МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Построение и анализ алгоритмов»

Тема: Коммивояжер(TCP)

Студент гр. 1304 _____ Басыров В.А.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург

2023

Цель работы.

Изучение алгоритмов на графах, нахождение гамильтонова цикла в взвешенных графах и реализация метода ветвей и границ.

Задание.

Дана карта городов в виде ассиметричного, неполного графа G = (V, E), где V(|V|=n) – это вершины графа, соответствующие городам; E(|E|=m) – это ребра между вершинами графа, соответствующие путям сообщения между этими городами.

Каждому ребру m_{ij} (переезд из города i в город j) можно сопоставить критерий выгодности маршрута (вес ребра) равный w_i (натуральное число [1, 1000]), m_{ij} =inf, если i=j.

Если маршрут включает в себя ребро m_{ij} , то x_{ij} =1, иначе x_{ij} =0.

Требуется найти минимальный маршрут (минимальный гамильтонов цикл):

Выполнение работы.

В ходе выполнения лабораторной работы был применен метод ветвей и границ, который использовал следующий алгоритм решения поставленной задачи:

1) Нахождение минимумов по строкам — в каждой строке определяется минимальное число и выписывается в отдельный столбец.

2) Редукция строк — из значений ячеек каждой строки вычитаем соответствующий минимум.

3) Нахождение минимумов по столбцам — в каждом столбце определяется минимальное число и выписывается в отдельную строку.

4) Редукция столбцов — из значений ячеек каждого столбца вычитаем соответствующий минимум.

5) Нахождение корневой нижней границы — вычисляем нижнюю границу (минимально возможную на текущем этапе длину маршрута) в

стартовой (корневой) точке решения, как сумму найденных ранее минимумо и начинаем построение графа (схемы) решения с внесения в него корневой вершины.

6) Вычисление оценок нулевых клеток — считаем оценки для каждой ячейки с нулями, как сумму минимумов по строке и столбцу, в которых располагается нулевая клетка, не учитывая при этом саму нулевую клетку

7) Выбор нулевой клетки с максимальной оценкой — ищем среди нулевых клеток обладающую наибольшей оценкой (если таких ячеек несколько, выбираем любую), и получаем пару ветвей (вариантов) решения задачи: с включением в маршрут отрезка пути относящегося к выбранной ячейке и без включения.

8) Редукция матрицы — вычеркиваем относящиеся к выбранной клетке строку и столбец

9) Вычисление нижней границы первой ветви (включающей отрезок пути) — вновь находим минимумы по строкам, проводим редукцию строк, находим минимумы по столбцам, проводим редукцию столбцов, после чего вычисляем локальную нижнюю границу, как сумму предыдущей локальной нижней границы и минимумов.

10) Вычисление нижней границы второй ветви (не включающей отрезок пути) — считаем локальную нижнюю границу, как сумму предыдущей локальной нижней границы и оценки выбранной ранее нулевой клетки ($Hk^* = Hk-1 + pij$)

11) Выбор ветви с минимальным значением нижней границы — среди еще не ветвившихся вершин выбираем обладающую минимальным значением локальной нижней границы (вне зависимости от того, какую ветвь рассматриваем в данный момент)

12) Если полный маршрут еще не найден, продолжаем решение, если найден — проверяем, что это гамильтонов цикл и если да, то завершаем работу алгоритма.

Для реализации данного алгоритма были написаны следующие классы:

Класс *My_matrix* — класс, который содержит таблицу весов ребер. Содержит двумерный массив, ранг этой матрицы, а также список в этой матрице исходящих и входящих вершин. Содержит ряд следующих методов:

1) *get*_rang — метод получения ранга матрицы.Ничего не принимает и не возвращает.

2) *minimum_of_row* — метод, принимающий номер строчки и возвращающий минимум в этой строчке.

3) *minimum_of_column* — метод, принимающий номер столбца и возвращающий минимум в этом столбце.

4) *reduction_row* —метод редукии по строке. В строке из каждого элемента вычитается *subrtrahend*.Ничего не возвращает и принимает номер строки и *subrtrahend.*

5) reduction_column — метод редукии по столбцу. В строке из каждого элемента вычитается *subrtrahend*.Ничего не возвращает и принимает номер столбца и *subrtrahend.*

6) *reduction_rows* — метод, который в каждой строке вычитает минимум по этой строке. Ничего не принимает и не возвращает.

7) *reduction_columns* — метод, который в каждом столбце вычитает минимум по этому столбцу. Ничего не принимает и не возвращает.

