«Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчёт защищён с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель Астахова А.В.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

Отчёт

Лабораторной работе №3

**«Управление проектами. Построение линейной диаграммы проекта»**

Студент группы ПИ 92 В.М. Шульпов

Преподаватель доцент, к. эк. н. Астахова А.В.

Барнаул 2022

**Задание**1. Повторить параграф 2.5 (лекция 2).  
(см. в Илиас «Лекция 2» и файл с материалами учебника: часть 2)  
2. С учетом результатов выполнения предыдущих лабораторных работ составить и  
отладить программу на выбранном языке программирования, которая отвечает  
следующим требованиям:  
– выводит на экран монитора исходный список работ;  
– выводит на экран монитора упорядоченный список работ;  
– выводит на экран монитора диаграмму Ганта, построенную по этому списку;  
– позволяет в режиме диалога вносить изменения в СГ, добавляя и удаляя работы и  
изменяя их веса на дугах графа – с получением очередной диаграммы Ганта.  
Форму выдачи диаграммы – спроектировать самостоятельно. При этом учесть  
идентификацию работ и их отображение с учетом моментов времени – по ранним срокам  
свершения событий.  
3. Оформить отчет, внеся все необходимые комментарии в программу. Защитить  
работу.

Структура графа:

self.fictive\_start\_top = -100 *# шифр фиктивной стартовой вершины*self.fictive\_end\_top = -101 *# шифр фиктивной конечной вершины*self.graph\_table = {**'arc\_start'**: [],  
 **'arc\_end'**: [],  
 **'weight'**: [],  
 **'is\_visited'**: [],  
 *# 'layer': [],* } *# таблица, содержащая дуги и веса графа*self.struct\_graph\_table = {**'arc\_start'**: [],  
 **'arc\_end'**: [],  
 **'weight'**: [],  
 **'is\_visited'**: [],  
 **'full\_time\_reserve'**: [],  
 **'independent\_time\_reserve'**: [],  
 } *# упоряденная таблица, содержащая дуги и веса графа  
# таблица, содержащая шифр события, его ранний срок, поздний срок и резервное время  
# количество входящих и выходящих из события работ*self.graph\_events = {**'event'**: [],  
 **'early\_term'**: [],  
 **'late\_term'**: [],  
 **'reserve\_time'**: [],  
 *# 'in\_works\_num': [],  
 # 'out\_works\_num': [],* **'in\_works'**: [],  
 **'out\_works'**: [],  
 **'layer'**: [],  
 **'visited'**: []  
 }  
self.first\_top = **None** *# первая вершина графа*self.last\_top = **None** *# последняя вершина графа*self.works\_num = 0 *# количество работ графа*self.struct\_graph\_works\_num = 0 *# количество работ упорядоченного графа*self.current\_way = [] *# текущий путь для вывода всех полных путей*

0) инициализация и получение исходного графа из файла происходит в конструкторе \_\_init\_\_(self, file\_in)

1) метод print\_row(self, row\_index, sorted\_graph=False)печатает строку таблицу по индексу

2) метод print\_graph(self, sorted\_graph=False)печатает всю таблицу графа, используя функцию print\_row. Есть параметр функции, который указывает какой граф выводить (упорядоченный или нет).

3) метод add\_row\_in\_graph\_table(self, start, end, weight, is\_visited=False, adding\_is\_in\_sort=False) добавляет строку в таблицу графа. Есть параметр функции, который указывает в какой граф добавлять (упорядоченный или нет). При добавлении метод увеличивает счётчик количества строк соответствующего графа на 1.

4) метод def copy\_row\_to\_struct\_graph\_table(self, index) копирует строку по индексу из неупорядоченного графа и добавляет её в конец упорядоченного. Метод вызывает метод add\_row\_in\_graph\_table.

5) метод delete\_row\_from\_graph\_table(self, index) удаляет строку из неупорядоченного графа. Уменьшает счётчик количества строк в неупорядоченном графе на 1.

6) метод search\_first\_top(self) ищет начальную вершину СГ.

Проходит во внешнем цикле по всем строкам таблицы путей графа  
Во внутреннем цикле проверяется, не является ли стартовой (в неё входит дуга, кроме петель) текущая вершина. Если найдена стартовая вершина (и не посещенная), проверяется, первая ли она такая. Если первая, то происходит запоминание её шифра и переход к следующей вершине. Если стартовая вершина уже существует, то происходит добавление фиктивной вершины (добавляется дополнительная запись в таблицу СГ.

