## 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:

Николаева Е.Д., 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.

1. **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.

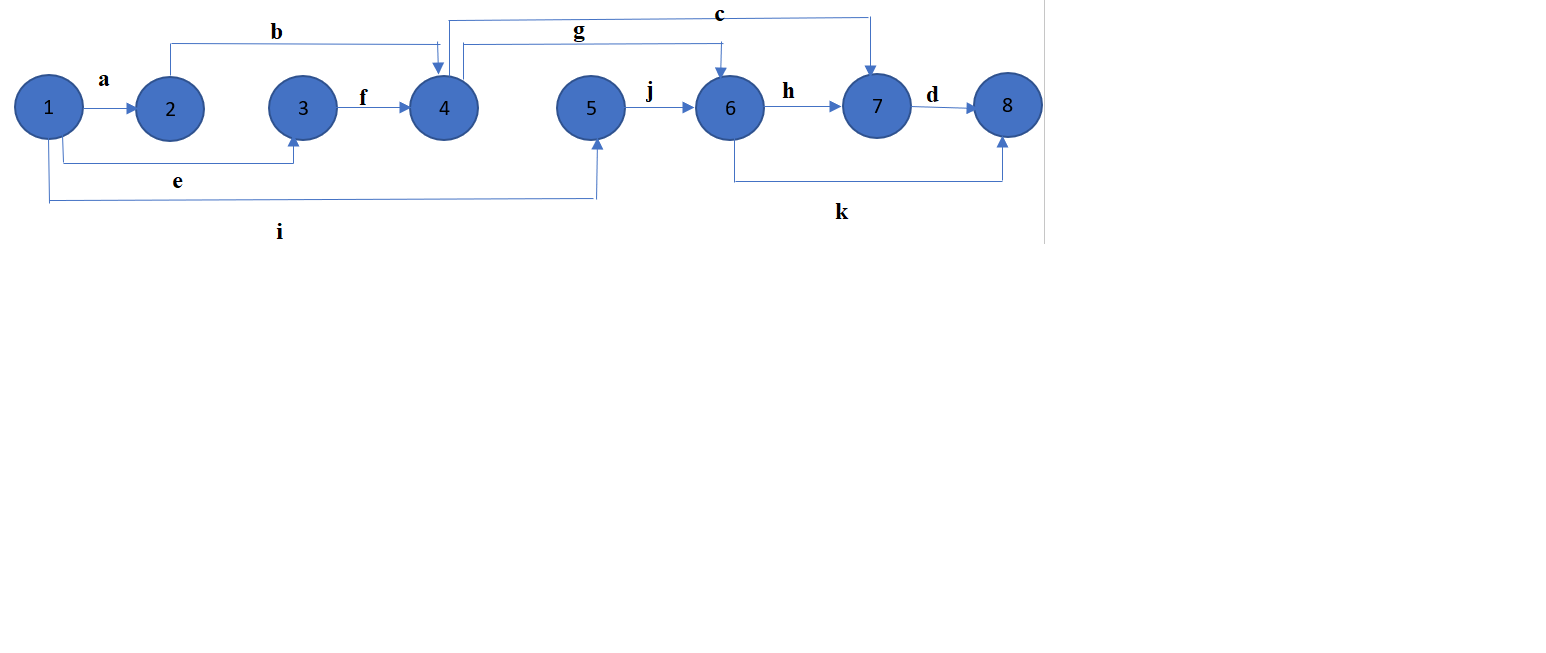


Таблица 1. Длительность работ.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **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.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *Pa* | *Pb* | *Pc* | *Pd* | *Pe* | *Pf* | *Pg* | *Ph* | *Pi* | *Pj* | *Pk* |
|  | - | a | b,f | c,h | - | e | b, f | g,j | - | i | g, j |

**3.** **Таблицы**

Таблица 3. Параметры событий.

**Number**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| 0 | 3 | 3 | 4 | 3 | 5 | 8 | 12 |
| 0 | 3 | 3 | 8 | 3 | 12 | 11 | 15 |
| 0 | 0 | 0 | 4 | 0 | 7 | 3 | 3 |

*p j*

*T*

*п j*

*T*

*R* =*Tp*−*T*

*j j*

Таблица 4. Параметры работ.

