ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»

(СПБГУТ)

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ СЕТЕЙ И СИСТЕМ (ИКСС)

КАФЕДРА СЕТЕЙ СВЯЗИ И ПЕРЕДАЧИ ДАННЫХ

КУРСОВОЙ ПРОЕКТ

«РАСЧЕТ ПРОПУСКНОЙ СПОСОБНОСТИ ЛИНИЙ СВЯЗИ»

ПО ДИСЦИПЛИНЕ «МАТЕМАТИЧЕСКИЕ МОДЕЛИ В СЕТЯХ СВЯЗИ»

КОД ВАРИАНТА 0514

Выполнил:

студент 2-го курса группы ИКПИ-05

Молошников Ф.А.

Санкт-Петербург

2021

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

Цель работы: расчет требуемых пропускных способностей линий связи в сети с заданными структурными параметрами и требованиями.

2. ИСХОДНЫЕ ДАННЫЕ

Параметры сети:

- Количество узлов в сети связи: $n = 20$,
- Интенсивность удельной абонентской нагрузки: $y_0 = 0{,}1$ Эрл,
- Тип кодека: G.711,
- Скорость потока для данного типа кодека: $a_0 = 85600$ бит/с,
- Длина пакета: $L = 200$ байт (1600 бит),
- Количество абонентов в каждом узле связи представлены в таблице 2.1:

n	1	2	3	4	5	6	7	8	9	10
Абонентов	2441	7510	3015	1799	6242	3464	4327	5590	9935	3437
n	11	12	13	14	15	16	17	18	19	20
Абонентов	9510	3993	2126	3917	7158	7035	7292	1790	2678	1306

Таблица 2.1 — Исходные данные по количеству абонентов в узлах сети

- Матрица расстояний представлена в таблице 2.2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0		95.7						35.8	63.3	36.3	14.2		80.4	95.3		67.1	10	17.6	
2		0	25.9		7.11	53.1	50.7				39.1		3.37					2.41		5.02
3	95.7	25.9	0			57.3			74			76.8		26.8		74.1		81.7	62.7	37.4
4				0	58.8					83.7	26.7	38			66.2	81			5.48	
5		7.11		58.8	0	43		13.9	53.5	64.2			62.5	4.94	25	8.62			74.2	53.3
6		53.1	57.3		43	0			73.5	80.6	71.7	64.5		96	90.2		55	15.5	94.9	
7		50.7					0			1.74	83.4	45.8		79.9	51.9		79.8		45.7	52.2
8					13.9			0	7.15	5.95		98.5		64.8				57.9		
9	35.8		74		53.5	73.5		7.15	0	18		17.4	69.5		44.1	27.7	52.1	26.8	62.7	
10	63.3			83.7	64.2	80.6	1.74	5.95	18	0					81.8	16.6	59.6		22.8	4.88
11	36.3	39.1		26.7		71.7	83.4				0	24.2	31.8						85.9	
12	14.2		76.8	38		64.5	45.8	98.5	17.4		24.2	0	40.8	76.8		15.4			3.4	47.8
13		3.37			62.5				69.5		31.8	40.8	0	57.9		3.27	3.73	28.5	58.9	77.8
14	80.4		26.8		4.94	96	79.9	64.8				76.8	57.9	0	46.6	10.2				29.7
15	95.3			66.2	25	90.2	51.9		44.1	81.8				46.6	0			93.5		86.8
16			74.1	81	8.62				27.7	16.6		15.4	3.27	10.2		0	60.2	56.6		
17	67.1					55	79.8		52.1	59.6			3.73			60.2	0			5.02
18	10	2.41	81.7			15.5		57.9	26.8				28.5		93.5	56.6		0	10	
19	17.6		62.7	5.48	74.2	94.9	45.7		62.7	22.8	85.9	3.4	58.9					10	0	
20		5.02	37.4		53.3		52.2			4.88		47.8	77.8	29.7	86.8		5.02			0

Таблица 2.2 — Матрица расстояний

Требования к качеству обслуживания:

- Начальное требование к величине задержки: $T_0 = 0{,}1$ с,
- Доля вызовов, обслуженных с гарантированным качеством: $q = 98\%$.

Структура сети задана неориентированным графом. Вершинами графа являются узлы сети, а ребрами линии связи (см. рисунок 2.1).

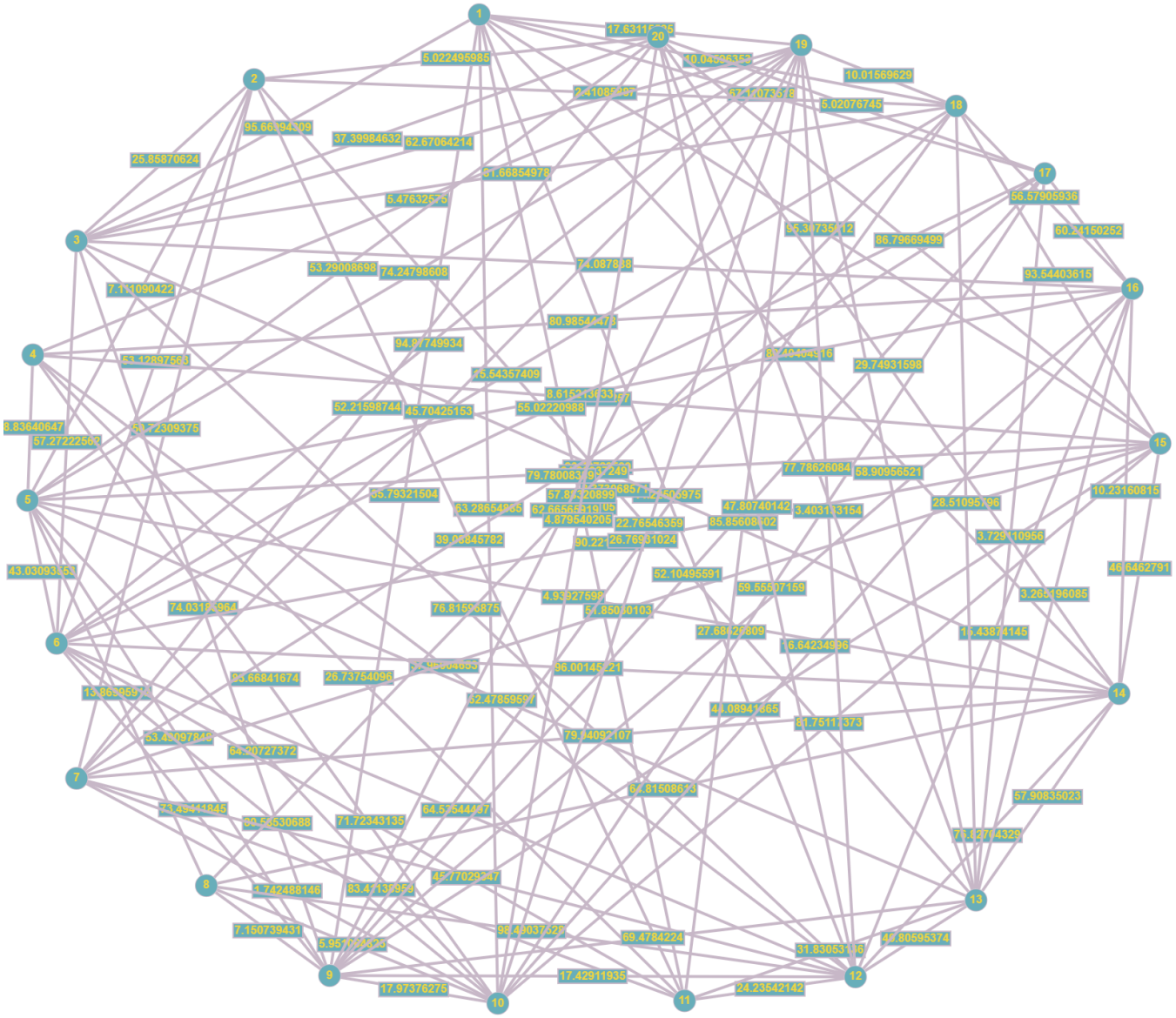

Рисунок 2.1 — Неориентированный граф

- Каждый из узлов сети может производить «собственный» трафик и может выполнять функции маршрутизации трафика других узлов сети.
- «Собственный» трафик узла связи производят абоненты, подключенные к этому узлу.
- Линии связи в сети являются двунаправленными (т.е. если скорость передачи равна b Мбит/с, то данные по ней могут передаваться независимо в двух направлениях со скоростями b Мбит/с).
- Трафик между узлами сети распределен пропорционально их абонентской емкости.

- Маршрутизация трафика производится по кратчайшему маршруту, при этом кратчайшим маршрутом является маршрут с минимальной длиной, вычисленной согласно заданной матрице расстояний между узлами.
- Все абоненты сети используют только услугу IP телефонии (VoIP).

3. ЗАДАНИЕ

1) Требуется выполнить расчет величины пропускной способности линий связи, для обеспечения нормы качества обслуживания (q и T_0).

2) Оптимизировать величины пропускной способности линий связи в сети, при целевом значении «сквозной» задержки абонент-абонент: $\frac{T_0}{2} = 50$ мс.

