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 №4

«Расчет параметров сетевого графика табличным 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 табличным способом.

**Задание на лабораторную работу:**

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) Сформировать табличное представление сетевого графика.

4) Произвести расчет временных параметров табличным методом.

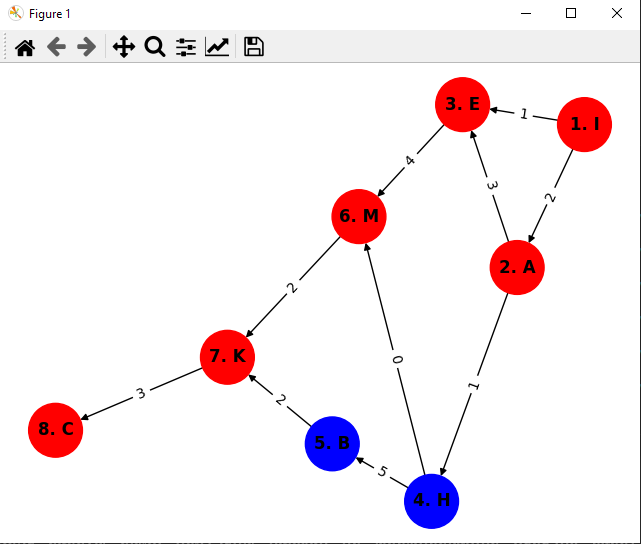
5) Оформить отчет.

6) Защитить отчет.

**Ход работы:**

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

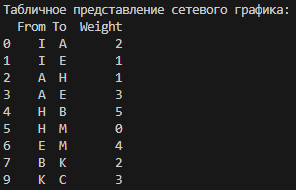
Построение сетевого графика:



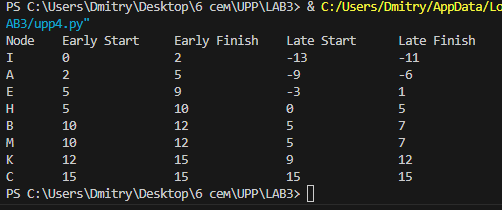
На графике присутствует нумерация, данные сходятся.

|  |
| --- |
| 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:



Произведем расчет временных параметров табличным методом:



|  |
| --- |
| class Node:      def \_\_init\_\_(self, name):          self.name = name          self.incoming\_edges = []          self.outgoing\_edges = []          self.early\_start = 0          self.early\_finish = 0          self.late\_start = 0          self.late\_finish = 0          self.duration = 0  class Edge:      def \_\_init\_\_(self, source, target, duration):          self.source = source          self.target = target          self.duration = duration  def calculate\_early\_times(node, current\_time):      if node.early\_start < current\_time:          node.early\_start = current\_time      node.early\_finish = node.early\_start + node.duration      for edge in node.outgoing\_edges:          calculate\_early\_times(edge.target, node.early\_finish)  def calculate\_late\_times(node, current\_time):      if node.late\_finish == 0 or node.late\_finish > current\_time:          node.late\_finish = current\_time      node.late\_start = node.late\_finish - node.duration      for edge in node.incoming\_edges:          calculate\_late\_times(edge.source, node.late\_start - edge.duration)  def print\_network(nodes):      print("Node\tEarly Start\tEarly Finish\tLate Start\tLate Finish")      for node in nodes:          print(f"{node.name}\t{node.early\_start}\t\t{node.early\_finish}\t\t{node.late\_start}\t\t{node.late\_finish}")  def main():      edges = [          ("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)      ]      nodes = {}      for edge in edges:          source\_name, target\_name, duration = edge          if source\_name not in nodes:              nodes[source\_name] = Node(source\_name)          if target\_name not in nodes:              nodes[target\_name] = Node(target\_name)          source\_node = nodes[source\_name]          target\_node = nodes[target\_name]          new\_edge = Edge(source\_node, target\_node, duration)          source\_node.outgoing\_edges.append(new\_edge)          target\_node.incoming\_edges.append(new\_edge)          source\_node.duration = max(source\_node.duration, duration)      start\_node = nodes["I"]      calculate\_early\_times(start\_node, 0)      calculate\_late\_times(nodes["C"], nodes["C"].early\_finish)      print\_network(nodes.values())  if \_\_name\_\_ == "\_\_main\_\_":      main() |

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 табличным способом.