p += .1;
		for i: 1..n_exp {
			mask := 0; w := 0;
			for ii: 1..g.edge_count {
				r := cast(s64)(random_get_within_range(.0, 1.) <= p);
				w += r;
				if g.edge_count - ii + w < L_min continue i;
				if w > L_max { p_out[cast(s64)(p * 10)] += 1; continue i; }
				mask = (mask << 1) | r;
			}

			Graph.set_fakes(*g, mask);
			if Graph.has_way(g, 1, 3, vis) p_out[cast(s64)(p * 10)] += 1;
		}
		p_out[cast(s64)(p * 10)] /= n_exp;
	}
	//print("%\n", p_out);
	print_with_tabs(p_out);
	}
```

```
part_3 :: (vn : (mask := -1)->(Graph), from : int, to : int, n_exp : int) {
 g := vn();
 vis := lf(NewArray(g.edges.count, bool));

 p_out := lf(NewArray(11, float64));

 p := .0; while p < 1.1 { defer p += .1;
        p_js := lf(NewArray(g.edge_count + 1, float64));

        print("P = %\n", p);
        for j : 1..g.edge_count {
                pi_j := c_n_k(g.edge_count, j) * pow(p, xx j) * pow(1. - p, xx (g.edge_count
- j));
                n_j := n_exp * pi_j;

                print("j = % n_j = %\n", j, n_j);

                for 1..cast(s64)n_j {
                        mask := 0;
                        n := 0; while n < j {
                                pos := 1 << (random_get() % g.edge_count);
                                if !(pos & mask) { mask |= pos; n += 1; }
                        }

                        Graph.set_fakes(*g, mask);
                        if Graph.has_way(g, 1, 3, vis) p_js[j] += 1;
                }
                p_js[j] /= n_j;
                p_js[j] *= pi_j;
        }
        print("\n");

        for p_js p_out[cast(s64)(p * 10)] += it;
 }
 p_out[0] = 0; p_out[10] = 1;
 //print("%\n", p_out);
 print_with_tabs(p_out);
}

main :: () {
 random_seed(xx(to_milliseconds(current_time_consensus()) % (1 << 32)));

 vn := v12;
 eps := .01;
```

```
from, to, n_exp := 1, 3, cast(s64)(2.25 / (eps * eps));

part_0(vn, from, to);
part_1(vn, from, to, n_exp);
part_2(vn, from, to, n_exp, 2, 7);
part_3(vn, from, to, n_exp);
}
```