8) *return_vector_rows* — метод, который ничего не принимает и возвращает вектор минимумов по строкам.

9) *return_vector_columns* — метод, который ничего не принимает и возвращает вектор минимумов по столбцам.

10) *sum_vector_rows* — метод, который ничего не принимает и возвращает сумму минимумов по строкам.

11) *sum_vector_columns* — метод, который ничего не принимает и возвращает сумму минимумов по столбцам.

12) *estimate_null_square* — метод, который ничего не принимает и возвращает нулевой элемент с максимумом по минимуму из строк и столбцов(не включая нулевой элемент).

13) *delete_column_number* — метод, который принимает номер, который необходимо удалить и возвращает вектор, который получается путем удаления из вектора-столбца i элемента.

14) *delete_row_number* — метод, который принимает номер, который необходимо удалить и возвращает вектор, который получается путем удаления из вектора-строки номера элемента.

15) *set_inf_of_number* — метод, который принимает положение элемента, который необходимо обратить в бесконечность. Ничего не возвращает.

16) *return_edge* — метод, который возвращает ребро и принимает индексы матриц, которые указывают на ребро.

17) *reduction_matrix* — метод, который удаляет строку и столбец. Принимает индексы этих строк и столбцов и возвращает получившуюся матрицу.

18) *copy* — метод, который копирует обьект класса *My_Matrix*. Ничего не принимает и возвращает скопированную матрицу.

19) *is_deadlock* — метод, который ничего не принимает и возвращает *True* — если данная ветка является тупиковой и *False* иначе.

Класс *Node_Tree* — *класс*-узел дерева, который в констукторе принимает матрицу путей из одной вершину к другой, а также отца исходного узла, и ребро, включенное на этом уровне(если ребро не включено, то значение None). Также инициализирует переменную оценки исходного узла. Содержит следующие методы:

1) *set_lower_estimate* — метод, которые устанавливает оценку для текущего узла. Принимает значение, является ли данный узел корнем и элемент, который необходим для формирования оценки в случае, если на предыдущем шаге ребро не было включено. Ничего не возвращает.

2) g*et_lower_estimate* — метод, который ничего не принимает и возвращает исходную оценку.

Класс *Priory_queue* — класс, являющийся приоритетной очередью. Содержит следующие методы:

1) Метод *add* — метод, принимающий узел дерева. Метод добавляет элемент в очередь с приоритетом и ничего не возвращает.

2) Метод *remove* — метод, который ничего не принимает и возращает приоритетный элемент, который удаляется из очереди.

3) Метод *get_len* — метод, который ничего не принимает и возвращает длину очереди.

Класс Algoritm — класс, который представляет алгоритм решения поставленной задачи. Содержит 2 поля: корень дерева и очередь с приоритетом. Содержит следующие методы:

1) *get_edges* — метод, который ничего не принимает и возвращает ребра, по которым был построен минимальный путь.

2) *get_near_answer* — метод, который возвращает последовательность вершин. Это последовательность вершин далее должна пройти проверку на то, является ли она гамильтоновым циклом. Ничего не принимает и возвращает эту последовательность.

3) *get_answer* — метод, который ничего не принимает и возвращает ответ на исходную задачу.

Исходный код предствлен в Приложение А Исходный код программы.

Тестирование.

Здесь результаты тестирования, которые помещаются на одну страницу. Ссылка на таблицу с тестированием обязательно должна быть в тексте до самой таблицы. Не оставляйте пустых столбцов в таблице, удаляйте их.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии

1.	inf 1 2 2 - inf 1 2 - 1 inf 1 1 1 - inf	([4, 1, 2, 3, 4], 4) 0:00:00.000399	Пример из задания.
2.	inf 10 1 10 inf 2 3 7 inf	([3, 1, 2, 3], 15) 0:00:00.000207	Проверка, что при локально выгодных решениях, алгоритм не возьмет их и получит глобально правильное решение.
3.	inf 2 2 inf	([1, 2], 4) 0:00:00.000043	Проверка, вырожденного случая, когда разменость матрицы равна 2.
4	inf 5 65 2 18 26 21 67 48 17 91 74 78 21 35 26 85 87 21 43 35 inf 88 63 24 43 46 75 5 22 3 27 87 55 50 25 65 14 10 68 3 68 inf 24 44 28 19 17 13 66 43 93 38 63 42 34 58 6 91 36 12 50 75 inf 87 62 89 21 60 41 45 89 68 35 32 9 16 88 23 75 84 19 89 90 inf 93 69 52 71 3 62 62 23 71 77 93 68 24 20 38 17 77 48 19 70 inf 22 43 5 63 43 78 10 25 91 8 89 79 35 50 8 29 93 19 94 15 inf 20 60 79 43 82 5 7 61 60 49 30 25 15 7 1 76 60 64 20 1 inf 12 4 42 15 75 100 34 71 9 35 69 79 7 41 90 38 88 68 22 49 inf 91 87 50 58 81 6 47 48 6 100 78 21 20 72 97 90 22 30 78 50 inf 22 47 26 71 72 59 11 100 30 41 15 60 98 97 34 45 7 55 1 47 inf 8 47 38 35 97 15 53 61 95 64 51 21 64 55 92 64 41 68 66 56 inf 70 25 77 84 55 87 82 48 95 23 49 54 88 34 60 97 18 76 43 40 inf 54 46 22 77 1 84 42 50 63 93 4 73 53 79 66 73 17 95 10 29 inf 1 27 71 11 85 5	([16, 6, 9, 8, 2, 11, 12, 3, 1, 4, 5, 10, 14, 15, 19, 7, 13, 18, 20, 17, 16], 304) 0:00:00.003853	Проверка, что при размерности матрицы 20, время работы программы будет меньше, чем 3 секунды.

	69 80 81 11 76 68 83 28 67 16 45 74 1 84 inf 74 81 100 15 26 20 54 97 47 16 8 56 80 42 84 20 83 76 62 61 inf 84 30 74 64 27 12 61 96 41 46 12 83 96 37 34 100 46 53 36 11 inf 13 87 49 94 70 50 4 75 58 96 60 24 9 100 76 10 61 16 98 30 inf 25 4 63 85 47 77 49 32 4 29 16 50 82 11 76 71 33 92 70 8 inf 6 91 29 72 5 36 43 55 22 95 63 87 52 33 40 5 60 2 41 16 inf		

Выводы.

Был реализован алгоритм нахождения кратчайшего гамильтонова пути в графе, а также реализован метод ветвей и границ, который для полного графа ранга 20 выполняет поиск минимального пути в среднем примерно за половину миллисекунды. Такой быстрой скорости алгоритма удалось получить благодаря использованию эффективного метода оценок, а также отсечению заведомо неверных вариантов. Стоит отметить, что при построении алгоритма был применен метод математического модерлирования, когда задача перенеслась на граф и свелась к более легкой задачи — задачи нахождение гамильтонова цкила во взвешенном ассиметричном графе.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Сначала указываем имя файла, в котором код лежит в репозитории:

Название файла: main.py

```
import datetime
from random import randint
inf=100000#бесконечное число
maximum_number=inf-10000#граница, все числа выше которой приравниваются к бесконечности.
class My_matrix:#класс матрицы весов.
                                                                    def __init__(self,matrix,rang:int,rows:list,columns:list):#иницализация: matrix - двумерный массив, хранящий таблицу вершин

#rang - ранг этой матрицы

#rows - числа от 1 до rang по горизонтали.Нужно для восстановление пути.

#columns - числа от 1 до rang по вертикали. Нужно для восстановления пути..
            self.matrix=matrix
            self.rang=rang
            self.rows=rows
            self.columns=columns
        def get_rang(self):#Метод, ничего не принимает и возвращает ранг матрицы
            return self.rang
             def minimum_of_row(self,i:int):#метод, принимающий i и возвращающий минимум в i строчке
            return min(self.matrix[i])
           def minimim_of_column(self,i:int):#метод, принимающий i и возвращающий минимум в i столбце.
            return min([x[i] for x in self.matrix])
        def reduction_row(self,i:int,subtrahend:int):#метод редукии по строке.В i строке из каждого элемента вычитается subrtrahend.Ничего не возвращает.
            self.matrix[i]=[x-subtrahend for x in self.matrix[i]]
        def reduction_column(self,i:int,subtrahend:int):#метод редукии по столбцу.В i столбце из каждого элемента вычитается subrtrahend.Ничего не возвращает.
            for j in range(self.rang):
                self.matrix[i][j]-=subtrahend
          def reduction_rows(self):#метод ничего не принимает и не возвращает.В каждой строке вычитает минимум этой строки.
            vector_rows=self.return_vector_rows()
            for i in range(self.rang):
                if vector_rows[i]<maximum_number:
                    self.reduction_row(i,vector_rows[i])
        def reduction_columns(self):#метод ничего не принимает и не возвращает.В каждом столбце вычитает минимум этого столбца.
            vector_columns = self.return_vector_columns()
            for i in range(self.rang):
```

```
                if vector_columns[i]<maximum_number:
                    self.reduction_column(i, vector_columns[i])
        def return_vector_rows(self):#функция ничего не принимает и возвращает вектор минимальных значений по строкам.
            return [self.minimum_of_row(x) for x in range(self.rang)]
        def return_vector_columns(self):#функция ничего не принимает и возвращает вектор минимальных значений по столбцам.
                        return [self.minimim_of_column(x) for x in range(self.rang)]
            def sum_vector_rows(self):#вычисляет сумму по минимумам строк.Метод ничего не принимает.
            return sum(self.return_vector_rows())
        def sum_vector_columns(self):#вычисляет сумму по минимумам столбцов.Метод ничего не принимает
            return sum(self.return_vector_rows())
            def estimate_null_square(self):#метод, который ничего не принимает и возвращает нулевую клетку с максимальной оценкой.
            null_elements=[]
            for i in range(self.rang):
                for j in range(self.rang):
                    if not self.matrix[i][j]:
                        save_element=self.matrix[i][j]
                        self.set_inf_of_number(i,j)
                            null_elements.append((self.minimum_of_row(i)+self.minimim_of_column(j),i,j))
                        self.matrix[i][j]=save_element
            return max(null_elements,key=lambda x:x[0])
        def delete_column_number(self,i):#метод, который удаляет в column i номер.Принимает i и возвращает новый вектор.
            vector=self.columns.copy()
            vector.pop(i)
            return vector
        def delete_row_number(self,i):#метод, который удаляет в row i номер. Принимает i и возвращает новый вектор.
            vector=self.rows.copy()
            vector.pop(i)
            return vector
            def set_inf_of_number(self,i:int,j:int):#метод, который устанавливает клетке i строки и j столбца inf. Принимает i и j и ничего не возвращает.
            self.matrix[i][j]=inf
        def return_edge(self,i,j):#метод, который возвращает ребро, состощее из i и j вершины.Вовзращает это ребро и принимает i и j.
            return (self.rows[i],self.columns[j])
        def reduction_matrix(self,i:int,j:int):#метод, который удаляет i строку и j столбец.Возвращает получившуюся матрицу и принимает i и j .
            self.set_inf_of_number(j,i)
            return [[col[1] for col in enumerate(row[1]) if col[0]!=j] for row in enumerate(self.matrix) if row[0]!=i]
        def copy(self):#метод, который копирует исходную матрицу. Ничего не принимает и возвращает копию Mymatrix.
                                                                    return My_matrix(self.matrix.copy(),self.rang,self.rows.copy(),self.columns.copy())
        def is_deadlock(self):#метод, который проверяет, является ли данная матрица тупиковой(нельзя пойти в новую вершину). Ничего не принимает и возвращает True - если тупик, иначе False.
            for i in range(self.rang):
```

```
                if self.minimum_of_row(i)<maximum_number:
                    return False
            return True
    class Node_tree:#класс дерева метода ветвей и границ.
                                                                    def
__init__(self,matrix:My_matrix,parent,edge=None):#инициализация: matrix -
класс матрицы Mymatrx.

#lower_estimate - оценка, необходимая очереди с приоритетом.
                                                        self.matrix=matrix
#parent - родитель исходного узла
                                                    self.lower_estimate=0
#edge   -   ребро,включенное   на   данном   уровне.Если   ребро   не
включено,edge=None.
            self.parent=parent
            self.edge=edge
                                                                    def
set_lower_estimate(self,is_root:bool,choose_element:int=None):#метод,
