САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3
по курсу «Алгоритмы и структуры данных»
Тема: Графы
Вариант 2

Выполнила:
Бочкарева Е.А.
К3144

Проверила:
Артамонова В.Е.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №2. Компоненты

Теперь вы решаете сделать так, чтобы в лабиринте не было мертвых зон, то есть чтобы из каждой клетки был доступен хотя бы один выход. Для этого вы находите связные компоненты соответствующего неориентированного графа и следите за тем, чтобы каждый компонент содержал выходную ячейку.

Дан неориентированный граф с n вершинами и m ребрами. Нужно посчитать количество компонент связяности в нем.

- **Формат ввода / входного файла (input.txt).** Неориентированный граф с n вершинами и m ребрами по формату 1.
- **Ограничения на входные данные.** $1 \leq n \leq 10^3, 0 \leq m \leq 10^3$.
- **Формат вывода / выходного файла (output.txt).** Выведите количество компонент связности.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.
- Пример:

input	output
4 2 1 2 3 2	2

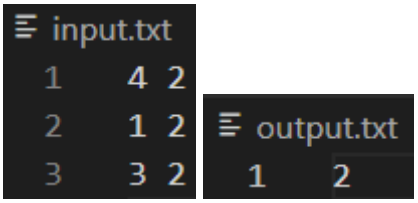
В этом графе есть два компонента связности: 1, 2, 3 и 4.

Листинг кода

```
def maze(graph, visited, node):
    visited[node] = True
    for neighbor in graph[node]:
        if not visited[neighbor]:
            maze(graph, visited, neighbor)


def components(graph, n):
    visited = [False] * n
    count = 0
    for vertex in range(n):
        if not visited[vertex]:
            count += 1
            maze(graph, visited, vertex)
    return count

def main():
    with open('input.txt', 'r') as f:
        n, m = map(int, f.readline().split())
        graph = [[] for _ in range(n)]
```

```
        for _ in range(m):
            u, v = map(int, f.readline().split())
            graph[u - 1].append(v - 1)
            graph[v - 1].append(u - 1)

    with open('output.txt', 'w') as f:
        f.write(str(components(graph, n)))

if __name__ == "__main__":
    main()
```

Примеры работы кода на данных из условий:



Вывод по задаче:

Сделали так, чтобы в лабиринте не было мертвых зон.

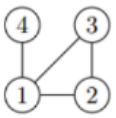
Задача №7 Двудольный граф

Неориентированный граф называется **двудольным**, если его вершины можно разбить на две части так, что каждое ребро графа соединяет вершины из разных частей, то есть не существует рёбер между вершинами одной и той же части графа. Двудольные графы естественным образом возникают в задачах, где граф используется для моделирования связей между объектами двух разных типов (например, мальчиками и девочками, или студентами и общежитиями). Альтернативное определение таково: граф двудольный, если его вершины можно раскрасить двумя цветами (например, черным и белым) так, что концы каждого ребра окрашены в разные цвета.

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

- **Формат ввода / входного файла (input.txt).** Неориентированный граф задан по формату 1.
- **Ограничения на входные данные.** $1 \le n \le 10^5, 0 \le m \le 10^5$.
- **Формат вывода / выходного файла (output.txt).** Выведите 1, если граф двудольный; и 0 в противном случае.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 512 мб.
- Пример 1:

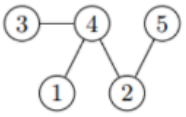
input	output
4 4 1 2 4 1 2 3 3 1	0



Этот граф не является двудольным. Чтобы убедиться в этом, предположим, что вершина 1 окрашена в белый цвет. Тогда вершины 2 и 3 нужно покрасить в черный цвет, так как граф содержит ребра 1, 2 и 1, 3. Но тогда ребро 2, 3 имеет оба конца одного цвета.

- Пример 2:

input	output
5 4 5 2 4 2 3 4 1 4	1



Этот граф двудольный: вершины 4 и 5 покрасим в белый цвет, все остальные вершины – в черный цвет.

