ГУАП

КАФЕДРА 43

|  |  |  |
| --- | --- | --- |
| ОТЧЕТ ЗАЩИЩЕН С ОЦЕНКОЙ: |  |  |

ПРЕПОДАВАТЕЛЬ:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| доцент | / |  | / |  | / | Д. В. Богданов |
| (должность, учёная степень, звание) |  | (подпись) |  | (дата защиты) |  | (инициалы, фамилия) |

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

«Расчет параметров сетевого графика матричным методом»

по дисциплине: Управление программными проектами

|  |  |  |  |
| --- | --- | --- | --- |
| РАБОТУ ВЫПОЛНИЛ (-А) СТУДЕНТ (-КА): | 4134К | / | Д. В. Самарин |
|  | (номер группы) |  | (инициалы, фамилия) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | / |  | / |  |
|  |  | (подпись студента) |  | (дата отчета) |

**Цель работы:**

Получение навыков по расчету временных параметров сетевых моделей, представленных в виде матрицы.

**Задание на лабораторную работу:**

Произвести расчет временных параметров сетевой модели матричным методом в соответствии с вариантом, представленным в приложении.

Вариант №1:

|  |
| --- |
| I, A 2  I, E 1  A, H 1  A, E 3  H, B 5  H, M 0  E, M 4  B, K 2  M, K 2  K, C 3 |

**Порядок выполнения работы:**

1) Проанализировать вариант задания.

2) Построить сетевой график.

3) Произвести нумерацию вершим сетевого графика.

4) Сформировать матричное представление сетевого графика.

5) Произвести расчет временных параметров матричным методом.

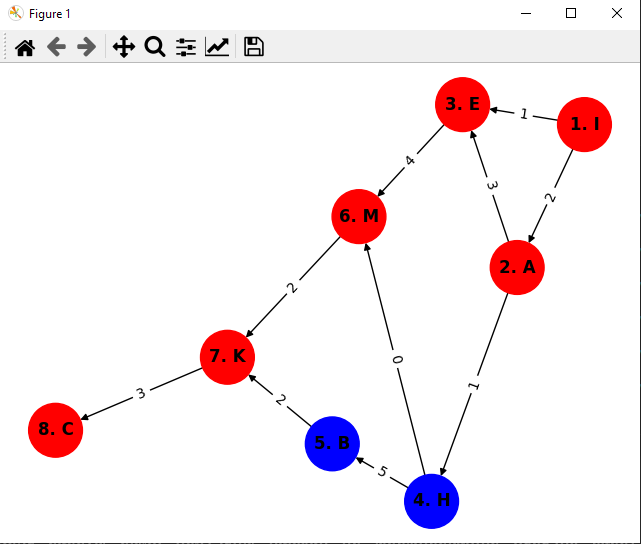
6) Оформить отчет.

7) Защитить отчет.

**Ход работы:**

Расчёты были выполнены на языке Python. В коде были реализованы методы нахождения нужных параметров.

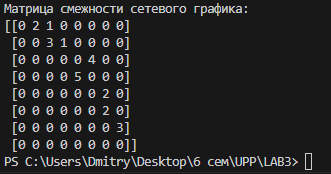
Построение сетевого графика:



На графике присутствует нумерация, данные сходятся.

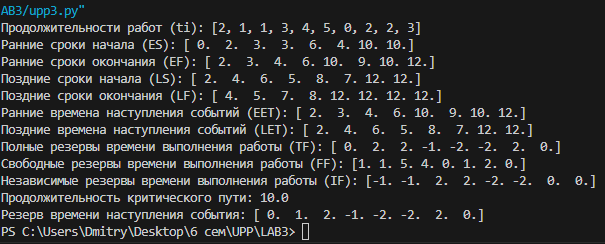
|  |
| --- |
| import networkx as nx  import matplotlib.pyplot as plt  # Создаем пустой граф  G = nx.DiGraph()  # Добавляем задачи и их зависимости  tasks = [      ("I", "A", {"weight": 2}),      ("I", "E", {"weight": 1}),      ("A", "H", {"weight": 1}),      ("A", "E", {"weight": 3}),      ("H", "B", {"weight": 5}),      ("H", "M", {"weight": 0}),      ("E", "M", {"weight": 4}),      ("B", "K", {"weight": 2}),      ("M", "K", {"weight": 2}),      ("K", "C", {"weight": 3})  ]  # Добавляем задачи и ребра в граф  G.add\_edges\_from(tasks)  # Присваиваем номера и буквенные обозначения вершинам  node\_labels = {node: f"{i+1}. {node}" for i, node in enumerate(G.nodes)}  # Раскрасим вершины в зависимости от критического пути  critical\_path = nx.dag\_longest\_path(G)  node\_colors = ["red" if node in critical\_path else "blue" for node in G.nodes]  # Рисуем граф  pos = nx.spring\_layout(G)  nx.draw(G, pos, with\_labels=True, labels=node\_labels, node\_color=node\_colors, node\_size=1500, font\_size=12, font\_weight="bold")  edge\_labels = nx.get\_edge\_attributes(G, "weight")  nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_color="black")  # Показываем граф  plt.title("Сетевой график с нумерацией и буквенными обозначениями вершин")  plt.show() |

Сформируем матричное представление сетевого графика:



|  |
| --- |
| import numpy as np  import networkx as nx  # Создаем пустой граф  G = nx.DiGraph()  # Добавляем задачи и их зависимости  tasks = [      ("I", "A", {"weight": 2}),      ("I", "E", {"weight": 1}),      ("A", "H", {"weight": 1}),      ("A", "E", {"weight": 3}),      ("H", "B", {"weight": 5}),      ("H", "M", {"weight": 0}),      ("E", "M", {"weight": 4}),      ("B", "K", {"weight": 2}),      ("M", "K", {"weight": 2}),      ("K", "C", {"weight": 3})  ]  # Добавляем задачи и ребра в граф  G.add\_edges\_from(tasks)  # Получаем матрицу смежности  adjacency\_matrix = nx.adjacency\_matrix(G).todense()  # Выводим матрицу смежности  print("Матрица смежности сетевого графика:")  print(adjacency\_matrix) |