7) метод delete\_top\_loops(self, index) проверяет строку на наличие петли и удаляет строку, если петля есть. Возвращает счётчик цикла: если было удаление, то возвращает 0, чтобы запустить цикл сначала, если не было удаления, то возвращает тот же индекс цикла и ничего не изменяется. (так как нет goto)

8) метод check\_duplication(self, i, j) проверяет наличие дублирования строки (с одинаковым и разным весом). При разных весах предлагает выбрать, какую работу удалить. При одинаковом удаляет автоматически вторую (разницы какую удалять нет). Метод аналогично возвращает счётчик цикла для запуска цикла сначала. (так как нет goto).

9) метод search\_last\_top(self) работает аналогично методу с поиском первой вершины. В нём происходит поиск последней вершины. Если вершина не одна, создается фиктивная конечная вершина.

10) метод optimize\_graph(self) вызывает методы delete\_top\_loops, check\_duplication, search\_last\_top и управляет порядком их вызова. Когда нужн повторно запустить метод, когда нет он понимает по возвращаемым значениям из этих функций (так как нет goto).

11) метод struct\_graph(self) упорядочивает граф, используя очередь (проходит граф в ширину). Новый упорядоченный граф хранится другом объекте.

12) метод search\_BFS(self) проходит по графу в ширину

Если последний элемент в текущем пути равен последнему элементу в графе, запускается цикл печати пути, удаляется последняя вершина из текущего пути и происходит выход из функции.

Иначе происходит вход в цикл( i от 0 до количества строк в таблице графа). Если стартовая вершина i-ой работы равна последней работе в текущем пути, то добавляем в текущий путь конечную вершину i-ой работы и рекурсивно запускаем search\_BFS

После этого цикла удаляется последняя вершина из текущего пути.

13) метод search\_full\_ways(self) находит все полные пути графа, вызывая рекурсивный метод recursive\_search(self). Перед этим вызовом находится первая строка, в которой хранится первая вершина. Добавляет эту вершину в текущий путь.

15) **def** append\_new\_event\_in\_events\_graph(self, event):  
 *""" добавлние новой вершины в graph\_events, инит параметров нулями """*

16) **def** swap\_events(self, i, j):  
 *""" поменять местами строки событий"""*

17) **def** sort\_events\_by\_layers(self):  
 *""" сортировка событий по слоям по возрастанию """*

18) **def** find\_early\_term\_for\_event(self, event):  
 *""" нахождение раннего срока события """*

19) **def** find\_early\_term\_for\_all\_event(self):  
 *""" нахождение раннего срока всех событий """*

20) **def** find\_late\_term\_for\_event(self, event):  
 *""" нахождение позднего срока события """*

21) **def** find\_late\_term\_for\_all\_event(self):  
 *""" нахождение позднего срока всех событий """*

22) **def** find\_time\_reserves\_for\_all\_event(self):  
 *""" нахождение резервов времени для всех событий """*

24) **def** get\_critical\_path\_length(self):  
 *""" получение критического пути из таблицы событий """*

25) **def** find\_time\_reserve\_for\_work(self, index, full=**True**):  
 *""" нахождение полного или независмого временного резерва для работы """  
 # ПОЛНЫЙ ВРЕМЕННОЙ РЕЗЕРВ = раннее время конечного события работы - позднее время начального события работы -  
 # - время выполнения работы (вес работы)  
 # НЕЗАВИСИМЫЙ ВРЕМЕННОЙ РЕЗЕРВ = раннее время начального события работы -  
 # - позднее время конечного события работы - время выполнения работы (вес работы)  
 # full = True, то находится полный временной резерв*

26) **def** find\_time\_reserves\_for\_all\_works(self):  
 *""" нахождение полных и независимых временных резервов для всех работ """*

27) **def** find\_index\_by\_event(self, event):  
 *""" нахождение номера (индекса) события в структуре по его шифру """*

28) **def** check\_vertex\_includes\_only\_prev\_layers\_vertices(self, index):  
 *""" проверить, что в вершину входят только вершины предыдущих слоев """*

29) **def** get\_max\_layer\_form\_prev\_vertices(self, index):  
 *""" получаем максимальный слой предыдущих вершин данной вершины """*

30) **def** find\_vertex\_layers(self):  
 *""" находим слой каждой вершины обходом в ширину"""*

