Университет ИТМО

Факультет инфокоммуникационных технологий
1 курс
Лабораторная работа №3

Выполнила:
Чагина Вероника Александровна

группа К3144

Преподаватель:

Харьковская Татьяна Александровна

Дата выполнения: 30.05.2022

Санкт-Петербург

Основная часть

6 Задача. Количество пересадок

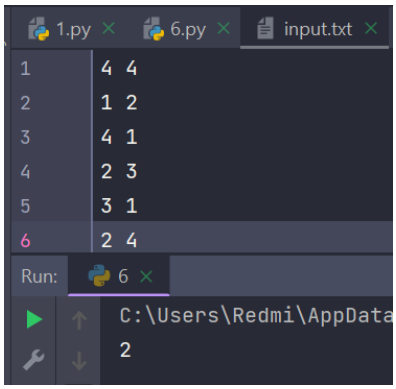
Дан неориентироанный граф с n вершинами и m ребрами, а также две вершины u и v, нужно посчитать длину кратчайшего пути между u и v (то есть, минимальное количество ребер в пути из u в v).

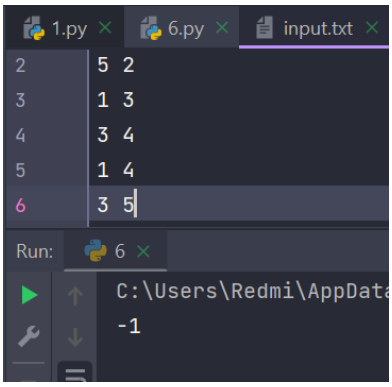
Решение:

```
def shortPath(u, v):
    global sides
    search_queue = deque()
    search_queue.append((u, 0))
    visited = []
    while search_queue:
        cur_node, path = search_queue.popleft()
        if cur_node == v:
            return path
        path += 1
        if cur_node not in visited:
            visited.append(cur_node)
            for node in sides[cur_node]:
                search_queue.append((node, path))
    return -1

with open('input.txt') as f:
    n, m = map(int, f.readline().split())
    sides = {}
    for i in range(n+1):
        sides[i] = []
    for i in range(m):
        v1, v2 = map(int, f.readline().split())
        sides[v1].append(v2)
        sides[v2].append(v1)
    u, v = map(int, f.readline().split())
```

Для решения данной задачи используется обход графа в ширину. То есть в нем нужна очередь. В очередь добавляются все не посещенные вершины, к которым есть доступ из текущей, вместе с количеством ребер, которое надо пройти, чтобы достичь данной вершины из исходной. Считываем неориентированный граф и кладем результат функции “shortPath” в переменную “result”, значение которой и есть ответ.

Тесты:





8 Задача. Стоимость полета

Дан ориентированный граф с положительными весами ребер, n - количество вершин и m - количество ребер, а также даны две вершины u и v. Вычислить вес кратчайшего пути между u и v (то есть минимальный общий вес пути из u в v).

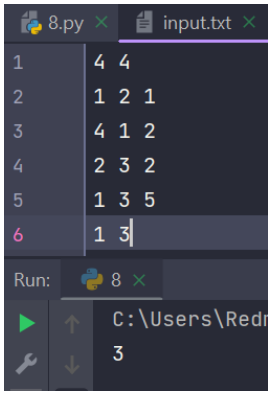
Решение:

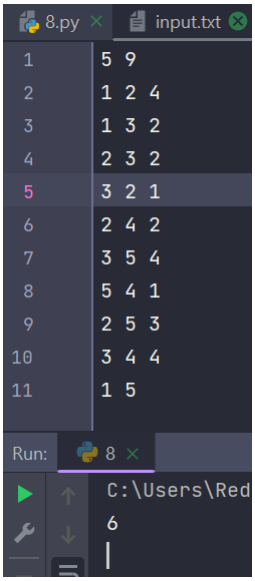
```
c_nodes, c_ribs = map(int, inp.readline().split())

edges = []

for _ in range(c_ribs):
    node, link, weight = map(int, inp.readline().split())
    edges.append((node - 1, link - 1, weight))

src, dst = map(int, inp.readline().split())
src -= 1
dst -= 1

inp.close()

ways = [None] * c_nodes
ways[src] = 0

for i in range(c_nodes-1):
    for source, destination, weight in edges:
        if ways[source] is not None:
            ways[destination] = min((math.inf if (ways[destination] is None) else
ways[destination]), ways[source] + weight)
```

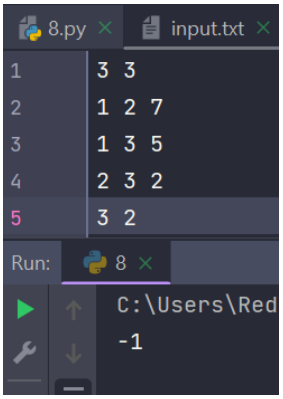
Для решения данной задачи используется обход графа в ширину. То есть в нем нужна очередь. В очереди хранится путь до текущей вершины, а также ее ‘родителя’. Если текущая вершина искомая, а текущий путь меньше минимального, то обновляем значение минимального пути. Если не было найдено ни одного пути, то возвращаем ‘-1’.