4. РАСЧЕТ

4.1. Расчет интенсивностей производимого в узлах сети трафика

Интенсивность исходящего трафика от каждого из узлов сети:
$y_i = N_i\, y_0$ Эрл, $\quad (i = 1..n)$
$N_i -$ количество абонентов в -м узле связи.
Расчет представлен в таблице 4.1.

n	1	2	3	4	5	6	7	8	9	10
Абонентов	2441	7510	3015	1799	6242	3464	4327	5590	9935	3437
y_i	244.1	751	301.5	179.9	624.2	346.4	432.7	559	993.5	343.7

n	11	12	13	14	15	16	17	18	19	20
Абонентов	9510	3993	2126	3917	7158	7035	7292	1790	2678	1306
y_i	951	399.3	212.6	391.7	715.8	703.5	729.2	179	267.8	130.6

Таблица 4.1 — Интенсивность исходящего трафика

4.2. Расчет коэффициентов распределения трафика по направлениям связи

Коэффициенты распределения трафика по направлениям связи

$$k_i = \frac{y_i}{y_\Sigma}, \qquad i = 1 \ldots n, \qquad y_\Sigma = \sum_{j=1}^{n} y_j\,, \quad j = 1 \ldots n$$

Расчет представлен в таблице 4.2.

n	1	2	3	4	5	6	7	8	9	10
Абонентов	2441	7510	3015	1799	6242	3464	4327	5590	9935	3437
k_i	0.026	0.079	0.032	0.019	0.066	0.037	0.046	0.059	0.105	0.036

n	11	12	13	14	15	16	17	18	19	20
Абонентов	9510	3993	2126	3917	7158	7035	7292	1790	2678	1306
k_i	0.101	0.042	0.022	0.041	0.076	0.074	0.077	0.019	0.028	0.014

Таблица 4.2 — Коэффициенты распределения трафика по направлениям связи

4.3. Расчет интенсивностей трафика в направлениях связи

Матрица интенсивностей трафика в направлениях связи

$$Y = [y_{i,j}], \qquad y_{i,j} = y_i * k_j, \qquad i,j = 1 \dots n$$

Расчет представлен в таблице 4.3

yij=kij *yi	ki=1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
yi=1	6.30	19.39	7.78	4.64	16.11	8.94	11.17	14.43	25.65	8.87	24.55	10.31	5.49	10.11	18.48	18.16	18.82	4.62	6.91	3.37
2	19.39	59.64	23.94	14.29	49.57	27.51	34.36	44.39	78.90	27.30	75.52	31.71	16.88	31.11	56.85	55.87	57.91	14.22	21.27	10.37
3	7.78	23.94	9.61	5.74	19.90	11.04	13.80	17.82	31.68	10.96	30.32	12.73	6.78	12.49	22.82	22.43	23.25	5.71	8.54	4.16
4	4.64	14.29	5.74	3.42	11.87	6.59	8.23	10.63	18.90	6.54	18.09	7.60	4.04	7.45	13.62	13.38	13.87	3.41	5.09	2.48
5	16.11	49.57	19.90	11.87	41.20	22.87	28.56	36.90	65.58	22.69	62.77	26.36	14.03	25.86	47.25	46.44	48.13	11.82	17.68	8.62
6	8.94	27.51	11.04	6.59	22.87	12.69	15.85	20.48	36.39	12.59	34.84	14.63	7.79	14.35	26.22	25.77	26.71	6.56	9.81	4.78
7	11.17	34.36	13.80	8.23	28.56	15.85	19.80	25.58	45.46	15.73	43.51	18.27	9.73	17.92	32.75	32.19	33.37	8.19	12.25	5.98
8	14.43	44.39	17.82	10.63	36.90	20.48	25.58	33.04	58.73	20.32	56.22	23.60	12.57	23.15	42.31	41.59	43.11	10.58	15.83	7.72
9	25.65	78.90	31.68	18.90	65.58	36.39	45.46	58.73	104.38	36.11	99.91	41.95	22.34	41.15	75.20	73.91	76.61	18.81	28.14	13.72
10	8.87	27.30	10.96	6.54	22.69	12.59	15.73	20.32	36.11	12.49	34.56	14.51	7.73	14.24	26.02	25.57	26.50	6.51	9.73	4.75
11	24.55	75.52	30.32	18.09	62.77	34.84	43.51	56.22	99.91	34.56	95.64	40.16	21.38	39.39	71.98	70.75	73.33	18.00	26.93	13.13
12	10.31	31.71	12.73	7.60	26.36	14.63	18.27	23.60	41.95	14.51	40.16	16.86	8.98	16.54	30.22	29.71	30.79	7.56	11.31	5.51
13	5.49	16.88	6.78	4.04	14.03	7.79	9.73	12.57	22.34	7.73	21.38	8.98	4.78	8.81	16.09	15.82	16.39	4.02	6.02	2.94
14	10.11	31.11	12.49	7.45	25.86	14.35	17.92	23.15	41.15	14.24	39.39	16.54	8.81	16.22	29.65	29.14	30.20	7.41	11.09	5.41
15	18.48	56.85	22.82	13.62	47.25	26.22	32.75	42.31	75.20	26.02	71.98	30.22	16.09	29.65	54.18	53.25	55.20	13.55	20.27	9.89
16	18.16	55.87	22.43	13.38	46.44	25.77	32.19	41.59	73.91	25.57	70.75	29.71	15.82	29.14	53.25	52.34	54.25	13.32	19.92	9.72
17	18.82	57.91	23.25	13.87	48.13	26.71	33.37	43.11	76.61	26.50	73.33	30.79	16.39	30.20	55.20	54.25	56.23	13.80	20.65	10.07
18	4.62	14.22	5.71	3.41	11.82	6.56	8.19	10.58	18.81	6.51	18.00	7.56	4.02	7.41	13.55	13.32	13.80	3.39	5.07	2.47
19	6.91	21.27	8.54	5.09	17.68	9.81	12.25	15.83	28.14	9.73	26.93	11.31	6.02	11.09	20.27	19.92	20.65	5.07	7.58	3.70
20	3.37	10.37	4.16	2.48	8.62	4.78	5.98	7.72	13.72	4.75	13.13	5.51	2.94	5.41	9.89	9.72	10.07	2.47	3.70	1.80

Таблица 4.3 — Матрица интенсивностей трафика

4.4. Расчет кратчайших расстояний и маршрутов между узлами сети

Используя алгоритм Флойда, получим матрицу кратчайших расстояний и матрицу кратчайших маршрутов. Расчеты представлены в таблицах 4.4.1 и 4.4.2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	12.5	38.3	23.1	19.6	25.6	24.1	28.3	31.7	22.4	36.3	14.2	15.8	24.5	44.6	19.1	19.6	10	17.6	17.5
2	12.5	0	25.9	17.9	7.11	18	11.6	15.9	23	9.9	35.2	15.8	3.37	12.1	32.1	6.64	7.1	2.41	12.4	5.02
3	38.3	25.9	0	43.8	31.7	43.8	37.5	41.7	48.9	35.8	61.1	41.7	29.2	26.8	56.7	32.5	33	28.3	38.3	30.9
4	23.1	17.9	43.8	0	25	31	29.5	33.5	26.3	27.8	26.7	8.88	21.3	30	50	24.3	25	15.5	5.48	22.9
5	19.6	7.11	31.7	25	0	25.1	18.8	13.9	21	17	42.3	22.9	10.5	4.94	25	8.62	14.2	9.52	19.5	12.1
6	25.6	18	43.8	31	25.1	0	29.6	33.8	41	27.9	53.2	29	21.3	30	50.1	24.6	25.1	15.5	25.6	23

7	24.1	11.6	37.5	29.5	18.8	29.6	0	7.69	14.8	1.74	46.8	27.5	15	23.7	43.7	18.3	11.6	14.1	24.1	6.62
8	28.3	15.9	41.7	33.5	13.9	33.8	7.69	0	7.15	5.95	48.8	24.6	19.2	18.8	38.9	22.5	15.9	18.3	28	10.8
9	31.7	23	48.9	26.3	21	41	14.8	7.15	0	13.1	41.7	17.4	26.4	26	44.1	27.7	23	25.4	20.8	18
10	22.4	9.9	35.8	27.8	17	27.9	1.74	5.95	13.1	0	45.1	25.7	13.3	22	42	16.5	9.9	12.3	22.3	4.88
11	36.3	35.2	61.1	26.7	42.3	53.2	46.8	48.8	41.7	45.1	0	24.2	31.8	45.3	67.3	35.1	35.6	37.6	27.6	40.2
12	14.2	15.8	41.7	8.88	22.9	29	27.5	24.6	17.4	25.7	24.2	0	18.7	25.7	47.9	15.4	22.4	13.4	3.4	20.9
13	15.8	3.37	29.2	21.3	10.5	21.3	15	19.2	26.4	13.3	31.8	18.7	0	13.5	35.5	3.27	3.73	5.78	15.8	8.4
14	24.5	12.1	26.8	30	4.94	30	23.7	18.8	26	22	45.3	25.7	13.5	0	29.9	10.2	17.2	14.5	24.5	17.1
15	44.6	32.1	56.7	50	25	50.1	43.7	38.9	44.1	42	67.3	47.9	35.5	29.9	0	33.6	39.2	34.5	44.5	37.1
16	19.1	6.64	32.5	24.3	8.62	24.6	18.3	22.5	27.7	16.5	35.1	15.4	3.27	10.2	33.6	0	6.99	9.05	18.8	11.7
17	19.6	7.1	33	25	14.2	25.1	11.6	15.9	23	9.9	35.6	22.4	3.73	17.2	39.2	6.99	0	9.51	19.5	5.02
18	10	2.41	28.3	15.5	9.52	15.5	14.1	18.3	25.4	12.3	37.6	13.4	5.78	14.5	34.5	9.05	9.51	0	10	7.43
19	17.6	12.4	38.3	5.48	19.5	25.6	24.1	28	20.8	22.3	27.6	3.4	15.8	24.5	44.5	18.8	19.5	10	0	17.4
20	17.5	5.02	30.9	22.9	12.1	23	6.62	10.8	18	4.88	40.2	20.9	8.4	17.1	37.1	11.7	5.02	7.43	17.4	0

Таблица 4.4.1 — Матрица кратчайших расстояний

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	18	18	12	18	18	18	18	12	18	11	12	18	18	18	18	18	18	12	18
2	18	2	3	18	5	18	20	20	20	20	13	18	13	5	5	13	13	18	18	20
3	2	2	3	2	14	2	2	2	2	2	2	2	2	14	14	2	2	2	2	2
4	19	19	19	4	19	19	19	19	19	19	11	19	19	19	19	19	19	19	19	19
5	2	2	14	2	5	2	2	8	8	2	2	2	2	14	15	16	2	2	2	2
6	18	18	18	18	18	6	18	18	18	18	18	18	18	18	18	18	18	18	18	18
7	10	10	10	10	10	10	7	10	10	10	10	10	10	10	10	10	10	10	10	10
8	10	10	10	9	5	10	10	8	9	10	9	9	10	5	5	5	10	10	9	10
9	12	8	8	12	8	8	8	8	9	8	12	12	8	8	15	16	8	8	12	8
10	20	20	20	20	20	20	7	8	8	10	20	20	20	20	20	20	20	20	20	20
11	1	13	13	4	13	13	13	12	12	13	11	12	13	13	13	13	13	13	12	13
12	1	19	19	19	19	19	19	9	9	19	11	12	16	16	19	16	16	19	19	19
13	2	2	2	2	2	2	2	2	2	2	11	16	13	16	2	16	17	2	2	2
14	5	5	3	5	5	5	5	5	5	5	16	16	16	14	5	16	16	5	5	5
15	5	5	5	5	5	5	5	5	9	5	5	5	5	5	15	5	5	5	5	5
16	13	13	13	12	5	13	13	5	9	13	13	12	13	14	5	16	13	13	12	13
17	13	13	13	13	13	13	20	20	20	20	13	13	13	13	13	13	17	13	13	20
18	1	2	2	19	2	6	2	2	2	2	2	19	2	2	2	2	2	18	19	2
19	12	18	18	4	18	18	18	12	12	18	12	12	18	18	18	12	18	18	19	18
20	2	2	2	2	2	2	10	10	10	10	2	2	2	2	2	2	17	2	2	20