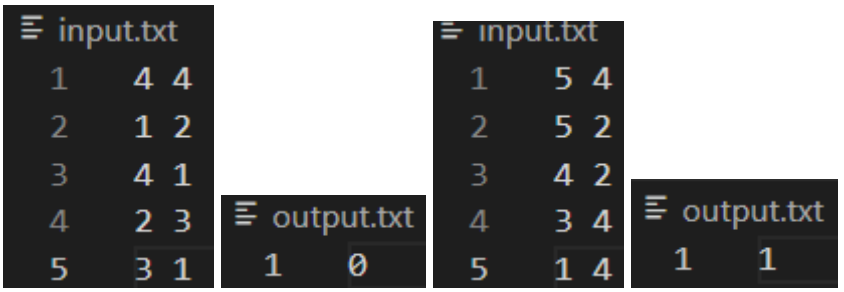
- Что делать? Адаптируйте поиск в ширину (BFS), чтобы решить эту проблему.

Листинг кода

```
from collections import deque


def half_graph(u):
    global sides, total_colors
    search_queue = deque()
    search_queue.append((u, 0))
    visited = []
    while search_queue:
        cur_node, color = search_queue.popleft()
        if cur_node not in visited:
            total_colors[cur_node] = color
            visited.append(cur_node)
            for node in sides[cur_node]:
```

```
                if color == 0:
                    search_queue.append((node, 1))
                else:
                    search_queue.append((node, 0))
        elif total_colors[cur_node] != color:
            return 0
    return 1

def main():
    f = open('input.txt')
    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)

    total_colors = [None] * (n+1)

    with open('output.txt', 'w') as f:
        f.write(str(half_graph(1)))

if __name__ == "__main__":
    main()
```

Текстовое объяснение решения.
Адаптировали поиск в ширину для решения задачи.
Результаты работы на примерах из кода:



Вывод по задаче:
Благодаря этой задаче я разобралась лучше в особенностях двудольных графов и поняла, как с ними работать.

Задача №13 Грядки

Прямоугольный садовый участок шириной N и длиной M метров разбит на квадраты со стороной 1 метр. На этом участке вскопаны грядки. Грядкой называется совокупность квадратов, удовлетворяющая таким условиям:

- из любого квадрата этой грядки можно попасть в любой другой квадрат этой же грядки, последовательно переходя по грядке из квадрата в квадрат через их общую сторону;
- никакие две грядки не пересекаются и не касаются друг друга ни по вертикальной, ни по горизонтальной сторонам квадратов (касание грядок углами квадратов допускается).

Подсчитайте количество грядок на садовом участке.

- **Формат входных данных (input.txt) и ограничения.** В первой строке входного файла INPUT.TXT находятся числа N и M через пробел, далее идут N строк по M символов. Символ # обозначает территорию грядки, точка соответствует незанятой территории. Других символов в исходном файле нет ($1 \le N, M \le 200$).
- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите количество грядок на садовом участке.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 16 мб.
- Примеры:

input.txt	output.txt
5 10 ##.....#. .#..#...#. .###....#. ..##....#. ....#.	3

input.txt	output.txt
5 10 ##..#####. .#.#.#.... ###..##.#. ..##.....# .###.#####	5

- Проверяем **обязательно** – на acmp.

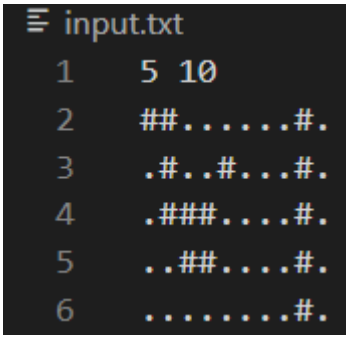
Листинг кода

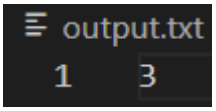
```
def main():
    f = open('input.txt')
    arr = []
    n, m = map(int, f.readline().split())
    for i in range(n):
        arr.append(f.readline()[:m])

    visited = [[False for _ in range(m)] for _ in range(n)]

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

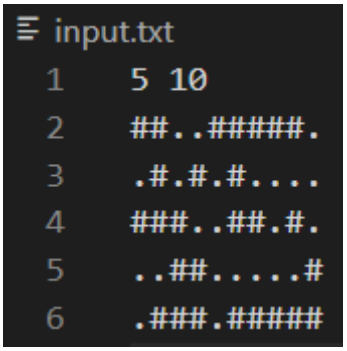
```
                            if x != 0:
                                stack.append((x-1, y))
                            if x != n-1:
                                stack.append((x+1, y))
                            if y != 0:
                                stack.append((x, y-1))
                            if y != m-1:
                                stack.append((x, y+1))