Родитель нужен, чтобы не зацикливать путь, тк ставить метки прохождения вершин мы не можем, из- за того что граф не ориентирован. При считывании графа убираем все ребра междувершинами, длина которых больше, чем длина минимального ребра между ними.

Тесты:







11 Задача. Алхимия

Результатом алхимической реакции является превращение одного вещества в другое. Задан набор алхимических реакций, описанных на найденных глиняных табличках, исходное вещество и требуемое вещество. Необходимо выяснить: возможно ли преобразовать исходное вещество в требуемое с помощью этого набора реакций, а в случае положительного ответа на этот вопрос – найти минимальное количество реакций, необходимое для осуществления такого преобразования.

Решение:

```
def shortPath(u, v):
    global sides
    search_queue = deque()
    search_queue.append((u, 0))
    visited = []
    while search_queue:
        cur_node, path = search_queue.popleft()
        if cur_node == v:
            return path
        path += 1
        if cur_node not in visited:
            visited.append(cur_node)
            if cur_node in sides:
                for node in sides[cur_node]:
                    search_queue.append((node, path))
    return -1
```

Задача по сути является копией задачи №6, только вместо чисел, ключами вершин графа являются названия веществ. Поэтому делаем поправкуна ввод ориентированного графа и задача решена.

Тесты:

Автор	Задача	Язык	Результат	Тест	Время	Память
Вероника Чагина	0743	Python	Accepted		0,046	794 Кб

Доп. Задачи (Задачи 1, 2, 7, 12, 13, 14, 15)

1 Задача. Лабиринт

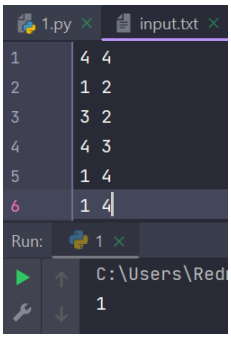
Дан неориентированный граф и две различные вершины u и v. Проверьте, есть ли путь между u и v.

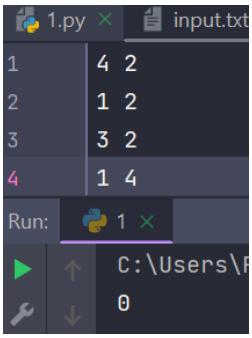
Решение:

```
visited = [False] * c_nodes
que = [start]

exist = False
visited[start] = True
while(len(que) > 0):
    node = que.pop()
    if(node == target):
        exist = True
        break
    for j in links[node]:
        if(not visited[j]):
            que.append(j)
            visited[j] = True
```

В начале считываем данные из исходного файла в словарь “links”, где номер ячейки – это ключ вершины графа, а в самой ячейке хранится массив вершин, с которыми у данной вершины есть ребра. Массив “visited” отвечает за метку прохождение вершины при обходе в глубину, а словарь “que” будет указывать на вершину графа, из которой попали в текущую вершину. В цикле происходит обход графа в глубину. Если встретится искомая вершина, то программа остановится и выведет ‘1’.

Тесты:





7 Задача. Двудольный граф

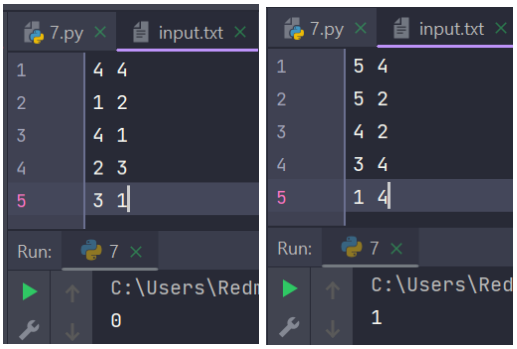
Дан неориентированный граф с n вершинами и m ребрами, проверьте, является ли он двудольным.

Решение:

```
def halfGraph(u):
    global sides, total_colours
    search_queue = deque()
    search_queue.append((u, 0))
    visited = []
    while search_queue:
        cur_node, colour = search_queue.popleft()
        if cur_node not in visited:
            total_colours[cur_node] = colour
            visited.append(cur_node)
            for node in sides[cur_node]:
                if colour == 0:
                    search_queue.append((node, 1))
                else:
                    search_queue.append((node, 0))
        elif total_colours[cur_node] != colour:
            return 0
    return 1
```

Для решения данной задачи также воспользуемся обходом в ширину. Только теперь вместо пути, мы будем записывать в очередь цвет вершины: условно ‘0’ и ‘1’. Цвет текущей вершины записываем в массив “total_colours”. Если вершина не посещена, то добавляем в очередь все вершины, доступные из нее с противоположным цветом. Иначе сверяем текущий цвет с записаннымв вышеупомянутый массив. Если они различаются, то возвращаем ‘0’. Иначе в конце возвращается ‘1’.