31) **def** find\_ins\_and\_outs(self):  
 *""" находим входящие и исходящие работы у вершин """*

32) **def** find\_all\_vertices(self):  
 *""" нахождение всех вершин графа """*

33) **def** recursive\_find\_all\_vertices(self):  
 *""" рекурсивный обход графа """*

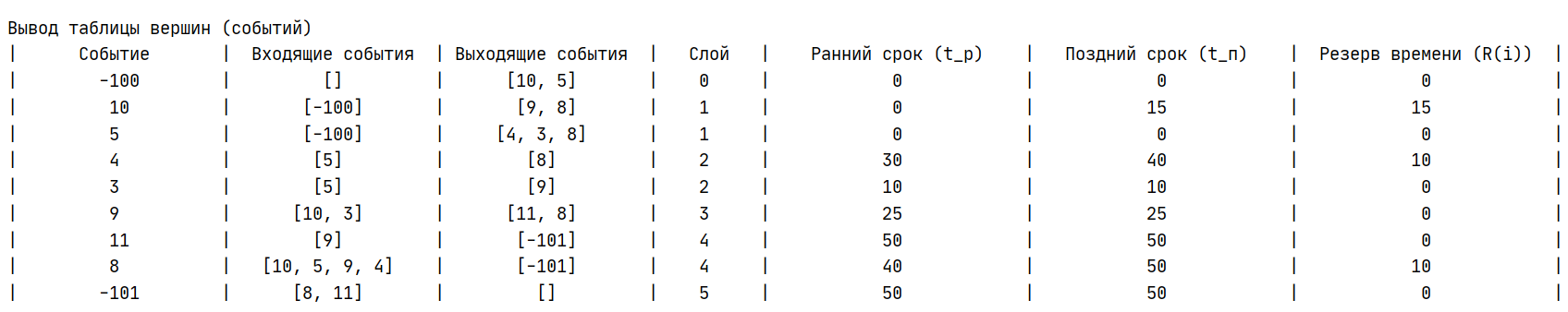
34) **def** print\_gant\_diagram(self):  
 *""" вывод диграммы Ганта """*

