Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

по лабораторной работе № 6
по дисциплине «Теория систем и системный анализ»

Тема: «Построение сетевого графа работ и его анализ методом критического пути (CPM)»

Вариант 12

Выполнил:
Мишков А.О.,
студент группы ИУ8-31

Проверил: Коннова Н.С.,
доцент каф. ИУ8

г. Москва,
2020 г.

1. Цель работы

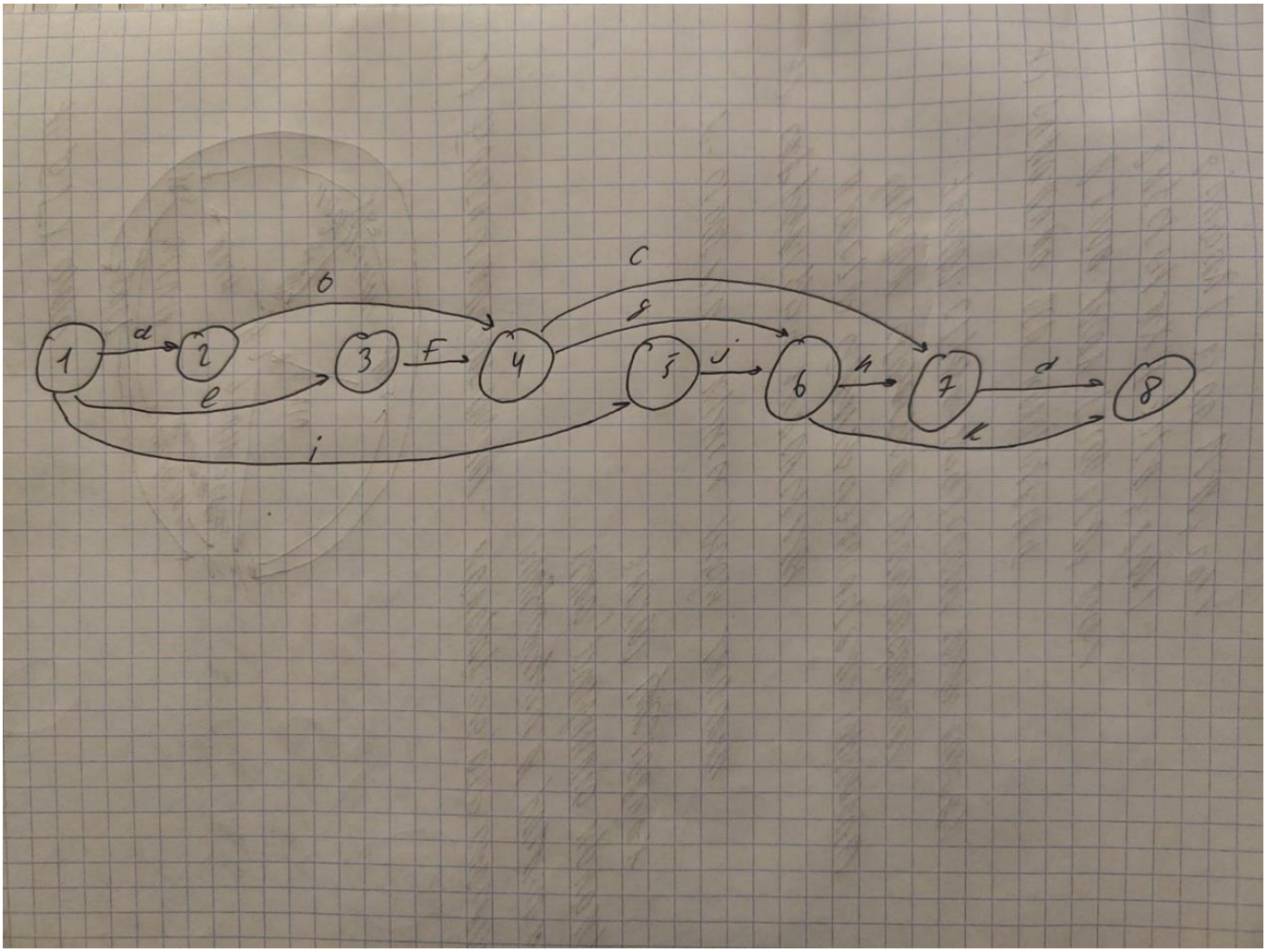
Изучить задачи сетевого планирования в управлении проектами и приобрести навыки их решения при помощи метода критического пути.