Тесты:



12 Задача. Цветной лабиринт

Цветной лабиринт состоит из n комнат, соединенных m двунаправленными коридорами. Каждый из коридоров покрашен в один из ста цветов, при этом от каждой комнаты отходит не более одного коридора каждого цвета. При этом две комнаты могут быть соединены любым количеством коридоров.

Человек, купивший билет на аттракцион, оказывается в комнате номер один. Он получает описание пути, по которому он может выбраться из лабиринта. Находясь в комнате, надо посмотреть на очередной цвет в этой последовательности, выбрать коридор такого цвета и пойти по нему. При этом если из комнаты нельзя пойти по коридору соответствующего цвета, то человеку приходится дальше самому выбирать, куда идти.

Решение:

```
def maze(k, path, sides):
    room_cur = 1
    for i in range(k):
        colour = path[i]
        flag = False
        for vertex, side in sides[room_cur]:
            if side == colour:
                room_cur = vertex
                flag = True
                break
        if not flag:
            return 'INCORRECT'
    return str(room_cur)
```

В данной нужно просто идти из первой вершины по коридору текущего цвета, пока это возможно. Если путь пройден, то возвращается номер конечной вершины, иначе – ‘INCORRECT’.

Тесты:

Вероника Чагина	0601	Python	Accepted		0,125	11 Мб

13 Задача. Грядки

Прямоугольный садовый участок шириной N и длиной M метров разбит на квадраты со стороной 1 метр. Подсчитайте количество грядок на садовом участке.

Решение:

```
count = 0
for i in range(rows):
    for j in range(cols):
        if not visited[i][j]:
            if field[i][j] == '#':
                count += 1
                stack = [(i, j)]
                while len(stack) != 0:
                    x, y = stack.pop()
                    if field[x][y] == '#' and not visited[x][y]:
                        visited[x][y] = True
                        if x != 0:
                            stack.append((x-1, y))
                        if x != rows-1:
                            stack.append((x+1, y))
                        if y != 0:
                            stack.append((x, y-1))
                        if y != cols-1:
                            stack.append((x, y+1))
```

Считываем исходное поле в массив строк. Создаем двумерный массив прохождения ячеек поля.

Далее идем по всем ячейкам. Если встречаем грядку, увеличиваем счетчик грядок, ищем и добавляем все ячейки данной грядки. В конце выводимколичество грядок.

Тесты:

Вероника Чагина	0432	Python	Accepted		0,156	5,6 Мб

15 Задача. Герои

Марлезонский балет вот-вот начнется; королева не в состоянии ждать героев больше L минут; ровно в начале L+1-й минуты королева покинет парк, и те мушкетеры, что не успеют к этому времени до нее добраться, не смогут передать ей подвески. На преодоление одного участка у мушкетеров уйдет ровно по минуте. С каждого участка мушкетеры могут перейти на 4 соседние. Требуется выяснить, сколько подвесок будет красоваться на платье королевы, когда она придет на бал.

Решение:

```
def toQueen(x, y, p):
    global x_queen, y_queen, L, garden, N, M
    search_queue = deque()
    search_queue.append((x, y, 0))
    visited = []
    while search_queue:
        x_cur, y_cur, path = search_queue.popleft()
        if x_cur == x_queen and y_cur == y_queen:
            print(p)
            return p
        if path < L:
            path += 1
            if (x_cur, y_cur) not in visited:
                visited.append((x_cur, y_cur))
                if x_cur != 1 and garden[x_cur-2][y_cur-1] == '0':
                    search_queue.append((x_cur-1, y_cur, path))
                if x_cur != N and garden[x_cur][y_cur-1] == '0':
                    search_queue.append((x_cur+1, y_cur, path))
                if y_cur != 1 and garden[x_cur-1][y_cur-2] == '0':
                    search_queue.append((x_cur, y_cur-1, path))
                if y_cur != M and garden[x_cur-1][y_cur] == '0':
                    search_queue.append((x_cur, y_cur+1, path))
    return 0
```

Для решения данной задачи также воспользуемся обходом в ширину. В очередь мы будем добавлять все доступные не пройденные ячейки поля, пока путь меньше отведенного времени или ячейка окажется искомой. В первом случае возвращаем 0, а во втором количество подвесок у соответствующего мушкетера.

В основном коде считываем поле, координаты королевы, мушкетеров, количество подвесок у каждого из них и время ожидания королевы. Затем суммируем значения принесенных подвесок и записываем его в файл.

Тесты:

Вероника Чагина	0846	Python	Accepted		0,046	822 Кб

Вывод:

В данной лабораторной работе я изучила графы, поиск в глубину, поиск в ширину, алгоритм Дейкстры, алгоритм Беллмана-Форда. Рассмотрены различные виды графов, типы их обходов и другие алгоритмы, связанные с графами.