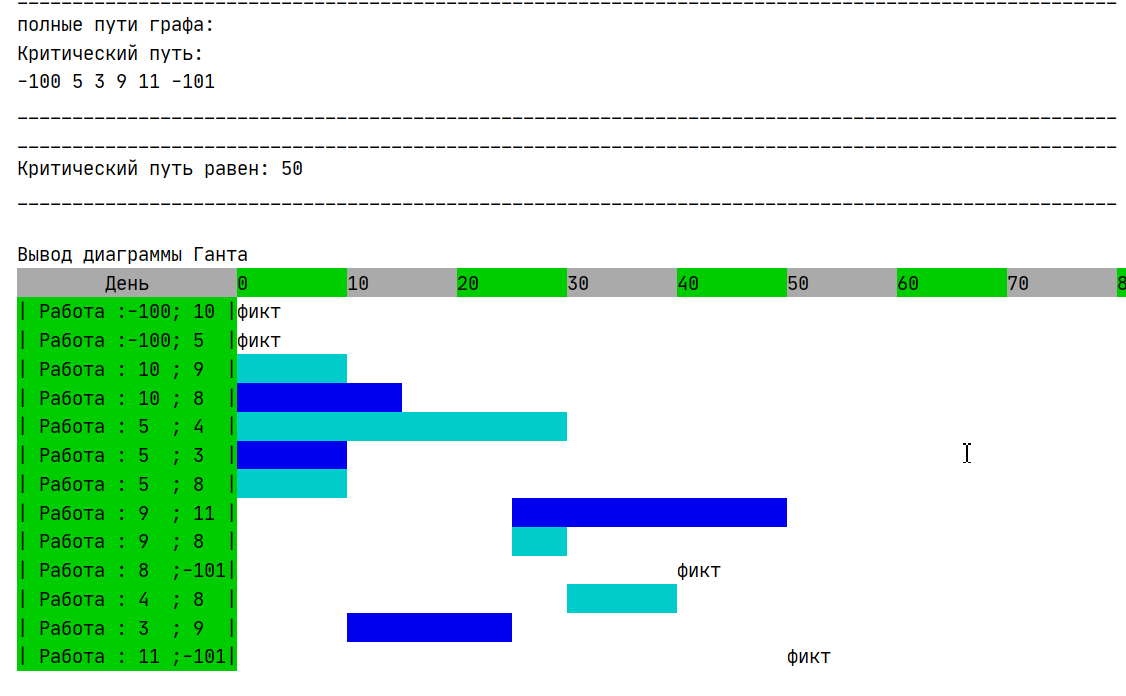
35) код основой программы

**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
**def** main():  
 in\_file = **"input.csv"** graph = Graph()  
 graph.read\_from\_file(in\_file)  
 loop\_counter = 0  
 **while True**:  
 **if** loop\_counter > 0:  
 graph.clear\_graph\_params()  
 graph.print\_graph()  
 *# находим первую вершину* graph.search\_first\_top()  
 *# оптимизируем (удаляем петли, дубли), там же находим конечную вершину* graph.optimize\_graph()  
 *# структурируем граф* graph.struct\_graph()  
 graph.print\_graph(sorted\_graph=**True**)  
 *# находим полные пути* graph.search\_full\_ways()  
 *# зануление перепроверяемых параметров (для циклического запуска функций на повтор)  
  
 # находим все вершины* graph.find\_all\_vertices()  
 *# находим входящие и исходящие вершины* graph.find\_ins\_and\_outs()  
 *# находим слои у вершин* graph.find\_vertex\_layers()  
 *# сортируем таблицу событий по слоям* graph.sort\_events\_by\_layers()  
 *# находим ранние сроки для событий* graph.find\_early\_term\_for\_all\_event()  
 *# находим поздние сроки для событий* graph.find\_late\_term\_for\_all\_event()  
 *# находим временной резерв для событий* graph.find\_time\_reserves\_for\_all\_event()  
 graph.print\_events\_table()  
 *# находим полные и независимые временные резервы для работы* graph.find\_time\_reserves\_for\_all\_works()  
 graph.print\_graph(sorted\_graph=**True**)  
 *# выводим критические пути* graph.search\_full\_ways(check\_critical=**True**)  
 *# находим длину критического пути* critical\_path\_length = graph.get\_critical\_path\_length()  
 *# выводим диаграмму Ганта* graph.print\_gant\_diagram()  
 *# копируем структурированный граф в неструктурированный, чтобы производить добавление/удаление/изменение работ* graph.copy\_graph\_from\_structed\_graph()  
 *#* print(**'\n'** \* 10)  
 choice = input(**'Выберите действие:\n'  
 '\t1 - добавить работу\n'  
 '\t2 - удалить работу\n'  
 '\t3 - изменить время работы\n'  
 '\t0 - выйти\n'  
 '\t\t\tВаш выбор:\t'**)  
 wrong\_input = **True  
 while** wrong\_input:  
 wrong\_input = **False  
 if** choice == **'1'**:  
 print(**'Вы выбрали "добавить работу"'**)  
 start\_vertex = int(input(**'Введите шифр начальной вершины (например: 150):\t'**))  
 end\_vertex = int(input(**'Введите шифр конечной вершины (например: 149):\t'**))  
 weight = int(input(**'Введите продолжитлеьность работы (например: 15):\t'**))  
 graph.add\_row\_in\_graph\_table(start\_vertex, end\_vertex, weight)  
 print(**'Работа добавлена'**)  
 **elif** choice == **'2'**:  
 print(**'Вы выбрали "удалить работу"'**)  
 graph.print\_graph()  
 index = int(input(**'Введите номер работы, которую удалить:\t'**))  
 graph.delete\_row\_from\_graph\_table(index)  
 print(**'Работа удалена'**)  
 graph.print\_graph()  
 **elif** choice == **'3'**:  
 print(**'Вы выбрали "изменить время работы"'**)  
 graph.print\_graph()  
 index = int(input(**'Введите номер работы, время которой вы хотите изменить:\t'**))  
 time = int(input(**'Введите время работы:\t'**))  
 **if** graph.change\_work\_time(index, time):  
 print(**'Время работы изменено'**)  
 graph.print\_graph()  
 **else**:  
 print(**'Работы по такому индексу нет'**)  
 **elif** choice == **'0'**:  
 print(**'Вы выбрали "выйти"'**)  
 **return  
 else**:  
 print(**'Неверный ввод'**)  
 wrong\_input = **False** loop\_counter += 1

**Тесты:**

**Тест 1 (есть фиктивные вершины, оптимизация)**





**Тест 2 (удаление, добавление, изменение)**