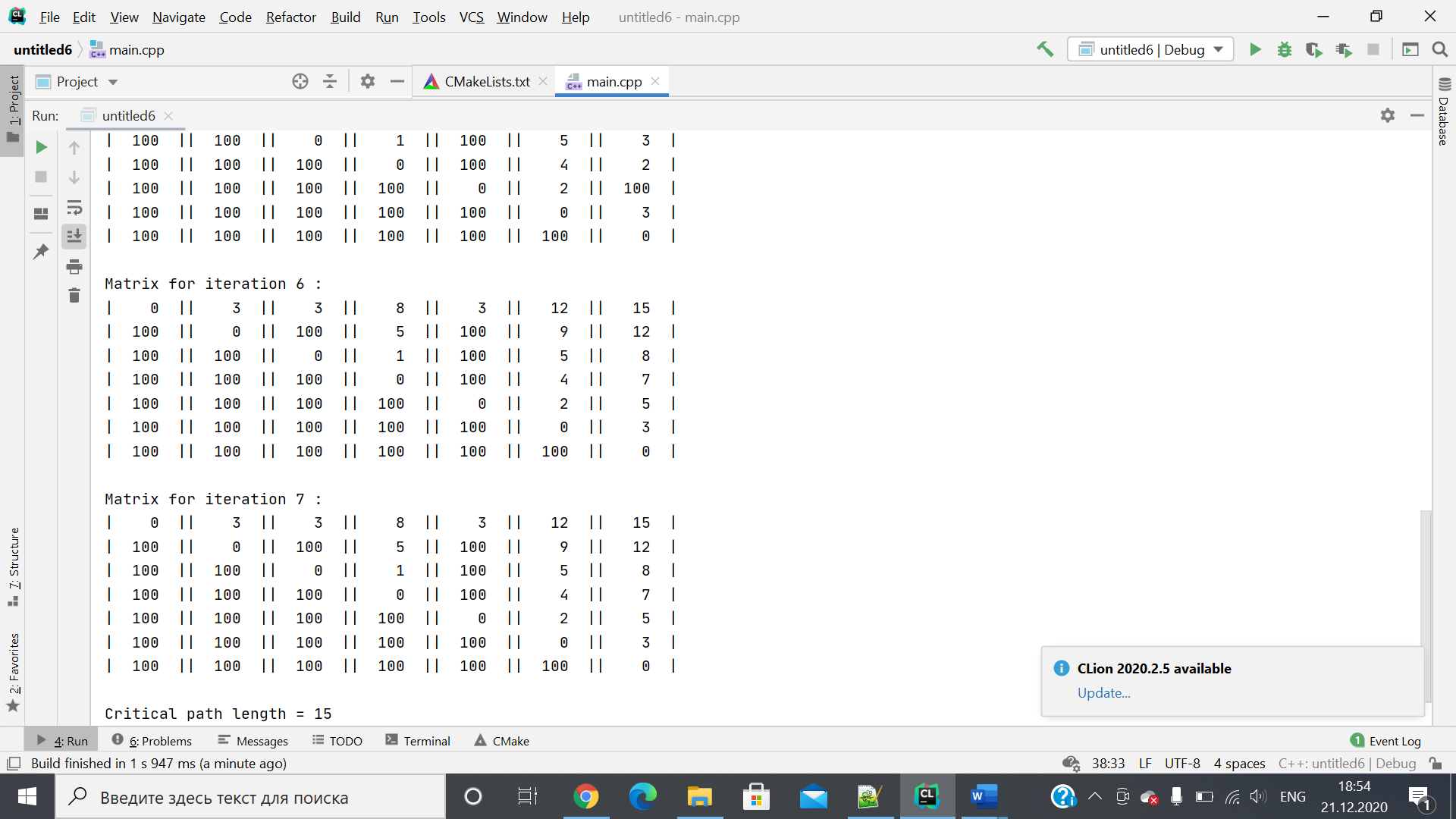