    d = open('output.txt', 'w')
    d.write(str(count))

if __name__ == "__main__":
    main()
```

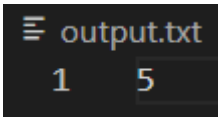
Создаётся список visited размером n x m, где все значения изначально равны False. Этот список отслеживает, была ли клетка уже обработана. Для исследования новой области используется алгоритм поиска в глубину (DFS), реализованный с помощью стека stack. Этот алгоритм помечает все клетки области как посещённые.

Результат работы кода на примерах из текста задачи:









Вывод: научилась решать задачу про грядки

Дополнительные задачи

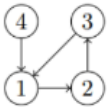
Задача №3 Циклы

Учебная программа по инфокоммуникационным технологиям определяет пререквизиты для каждого курса в виде списка курсов, которые необходимо пройти перед тем, как начать этот курс. Вы хотите выполнить проверку согласованности учебного плана, то есть проверить отсутствие циклических зависимостей. Для этого строится следующий ориентированный граф: вершины соответствуют курсам, есть направленное ребро (u, v) – курс u следует пройти перед курсом v. Затем достаточно проверить, содержит ли полученный граф цикл.

Проверьте, содержит ли данный граф циклы.

- **Формат ввода / входного файла (input.txt).** Ориентированный граф с n вершинами и m ребрами по формату 1.
- **Ограничения на входные данные.** $1 \le n \le 10^3, 0 \le m \le 10^3$.
- **Формат вывода / выходного файла (output.txt).** Выведите 1, если данный граф содержит цикл; выведите 0, если не содержит.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.
- Пример 1:

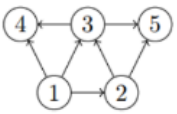
input	output
4 4 1 2 4 1 2 3 3 1	1



Этот граф содержит цикл: $3 \to 1 \to 2 \to 3$.

- Пример 2:

input	output
5 7 1 2 2 3 1 3 3 4 1 4 2 5 3 5	0



В этом графе нет циклов. Это можно увидеть, например, отметив, что все ребра в этом графе идут от вершины с меньшим номером к вершине с большим номером.

Листинг кода

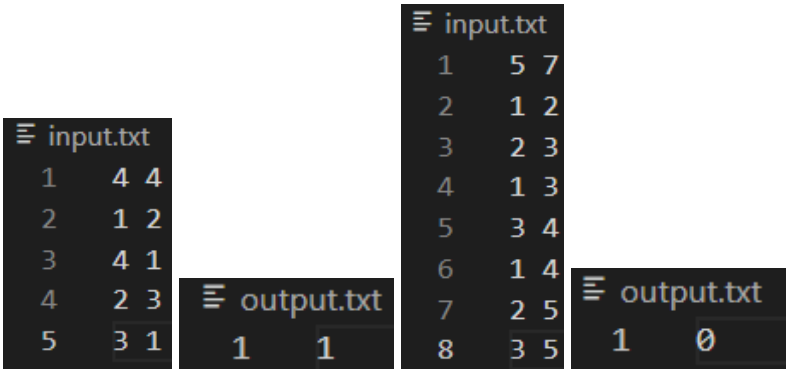
```
def solve():
    f = open('input.txt')
    n, m = map(int, f.readline().split())
    sides = {}
    for i in range(1, n+1):
        sides[i] = []
    for i in range(m):
```

