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 4

Выполнила: Бояркина Е.Р.,
студент группы ИУ8-31

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

г. Москва, 2020 г.

Цель работы

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

Условие задачи

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
3	5	2	4	3	1	4	3	3	2	5

Таблица 1 – Длительности работ.

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

Таблица 2 – Множества предшествующих работ.

Решение задачи

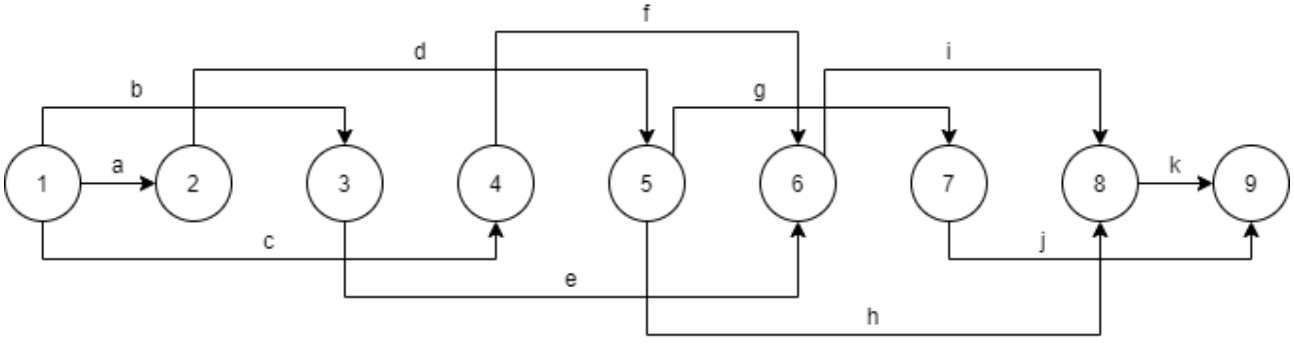

Рисунок 1 – Сетевой граф задачи.

Рассчитанные программой ранние и поздние сроки наступления событий, резерв времени событий:

Number	T_r	T_p	R
1	0	0	0
2	3	4	1
3	5	5	0
4	2	7	5
5	7	8	1
6	8	8	0
7	11	14	3
8	11	11	0
9	16	16	0

Рассчитанные программой поздние сроки начала и ранние сроки окончания работ, а также полный и свободный резервы времени работ:

Number	Length	Previous	t_ro	t_pn	r_p	r_s
1-2	3	-	3	1	1	0
1-3	5	-	5	0	0	0
1-4	2	-	2	5	5	0
2-5	4	1-2	7	4	1	-1
3-6	3	1-3	8	5	0	0
4-6	1	1-4	3	7	5	0
5-7	4	2-5	11	10	3	-1
5-8	3	2-5	10	8	1	0
6-8	3	3-6, 4-6	11	8	0	0
7-9	2	5-7	13	14	3	0
8-9	5	5-8, 6-8	16	11	0	0