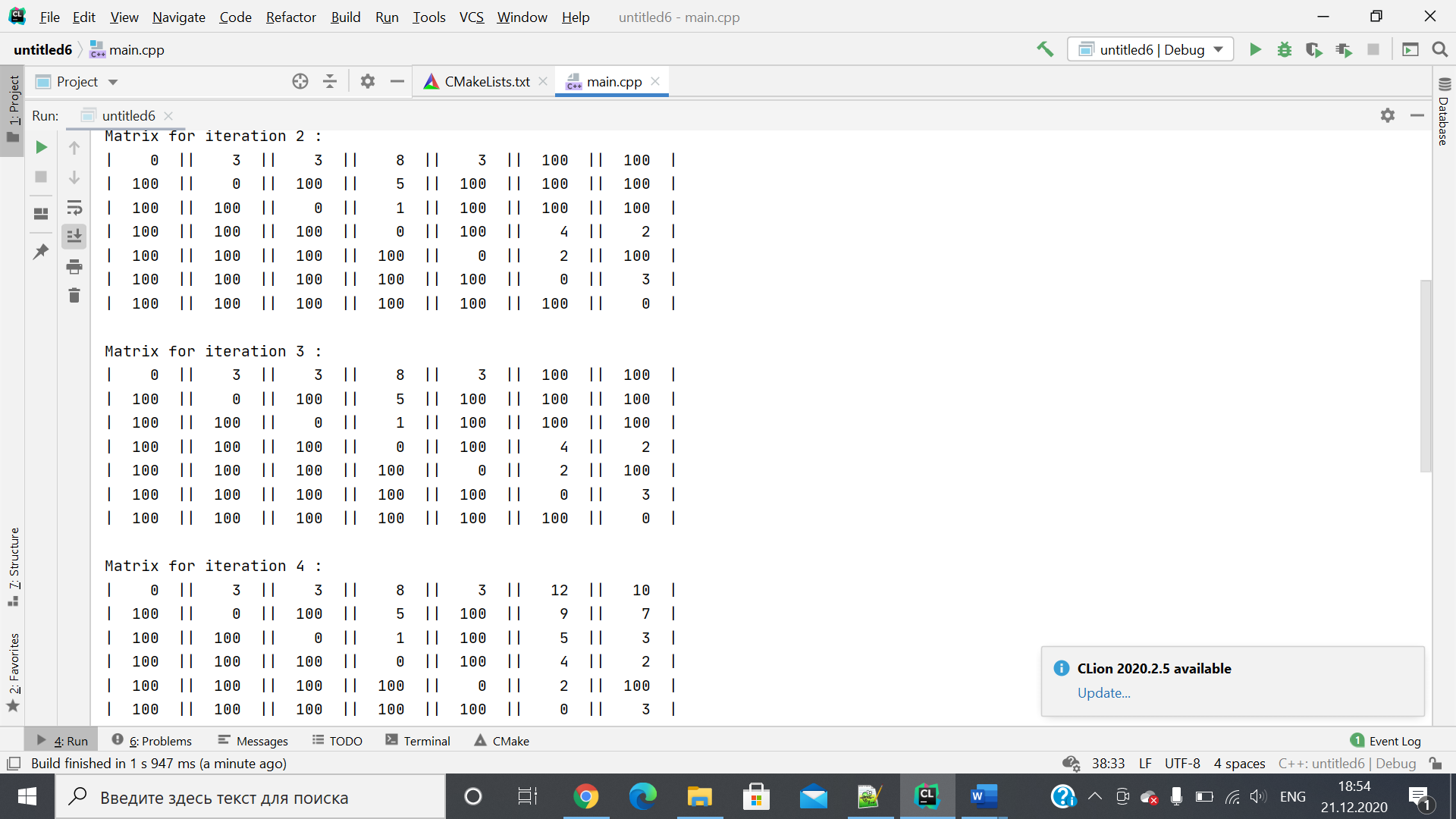