Таблица 4.4.2 — Матрица кратчайших маршрутов

4.5. Расчет интенсивностей нагрузок на линиях связи

На основе матрицы Y и матрицы R найдем матрицу $\tilde{Y} = [\tilde{y}_{i,j}]; i,j = 1 \dots n.$

Расчет представлен в таблице 4.5.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	6.3	0	0	0	0	0	0	0	0	0	24.5	47.5	0	0	0	0	0	166	0	0
2	0	59.6	237	0	851	0	0	0	0	0	0	0	1002	0	0	0	0	946	0	910

3	0	237	9.61	0	0	0	0	0	0	0	0	0	0	55.2	0	0	0	0	0	0
4	0	0	0	3.42	0	0	0	0	0	0	18.1	0	0	0	0	0	0	0	158	0
5	0	851	0	0	41.2	0	0	251	0	0	0	0	0	282	586	141	0	0	0	0
6	0	0	0	0	0	12.7	0	0	0	0	0	0	0	0	0	0	0	334	0	0
7	0	0	0	0	0	0	19.8	0	0	413	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	251	0	0	33	632	577	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	632	104	0	0	321	0	0	75.2	73.9	0	0	0	0
10	0	0	0	0	0	0	413	577	0	12.5	0	0	0	0	0	0	0	0	0	1035
11	24.5	0	0	18.1	0	0	0	0	0	0	95.6	223	590	0	0	0	0	0	0	0
12	47.5	0	0	0	0	0	0	0	321	0	223	16.9	0	0	0	119	0	0	326	0
13	0	1002	0	0	0	0	0	0	0	0	590	0	4.78	0	0	462	483	0	0	0
14	0	0	55.2	0	282	0	0	0	0	0	0	0	0	16.2	0	124	0	0	0	0
15	0	0	0	0	586	0	0	0	75.2	0	0	0	0	0	54.2	0	0	0	0	0
16	0	0	0	0	141	0	0	0	73.9	0	0	119	462	124	0	52.3	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	483	0	0	0	56.2	0	0	190
18	166	946	0	0	0	334	0	0	0	0	0	0	0	0	0	0	0	3.39	406	0
19	0	0	0	158	0	0	0	0	0	0	0	326	0	0	0	0	0	406	7.58	0
20	0	910	0	0	0	0	0	0	0	1035	0	0	0	0	0	0	190	0	0	1.8

Таблица 4.5 — Матрица интенсивностей нагрузок

4.6. Расчет количества потоков в линиях связи

На основе матрицы интенсивностей нагрузок на линии связи Y и требований к качеству обслуживания p_0, найдем матрицу потоков:

$$V = [v_{i,j}],\ \ i,j = 1 \dots n$$

$$v_{i,j} = \arg\min\left|p(\tilde{y}_{i,j}, v_{i,j}) - p_0\right|,\quad p(\tilde{y}_{i,j}, v_{i,j}) \le p_0$$

$$p(\tilde{y}_{i,j}, v_{i,j}) = \frac{\frac{\tilde{y}_{i,j}^{v_{i,j}}}{v_{i,j}!}}{\sum_{k=0}^{v_{i,j}} \frac{\tilde{y}_{i,j}^{k}}{k!}}$$

$$p_0 = 1 - \frac{q}{100}, \text{где } q \text{ в \%}$$

Расчет представлен в таблице 4.6.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	12	0	0	0	0	0	0	0	0	0	33	58	0	0	0	0	0	180	0	0
2	0	71	251	0	862	0	0	0	0	0	0	0	1011	0	0	0	0	956	0	919
3	0	251	16	0	0	0	0	0	0	0	0	0	0	66	0	0	0	0	0	0
4	0	0	0	8	0	0	0	0	0	0	26	0	0	0	0	0	0	0	172	0
5	0	862	0	0	52	0	0	265	0	0	0	0	0	296	600	155	0	0	0	0
6	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	348	0	0
7	0	0	0	0	0	0	28	0	0	427	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	265	0	0	43	645	590	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	645	117	0	0	336	0	0	87	86	0	0	0	0
10	0	0	0	0	0	0	427	590	0	20	0	0	0	0	0	0	0	0	0	1043
11	33	0	0	26	0	0	0	0	0	0	108	238	603	0	0	0	0	0	0	0
12	58	0	0	0	0	0	0	0	336	0	238	25	0	0	0	133	0	0	340	0

13	0	1011	0	0	0	0	0	0	0	0	603	0	10	0	0	476	497	0	0	0
14	0	0	66	0	296	0	0	0	0	0	0	0	0	24	0	138	0	0	0	0
15	0	0	0	0	600	0	0	0	87	0	0	0	0	0	65	0	0	0	0	0
16	0	0	0	0	155	0	0	0	86	0	0	133	476	138	0	63	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	497	0	0	0	67	0	0	204
18	180	956	0	0	0	348	0	0	0	0	0	0	0	0	0	0	0	8	420	0
19	0	0	0	172	0	0	0	0	0	0	0	340	0	0	0	0	0	420	14	0
20	0	919	0	0	0	0	0	0	0	1043	0	0	0	0	0	0	204	0	0	6

Таблица 4.6 — Матрица потоков

4.7. Расчет интенсивностей трафика ПД для линий связи

На основе матрицы потоков V и данных о типе кодека (скорости одного потока) вычисляем матрицу:

$$A = [a_{i,j}], \quad a_{i,j} = v_{i,j} * a_0, \quad i,j = 1 \dots n$$

$a_0 = 85600 \frac{\text{бит}}{\text{сек}}$ (по условию)

Расчет представлен в таблице 4.7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1027200	0	0	0	0	0	0	0	0	0	2824800	4964800	0	0	0	0	0	15408000	0	0
2	0	6077600	21485600	0	73787200	0	0	0	0	0	0	0	86541600	0	0	0	0	81833600	0	78666400
3	0	21485600	1369600	0	0	0	0	0	0	0	0	0	0	5649600	0	0	0	0	0	0
4	0	0	0	684800	0	0	0	0	0	0	2225600	0	0	0	0	0	0	0	14723200	0
5	0	73787200	0	0	4451200	0	0	22684000	0	0	0	0	0	25337600	51360000	13268000	0	0	0	0
6	0	0	0	0	0	1712000	0	0	0	0	0	0	0	0	0	0	0	29788800	0	0
7	0	0	0	0	0	0	2396800	0	0	36551200	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	22684000	0	0	3680800	55212000	50504000	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	55212000	10015200	0	0	28761600	0	0	7447200	7361600	0	0	0	0
10	0	0	0	0	0	0	36551200	50504000	0	1712000	0	0	0	0	0	0	0	0	0	89280800
11	2824800	0	0	2225600	0	0	0	0	0	0	9244800	20372800	51616800	0	0	0	0	0	0	0
12	4964800	0	0	0	0	0	0	0	28761600	0	20372800	2140000	0	0	0	11384800	0	0	29104000	0
13	0	86541600	0	0	0	0	0	0	0	0	51616800	0	856000	0	0	40745600	42543200	0	0	0
14	0	0	5649600	0	25337600	0	0	0	0	0	0	0	0	2054400	0	11812800	0	0	0	0
15	0	0	0	0	51360000	0	0	0	7447200	0	0	0	0	0	5564000	0	0	0	0	0
16	0	0	0	0	13268000	0	0	0	7361600	0	0	11384800	40745600	11812800	0	5392800	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	42543200	0	0	0	5735200	0	0	17462400
18	15408000	81833600	0	0	0	29788800	0	0	0	0	0	0	0	0	0	0	0	684800	35952000	0
19	0	0	0	14723200	0	0	0	0	0	0	0	29104000	0	0	0	0	0	35952000	1198400	0
20	0	78666400	0	0	0	0	0	0	0	89280800	0	0	0	0	0	0	17462400	0	0	513600

Таблица 4.7 — Матрица интенсивностей трафика ПД для линий связи

4.8. Расчет пропускной способности линий связи

На основе матрицы интенсивностей трафика ПД $\boldsymbol{A}$ и данных и требований к величине задержки T_0, выбрав для расчета модель M/M/1 вычислить матрицу пропускных способностей:

$$B = [b_{i,j}], \quad b_{i,j} = a_{i,j} + \frac{L}{T_0}, \quad \text{где } L - \text{в битах}, \quad i,j = 1 \dots n$$