2. Условие задачи

Вариант № 12.

Задан набор работ с множествами непосредственно предшествующих работ (по варианту).

1. Построить сетевой граф, произвести его топологическое упорядочение и нумерацию.
2. Рассчитать и занести в таблицу поздние сроки начала и ранние сроки окончания работ.
3. Рассчитать и занести в таблицу ранние и поздние сроки наступления событий.
4. Рассчитать полный и свободный резервы времени работ.
5. Рассчитать резерв времени событий, определить и выделить на графе критический путь.



	a	b	c	d	e	f	g	h	i	j	k
t	3	5	2	4	3	1	4	3	3	2	5

12	∅	a	b,f	c,h	∅	e	b,f	g,j	∅	i	g,j

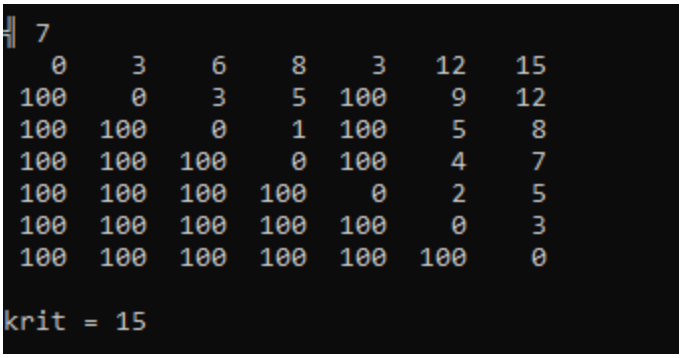
3. Таблицы

	Tpj	Tnj	Rj
1	0	0	0
2	3	3	0
3	3	3	0
4	4	8	4
5	3	3	0
6	5	12	7
7	8	11	3
8	12	15	3

	tij	tпнij	tpoij	rПij	rcij
12	3	3	0	0	0
13	3	3	0	0	0
15	3	3	0	0	0
24	5	8	3	-4	0
34	1	4	7	0	4
46	4	8	8	-7	4
47	2	6	9	-2	5
56	2	5	10	0	7
67	3	8	8	-7	3
68	5	15	10	-5	5
78	4	12	11	-3	3

4. Результат работы программ

```
k0
   0    3  100  100    3  100  100
 100    0    3    5  100  100  100
 100  100    0    1  100  100  100
 100  100  100    0  100    4    2
 100  100  100  100    0    2  100
 100  100  100  100  100    0    3
 100  100  100  100  100  100    0

k 1
   0    3  100  100    3  100  100
 100    0    3    5  100  100  100
 100  100    0    1  100  100  100
 100  100  100    0  100    4    2
 100  100  100  100    0    2  100
 100  100  100  100  100    0    3
 100  100  100  100  100  100    0

k 2
   0    3    6    8    3  100  100
 100    0    3    5  100  100  100
 100  100    0    1  100  100  100
 100  100  100    0  100    4    2
 100  100  100  100    0    2  100
 100  100  100  100  100    0    3
 100  100  100  100  100  100    0

k 3
   0    3    6    8    3  100  100
 100    0    3    5  100  100  100
 100  100    0    1  100  100  100
 100  100  100    0  100    4    2
 100  100  100  100    0    2  100
 100  100  100  100  100    0    3
 100  100  100  100  100  100    0

k 4
   0    3    6    8    3   12   10
 100    0    3    5  100    9    7
 100  100    0    1  100    5    3
 100  100  100    0  100    4    2
 100  100  100  100    0    2  100
 100  100  100  100  100    0    3
 100  100  100  100  100  100    0

k 5
   0    3    6    8    3   12   10
 100    0    3    5  100    9    7
 100  100    0    1  100    5    3
 100  100  100    0  100    4    2
 100  100  100  100    0    2  100
 100  100  100  100  100    0    3
 100  100  100  100  100  100    0

k 6
   0    3    6    8    3   12   15
 100    0    3    5  100    9   12
 100  100    0    1  100    5    8
 100  100  100    0  100    4    7
 100  100  100  100    0    2    5
 100  100  100  100  100    0    3
 100  100  100  100  100  100    0
```