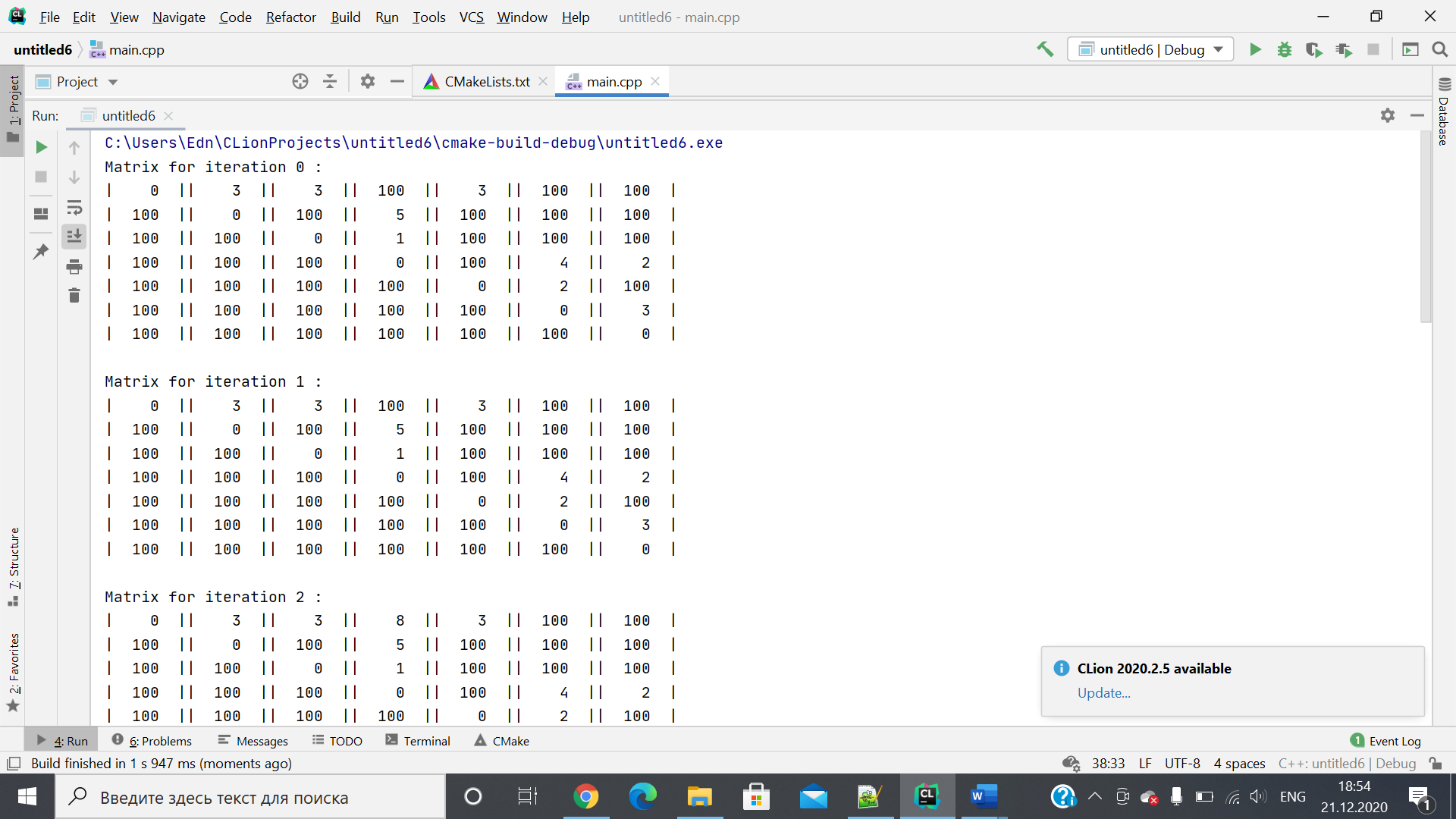
*tij t ро t пн rc rп*