```
        v1, v2 = map(int, f.readline().split())
        sides[v1].append(v2)

    for u in sides:
        visited = []
        parent = {}
        cur_node = u
        node_found = 1
        node_completed = 0
        while True:
            visited.append(cur_node)
            flag = False
            for i in sides[cur_node]:
                if i == u:
                    return 1
                if i not in visited:
                    parent[i] = cur_node
                    cur_node = i
                    node_found += 1
                    flag = True
                    break
            if not flag:
                node_completed += 1
                if node_found == node_completed:
                    break
                cur_node = parent[cur_node]
    return 0

def main():
    with open('output.txt', 'w') as f:
        f.write(str(solve()))

if __name__ == "__main__":
    main()
```

Пояснение к решению:

Основная идея — попытаться пройти по всем рёбрам, начиная с каждой вершины, и проверять, возвращаемся ли мы в начальную точку (образуется ли цикл).

Результат работы на примерах из текста к задаче:



Вывод по задаче:

Научилась решать задачу обнаружения цикла в ориентированном графе.

Задача №11 Алхимия

Алхимики средневековья владели знаниями о превращении различных химических веществ друг в друга. Это подтверждают и недавние исследования археологов.

В ходе археологических раскопок было обнаружено m глиняных табличек, каждая из которых была покрыта непонятными на первый взгляд символами. В результате расшифровки выяснилось, что каждая из табличек описывает одну алхимическую реакцию, которую умели проводить алхимики.

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.

- **Формат входных данных (input.txt) и ограничения.** Первая строка входного файла INPUT.TXT содержит целое число m ($0 \leq m \leq 1000$) – количество записей в книге. Каждая из последующих m строк описывает одну алхимическую реакцию и имеет формат «вещество1 -> вещество2», где «вещество1» – название исходного вещества, «вещество2» – название продукта алхимической реакции. $m + 2$-ая строка входного файла содержит название вещества, которое имеется исходно, $m + 3$-ая – название вещества, которое требуется получить.

  Во входном файле упоминается не более 100 различных веществ. Название каждого из веществ состоит из строчных и заглавных английских букв и имеет длину не более 20 символов. Строчные и заглавные буквы различаются.

- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите минимальное количество алхимических реакций, которое требуется для получения требуемого вещества из исходного, или -1, если требуемое вещество невозможно получить.

- Ограничение по времени. 1 сек.

- Ограничение по памяти. 16 мб.

- Примеры:

input.txt	output.txt
5 Aqua -> AquaVita AquaVita -> PhilosopherStone AquaVita -> Argentum Argentum -> Aurum AquaVita -> Aurum Aqua Aurum	2

input.txt	output.txt
5 Aqua -> AquaVita AquaVita -> PhilosopherStone AquaVita -> Argentum Argentum -> Aurum AquaVita -> Aurum Aqua Osmium	-1

- Проверяем **обязательно** – на acmp.

Листинг кода

```
from collections import deque

sides = {}

def short_path(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

def main():
    f = open('input.txt')
    m = int(f.readline())
    global sides
    for i in range(m):
        v1, sigh, v2 = map(str, f.readline().split())
        if v1 in sides:
            sides[v1].append(v2)
        else:
            sides[v1] = [v2]
    u = f.readline()[:-1]
    v = f.readline()[:-1]

    result = short_path(u, v)

    d = open('output.txt', 'w')
    d.write(str(result))

if __name__ == "__main__":
    main()
```

Пояснение к решению:

Используется алгоритм BFS (поиск в ширину) для нахождения кратчайшего пути. BFS гарантирует, что найдёт кратчайший путь в графе, поскольку он обрабатывает все узлы на каждом уровне до перехода на следующий.

Результат работы на примерах из текста задачи:

input.txt

```
5
Aqua -> AquaVita
AquaVita -> PhilosopherStone
AquaVita -> Argentum
Argentum -> Aurum
AquaVita -> Aurum
Aqua
Aurum
```

output.txt

```
1 2
```

input.txt

```
5
Aqua -> AquaVita
AquaVita -> PhilosopherStone
AquaVita -> Argentum
Argentum -> Aurum
AquaVita -> Aurum
Aqua
Osmium
```

output.txt

