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 3

Выполнил: Бакаев Ф. Б.,
студент группы ИУ8-31

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

Москва 2020

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.

Ход работы

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

Множества предшествующих работ для варианта №3:

№пп	P_a	P_b	P_c	P_d	P_e	P_f	P_g	P_h	P_i	P_j	P_k
3	∅	∅	a	a	d	d	d	c, b	g	f, i	e, j

Реализация алгоритма нахождения критического пути сетевого графа производилось с помощью библиотеки boost/graph.

Результат работы программы представлен на следующей фотографии.

```
/home/prostofil/labs/tsisa-lab-06/cmake-build-debug/main
Graph successfully created
Number of edges: 11
List of edges:
(1,2) (1,3) (2,3) (2,4) (3,8) (4,5) (4,6) (4,7) (5,6) (6,7) (7,8)
Path from 1 to 8
Shortest path: 1 3 8
Shortest path size: 3

Process finished with exit code 0
```

Рисунок 1 – результат работы программы

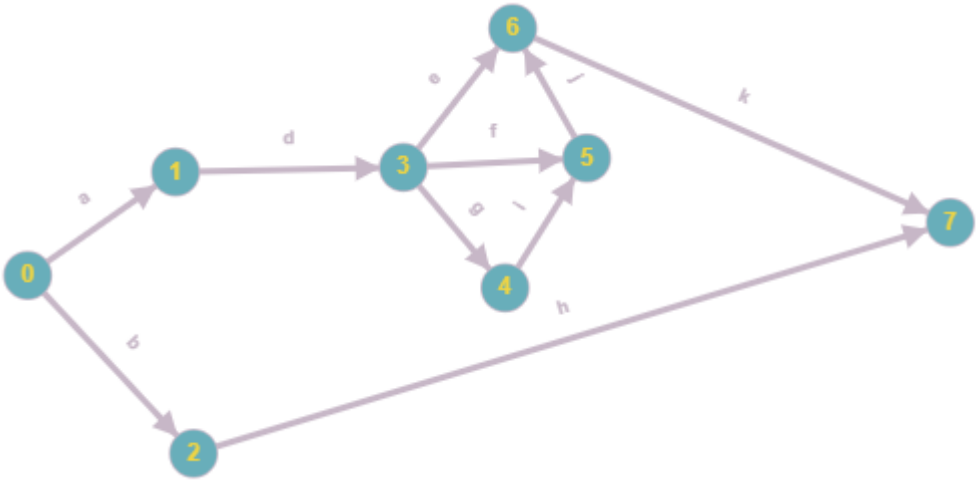

Рисунок 2 – исходный граф

Выводы

С помощью библиотеки boost/graph реализовал решение поставленной задачи. Результат работы программы совпадает со значениями, полученными аналитическим путем.

Приложение. Исходный код программы

```
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/dijkstra_shortest_paths.hpp>
#include <boost/graph/graph_traits.hpp>

#include <iostream>
#include <vector>

using boost::add_edge;
using std::cout;
using std::endl;

// Typedef for DirectedGraph
typedef boost::property<boost::edge_weight_t, int> EdgeWeightProperty;
// Declaring an adjacency list for a directed graph
typedef boost::adjacency_list<boost::listS, boost::vecS, boost::directedS,
boost::no_property, EdgeWeightProperty> DirectedGraph;
typedef boost::graph_traits<DirectedGraph>::edge_iterator edge_iterator;
typedef boost::graph_traits<DirectedGraph>::vertex_descriptor vertex_descriptor;

int main() {

    // Weights of edges
    uint a = 3;
    uint b = 5;
    uint c = 2;
    uint d = 4;
    uint e = 3;
    uint f = 1;
    uint g = 4;
    uint h = 3;
    uint i = 3;
    uint j = 2;
    uint k = 5;

    DirectedGraph graph;

    // Adding edges
    add_edge(1, 2, a, graph);
    add_edge(1, 3, b, graph);
    add_edge(2, 3, c, graph);
    add_edge(3, 8, h, graph);
    add_edge(2, 4, d, graph);
    add_edge(4, 5, g, graph);
    add_edge(4, 6, f, graph);
    add_edge(4, 7, e, graph);
    add_edge(5, 6, i, graph);
    add_edge(6, 7, j, graph);
    add_edge(7, 8, k, graph);

    // Iterator for printing
    std::pair<edge_iterator, edge_iterator> ei = edges(graph);

    cout << "Graph successfully created" << endl;
    cout << "Number of edges: " << num_edges(graph) << endl;
    cout << "List of edges: " << endl;
    for (auto it = ei.first; it != ei.second; ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    // Vectors for finding shortest path
```

```
    std::vector<vertex_descriptor> p(num_vertices(graph));
    std::vector<int> d_vert(num_vertices(graph));

    auto start = vertex(1, graph);
    auto final = vertex(8, graph);

    boost::dijkstra_shortest_paths(graph, start,
boost::predecessor_map(&p[0]).distance_map(&d_vert[0]));

    // Shortest path
    std::vector<boost::graph_traits<DirectedGraph>::vertex_descriptor> path;
    auto current = final;

    while (current != start) {
        path.push_back(current);
        current = p[current];
    }

    path.push_back(start);

    cout << "Path from " << 1 << " to " << 8 << endl;
    cout << "Shortest path: ";
    for (auto it = path.rbegin(); it != path.rend(); it++) {
        cout << *it << " ";
    }
    cout << endl << "Shortest path size: " << path.size() << endl;

    return 0;
}
```

Ответ на контрольный вопрос

1. Какие основные данные необходимы для использования метода критического пути?

Для использования метода критического пути необходимы следующие данные:

1) список работ проекта;

2) связи между работами;

3) длительность выполнения работ;

4) календарный план рабочего времени;

5) календарный срок начала проекта.