Крит путь : 1-2-4-6-7-8 15

5. Выводы

В результате работы были изучены задачи сетевого планирования в управлении проектами и приобретены навыки их решения при помощи метода критического пути.

6. Ответ на контрольный вопрос

Какие исходные данные необходимы для использования метода критического пути?

Для использования метода критического пути нужно знать длительность работ и множество предшествующих работ для каждой работы.

Приложение 1. Исходный код программы

```
#include <iostream>
#include<iomanip>
#include <map>
#include <vector>
#include <algorithm>

struct gr {
public:
   int n;
   std::map<gr, int> start;
   std::map<gr, int> finish;

   gr(const int& n) : n(n) {}
   bool nu(const gr& n);
   int pol(const gr& n);
   friend bool operator<(const gr& lev, const gr& pra)
   {
      return lev.n < pra.n;
   }

};
int gr::pol(const gr& n)
{
   for (const auto& i : finish) {
      if (i.first.n == n.n) return i.second;
   }
   return 0;
}

void ris(gr& lev, gr& pra, int w) {
   lev.finish.insert(std::make_pair(pra, w));
   pra.start.insert(std::make_pair(lev, w));
}

void print(const std::vector<std::vector<int>>& mat) {
   for (const auto& i : mat) {
      for (const auto& m : i) {
         std::cout << ' ' << std::setw(3) << m << std::setw(1) << ' ';
      }
      std::cout << std::endl;
   }
   std::cout << std::endl;
}

int FloydAlgorithm(std::vector<gr>& gr) {
   int inf = 100;
   std::vector<std::vector<int>> mat(gr.size());
   for (auto& i : mat) {
      i.resize(gr.size());
   }
   for (int i = 0; i < gr.size(); ++i) {
      for (int j = 0; j < gr.size(); ++j) {
         if (i == j) {
            mat[i][j] = 0;
```

```
            }
            else if (gr[i].nu(gr[j])) {
                mat[i][j] = gr[i].pol(gr[j]);
            }
            else {
                mat[i][j] = inf;
            }
        }
    }
    std::cout << "№0" << std::endl;
    print(mat);

    for (int k = 0; k < gr.size(); ++k) {
        for (int i = 0; i < gr.size(); ++i) {
            for (int j = 0; j < gr.size(); ++j) {
                if (mat[i][k] != inf && mat[k][j] != inf) {
                    mat[i][j] = (mat[i][j] == inf ? mat[i][k] + mat[k][j] :
                        std::max(mat[i][j], mat[i][k] + mat[k][j]));
                }
            }
        }
        std::cout << "№ " << k + 1<< std::endl;
        print(mat);
    }
    return mat[0][gr.size() - 1];
}

bool gr::nu(const gr& n) {
    for (const auto& i : finish) {
        if (i.first.n == n.n) return true;
    }
    return false;
}



int main()
{
    int a = 3, b = 5, f = 1, c = 2,  h = 3, e = 3, d = 4, g = 4, j = 2, i = 3, k = 5;
    gr n1(1); gr n2(2); gr n3(3); gr n4(4); gr n5(5); gr n6(6); gr n7(7); gr n8(8);
    ris(n1, n2, a); ris(n2, n3, e); ris(n2, n4, b); ris(n3, n4, f); ris(n1, n5, i); ris(n5, n6, j); ris(n4, n6,
g); ris(n6, n7, h); ris(n4, n7, c); ris(n6, n8, k); ris(n7, n8, d);
    std::vector<gr> gr = { n1, n2, n3, n4, n5, n6, n7 };
    int l = FloydAlgorithm(gr);
    std::cout << "krit = " << l << std::endl;
    return 0;
}
```