Расчет представлен в таблице 4.8.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1043200	0	0	0	0	0	0	0	0	0	2840800	4980800	0	0	0	0	0	15424000	0	0
2	0	6093600	21501600	0	73803200	0	0	0	0	0	0	0	86557600	0	0	0	0	81849600	0	78682400
3	0	21501600	1385600	0	0	0	0	0	0	0	0	0	0	5665600	0	0	0	0	0	0
4	0	0	0	700800	0	0	0	0	0	0	2241600	0	0	0	0	0	0	0	14739200	0
5	0	73803200	0	0	4467200	0	0	22700000	0	0	0	0	0	25353600	51376000	13284000	0	0	0	0
6	0	0	0	0	0	1728000	0	0	0	0	0	0	0	0	0	0	0	29804800	0	0
7	0	0	0	0	0	0	2412800	0	0	36567200	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	22700000	0	0	3696800	55228000	50520000	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	55228000	10031200	0	0	28777600	0	0	7463200	7377600	0	0	0	0
10	0	0	0	0	0	0	36567200	50520000	0	1728000	0	0	0	0	0	0	0	0	0	89296800
11	2840800	0	0	2241600	0	0	0	0	0	0	9260800	20388800	51632800	0	0	0	0	0	0	0
12	4980800	0	0	0	0	0	0	0	28777600	0	20388800	2156000	0	0	0	11400800	0	0	29120000	0
13	0	86557600	0	0	0	0	0	0	0	0	51632800	0	872000	0	0	40761600	42559200	0	0	0
14	0	0	5665600	0	25353600	0	0	0	0	0	0	0	0	2070400	0	11828800	0	0	0	0
15	0	0	0	0	51376000	0	0	0	7463200	0	0	0	0	0	5580000	0	0	0	0	0
16	0	0	0	0	13284000	0	0	0	7377600	0	0	11400800	40761600	11828800	0	5408800	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	42559200	0	0	0	5751200	0	0	17478400
18	15424000	81849600	0	0	0	29804800	0	0	0	0	0	0	0	0	0	0	0	700800	35968000	0
19	0	0	0	14739200	0	0	0	0	0	0	0	29120000	0	0	0	0	0	35968000	1214400	0
20	0	78682400	0	0	0	0	0	0	0	89296800	0	0	0	0	0	0	17478400	0	0	529600

Таблица 4.8 — Матрица пропускных способностей (бит/с)

4.9. Оптимизация пропускной способности линий связи

Целью оптимизации пропускной способности является распределение минимального объема ресурса пропускной способности по линиям связи, необходимого для обслуживания трафика при величине задержки Topt не превышающей $\frac{T_0}{2}$ от абонента до абонента на всех маршрутах.

Алгоритм решения(рис. 4.9.1):

1) Ввод исходных данных

2) Получение решения задачи пунктов 1-7 для некоторой начальной величины T_0, заданной для каждой из линий связи.

3) Зададим задержку ToptCurrent=$\frac{T_0}{2} + dTopt$, где dTopt – заданный шаг dTopt =0.001 с.

4) Уменьшим задержку ToptCurrent на dTopt.

5) Заполним B0 начальными значениями (пункт 8 задания)

6) Вычисление матрицы значений целевой функции (В программе обозначена как MO). Вычисление каждого из элементов матрицы происходит во вложенном цикле по iO, jO=1...n. Рассмотрим вычисление одного такого элемента:

1) Задаемся некоторой величиной шага изменения пропускной способности dc (значением dc=1000 бит/с).

Вычисляем матрицу Del (В программе MDel) с учетом добавки dc, в линии между iO и jO. Вычисления производятся с использованием модели M/M/1.

$$T_m = T_{ij} = \frac{\rho\bar{t}}{1-\rho} + \bar{t} = \begin{pmatrix} \bar{t} = \frac{L}{b_m} \\ \rho = \frac{a_m}{b_m} \end{pmatrix} = \frac{L}{b_m - a_m} = [dc \neq 0] = \frac{L}{b_{ij} + dc - a_{ij}}$$

где $\mathrm{b_m}$ — пропускная способность линии (без учета добавки) (бит/сек),
$\mathrm{a_m}$ — интенсивность трафика на линии (бит/сек),
dc — произведенная «добавка» пропускной способности (бит/сек).

2) Вычисляем матрицу Dl (В программе MDl) - задержки на маршрутах с учетом изменения задержки линии во всех маршрутах абонент-абонент. Маршруты между узлами остаются прежними.

3) Вычисляем значение целевой функции:

$$O_{ij} = \sum_{i}^{n}\sum_{j}^{n}\left(T_{ij} - \mathrm{ToptCurrent}\right)^2$$

7) Находим минимальное значение в O и записываем в переменную OminCurrent.

8) Сохраняем текущее найденное минимальное значение целевой функции как ранее достигнутое. (OminPrevious= OminCurrent)

9) В матрице B0 увеличиваем пропускную способность той линии, после добавки к которой dc значение в O оказалось минимальным. (В программе MB0[iOmin][jOmin]+=dc;)

10) Вычисляем матрицу значений целевой функции аналогично пункту 6.

11) Сравниваем текущее минимальное значение целевой функции с ранее достигнутым. Если оно оказалось меньше предыдущего, то переходим к пункту 5, иначе выходим из цикла.

12) Вычилсяем Del(Таблица 4.9.3) и Dl(Таблица 4.9.4). Ищем в Dl элементы, которые больше Topt. Если такие элементы найдены, то переходим к пункту 4, иначе выходим из цикла

Таким образом, после завершения работы алгоритма мы получаем оптимизированные значения пропускных способностей в матрице B0 (Таблица 4.9.1). Del(Таблица 4.9.3) и Dl(Таблица 4.9.4) содержат задержки, соответствующие оптимизированным пропускным способностям.

Для демонстрации матрицы O (Таблица 4.9.2), в которой содержалось наименьшее достигнутое в ходе оптимизации значение целевой функции, матрица B0 возвращается в предыдущее состояние, после чего вычисляется матрица O.

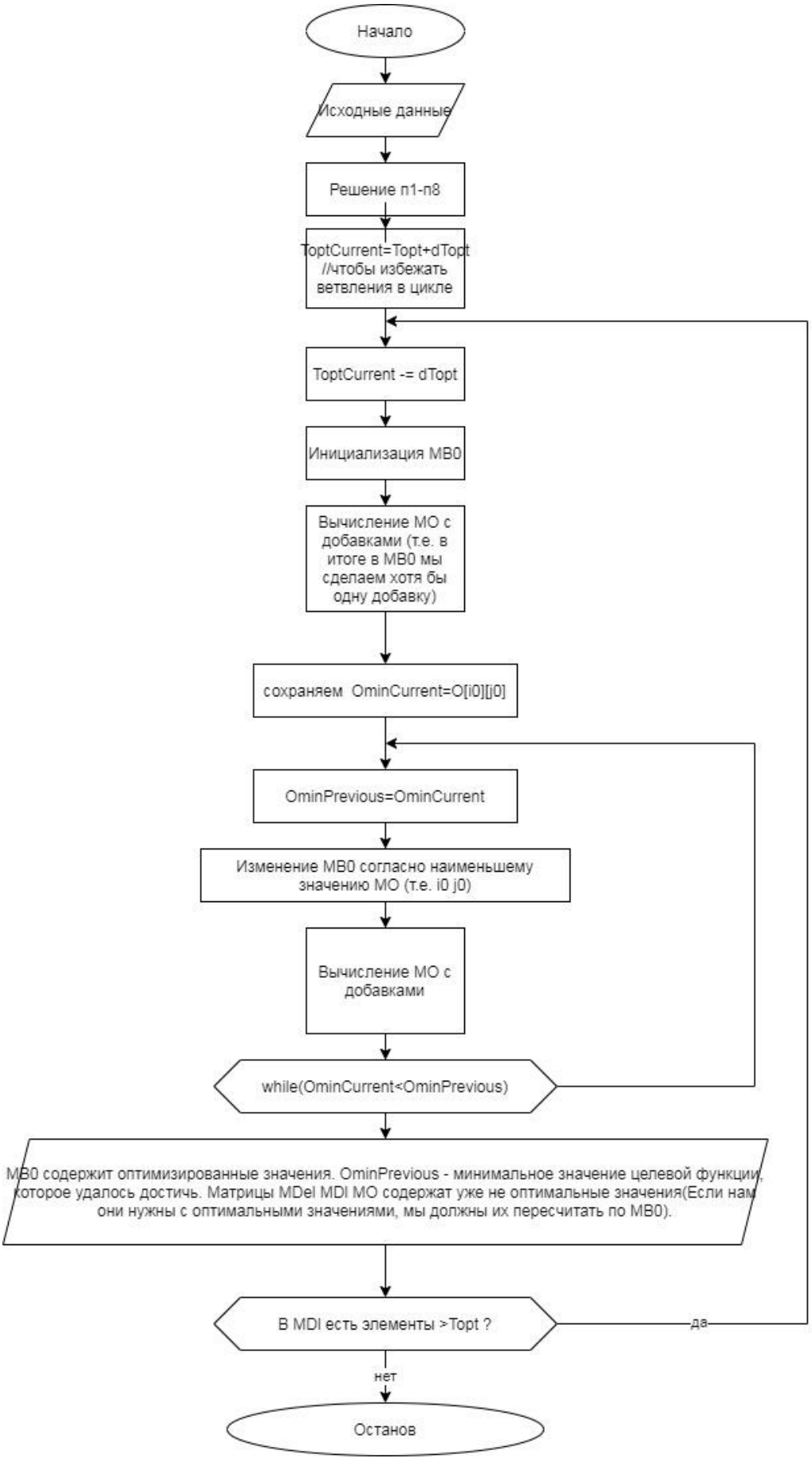

Рисунок 4.9.1 —Алгоритм оптимизации пропускных способностей

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1075200	0	0	0	0	0	0	0	0	0	2872800	5037800	0	0	0	0	0	15528000	0	0
2	0	6125600	21590600	0	73934200	0	0	0	0	0	0	0	86694600	0	0	0	0	82084600	0	78865400
3	0	21590600	1417600	0	0	0	0	0	0	0	0	0	0	5718600	0	0	0	0	0	0
4	0	0	0	732800	0	0	0	0	0	0	2273600	0	0	0	0	0	0	0	14951200	0
5	0	73934200	0	0	4499200	0	0	22757000	0	0	0	0	0	25490600	51545000	13338000	0	0	0	0
6	0	0	0	0	0	1760000	0	0	0	0	0	0	0	0	0	0	0	29916800	0	0
7	0	0	0	0	0	0	2444800	0	0	36745200	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	22757000	0	0	3728800	55457000	50645000	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	55457000	10063200	0	0	28858600	0	0	7495200	7409600	0	0	0	0
10	0	0	0	0	0	0	36745200	50645000	0	1760000	0	0	0	0	0	0	0	0	0	89634800
11	2872800	0	0	2273600	0	0	0	0	0	0	9292800	20450800	51738800	0	0	0	0	0	0	0
12	5037800	0	0	0	0	0	0	0	28858600	0	20450800	2188000	0	0	0	11465800	0	0	29257000	0
13	0	86694600	0	0	0	0	0	0	0	0	51738800	0	904000	0	0	40877600	42691200	0	0	0
14	0	0	5718600	0	25490600	0	0	0	0	0	0	0	0	2102400	0	11906800	0	0	0	0
15	0	0	0	0	51545000	0	0	0	7495200	0	0	0	0	0	5612000	0	0	0	0	0
16	0	0	0	0	13338000	0	0	0	7409600	0	0	11465800	40877600	11906800	0	5440800	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	42691200	0	0	0	5783200	0	0	17536400
18	15528000	82084600	0	0	0	29916800	0	0	0	0	0	0	0	0	0	0	0	732800	36115000	0
19	0	0	0	14951200	0	0	0	0	0	0	0	29257000	0	0	0	0	0	36115000	1246400	0
20	0	78865400	0	0	0	0	0	0	0	89634800	0	0	0	0	0	0	17536400	0	0	561600