Найденная по алгоритму Флойда длина критического пути:

```
Matrix on iteration 0:
| -999    3    5    2 -999 -999 -999 -999 -999  |
| -999 -999 -999 -999    4 -999 -999 -999 -999  |
| -999 -999 -999 -999 -999    3 -999 -999 -999  |
| -999 -999 -999 -999 -999    1 -999 -999 -999  |
| -999 -999 -999 -999 -999 -999    4    3 -999  |
| -999 -999 -999 -999 -999 -999 -999    3 -999  |
| -999 -999 -999 -999 -999 -999 -999 -999    2  |
| -999 -999 -999 -999 -999 -999 -999 -999    5  |
| -999 -999 -999 -999 -999 -999 -999 -999 -999  |

Matrix on iteration 1:
| -999    3    5    2 -999 -999 -999 -999 -999  |
| -999 -999 -999 -999    4 -999 -999 -999 -999  |
| -999 -999 -999 -999 -999    3 -999 -999 -999  |
| -999 -999 -999 -999 -999    1 -999 -999 -999  |
| -999 -999 -999 -999 -999 -999    4    3 -999  |
| -999 -999 -999 -999 -999 -999 -999    3 -999  |
```

```
| -999 -999 -999 -999 -999 -999 -999 -999    2  |
| -999 -999 -999 -999 -999 -999 -999 -999    5  |
| -999 -999 -999 -999 -999 -999 -999 -999 -999  |

Matrix on iteration 2:
| -999    3    5    2    7 -999 -999 -999 -999  |
| -999 -999 -999 -999    4 -999 -999 -999 -999  |
| -999 -999 -999 -999 -999    3 -999 -999 -999  |
| -999 -999 -999 -999 -999    1 -999 -999 -999  |
| -999 -999 -999 -999 -999 -999    4    3 -999  |
| -999 -999 -999 -999 -999 -999 -999    3 -999  |
| -999 -999 -999 -999 -999 -999 -999 -999    2  |
| -999 -999 -999 -999 -999 -999 -999 -999    5  |
| -999 -999 -999 -999 -999 -999 -999 -999 -999  |

Matrix on iteration 3:
| -999    3    5    2    7    8 -999 -999 -999  |
| -999 -999 -999 -999    4 -999 -999 -999 -999  |
| -999 -999 -999 -999 -999    3 -999 -999 -999  |
| -999 -999 -999 -999 -999    1 -999 -999 -999  |
| -999 -999 -999 -999 -999 -999    4    3 -999  |
| -999 -999 -999 -999 -999 -999 -999    3 -999  |
| -999 -999 -999 -999 -999 -999 -999 -999    2  |
| -999 -999 -999 -999 -999 -999 -999 -999    5  |
| -999 -999 -999 -999 -999 -999 -999 -999 -999  |

Matrix on iteration 4:
| -999    3    5    2    7    8 -999 -999 -999  |
| -999 -999 -999 -999    4 -999 -999 -999 -999  |
| -999 -999 -999 -999 -999    3 -999 -999 -999  |
| -999 -999 -999 -999 -999    1 -999 -999 -999  |
| -999 -999 -999 -999 -999 -999    4    3 -999  |
| -999 -999 -999 -999 -999 -999 -999    3 -999  |
| -999 -999 -999 -999 -999 -999 -999 -999    2  |
| -999 -999 -999 -999 -999 -999 -999 -999    5  |
| -999 -999 -999 -999 -999 -999 -999 -999 -999  |

Matrix on iteration 5:
| -999    3    5    2    7    8   11   10 -999  |
| -999 -999 -999 -999    4 -999    8    7 -999  |
| -999 -999 -999 -999 -999    3 -999 -999 -999  |
| -999 -999 -999 -999 -999    1 -999 -999 -999  |
| -999 -999 -999 -999 -999 -999    4    3 -999  |
| -999 -999 -999 -999 -999 -999 -999    3 -999  |
| -999 -999 -999 -999 -999 -999 -999 -999    2  |
| -999 -999 -999 -999 -999 -999 -999 -999    5  |
| -999 -999 -999 -999 -999 -999 -999 -999 -999  |

Matrix on iteration 6:
| -999    3    5    2    7    8   11   11 -999  |
| -999 -999 -999 -999    4 -999    8    7 -999  |
| -999 -999 -999 -999 -999    3 -999    6 -999  |
```

```
| -999 -999 -999 -999 -999    1 -999    4 -999  |
| -999 -999 -999 -999 -999 -999    4    3 -999  |
| -999 -999 -999 -999 -999 -999 -999    3 -999  |
| -999 -999 -999 -999 -999 -999 -999 -999    2  |
| -999 -999 -999 -999 -999 -999 -999 -999    5  |
| -999 -999 -999 -999 -999 -999 -999 -999 -999  |

Matrix on iteration 7:
| -999    3    5    2    7    8   11   11   13  |
| -999 -999 -999 -999    4 -999    8    7   10  |
| -999 -999 -999 -999 -999    3 -999    6 -999  |
| -999 -999 -999 -999 -999    1 -999    4 -999  |
| -999 -999 -999 -999 -999 -999    4    3    6  |
| -999 -999 -999 -999 -999 -999 -999    3 -999 |
| -999 -999 -999 -999 -999 -999 -999 -999    2  |
| -999 -999 -999 -999 -999 -999 -999 -999    5  |
| -999 -999 -999 -999 -999 -999 -999 -999 -999  |

Matrix on iteration 8:
| -999    3    5    2    7    8   11   11   16  |
| -999 -999 -999 -999    4 -999    8    7   12  |
| -999 -999 -999 -999 -999    3 -999    6   11  |
| -999 -999 -999 -999 -999    1 -999    4    9  |
| -999 -999 -999 -999 -999 -999    4    3    8  |
| -999 -999 -999 -999 -999 -999 -999    3    8  |
| -999 -999 -999 -999 -999 -999 -999 -999    2  |
| -999 -999 -999 -999 -999 -999 -999 -999    5  |
| -999 -999 -999 -999 -999 -999 -999 -999 -999  |

Matrix on iteration 9:
| -999    3    5    2    7    8   11   11   16  |
| -999 -999 -999 -999    4 -999    8    7   12  |
| -999 -999 -999 -999 -999    3 -999    6   11  |
| -999 -999 -999 -999 -999    1 -999    4    9  |
| -999 -999 -999 -999 -999 -999    4    3    8  |
| -999 -999 -999 -999 -999 -999 -999    3    8  |
| -999 -999 -999 -999 -999 -999 -999 -999    2  |
| -999 -999 -999 -999 -999 -999 -999 -999    5  |
| -999 -999 -999 -999 -999 -999 -999 -999 -999  |

Critical path's length: 16
```

Действительно, путь 1-3-6-8-9 имеет наибольшую длину, равную 16.