Произведем расчет временных параметров матричным методом:



|  |
| --- |
| import numpy as np  import networkx as nx  # Создаем пустой граф  G = nx.DiGraph()  # Добавляем задачи и их зависимости  tasks = [      ("I", "A", {"weight": 2}),      ("I", "E", {"weight": 1}),      ("A", "H", {"weight": 1}),      ("A", "E", {"weight": 3}),      ("H", "B", {"weight": 5}),      ("H", "M", {"weight": 0}),      ("E", "M", {"weight": 4}),      ("B", "K", {"weight": 2}),      ("M", "K", {"weight": 2}),      ("K", "C", {"weight": 3})  ]  # Добавляем задачи и ребра в граф  G.add\_edges\_from(tasks)  # Получаем матрицу смежности  adjacency\_matrix = nx.adjacency\_matrix(G).todense()  # Расчет временных параметров матричным методом  def calculate\_parameters(adjacency\_matrix):      num\_tasks = len(adjacency\_matrix)      ES = np.zeros(num\_tasks)  # Ранние сроки начала выполнения задач      EF = np.zeros(num\_tasks)  # Ранние сроки окончания выполнения задач      LS = np.zeros(num\_tasks)  # Поздние сроки начала выполнения задач      LF = np.zeros(num\_tasks)  # Поздние сроки окончания выполнения задач      EET = np.zeros(num\_tasks)  # Ранние времена наступления событий      LET = np.zeros(num\_tasks)  # Поздние времена наступления событий      TF = np.zeros(num\_tasks)  # Полные резервы времени выполнения работы      FF = np.zeros(num\_tasks)  # Свободные резервы времени выполнения работы      IF = np.zeros(num\_tasks)  # Независимые резервы времени выполнения работы      ti = np.array([e[2]['weight'] for e in G.edges(data=True)])  # Продолжительности работ      # Рассчитываем ранние сроки начала и окончания      for i in range(num\_tasks):          if np.all(adjacency\_matrix[:, i] == 0):  # Если задача не имеет предшественников              ES[i] = 0              EF[i] = ES[i] + np.max(adjacency\_matrix[i, :])          else:              ES[i] = np.max(EF[list(np.where(adjacency\_matrix[:, i] > 0)[0])])              EF[i] = ES[i] + ti[i]      # Рассчитываем поздние сроки начала и окончания      LS[num\_tasks - 1] = LF[num\_tasks - 1] = EF[num\_tasks - 1]      for i in range(num\_tasks - 2, -1, -1):          if np.any(adjacency\_matrix[i, :] > 0):  # Если у задачи есть последователи              LF[i] = np.min(LS[list(np.where(adjacency\_matrix[i, :] > 0)[0])])              LS[i] = LF[i] - ti[i]      # Рассчитываем ранние и поздние времена наступления событий      for i in range(num\_tasks):          EET[i] = EF[i]          LET[i] = LS[i]      # Рассчитываем полные резервы времени выполнения работы      for i in range(num\_tasks):          if np.any(adjacency\_matrix[i, :] > 0):  # Если у задачи есть последователи              TF[i] = LET[np.where(adjacency\_matrix[i, :] > 0)[0][0]] - EET[i] - ti[i]      # Рассчитываем свободные резервы времени выполнения работы      for i in range(num\_tasks):          if np.any(adjacency\_matrix[i, :] > 0):  # Если у задачи есть последователи              FF[i] = EET[np.where(adjacency\_matrix[i, :] > 0)[0][0]] - ES[i] - ti[i]      # Рассчитываем независимые резервы времени выполнения работы      for i in range(num\_tasks):          if np.any(adjacency\_matrix[i, :] > 0):  # Если у задачи есть последователи              IF[i] = EET[np.where(adjacency\_matrix[i, :] > 0)[0][0]] - LET[i] - ti[i]        # Рассчитываем продолжительность критического пути      critical\_path\_duration = LF[-1] - EF[0]      # Рассчитываем резерв времени наступления события      event\_reserve = LET - EET      return ES, EF, LS, LF, EET, LET, TF, FF, IF, critical\_path\_duration, event\_reserve  # Выполним расчет временных параметров  ES, EF, LS, LF, EET, LET, TF, FF, IF, critical\_path\_duration, event\_reserve = calculate\_parameters(adjacency\_matrix)  # Выведем результаты  print("Продолжительности работ (ti):", [e[2]['weight'] for e in G.edges(data=True)])  print("Ранние сроки начала (ES):", ES)  print("Ранние сроки окончания (EF):", EF)  print("Поздние сроки начала (LS):", LS)  print("Поздние сроки окончания (LF):", LF)  print("Ранние времена наступления событий (EET):", EET)  print("Поздние времена наступления событий (LET):", LET)  print("Полные резервы времени выполнения работы (TF):", TF)  print("Свободные резервы времени выполнения работы (FF):", FF)  print("Независимые резервы времени выполнения работы (IF):", IF)  print("Продолжительность критического пути:", critical\_path\_duration)  print("Резерв времени наступления события:", event\_reserve) |

В выполненной лабораторной работе присутствуют исходные данные (в виде выданного варианта). Итоговые результаты сформированы в отчёте выше.

**Вывод:**

Выполнив работу, я получил навыки по расчету временных параметров сетевых моделей, представленных в виде матрицы.