Таблица 4.9.1 — Матрица оптимизированных пропускных способностей (бит/с)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0.21208	0	0	0	0	0	0	0	0	0	0.21208	0.212097	0	0	0	0	0	0.212109	0	0
2	0	0.21208	0.212135	0	0.212148	0	0	0	0	0	0	0	0.212152	0	0	0	0	0.212114	0	0.212133
3	0	0.212135	0.21208	0	0	0	0	0	0	0	0	0	0	0.212091	0	0	0	0	0	0
4	0	0	0	0.21208	0	0	0	0	0	0	0.21208	0	0	0	0	0	0	0	0.212075	0
5	0	0.212148	0	0	0.21208	0	0	0.212118	0	0	0	0	0	0.212091	0.212085	0.21209	0	0	0	0
6	0	0	0	0	0	0.21208	0	0	0	0	0	0	0	0	0	0	0	0.212119	0	0
7	0	0	0	0	0	0	0.21208	0	0	0.212084	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0.212118	0	0	0.21208	0.212071	0.212111	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	**0.212071**	0.21208	0	0	0.212109	0	0	0.21208	0.21208	0	0	0	0
10	0	0	0	0	0	0	0.212084	0.212111	0	0.21208	0	0	0	0	0	0	0	0	0	0.212077
11	0.21208	0	0	0.21208	0	0	0	0	0	0	0.21208	0.212092	0.212104	0	0	0	0	0	0	0
12	0.212097	0	0	0	0	0	0	0	0.21211	0	0.212092	0.21208	0	0	0	0.212106	0	0	0.212103	0
13	0	0.212152	0	0	0	0	0	0	0	0	0.212104	0	0.21208	0	0	0.212106	0.212091	0	0	0
14	0	0	0.212091	0	0.212091	0	0	0	0	0	0	0	0	0.21208	0	0.212087	0	0	0	0
15	0	0	0	0	0.212085	0	0	0	0.21208	0	0	0	0	0	0.21208	0	0	0	0	0
16	0	0	0	0	0.21209	0	0	0	0.21208	0	0	0.212106	0.212106	0.212087	0	0.21208	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0.212091	0	0	0	0.21208	0	0	0.212107
18	0.212109	0.212114	0	0	0	0.212119	0	0	0	0	0	0	0	0	0	0	0	0.21208	0.212125	0
19	0	0	0	0.212075	0	0	0	0	0	0	0	0.212103	0	0	0	0	0	0.212125	0.21208	0
20	0	0.212133	0	0	0	0	0	0	0	0.212077	0	0	0	0	0	0	0.212107	0	0	0.21208

Таблица 4.9.2 — Матрица значений целевой функции МО, содержащая минимальное значение целевой функции

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0.033	0	0	0	0	0	0	0	0	0	0.033	0.022	0	0	0	0	0	0.013	0	0
2	0	0.033	0.015	0	0.011	0	0	0	0	0	0	0	0.01	0	0	0	0	0.006	0	0.008
3	0	0.015	0.033	0	0	0	0	0	0	0	0	0	0	0.023	0	0	0	0	0	0
4	0	0	0	0.033	0	0	0	0	0	0	0.033	0	0	0	0	0	0	0	0.007	0
5	0	0.011	0	0	0.033	0	0	0.022	0	0	0	0	0	0.01	0.009	0.023	0	0	0	0
6	0	0	0	0	0	0.033	0	0	0	0	0	0	0	0	0	0	0	0.013	0	0
7	0	0	0	0	0	0	0.033	0	0	0.008	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0.022	0	0	0.033	0.007	0.011	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.007	0.033	0	0	0.016	0	0	0.033	0.033	0	0	0	0
10	0	0	0	0	0	0	0.008	0.011	0	0.033	0	0	0	0	0	0	0	0	0	0.005
11	0.033	0	0	0.033	0	0	0	0	0	0	0.033	0.021	0.013	0	0	0	0	0	0	0
12	0.022	0	0	0	0	0	0	0	0.016	0	0.021	0.033	0	0	0	0.02	0	0	0.01	0
13	0	0.01	0	0	0	0	0	0	0	0	0.013	0	0.033	0	0	0.012	0.011	0	0	0
14	0	0	0.023	0	0.01	0	0	0	0	0	0	0	0	0.033	0	0.017	0	0	0	0
15	0	0	0	0	0.009	0	0	0	0.033	0	0	0	0	0	0.033	0	0	0	0	0
16	0	0	0	0	0.023	0	0	0	0.033	0	0	0.02	0.012	0.017	0	0.033	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0.011	0	0	0	0.033	0	0	0.022
18	0.013	0.006	0	0	0	0.013	0	0	0	0	0	0	0	0	0	0	0	0.033	0.01	0
19	0	0	0	0.007	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0.01	0.033	0
20	0	0.008	0	0	0	0	0	0	0	0.005	0	0	0	0	0	0	0.022	0	0	0.033

Таблица 4.9.3 —Матрица задержек на линиях связи MDel

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0.033	0.02	0.035	0.039	0.031	0.026	0.041	0.044	0.038	0.032	0.033	0.022	0.03	0.041	0.039	0.042	0.041	0.013	0.032	0.028
2	0.02	0.033	0.015	0.023	0.011	0.019	0.021	0.024	0.03	0.013	0.024	0.027	0.01	0.021	0.02	0.023	0.021	0.006	0.016	0.008
3	0.035	0.015	0.033	0.038	0.034	0.034	0.036	0.039	0.046	0.028	0.039	0.042	0.026	0.023	0.042	0.038	0.037	0.022	0.031	0.023
4	0.039	0.023	0.038	0.033	0.034	0.029	0.044	0.041	0.034	0.036	0.033	0.017	0.034	0.045	0.043	0.037	0.044	0.017	0.007	0.031
5	0.031	0.011	0.034	0.034	0.033	0.03	0.032	0.022	0.028	0.023	0.034	0.038	0.021	0.01	0.009	0.023	0.032	0.017	0.027	0.019
6	0.026	0.019	0.034	0.029	0.03	0.033	0.04	0.043	0.049	0.031	0.042	0.033	0.029	0.04	0.038	0.041	0.04	0.013	0.022	0.027
7	0.041	0.021	0.036	0.044	0.032	0.04	0.033	0.02	0.026	0.008	0.044	0.047	0.031	0.042	0.04	0.043	0.034	0.027	0.037	0.013
8	0.044	0.024	0.039	0.041	0.022	0.043	0.02	0.033	0.007	0.011	0.044	0.023	0.034	0.032	0.031	0.045	0.037	0.03	0.033	0.016
9	0.038	0.03	0.046	0.034	0.028	0.049	0.026	0.007	0.033	0.018	0.037	0.016	0.041	0.039	0.033	0.033	0.044	0.037	0.027	0.022
10	0.032	0.013	0.028	0.036	0.023	0.031	0.008	0.011	0.018	0.033	0.036	0.039	0.023	0.034	0.032	0.035	0.026	0.019	0.029	0.005
11	0.033	0.024	0.039	0.033	0.034	0.042	0.044	0.044	0.037	0.036	0.033	0.021	0.013	0.042	0.043	0.025	0.024	0.03	0.031	0.032
12	0.022	0.027	0.042	0.017	0.038	0.033	0.047	0.023	0.016	0.039	0.021	0.033	0.032	0.037	0.046	0.02	0.043	0.02	0.01	0.035
13	0.03	0.01	0.026	0.034	0.021	0.029	0.031	0.034	0.041	0.023	0.013	0.032	0.033	0.029	0.03	0.012	0.011	0.017	0.027	0.018
14	0.041	0.021	0.023	0.045	0.01	0.04	0.042	0.032	0.039	0.034	0.042	0.037	0.029	0.033	0.019	0.017	0.04	0.028	0.038	0.029
15	0.039	0.02	0.042	0.043	0.009	0.038	0.04	0.031	0.033	0.032	0.043	0.046	0.03	0.019	0.033	0.032	0.041	0.026	0.036	0.028
16	0.042	0.023	0.038	0.037	0.023	0.041	0.043	0.045	0.033	0.035	0.025	0.02	0.012	0.017	0.032	0.033	0.023	0.029	0.03	0.031
17	0.041	0.021	0.037	0.044	0.032	0.04	0.034	0.037	0.044	0.026	0.024	0.043	0.011	0.04	0.041	0.023	0.033	0.028	0.037	0.022
18	0.013	0.006	0.022	0.017	0.017	0.013	0.027	0.03	0.037	0.019	0.03	0.02	0.017	0.028	0.026	0.029	0.028	0.033	0.01	0.014
19	0.032	0.016	0.031	0.007	0.027	0.022	0.037	0.033	0.027	0.029	0.031	0.01	0.027	0.038	0.036	0.03	0.037	0.01	0.033	0.024
20	0.028	0.008	0.023	0.031	0.019	0.027	0.013	0.016	0.022	0.005	0.032	0.035	0.018	0.029	0.028	0.031	0.022	0.014	0.024	0.033

Таблица 4.9.4 —Матрица задержек на маршрутах MDl

5. ВЫВОДЫ

1) Пропускные способности линий связи рассчитаны согласно заданным параметрам и требованиям.

2) Выполнена оптимизация пропускных способностей линий связи для заданной задержки на маршрутах ($\frac{T_0}{2}$).

3) На языке C++ написана программа, позволяющая производить вычисления матриц, необходимых для выполнения данной работы.

