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 7

Выполнил: Кидинова Д.Д.,
студент группы ИУ8-31

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

г. Москва,
2020 г.

1. Цель работы

Изучить задачи сетевого планированияв управлении проектами и приобрести навыки их решения при помощиметода критического пути.

2. Условие задачи

Вариант № 7.

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.

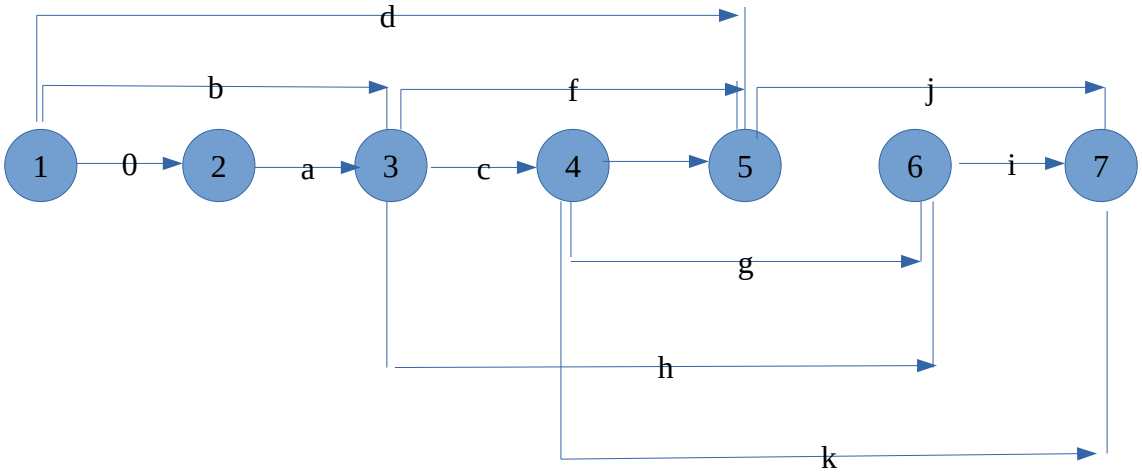

Таблица 1. Длительность работ.

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

Таблица 2. Множества предшествующих работ.

вариант	P_a	P_b	P_c	P_d	P_e	P_f	P_g	P_h	P_i	P_j	P_k
7	-	-	a, b	-	b, c	a	b, c	a	h, g	f, e, d	b, c

3. Таблицы

Таблица 3. Параметры событий.

j	**1**	**2**	**3**	**4**	**5**	**6**	**7**
T_j^p	0	0	5	7	10	11	14
T_j^n	0	2	5	7	12	11	14
$R_j = T_j^p - T$	0	2	0	0	2	0	0

Таблица 4. Параметры работ.

	t_{ij}	t_{ij}^{po}	$t_{ij}^{пн}$	r_{ij}^c	r_{ij}^n
1-2	0	0	2	-2	2
2-3	3	3	2	3	5
1-3	5	5	0	5	5
3-4	2	7	5	2	2
4-5	3	10	9	3	5
3-5	1	6	11	5	7
1-5	4	4	8	10	12
4-6	4	11	7	4	4
3-6	3	8	8	6	6
4-7	5	12	9	7	7
6-7	3	14	11	3	3
5-7	2	12	12	2	4

4. Результат работы программы

С помощью алгоритма Флойда можно найти длину критического пути:

Matrix for iteration 0 :

0	0	5	999	4	999	999
999	0	3	999	999	999	999
999	999	0	2	1	3	999
999	999	999	0	3	4	5
999	999	999	999	0	999	2
999	999	999	999	999	0	3
999	999	999	999	999	999	0

Matrix for iteration 1 :

0	0	5	999	4	999	999
999	0	3	999	999	999	999
999	999	0	2	1	3	999
999	999	999	0	3	4	5
999	999	999	999	0	999	2
999	999	999	999	999	0	3
999	999	999	999	999	999	0

Matrix for iteration 2 :

0	0	5	999	4	999	999

999	0	3	999	999	999	999
999	999	0	2	1	3	999
999	999	999	0	3	4	5
999	999	999	999	0	999	2
999	999	999	999	999	0	3
999	999	999	999	999	999	0

Matrix for iteration 3 :

0	0	5	7	6	8	999
999	0	3	5	4	6	999
999	999	0	2	1	3	999
999	999	999	0	3	4	5
999	999	999	999	0	999	2
999	999	999	999	999	0	3
999	999	999	999	999	999	0

Matrix for iteration 4 :

0	0	5	7	10	11	12
999	0	3	5	8	9	10
999	999	0	2	5	6	7

999	999	999	0	3	4	5
999	999	999	999	0	999	2
999	999	999	999	999	0	3
999	999	999	999	999	999	0

