|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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: Быкова П. Е.,

студентка группы ИУ8-32

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. Постановка задачи

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.

**3. Ход работы**

Таблица 1. Расшифровка работ

|  |  |  |
| --- | --- | --- |
| **№ работы** | **Длительность работы** | **Предшествующие работы** |
| 1-2 | 2 | - |
| 2-3 | 3 | 1-2 |
| 1-4 | 5 | - |
| 3-7 | 2 | 2-3 |
| 4-5 | 3 | 1-4 |
| 4-6 | 3 | 1-4 |
| 7-8 | 3 | 3-7 |
| 5-8 | 2 | 4-5 |
| 8-9 | 3 | 7-8, 5-8 |
| 6-9 | 5 | 4-6 |
| 9-10 | 4 | 8-9, 6-9 |

Таблицы 2,3. Найденные параметры работ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **T** | **t-ПН** | **t-PO** | **r-П** | **r-С** |
| 1-2 | 1 | 0 | 2 | 1 | 1 |
| 2-3 | 2 | 2 | 5 | 1 | 1 |
| 1-4 | 4 | 0 | 5 | 0 | 1 |
| 3-7 | 2 | 5 | 7 | 0 | 0 |
| 4-5 | 4 | 4 | 8 | 0 | 0 |
| 4-6 | 3 | 5 | 8 | 0 | 1 |
| 7-8 | 3 | 7 | 10 | 0 | 0 |
| 5-8 | 2 | 8 | 10 | 0 | 0 |
| 8-9 | 2 | 10 | 13 | 1 | 1 |
| 6-9 | 5 | 8 | 13 | 0 | 0 |
| 9-10 | 4 | 13 | 17 | 0 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **t-P** | **t-П** | **R** |
| 1 | 0 | 0 | 0 |
| 2 | 2 | 2 | 0 |
| 3 | 5 | 5 | 0 |
| 4 | 5 | 4 | 1 |
| 5 | 8 | 8 | 0 |
| 6 | 8 | 8 | 0 |
| 7 | 7 | 7 | 0 |
| 8 | 10 | 10 | 0 |
| 9 | 13 | 13 | 0 |
| 10 | 17 | 17 | 0 |

# 4. Выводы

По результатам лабораторной работы длина критического пути равна 17, что совпало с аналитическим значением.

**5. Контрольный вопрос.**

*Вопрос:* *Какие исходные данные необходимы для использования метода критического пути?*

*Ответ: Необходимо задать следующие параметры работ: список работ, их продолжительность, связи между работами, календарный срок начала проекта = 0, директивный срок окончания не задан. Таким образом, в нашем распоряжении имеются все необходимые данные для применения метода критического пути.*

**Приложение 1. Исходный код программы «Нейронные сети»**

*Файл* ***Critical\_path.cpp***

#include <iostream>

#include <vector>

#include <algorithm>

// объявляем переменную, которая будет отвечать за размер нашего графа-матрицы

const int N = 10;

// метод, который печатает наш граф-матрицу

void Print\_Graf(const std::vector<std::vector<int>>& Graf)

{

// "бежим" по всем строчкам и столбцам графа-матрицы:

for (const auto& x : Graf) {

for (const auto& y : x) {

// если встречаем обозначение "пути нет", то заменяем его знаком "-" для удобства чтения

if (y == INT\_MAX)

std::cout << "-" << " ";

else

// иначе, если такого обозначения не встречается, то просто записываем расстояние между соответствующими вершинами

std::cout << y << " ";

}

std::cout << std::endl;

}

std::cout << std::endl;

}

// метод, возвращающий критический путь между начальной и конечной вершинами

int Search\_of\_Critical\_Path(std::vector<std::vector<int>>& Graf)

{

// "бежим" по всему графу-матрице и смотрим конкретные строки и столбцы "K":

for (int k = 0; k < N; k++) {

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

{

// ставим условия, чтобы путь между двумя просматриваемыми вершинами "существовал" и

// чтобы эти две вершины графа были различны, то есть не рассматриваем пути вершимы от

// самой себя до себя

if ((i != j)&&(Graf[i][k] != INT\_MAX)&&(Graf[k][j] != INT\_MAX))

{

// ищем как бы "пустые/несуществующие" места между вершинами:

if (Graf[i][j] == INT\_MAX)

// если все условия выполнены, то с каждым разом складываем найденные пути

// и записываем их в "пустые" места

Graf[i][j] = Graf[i][k] + Graf[k][j];

else

// иначе нужно посмотреть, какой путь больше: тот, который получится сложением или тот, который

// представлен на данный момент

Graf[i][j] = std::max(Graf[i][j], Graf[i][k] + Graf[k][j]);

}

}

}

// возвращаем наибольший путь между вершинами "0-начало" и "9-конец"

return Graf[0][9];

}

int main()

{

// создаём наш граф в виде матрицы

// столбцы слева-направо: путь между вершинами [1][1], [1][2], [1][3],...

// допустим, путь от 1-1 = 0, 1-2 = 2 и т. д.

// строчки сверху-вниз: путь между вершинами [1][1], [2][1], [3][1],...

// допустим, путь от 1-1 = 0, 2-1 = INT\_MAX и т. д.

std::vector<std::vector<int>> Graf =

{

// обозначение "INT\_MAX" означает, что из такой-то вершины в такую-то пути не существует

{0,2,INT\_MAX,5,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX},

{INT\_MAX,0,3,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX},

{INT\_MAX,INT\_MAX,0,INT\_MAX,INT\_MAX,INT\_MAX,2,INT\_MAX,INT\_MAX,INT\_MAX},

{INT\_MAX,INT\_MAX,INT\_MAX,0,3,3,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX},

{INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,0,INT\_MAX,INT\_MAX,2,INT\_MAX,INT\_MAX},

{INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,0,INT\_MAX,INT\_MAX,5,INT\_MAX},

{INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,0,3,INT\_MAX,INT\_MAX},

{INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,0,3,INT\_MAX},

{INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,0,4},

{INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,0}

};

// печатаем изначальный граф

Print\_Graf(Graf);

// ищем критический путь

int critical\_path = Search\_of\_Critical\_Path(Graf);

// печатаем уже изменённый граф

Print\_Graf(Graf);

std::cout << "Critical path: " << critical\_path << std::endl;

}

**Приложение 2. Ссылка на git-репозиторий**

https://github.com/PolkaBBB/TSISA-lab06