который устанавливает исходную оценку.Принимает is_root- явлеятся ли узел
корнем и choose_element - элемент, который при не включении ребра
принимает участие в формировании оценки.Ничего не возвращает
            if choose_element:
                 self.lower_estimate = self.parent.get_lower_estimate()
+choose_element
            else:
                if is_root:
                     self.lower_estimate=self.matrix.sum_vector_rows() +
self.matrix.sum_vector_columns()
                else:
                                                    self.lower_estimate   =
self.parent.get_lower_estimate()   +   self.matrix.sum_vector_rows()   +
self.matrix.sum_vector_columns()
             def  get_lower_estimate(self):#метод,  который  ничего  не
принимает и возвращает оценку.
            return self.lower_estimate
     class  Priory_queue:#Класс  очереди  с  приоритетом,  необходимый  для
выполнения поставленной задачи
         def __init__(self):
            self.arr=[]
            self.len=0
           def  add(self,node:Node_tree):#добавляет  элемент  в  очередь.
node- добавляемый элемент. Ничего не возвращает
            len_arr=self.get_len()
            i=0
                                                while     i<len_arr     and
node.get_lower_estimate()>self.arr[i].get_lower_estimate():
                i+=1
            if i==len_arr:
                self.arr.append(node)
            else:
                self.arr.insert(i,node)
            self.len+=1
           def  remove(self)->Node_tree:#Удаление  самого  приоритетного
элемента. Ничего не принимает и возвращает удаляемый элемент.
            self.len-=1
            return self.arr.pop(0)
         def get_len(self):#Возвращает длину очереди.
            return self.len
```

```
    class Algoritm:#класс-алгоритм.
        def __init__(self,root_node:Node_tree):#иницализация:root_node
- корень дерева, priory_queue - очередь с приоритетом.
            self.root_node=root_node
            self.priory_queue=Priory_queue()
        def get_edges(self,vertex):#метод, которая получает ребра, по
которым был построен итоговый путь.Возвращает список ребер и принимает
vertex - лист исходного дерева.
            res=[(vertex.matrix.rows[0],vertex.matrix.columns[0])]
            while vertex.parent:
                if vertex.edge:
                    res.append(vertex.edge)
                vertex=vertex.parent
            res.reverse()
            return res
        def get_near_answer(self,vertex):#метод, который возвращает
список вершин и вес итогового пути.Принимает vertex- лист дерева.
            arr=self.get_edges(vertex)
            answer=[arr[0][0]]
            vertex=arr[0][1]
            answer.append(vertex)
            n=0
            while answer[0]!=vertex:
                for i in arr:
                    if i[0]==vertex:
                        n+=1
                        vertex=i[1]
                        answer.append(vertex)
                        break
            weight=0
            for i in arr:
                weight+=self.root_node.matrix.matrix[i[0]-1][i[1]-1]
            return answer,weight
        def get_answer(self):#метод, который представляет исходный
алгоритм и возвращает результат - список вершин, в том порядке, по
которому по ним прошел коммивояжер и вес пути.
            save_matrix=self.root_node.matrix.matrix.copy()
            self.root_node.matrix.reduction_rows()
            self.root_node.matrix.reduction_columns()
            self.root_node.set_lower_estimate(True)
            self.priory_queue.add(self.root_node)
            while True:
                leaf=self.priory_queue.remove()
                if leaf.matrix.is_deadlock():
                    continue
                if leaf.matrix.get_rang()==1:
                    self.root_node.matrix.matrix = save_matrix
                    answer,weight=self.get_near_answer(leaf)
                    if len(answer)==self.root_node.matrix.get_rang()+1:
                        return answer,weight
                    else:
                        continue
                leaf.matrix.reduction_rows()
                leaf.matrix.reduction_columns()
                remove_element=leaf.matrix.estimate_null_square()
                 new_matrix=My_matrix(leaf.matrix.reduction_matrix(remov
e_element[1],remove_element[2]),leaf.matrix.get_rang()-
```

```
1,leaf.matrix.delete_row_number(remove_element[1]),leaf.matrix.delete_col
umn_number(remove_element[2]))
                  left_vertex=Node_tree(new_matrix,leaf,leaf.matrix.retur
n_edge(remove_element[1],remove_element[2]))
                 left_vertex.set_lower_estimate(False)
                 right_vertex=Node_tree(leaf.matrix.copy(),leaf)
                  right_vertex.set_lower_estimate(False,right_vertex.matr
ix.matrix[remove_element[1]][remove_element[2]])
                  right_vertex.matrix.set_inf_of_number(remove_element[1]
,remove_element[2])
                 self.priory_queue.add(right_vertex)
                 self.priory_queue.add(left_vertex)
     def main():
         f=open("test3.txt",'r')
         arr=f.readlines()
         matrix=[]
         rang=0
         for i in arr:
             rang+=1
             vector=i.replace('-',str(inf)).split()
             vector[rang-1]=inf
             vector=list(map(int,vector))
             matrix.append(vector)
         rang=20
         matrix=[]
         vector=[]
         for i in range(20):
             vector=[]
             for j in range(20):
                 if i==j:
                     vector.append(inf)
                     continue
                 vector.append(randint(0,10))
             matrix.append(vector)
         time_start=datetime.datetime.now()
               root_Node=Node_tree(My_matrix(matrix,rang,[x  for  x  in
range(1,rang+1)],[x for x in range(1,rang+1)]),None)
         algoritm=Algoritm(root_Node)
         print(algoritm.get_answer(),datetime.datetime.now()-time_start)
     if __name__=='__main__':
         main()
```