*ij ij ij ij*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1-2** | 3 | 3 | 0 | 0 | 0 |
| **2-4** | 5 | 8 | 3 | -4 | 0 |
| **1-3** | 3 | 3 | 0 | 0 | 0 |
| **1-5** | 3 | 3 | 0 | 0 | 0 |
| **3-4** | 1 | 4 | 7 | 0 | 4 |
| **4-6** | 4 | 8 | 8 | -7 | 4 |
| **5-6** | 2 | 5 | 10 | 0 | 7 |
| **4-7** | 2 | 6 | 9 | -2 | 5 |
| **6-7** | 3 | 8 | 8 | -7 | 3 |
| **7-8** | 4 | 12 | 11 | -3 | 3 |
| **6-8** | 5 | 15 | 10 | -5 | 5 |
|  |  |  |  |  |  |

# Результат работы программы

С помощью алгоритма Флойда была найдена длина критического пути:



1-2-4-6-7-8 - критический путь

* 1. **Выводы**

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

* 1. **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<iomanip>

#include <map>

#include <vector>

#include <algorithm>

|  |
| --- |
| struct Graph { |
|  | public: |
|  | int num; |
|  | std::map<Graph, int> start; |
|  | std::map<Graph, int> finish; |
|  |  |
|  | Graph(const int& n) : num(n) {} |
|  | bool numb(const Graph& n) ; |
|  | int getw(const Graph& n) ; |
|  | friend bool operator<(const Graph& lhs, const Graph& rhs) |
|  | { |
|  | return lhs.num < rhs.num; |
|  | } |
|  |  |
|  | }; |
|  | int Graph::getw(const Graph& n) |
|  | { |
|  | for (const auto &i : finish) { |
|  | if (i.first.num == n.num) return i.second; |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  | bool Graph::numb(const Graph &n) { |
|  | for (const auto &i: finish) { |
|  | if (i.first.num == n.num) return true; |
|  | } |
|  | return false; |
|  | } |
|  |  |
|  | void make\_gr(Graph &lhs, Graph &rhs, int w) { |
|  | lhs.finish.insert(std::make\_pair(rhs, w)); |
|  | rhs.start.insert(std::make\_pair(lhs, w)); |
|  | } |
|  |  |
|  | void print (const std::vector<std::vector<int>> &matrix) { |
|  | for (const auto &i:matrix) { |
|  | for (const auto &el:i) { |
|  | std::cout << '|' << std::setw(5) << el << std::setw(3) << '|'; |
|  | } |
|  | std::cout << std::endl; |
|  | } |
|  | std::cout << std::endl; |
|  | } |
|  |  |
|  | int FloydAlgorithm(std::vector<Graph> &gr) { |
|  | int inf = 100; |
|  | std::vector<std::vector<int>> matrix(gr.size()); |
|  | for (auto &i:matrix) { |
|  | i.resize(gr.size()); |
|  | } |
|  | for (int i = 0; i < gr.size(); ++i) { |
|  | for (int j = 0; j < gr.size(); ++j) { |
|  | if (i == j) { |
|  | matrix[i][j] = 0; |
|  | } else if (gr[i].numb(gr[j])) { |
|  | matrix[i][j] = gr[i].getw(gr[j]); |
|  | } else { |
|  | matrix[i][j] = inf; |
|  | } |
|  | } |
|  | } |
|  | std::cout << "Matrix for iteration 0 :"<<std::endl; |
|  | print(matrix); |
|  |  |
|  | 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 (matrix[i][k] != inf && matrix[k][j] != inf) { |
|  | matrix[i][j] = (matrix[i][j] == inf ? matrix[i][k] + matrix[k][j] : |
|  | std::max(matrix[i][j], matrix[i][k] + matrix[k][j])); |
|  | } |
|  | } |
|  | } |
|  | std::cout << "Matrix for iteration " << k + 1 << " :"<< std::endl; |
|  | print(matrix); |
|  | } |
|  | return matrix[0][gr.size() - 1]; |
|  | } |
|  |  |
|  | int main() |
|  | { |
|  | int a = 3, b = 5, c = 2, d = 4, e = 3, f = 1, g = 4, h = 3, i = 3, j = 2, k = 5; |
|  | Graph n1(1); |
|  | Graph n2(2); |
|  | Graph n3(3); |
|  | Graph n4(4); |
|  | Graph n5(5); |
|  | Graph n6(6); |
|  | Graph n7(7); |
|  | Graph n8(8); |
|  | make\_gr(n1, n2, a); |
|  | make\_gr(n2, n3, e); |
|  | make\_gr(n2, n4, b); |
|  | make\_gr(n3, n4, f); |
|  | make\_gr(n1, n5, i); |
|  | make\_gr(n5, n6, j); |
|  | make\_gr(n4, n6, g); |
|  | make\_gr(n6, n7, h); |
|  | make\_gr(n4, n7, c); |
|  | make\_gr(n6, n8, k); |
|  | make\_gr(n7, n8, d); |
|  | std::vector<Graph> gr = {n1, n2, n3, n4, n5, n6, n7}; |
|  | int l = FloydAlgorithm(gr); |
|  | std::cout << "Critical path length = " << l << std::endl; |
|  | return 0; |
|  | } |