6. ПРИЛОЖЕНИЯ

6.1. Приложение 1. Код программы.

```
    #include <iostream>
    #include <algorithm>
    #include <fstream>
    //#include <iomanip> если не будет работать сброс ширины поля в консоли
    //Максимальное значение веса = 100
    #include <cmath>

    #define INF 101

    using namespace std;

    template<typename T>
    void printMatrix(T** matrix, int numberOfVert, int OrderFlag=0) //int OrderFlag - флаг нумерации значений в матрице. 0 с нуля; 1 - с единицы(будет прибалена 1 к каждому элементу).
    {
        double CtrlSum=0; //контрольная сумма
        for(int i = 0; i < numberOfVert; i++) {
            for(int j = 0; j < numberOfVert; j++)
            {
                cout.width(7);
                cout << matrix[i][j]+OrderFlag<< " ";
                CtrlSum+=matrix[i][j]+OrderFlag; //нумерация с 1
            }
            cout << endl;
        }
        cout << "CtrlSum=" << CtrlSum<<endl;
    }
    template<typename T>
    void printMatrixToFile(char* filename, T** matrix, int numberOfVert, int OrderFlag=0) //int OrderFlag - флаг нумерации значений в матрице. 0 с нуля; 1 - с единицы(будет прибалена 1 к каждому элементу).
    {
        ofstream ofile(filename);
        if(!filename) cout <<"Cannot open file " << filename << endl;

        double CtrlSum=0; //контрольная сумма
        for(int i = 0; i < numberOfVert; i++) {
```

```
            for(int j = 0; j < numberOfVert; j++)
            {
                ofile<<std::fixed;
            //    cout.width(7);
               // cout << matrix[i][j]+OrderFlag<< " ";
                ofile<<matrix[i][j]+OrderFlag<< " ";
                CtrlSum+=matrix[i][j]+OrderFlag; //нумерация с 1
                ofile.unsetf(ios::fixed | ios::scientific);
            }
         //   cout << endl;
            ofile <<endl;
        }
      //  cout << "CtrlSum=" << CtrlSum<<endl;
        ofile<<std::fixed;
        ofile << "CtrlSum=" << CtrlSum<<endl;
        ofile.unsetf(ios::fixed | ios::scientific);
        ofile.close();
       // cout << std::resetiosflags(std::ios::fixed); //если не будет
работать сброс ширины поля в консоли
    }
    //matrix - матрица смежности
    void originalFloydWarshall(double **matrix, int numberOfVert, int
**MPath) { //double **matrix-матрица исходных расстояний, туда же
пишется результат, int numberOfVert-число вершин, int **MPath-матрица
маршрутизации
      //     int  MPath[numberOfVert][numberOfVert];  //матрица  1-ых
элеметов в пути от i до j;  Т.е. путь от i до j такой: i->MPath[i][j]-
>...->j
        for(int  i=0;  i<  numberOfVert;  i++)  //заполняем  матрицу
стартовыми значениями.
        {
            for(int j=0; j< numberOfVert; j++)
            {
                MPath[i][j]=j;
            }
        }
        //Пробегаемся по всем вершинам и ищем более короткий путь
        //через вершину k
        for(int k = 0; k < numberOfVert; k++) {
            for(int i = 0; i < numberOfVert; i++) {
                for(int j = 0; j < numberOfVert; j++) {
                    //Новое  значение  ребра  равно  минимальному  между
старым
                    //и суммой ребер i <-> k + k <-> j (если через k
пройти быстрее)
                    if(matrix[i][j]>matrix[i][k]+matrix[k][j])
                    {
                        matrix[i][j] = matrix[i][k] + matrix[k][j];
                        MPath[i][j]=MPath[i][k];
                    }
                 //    matrix[i][j] = min(matrix[i][j], matrix[i][k] +
matrix[k][j]);
```

```
                }
            }
        }
    //    int CtrlSum=0; //контрольная сумма
      //  cout << "MPath[i][j]:" <<endl;
     //   for(int i=0; i< numberOfVert;i++)
    //    {
     //       for(int j=0;j< numberOfVert; j++)
      //      {
       //         cout << MPath[i][j]+1 << " "; //нумерация с 1
        //        CtrlSum+=MPath[i][j]+1; //нумерация с 1
         //   }
         //   cout <<endl;
       //  }
     //   cout << "CtrlSum=" << CtrlSum<<endl;
        return;
    }
    void   EdgeLoad(double   **MNodeToNodeLoad,   int   **MPath,   int
numberOfVert, double **MEdgeLoad)
    {
        int i, j;
        for(i=0; i<numberOfVert; i++)
        {
            for(j=0; j<numberOfVert; j++)
            {
                int k1=i;
                int k2=MPath[i][j]; //
                MEdgeLoad[k1][k2]+=MNodeToNodeLoad[i][j];
                while(k2!=j)
                {
                    k1=k2;
                    k2=MPath[k2][j];
                    MEdgeLoad[k1][k2]+=MNodeToNodeLoad[i][j];
                }
            }
        }
    }
    void ThreatNumber(double **MEdgeLoad, int numberOfVert, double p0,
int **MThreatNumber)
    {
        int i, j;
    /*     int k; //Нужно преобразовывать к long double. long long не
прокатит. */
        long double p, numerator, sum;
        for(i=0; i<numberOfVert; i++)
        {
            for(j=0;j<numberOfVert;j++)
            {
                if(MEdgeLoad[i][j]>0)
                {
```

```
            /*          MThreatNumber[i][j]=1; //все работает. Короче,
тут 2 варианта. Тот, который в комментариях, считает в лоб. Его
результаты отличаются от результатов 2го варианта в пределах единицы.
Надо полагать, что 2 вариант точнее, т.к. при его использовании меньше
операций с большими числами
                    sum=0;
                    for(k=0;    k<MThreatNumber[i][j];    k++)    //ищем
знаменатель формулы
                    {
                        sum+=pow((long          double)MEdgeLoad[i][j],
k)/tgamma((long double)k+1); //MEdgeLoad тоже нужно преобразовывать к
long double, иначе pow возвращает бесконечность.
                    }
                    p=pow((long                 double)MEdgeLoad[i][j],
MThreatNumber[i][j])/tgamma((long    double)MThreatNumber[i][j]+1)/sum;
*/
                    MThreatNumber[i][j]=1;
                    numerator=MEdgeLoad[i][j];
                    sum=1+MEdgeLoad[i][j];
                    p=numerator/sum;
                    while(p>p0)
                    {
            /*          MThreatNumber[i][j]++; //все работает
                        sum=0;
                     //   cout<< "sum:"<<endl;
                        for(k=0;  k<MThreatNumber[i][j];  k++)  //ищем
знаменатель формулы
                        {
                            sum+=pow((long      double)MEdgeLoad[i][j],
k)/tgamma((long double)k+1);
                       //       cout << sum <<endl;
                        }
                    //     cout<< "sum="<<sum<<endl;
                    //        cout<< "pow((long double)MEdgeLoad[i][j],
MThreatNumber[i][j])/tgamma((long  double)MThreatNumber[i][j]+1)="  <<
pow((long   double)MEdgeLoad[i][j],   MThreatNumber[i][j])/tgamma((long
double)MThreatNumber[i][j]+1)<< endl;
                        p=pow((long             double)MEdgeLoad[i][j],
MThreatNumber[i][j])/tgamma((long double)MThreatNumber[i][j]+1)/sum;
            */
                        MThreatNumber[i][j]++;

numerator=numerator*MEdgeLoad[i][j]/MThreatNumber[i][j];
                        sum=sum+numerator;
                        p=numerator/sum;
                    }
                }
                else MThreatNumber[i][j]=0;
             //   break;
            }
         //   break;
        }
```

```
    //          cout  <<"tgamma((long  double)1000+1)="<<  tgamma((long
double)1000+1) <<endl;
     //       cout  <<"tgamma((long  double)MThreatNumber[i][j]+1)="<<
tgamma((long double)MThreatNumber[i][j]+1) <<endl;
    }
    void ThreatNumberBruteForce(double **MEdgeLoad, int numberOfVert,
double p0, int **MThreatNumber)
    {
        int i, j;
        int k; //Нужно преобразыовывать к long double. long long не
прокатит.
        long double p, sum;
        for(i=11; i<numberOfVert; i++)
        {
            for(j=11;j<numberOfVert;j++)
            {
                if(MEdgeLoad[i][j]>0)
                {
                    MThreatNumber[i][j]=1; //все работает. Короче, тут
2 варианта. Тот, который в комментариях, считает в лоб. Его результаты
отличаются от результатов 2го варианта в пределах единицы. Надо
полагать, что 2 вариант точнее, т.к. при его использовании меньше
операций с большими числами
                    sum=0;
                    for(k=0;    k<MThreatNumber[i][j];    k++)    //ищем
знаменатель формулы
                    {
                        sum+=pow((long          double)MEdgeLoad[i][j],
k)/tgamma((long double)k+1); //MEdgeLoad тоже нужно преобразовывать к
long double, иначе pow возвращает бесконечность.
                    }
                    p=pow((long                 double)MEdgeLoad[i][j],
MThreatNumber[i][j])/tgamma((long double)MThreatNumber[i][j]+1)/sum;
                    while(p>p0)
                    {
                    MThreatNumber[i][j]++; //все работает
                        sum=0;
                     //   cout<< "sum:"<<endl;
                        for(k=0;  k<MThreatNumber[i][j];  k++)  //ищем
знаменатель формулы
                        {
                            sum+=pow((long      double)MEdgeLoad[i][j],
k)/tgamma((long double)k+1);
                       //       cout << sum <<endl;
                        }
                    //     cout<< "sum="<<sum<<endl;
                    //        cout<<  "pow((long  double)MEdgeLoad[i][j],
MThreatNumber[i][j])/tgamma((long  double)MThreatNumber[i][j]+1)="  <<
pow((long   double)MEdgeLoad[i][j],   MThreatNumber[i][j])/tgamma((long
double)MThreatNumber[i][j]+1)<< endl;
                        p=pow((long             double)MEdgeLoad[i][j],
MThreatNumber[i][j])/tgamma((long double)MThreatNumber[i][j]+1)/sum;
```

```