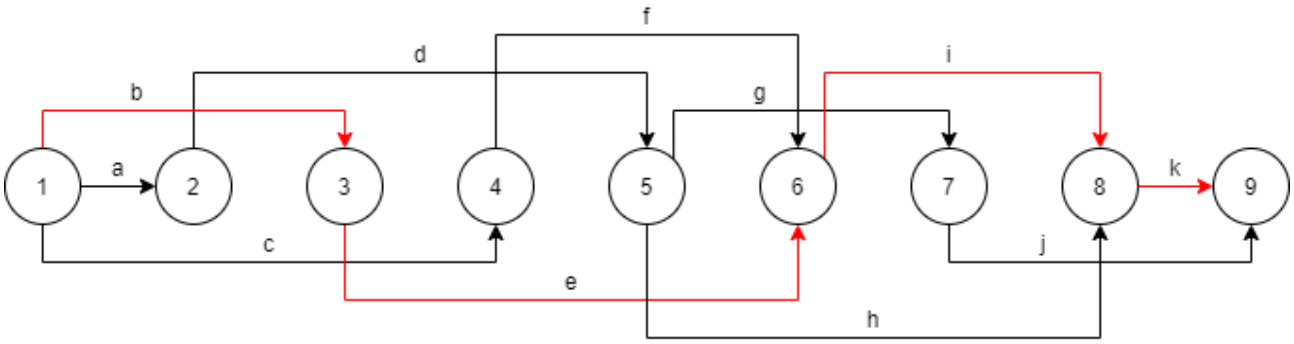

Рисунок 2 – Критический путь (выделен красным).

Выводы

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

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. Исходный код программы

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

struct GraphNode {                          // Event structure
    int number = 0;                         // Number of node
    int T_r = -99;                          // Early time of doing
    int T_p = -99;                          // Late time of doing
    int R = -99;                            // Time reserve
};

bool operator ==(const GraphNode& lhs, const GraphNode& rhs) {
    return lhs.number == rhs.number;
}

struct GraphLine {                          // Work structure
    GraphNode& start;                       // Starting event for this work
    GraphNode& finish;                      // Finish event for this work
    int length;                             // Length of work
    std::vector<GraphLine> previous;        // Previous works
    int t_ro = -99;                         // Time of early ending
    int t_pn = -99;                         // Time of late starting
    int r_p = -99;                          // Full reserve of time
    int r_s = -99;                          // Free reserve of time
};

std::vector<GraphLine> createGraph(std::vector<GraphNode>& events) {
    /* This function create graph from my variant task */

    std::vector<GraphLine> graph = {
            {events[0], events[1], 3}, {events[0], events[2], 5},
            {events[0], events[3], 2}, {events[1], events[4], 4},
            {events[2], events[5], 3}, {events[3], events[5], 1},
            {events[4], events[6], 4}, {events[4], events[7], 3},
            {events[5], events[7], 3}, {events[6], events[8], 2},
            {events[7], events[8], 5}
    };
    for (size_t i = 0; i < graph.size(); ++i) {
        for (size_t j = i; j < graph.size(); ++j) {
            if (graph[i].finish == graph[j].start) {
                graph[j].previous.push_back(graph[i]);
            }
        }
    }

    return graph;
}

void evaluateGraph(std::vector<GraphNode>& events, std::vector<GraphLine>& graph) {
    /* This function evaluate parameters of graph from my variant task */

    // Straight movement : find early time
    events[0].T_r = 0;
    while (events[events.size() - 1].T_r == -99) {
        for (auto& work : graph) {
            if (work.start.T_r != -99) {
                work.t_ro = work.start.T_r + work.length;
            }
```

```
        }
        for (auto& event : events) {
            for (auto& work : graph) {
                if (work.finish == event && work.t_ro != -99) {
                    event.T_r = std::max(event.T_r, work.t_ro);
                }
            }
        }
    }

    // Reversed movement
    events[events.size() - 1].T_p = events[events.size() - 1].T_r;
    while (events[0].T_p == -99) {
        for (auto& work : graph) {
            if (work.finish.T_p != -99) {
                work.t_pn = work.finish.T_p - work.length;
            }
        }
        for (auto& event : events) {
            for (auto& work : graph) {
                if (work.start == event && work.t_pn != -99) {
                    if (event.T_p == -99) event.T_p = work.t_pn;
                    else event.T_p = std::min(event.T_p, work.t_pn);
                }
            }
        }
    }