Matrix for iteration 5 :

0	0	5	7	10	11	12
999	0	3	5	8	9	10
999	999	0	2	5	6	7
999	999	999	0	3	4	5
999	999	999	999	0	999	2
999	999	999	999	999	0	3
999	999	999	999	999	999	0

Matrix for iteration 6 :

0	0	5	7	10	11	14
999	0	3	5	8	9	12
999	999	0	2	5	6	9
999	999	999	0	3	4	7
999	999	999	999	0	999	2
999	999	999	999	999	0	3

999	999	999	999	999	999	0

Matrix for iteration 7 :

0	0	5	7	10	11	**14**
999	0	3	5	8	9	12
999	999	0	2	5	6	9
999	999	999	0	3	4	7
999	999	999	999	0	999	2
999	999	999	999	999	0	3
999	999	999	999	999	999	0

Critical path length = 14

Действительно, путь 1-3-4-6-7 имеет наибольшую длину.

5. Выводы

Результаты работы совпали с результатами, полученными аналитически с помощью метода критического пути.

6. Ответ на контрольный вопрос

2.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 «Задача 1»

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

using std::map;
using std::vector;
using std::cout;

struct Node {
    size_t num;
    map<Node, size_t> input;
    map<Node, size_t> output;

    explicit Node(const size_t &n) : num(n) {}

    bool FindOutputNode(const Node &n) {
        for (const auto &node: output) {
            if (node.first.num == n.num) return true;
        }
        return false;
    }

    size_t GetWeightToOutputNode(const Node &n) {
        for (const auto &node : output) {
            if (node.first.num == n.num) return node.second;
        }
        return 0;
    }

    friend bool operator<(const Node &left, const Node &right) {
        return left.num < right.num;
    }

};

void Pair(Node &left, Node &right, const size_t &weight) {
    left.output.insert(std::make_pair(right, weight));
    right.input.insert(std::make_pair(left, weight));
}

void PrintMatrix(const vector<vector<size_t>> &matrix) {
    for (const auto &str:matrix) {
        for (const auto &el:str) {
            cout << el << "\t\t";
        }
        cout << '\n';
    }
```

```
    cout << '\n';
}

size_t FloydAlgorithm(std::vector<Node> &graph) {
    size_t infinity = 999;//std::numeric_limits<size_t>::max();
    vector<vector<size_t>> matrix(graph.size());
    for (auto &str:matrix) {
        str.resize(graph.size());
    }
    for (size_t i = 0; i < graph.size(); ++i) {
        for (size_t j = 0; j < graph.size(); ++j) {
            if (i == j) {
                matrix[i][j] = 0;
            } else if (graph[i].FindOutputNode(graph[j])) {
                matrix[i][j] = graph[i].GetWeightToOutputNode(graph[j]);
            } else {
                matrix[i][j] = infinity;
            }
        }
    }
    cout << "Matrix for iteration 0 :\n";
    PrintMatrix(matrix);

    for (size_t k = 0; k < graph.size(); ++k) {
        for (size_t i = 0; i < graph.size(); ++i) {
            for (size_t j = 0; j < graph.size(); ++j) {
                if (matrix[i][k] != infinity && matrix[k][j] != infinity) {
                    matrix[i][j] = //std::min(matrix[i][j], matrix[i][k] + matrix[k][j]);
                            (matrix[i][j] == infinity ? matrix[i][k] + matrix[k][j] :
                             std::max(matrix[i][j], matrix[i][k] + matrix[k][j]));
                }
            }
        }
        cout << "Matrix for iteration " << k + 1 << " :\n";
        PrintMatrix(matrix);
    }
    return matrix[0][graph.size() - 1];
}


int main() {
    size_t a = 3, b = 5, c = 2, d = 4, e = 3,
           f = 1, g = 4, h = 3, i = 3, j = 2, k = 5;
    Node n1(1);
    Node n2(2);
    Node n3(3);
    Node n4(4);
    Node n5(5);
    Node n6(6);
    Node n7(7);
    Pair(n1, n2, 0);
    Pair(n2, n3, a);
```

```
    Pair(n1, n3, b);
    Pair(n3, n4, c);
    Pair(n4, n5, e);
    Pair(n3, n5, f);
    Pair(n1, n5, d);
    Pair(n4, n6, g);
    Pair(n3, n6, h);
    Pair(n6, n7, i);
    Pair(n5, n7, j);
    Pair(n4, n7, k);
    std::vector<Node> graph = {n1, n2, n3, n4, n5, n6, n7};

    size_t len = FloydAlgorithm(graph);
    std::cout << "Critical path length = " << len << std::endl;

    return 0;
}
```