                        cout <<"p 11x11="<< p <<endl;
                    }
                }
                else MThreatNumber[i][j]=0;
                break;
            }
            break;
        }

     //       cout  <<"tgamma((long  double)MThreatNumber[i][j]+1)="<<
tgamma((long double)MThreatNumber[i][j]+1) <<endl;
    }
    /*  7 пункт. Матрица интенсивности трафика ПД для линий связи*/
    void  TrafficIntensity(int  **MThreatNumber,  int  numberOfVert,
double a0, double **MTrafficIntensity)
    {
        int i, j;
        for(i=0; i< numberOfVert; i++)
        {
            for(j=0; j<numberOfVert; j++)
            {
                MTrafficIntensity[i][j]=MThreatNumber[i][j]*a0;
            }
        }
    }
    /*  8 пункт. Матрица пропускной способности линий связи*/
    void     CommunLineCapacity(double     **MTrafficIntensity,     int
numberOfVert, double L, double T0, double **MCommunLineCapacity)
    {
        int i, j;
        for(i=0; i< numberOfVert; i++)
        {
            for(j=0; j< numberOfVert; j++)
            {
                if(MTrafficIntensity[i][j]!=0)
                {

MCommunLineCapacity[i][j]=MTrafficIntensity[i][j]+L/T0;
                }
                else
                {
                    MCommunLineCapacity[i][j]=0;
                }
            }
        }
    }
    /* 9 пункт. Матрица МО*/
    void OCalc(double **MB0, double **MTrafficIntensity, int **MPath,
int  numberOfVert,  double  L,  double  Topt,  double  dc,  double  **MO)
//результат в МО. Остальные параметры не меняются.
    {
        //MDel матрица задержек Tij на линиях связи
```

```
        double **MDel=(double**)malloc(sizeof(double *)*numberOfVert);
        for(int i=0; i<numberOfVert; i++)
        {
            MDel[i]=(double*)malloc(sizeof(double)*numberOfVert);
        }
        //MDl матрица задержек dlij между узлами по маршрутам.
        double **MDl=(double**)malloc(sizeof(double *)*numberOfVert);
        for(int i=0; i<numberOfVert; i++)
        {
            MDl[i]=(double*)malloc(sizeof(double)*numberOfVert);
        }
        for(int iO=0; iO< numberOfVert; iO++)
        {
            for(int jO=0; jO<numberOfVert; jO++) //в цикле ищем каждое
значение MO[iO][jO]
            {
                //Вычисляем MDel
                for(int i=0; i< numberOfVert; i++)
                {
                    for(int j=0; j<numberOfVert; j++)
                    {
                        if(MB0[i][j]>0)  //если  линия  используется  в
маршрутах
                        {
                            if(i==iO && j==jO)
                            {
                                MDel[i][j]=L*8/(MB0[i][j]+dc-
MTrafficIntensity[i][j]);
                            }
                            else
                            {
                                MDel[i][j]=L*8/(MB0[i][j]-
MTrafficIntensity[i][j]); //это можно вынести вне цикла
                            }
                        }
                        else
                        {
                            MDel[i][j]=0;
                        }
                    }
                }
         //       cout<<"MDel:"<<endl;
           //     printMatrix(MDel, numberOfVert);
                //Вычилсяем MDl
                for(int i=0; i<numberOfVert; i++)
                {
                    for(int   j=0;   j<numberOfVert;   j++)   //вычисляем
задержку для каждого маршрута
                    {
                        int k1=i;
                        int k2=MPath[i][j];
                        MDl[i][j]=0;        //инициализация
```

```
                        MDl[i][j]+=MDel[k1][k2];
                        while(k2!=j)
                        {
                            k1=k2;
                            k2=MPath[k2][j];
                            MDl[i][j]+=MDel[k1][k2];
                        }
                    }
                }
        //        cout<<"MDl:"<<endl;
         //       printMatrix(MDl, numberOfVert);
                //Вычисляем MO[iO][jO]
                MO[iO][jO]=0;
                for(int i=0; i<numberOfVert; i++)
                {
                    for(int j=0; j<numberOfVert; j++)
                    {
                        if(MB0[iO][jO]!=0)
                        {
                            MO[iO][jO]+=(MDl[i][j]-Topt)*(MDl[i][j]-
Topt);
                        }
                        else
                        {
                            MO[iO][jO]=0;
                        }
                    }
                }

        //    break;
            }
        //    break;
        }
        for(int i=0; i< numberOfVert; i++)
        {
            free(MDel[i]);
            free(MDl[i]);
        }
        free(MDel);
        free(MDl);
    }
    /* Ищем OminCurrent*/
    void OminCurrentCalc(double **MO, int numberOfVert, int *iOmin,
int *jOmin, double *OminCurrent)
    {
        *OminCurrent=MO[0][0];
        *iOmin=0;
        *jOmin=0;
        for(int iO=0; iO<numberOfVert; iO++)
        {
            for(int jO=0; jO<numberOfVert; jO++)
            {
```

```
                if(MO[iO][jO]<  *OminCurrent  &&  MO[iO][jO]>0)  //есть
некоторый шанс, что MO[iO][jO] станет равно 0;
                {
                    *OminCurrent=MO[iO][jO];
                    *iOmin=iO;
                    *jOmin=jO;
                }
            }
        }
    }
    bool IsGreaterThen(double **MDl, int numberOfVert, double Topt) //
если хотя бы один элемент больше заданного времени, то возвращается
true иначе false
    {
        int i, j;
        for(i=0;i<numberOfVert; i++)
        {
            for(j=0; j<numberOfVert; j++)
            {
                if(MDl[i][j]>Topt) return true;
            }
        }
        return false;
    }
    int main(int argc, char** argv) {
    /*  Данные курсового проекта*/
        double L=200; //байт
        double a0=85600; //бит/с
        double T0=0.1; //с
        double q=0.98; //в долях единицы

        // Для пункта 9
        double  Topt=T0/2; //Задержка по маршруту (от узла i до узла
j), к которой мы стремимся, увеличивая пропускную способность на линии
        double dc=1000; //Шаг изменения пропускной способности линии
связи
        int iOminPrevious; //Индексы минимального значения в матрице
MO. Нужны для получения предыдущего состояния MB0, чтобы по нему уже
построить все интересующие нас для ответа промежуточные матрицы.
        int jOminPrevious;

        double OminCurrent;
        double OminPrevious;
        int  iOmin,  jOmin;  //хранят  индекс  MO  такой,  в  котором
минимальное значение MO из всех

        //Для оптимизации времени (чтобы задержки не превышали
заданное время, а не просто были оптимальными для данного шага
пропускных способностей)
        double ToptCurrent; //Постепенно уменьшаем задержку до тех
пор, пока по всем маршрутам зареджка станет не больше Topt.
        double dTopt=0.001; //с ;Шаг уменьшения задержки.
```

```
        setlocale (LC_ALL,"");
    /*   Выделение памяти для matrix и MPath. Ввод matrix - матрицы
расстояний из файла*/
        ifstream file("matrix.txt");
        int numberOfVert;
        file >> numberOfVert;
        cout << numberOfVert << endl;

        //Матрица смежности с весами ребер графа(101 - ребра нет, 0
ребро в себя)
        double                          **matrix                          =
(double**)malloc(sizeof(double)*numberOfVert);
        for(int i = 0; i < numberOfVert; i++) {
            matrix[i]    =    (double    *)    malloc(sizeof(double)    *
numberOfVert);
        }
        //Считываем матрицу весов ребер
        for(int i = 0; i < numberOfVert; i++) {
            for(int  j  =  0;  j  <  numberOfVert;  j++)  {  file  >>
matrix[i][j];
            }
        }
        file.close();
        cout << "Distance matrix:" << endl;
        printMatrix(matrix, numberOfVert);
        //выделение памяти под матрицу маршрутов
        int     **MPath=(int     **)malloc(sizeof(int*)*numberOfVert);
//матрица маршрутизации не нужна для Флойда, но нужна для курсовика
        for(int i=0; i<numberOfVert; i++)
        {
            MPath[i]=(int*)malloc(sizeof(int)*numberOfVert);
        }

    /*  Получение в matrix матрицы кратчайших путей. и в MPath матрицы
маршрутов одновременно */
        originalFloydWarshall(matrix, numberOfVert, MPath);
        cout << "Shortest distance matrix(Floyd):" << endl;
        printMatrix(matrix, numberOfVert);
        printMatrixToFile((char      *)"matrixFloyd.txt",      matrix,
numberOfVert);
        cout << "Route matrix(Floyd):" <<endl;
        printMatrix(MPath, numberOfVert, 1);
        printMatrixToFile((char  *)"MPath.txt",  MPath,  numberOfVert,
1);

    /*  Ввод матрицы интенсивности трафика в направлениях связи*/
        double
**MNodeToNodeLoad=(double**)malloc(sizeof(double*)*numberOfVert);
        for(int i=0; i<numberOfVert; i++)
        {
```

```
MNodeToNodeLoad[i]=(double*)malloc(sizeof(double)*numberOfVert);
        }
        ifstream file1("MNodeToNodeLoad.txt");
        for(int i=0; i<numberOfVert; i++)
        {
            for(int j=0; j<numberOfVert; j++)
            {
                file1>>MNodeToNodeLoad[i][j];
            }
        }
        cout << "MNodeToNodeLoad:" <<endl;
        printMatrix(MNodeToNodeLoad, numberOfVert); //проверка
    /*  Расчет матрицы интенсивности нагрузок на линии связи*/
        double
**MEdgeLoad=(double**)malloc(sizeof(double*)*numberOfVert);
        for(int i=0; i<numberOfVert; i++)
        {
            MEdgeLoad[i]=(double*)malloc(sizeof(double)*numberOfVert);
        }
        for(int i=0; i<numberOfVert; i++)
        {
            for(int j=0; j<numberOfVert; j++)