```
1 -1
```

Вывод по задаче:

Научилась решать задачу, используя алгоритм поиска в ширину.

Задача №12 Цветной лабиринт

В одном из парков одного большого города недавно был организован новый аттракцион Цветной лабиринт. Он 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. Кроме билета, он также получает описание пути, по которому он может выбраться из лабиринта. Это описание представляет собой последовательность цветов $c_1 \dots c_k$. Пользоваться ей надо так: 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.

В последнее время в администрацию парка стали часто поступать жалобы от заблудившихся в лабиринте людей. В связи с этим, возникла необходимость написания программы, проверяющей корректность описания и пути, и, в случае ее корректности, сообщающей номер комнаты, в которую ведет путь.

Описание пути некорректно, если на пути, который оно описывает, возникает ситуация, когда из комнаты нельзя пойти по коридору соответствующего цвета.

- **Формат входных данных (input.txt) и ограничения.** Первая строка входного файла INPUT.TXT содержит два целых числа n ($1 \le n \le 10000$) и m ($1 \le m \le 100000$) - соответственно количество комнат и коридоров в лабиринте. Следующие m строк содержат описания коридоров. Каждое описание содержит три числа u ($1 \le u \le n$), v ($1 \le v \le n$), c ($1 \le c \le 100$) - соответственно номера комнат, соединенных этим коридором, и цвет коридора. Следующая, $(m + 2)$-ая строка входного файла содержит длину описания пути - целое число k ($0 \le k \le 100000$). Последняя строка входного файла содержит k целых чисел, разделенных пробелами, - описание пути по лабиринту.
- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите строку INCORRECT, если описание пути некорректно, иначе выведите номер комнаты, в которую ведет описанный путь. Помните, что путь начинается в комнате номер один.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 16 мб.
- Примеры:

input.txt	output.txt
3 2 1 2 10 1 3 5 5 10 10 10 10 5	3

input.txt	output.txt
3 2 1 2 10 2 3 5 5 5 10 10 10 10	INCORRECT

input.txt	output.txt
3 2 1 2 10 1 3 5 4 10 10 10 5	INCORRECT

Листинг кода

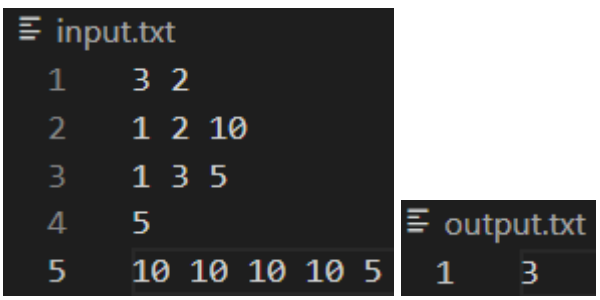
```
def labirint(k, path, sides):
    room_cur = 1
    for i in range(k):
        color = path[i]
        flag = False
        for vertex, side in sides[room_cur]:
            if side == color:
                room_cur = vertex
                flag = True
                break
        if not flag:
 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

```
def main():
    f = open('input.txt')
    n, m = map(int, f.readline().split())
    sides = {}
    for i in range(1, n+1):
        sides[i] = []
    for _ in range(m):
        u, v, c = map(int, f.readline().split())
        sides[u].append([v, c])
        sides[v].append([u, c])
    k = int(f.readline())
    path = list(map(int, f.readline().split()))

    d = open('output.txt', 'w')
    d.write(labirint(k, path, sides))

if __name__ == "__main__":
    main()
```

Пояснение к решению:

Этот код решает задачу о прохождении через лабиринт, представленный в виде графа. В лабиринте каждая комната связана с другими, и для перехода между комнатами необходимо двигаться через двери определённого цвета. Программа определяет, в какой комнате вы окажетесь после прохождения последовательности комнат через двери определённых цветов. Если в какой-то момент путь по нужному цвету невозможен, программа выводит "INCORRECT".

Результат работы на примерах из текста к задаче:



input.txt

```
3 2
1 2 10
2 3 5
5
5 10 10 10 10
```

output.txt

```
INCORRECT
```

input.txt

```
3 2
1 2 10
1 3 5
4
10 10 10 5
```

output.txt

```
INCORRECT
```

Вывод по задаче:

Научилась решать задачу про лабиринт