    // Model analyze
    for (auto& event : events) {
        event.R = event.T_p - event.T_r;
    }
    for (auto& work : graph) {
        work.r_p = work.finish.T_p - work.start.T_r - work.length;
        work.r_s = work.finish.T_r - work.start.T_p - work.length;
    }
}

void printMatrix(const std::vector<std::vector<int>>& matrix) {
    /* This function print matrix from Floyd algorithm */

    for (const auto& line : matrix) {
        std::cout << "| ";
        for (const auto item : line) {
            std::cout << std::setw(4) << item << ' ';
        }
        std::cout << " |" << std::endl;
    }
}

int algorithmFloyda(const std::vector<GraphNode>& events, const std::vector<GraphLine>&
graph) {
    /* This function realize Floyd algorithm */

    std::cout << "\nFloyd algorithm: " << std::endl;
    std::vector<std::vector<int>> currentMaximalPaths;
    for (size_t i = 0; i < events.size(); ++i) {
        currentMaximalPaths.emplace_back(events.size(), -999);
    }
    for (const auto& work : graph) {
        currentMaximalPaths[work.start.number - 1][work.finish.number - 1] =
work.length;
    }
```

```
    std::cout << "\nMatrix on iteration 0: " << std::endl;
    printMatrix(currentMaximalPaths);
    for (size_t k = 0; k < currentMaximalPaths.size(); ++k) {
        auto newMinimalPath = currentMaximalPaths;
        for (size_t i = 0; i < newMinimalPath.size(); ++i) {
            for (size_t j = 0; j < newMinimalPath.size(); ++j) {
                newMinimalPath[i][j] =
                        (currentMaximalPaths[i][k] == -999 || currentMaximalPaths[k][j]
== -999) ?
                        currentMaximalPaths[i][j] : std::max(
                                currentMaximalPaths[i][j],
                                currentMaximalPaths[i][k] + currentMaximalPaths[k][j]);
            }
        }
        currentMaximalPaths = newMinimalPath;
        std::cout << "\nMatrix on iteration " << k + 1 << ": " << std::endl;
        printMatrix(currentMaximalPaths);
    }

    return currentMaximalPaths[0][events.size() - 1];
}

void printEventsTable(const std::vector<GraphNode>& events) {
    /* This function print events vector */

    std::cout << std::string(28, '-') << std::endl;
    std::cout << "| " << "Number" << " | " << "T_r" << " | " << "T_p" << " | " << "  R"
<< " |" << std::endl;
    std::cout << std::string(28, '-') << std::endl;
    for (const auto& event : events) {
        std::cout << "| " << std::setw(6) << event.number << " | "
                  << std::setw(3) << event.T_r << " | "
                  << std::setw(3) << event.T_p << " | "
                  << std::setw(3) << event.R << " |" << std::endl;
    }
    std::cout << std::string(28, '-') << std::endl;
}

void printWorksTable(const std::vector<GraphLine>& graph) {
    /* This function print works vector */

    std::cout << std::string(58, '-') << std::endl;
    std::cout << "| " << "Number" << " | " << "Length" << " | " << "Previous" << " | "
<< "t_ro" << " | "
              << "t_pn" << " | " << " r_p" << " | " << " r_s" << " |" << std::endl;
    std::cout << std::string(58, '-') << std::endl;
    for (const auto& work : graph) {
        std::string work_number = std::to_string(work.start.number) + "-" +
std::to_string(work.finish.number);
        std::cout << "| " << std::setw(6) << work_number << " | "
                  << std::setw(6) << work.length << " | ";
        std::string previousPaths;
        for (const auto& prev : work.previous) {
            if (!previousPaths.empty()) previousPaths += ", ";
            previousPaths += std::to_string(prev.start.number) + "-" +
std::to_string(prev.finish.number);
        }
        if (previousPaths.empty()) previousPaths = "-";
        std::cout << std::setw(8) << previousPaths << " | "
                  << std::setw(4) << work.t_ro << " | "
                  << std::setw(4) << work.t_pn << " | "
                  << std::setw(4) << work.r_p << " | "
                  << std::setw(4) << work.r_s << " | " << std::endl;
```

```
    }
    std::cout << std::string(58, '-') << std::endl;
}

int main() {
    std::vector<GraphNode> events = {{1}, {2}, {3}, {4}, {5},
                                     {6}, {7}, {8}, {9}};
    auto graph = createGraph(events);
    evaluateGraph(events, graph);
    std::cout << "Events: " << std::endl;
    printEventsTable(events);
    std::cout << "\nWorks: " << std::endl;
    printWorksTable(graph);
    int criticalPathLength = algorithmFloyda(events, graph);
    std::cout << "\nCritical path's length: " << criticalPathLength << std::endl;

    return 0;
```