            {
                MEdgeLoad[i][j]=0;
            }
        }
        cout << "MEdgeLoad:" <<endl;
        printMatrix(MEdgeLoad,numberOfVert);
        EdgeLoad(MNodeToNodeLoad, MPath, numberOfVert, MEdgeLoad);
        cout << "MEdgeLoad:" <<endl;
        printMatrix(MEdgeLoad,numberOfVert);
        printMatrixToFile((char                    *)"MEdgeLoad.txt",
MEdgeLoad,numberOfVert);
    /*  Матрица потоков*/
        int **MThreatNumber=(int**)malloc(sizeof(int*)*numberOfVert);
        for(int i=0; i<numberOfVert; i++)
        {
            MThreatNumber[i]=(int*)malloc(sizeof(int)*numberOfVert);
        }
        ThreatNumber(MEdgeLoad, numberOfVert, 1-q , MThreatNumber);
        cout << "MThreatNumber:" <<endl;
        printMatrix(MThreatNumber, numberOfVert);
        printMatrixToFile((char  *)"MThreatNumber.txt",  MThreatNumber,
numberOfVert);
     //        ThreatNumberBruteForce(MEdgeLoad,   numberOfVert,   0.02,
MThreatNumber);
     //   cout << "MThreatNumberBruteForce:" <<endl;
    //    printMatrix(MThreatNumber, numberOfVert);

    /*  7 пункт. Матрица интенсивности трафика ПД для линий связи*/
```

```
        double       **MTrafficIntensity=(double**)malloc(sizeof(double
*)*numberOfVert);
        for(int i=0; i<numberOfVert; i++)
        {

MTrafficIntensity[i]=(double*)malloc(sizeof(double)*numberOfVert);
        }
        TrafficIntensity(MThreatNumber,        numberOfVert,        a0,
MTrafficIntensity);
        cout << "MTrafficIntensity:" <<endl;
        printMatrix(MTrafficIntensity, numberOfVert);
        printMatrixToFile((char              *)"MTrafficIntensity.txt",
MTrafficIntensity, numberOfVert);
    /*  8 пункт. Матрица пропускной способности линий связи*/
        double     **MCommunLineCapacity=(double**)malloc(sizeof(double
*)*numberOfVert);
        for(int i=0; i<numberOfVert; i++)
        {

MCommunLineCapacity[i]=(double*)malloc(sizeof(double)*numberOfVert);
        }
        CommunLineCapacity(MTrafficIntensity,  numberOfVert,  L*8,  T0,
MCommunLineCapacity);
        cout << "MCommunLineCapacity:" <<endl;
        printMatrix(MCommunLineCapacity, numberOfVert);
        printMatrixToFile((char            *)"MCommunLineCapacity.txt",
MCommunLineCapacity, numberOfVert);

    /*      9   пункт.   Оптимизация   пропускной   способности   линий
связи==========================================================================
==================*/
    /*  Выделение памяти*/
        //MB0   матрица,   в   которой   будут   оптимальные   пропускные
способности. Выделение памяти.
        double **MB0=(double**)malloc(sizeof(double *)*numberOfVert);
        for(int i=0; i<numberOfVert; i++)
        {
            MB0[i]=(double*)malloc(sizeof(double)*numberOfVert);
        }
        //MDel матрица задержек Tij на линиях связи
        double **MDel=(double**)malloc(sizeof(double *)*numberOfVert);
        for(int i=0; i<numberOfVert; i++)
        {
            MDel[i]=(double*)malloc(sizeof(double)*numberOfVert);
        }
        //MDl матрица задержек dlij между узлами по маршрутам.
        double **MDl=(double**)malloc(sizeof(double *)*numberOfVert);
        for(int i=0; i<numberOfVert; i++)
        {
            MDl[i]=(double*)malloc(sizeof(double)*numberOfVert);
        }
        //MO матрица значений целевой функции
```

```
        double **MO=(double**)malloc(sizeof(double *)*numberOfVert);
        for(int i=0; i<numberOfVert; i++)
        {
            MO[i]=(double*)malloc(sizeof(double)*numberOfVert);
        }



        ToptCurrent=Topt+dTopt; //для избежания ветвления в цикле.


        do{ //цикл для того чтобы значения в MDl не превышали Topt
        ToptCurrent-=dTopt;
    /*  Инициализация MB0*/
        for(int i=0;i< numberOfVert; i++)
        {
            for(int j=0; j<numberOfVert; j++)
            {
                MB0[i][j]=MCommunLineCapacity[i][j];
            }
        }
        cout<<"MB0 init:"<<endl;
        printMatrix(MB0, numberOfVert);


    /*  Вычисление MO*/
        OCalc(MB0,   MTrafficIntensity,   MPath,   numberOfVert,   L,
ToptCurrent, dc, MO);
        cout<<"MO firstcalc:"<<endl;
        printMatrix(MO, numberOfVert);

        //ищем OminCurrent;
        OminCurrentCalc(MO,     numberOfVert,     &iOmin,     &jOmin,
&OminCurrent);

        int counter=0; //просто счетчик для проверки
        do{
            OminPrevious=OminCurrent;
            iOminPrevious=iOmin;   //нужны   только   для   получения
промежуточных матриц для ответа.
            jOminPrevious=jOmin;
            MB0[iOmin][jOmin]+=dc;
         //   cout<<"iOmin=" <<iOmin <<"jOmin="<< jOmin<<endl;
            /*  Вычисление MO*/
            OCalc(MB0,  MTrafficIntensity,  MPath,  numberOfVert,  L,
ToptCurrent, dc, MO);

            //ищем OminCurrent;
            OminCurrentCalc(MO,    numberOfVert,    &iOmin,    &jOmin,
&OminCurrent);
```

```
      //           cout<<"MO"<<   counter <<":"<< " OminPrevious="<<
OminPrevious<<" OminCurrent=" << OminCurrent<<endl;
        //  printMatrix(MO, numberOfVert);
            counter++;

        }while(OminCurrent<OminPrevious);

        //Тут нам нужно получить в MDl матрицу, соответствующую
минимальному значению MO. Этому значению соответствует текущее
состояние MB0, поэтому считаем задержки по этой матрице.
                    //Вычисляем MDel
                for(int i=0; i< numberOfVert; i++)
                {
                    for(int j=0; j<numberOfVert; j++)
                    {
                        if(MB0[i][j]>0)  //если линия используется в
маршрутах
                        {
                       //      if(i==iO && j==jO) //здесь уже не нужно
делать добавки. MB0 содержит уже оптимальные пропускные способности, и
мы просто находим по ним задержки.
                        //     {
                        //                  MDel[i][j]=L*8/(MB0[i][j]+dc-
MTrafficIntensity[i][j]);
                        //     }
                        //     else
                           //  {
                                 MDel[i][j]=L*8/(MB0[i][j]-
MTrafficIntensity[i][j]); //это можно вынести вне цикла
                             //  }
                        }
                        else
                        {
                            MDel[i][j]=0;
                        }
                    }
                }
         //       cout<<"MDel:"<<endl;
           //     printMatrix(MDel, numberOfVert);
                //Вычисляем MDl
                for(int i=0; i<numberOfVert; i++)
                {
                    for(int   j=0;   j<numberOfVert;   j++)   //вычисляем
задержку для каждого маршрута
                    {
                        int k1=i;
                        int k2=MPath[i][j];
                        MDl[i][j]=0;        //инициализация
                        MDl[i][j]+=MDel[k1][k2];
                        while(k2!=j)
                        {
                            k1=k2;
```

```
                            k2=MPath[k2][j];
                            MDl[i][j]+=MDel[k1][k2];
                        }
                    }
                }

                cout<<"MDl:"<<endl;
                printMatrix(MDl, numberOfVert);
            //   system("pause"); //для отладки
                cout<<"ToptCurrent="<<ToptCurrent<<endl;

        }while(IsGreaterThen(MDl, numberOfVert, Topt));

    /*  Вывод результатов 9 пункта*/
        cout<<"MB0:"<<endl;
        printMatrix(MB0, numberOfVert);
        printMatrixToFile((char *)"MB0.txt", MB0, numberOfVert);

        cout<<"MDel:"<<endl;
        printMatrix(MDel, numberOfVert);
        printMatrixToFile((char *)"MDel.txt", MDel, numberOfVert);

        cout<<"MDl:"<<endl;
        printMatrix(MDl, numberOfVert);
        printMatrixToFile((char *)"MDl.txt", MDl, numberOfVert);

        /*  Вычисление MO с минимальным значением из всех достигнутых
для ответа*/
        MB0[iOminPrevious][jOminPrevious]-=dc;              //Матрица,
соответствующая предыдущему состоянию.
        OCalc(MB0, MTrafficIntensity, MPath, numberOfVert, L, Topt,
dc, MO);

        cout<<"MOPrevious    (have    minimum    value    which    have
achived):"<<endl;
        printMatrix(MO, numberOfVert);
        printMatrixToFile((char *)"MOPrevious.txt", MO, numberOfVert);
//пишем матрицу, в которой содержится наименьшее достигнутое значение
целевой функции.

        //ищем OminCurrent;
        OminCurrentCalc(MO,     numberOfVert,     &iOmin,     &jOmin,
&OminCurrent);

        cout<<"Omin="<<OminCurrent<<"  In  cell(counting  from  1):"<<
iOmin+1<< " "<<jOmin+1 <<endl;
        ofstream ofile("MOPrevious.txt", ios_base::app);
        ofile<<"Omin="<<OminCurrent<<"  In  cell(counting  from  1):"<<
iOmin+1<< " "<<jOmin+1 <<endl; //пишем минимальное значение функции и
индекс ячейки, в которой оно достигается.
        cout<<"ToptCurrent="<<ToptCurrent<<endl;
        ofile<<"ToptCurrent="<<ToptCurrent<<endl;
```

```
    ofile.close();

/*  Освобождение памяти*/
    for(int i=0;i<numberOfVert;i++)
    {
        free(matrix[i]);
        free(MPath[i]);
        free(MNodeToNodeLoad[i]);
        free(MEdgeLoad[i]);
        free(MThreatNumber[i]);
        free(MTrafficIntensity[i]);
        free(MCommunLineCapacity[i]);
        //Для 9 пункта
        free(MB0[i]);
        free(MDel[i]);
        free(MDl[i]);
        free(MO[i]);
    }
    free(matrix);
    free(MPath);
    free(MNodeToNodeLoad);
    free(MEdgeLoad);
    free(MThreatNumber);
    free(MTrafficIntensity);
    free(MCommunLineCapacity);
    //Для 9 пункта
    free(MB0);
    free(MDl);
    free(MDel);
    free(MO);
    return 0;
}
```