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

Выполнил:
Вилис З.М. (фамилия имя)
К3244 (номер группы)

Проверила:
Артамонова В.Е.

Санкт-Петербург
2023 г.

Содержание отчета

Задачи по варианту

Задание №1. Лабиринт[1 балл]

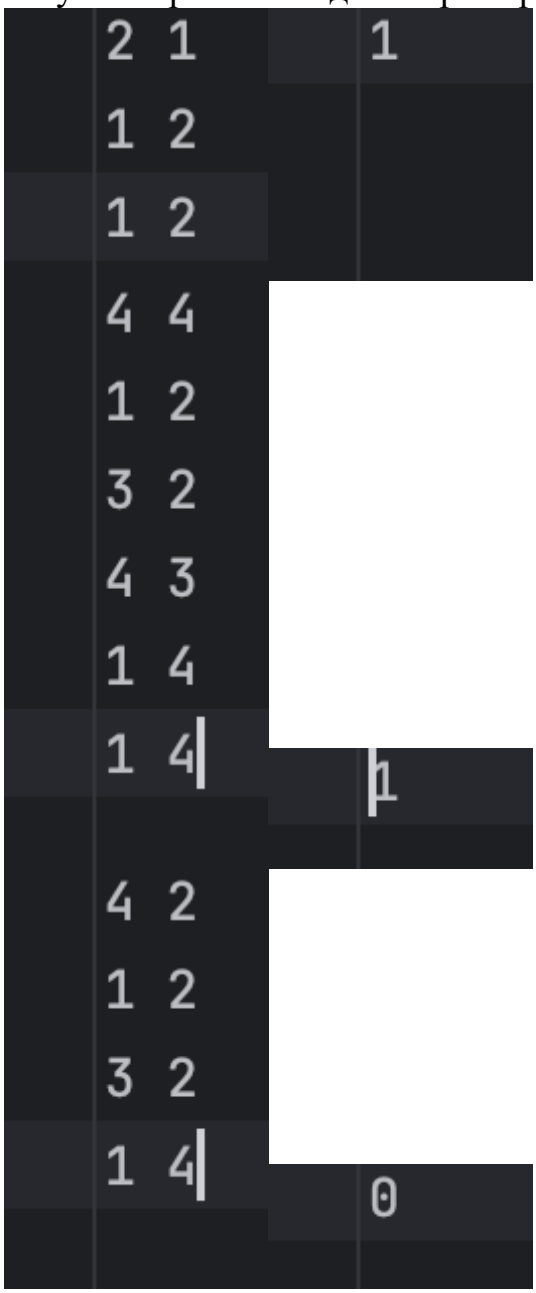
Лабиринт представляет собой прямоугольную сетку ячеек со стенками между некоторыми соседними ячейками. Вы хотите проверить, существует ли путь от данной ячейки к данному выходу из лабиринта, где выходом также является ячейка, лежащая на границе лабиринта (в примере, показанном на рисунке, есть два выхода: один на левой границе и один на правой границе). Для этого вы представляете лабиринт в виде неориентированного графа: вершины графа являются ячейками лабиринта, две вершины соединены неориентированным ребром, если они смежные и между ними нет стены. Тогда, чтобы проверить, существует ли путь между двумя заданными ячейками лабиринта, достаточно проверить, что существует путь между соответствующими двумя вершинами в графе.

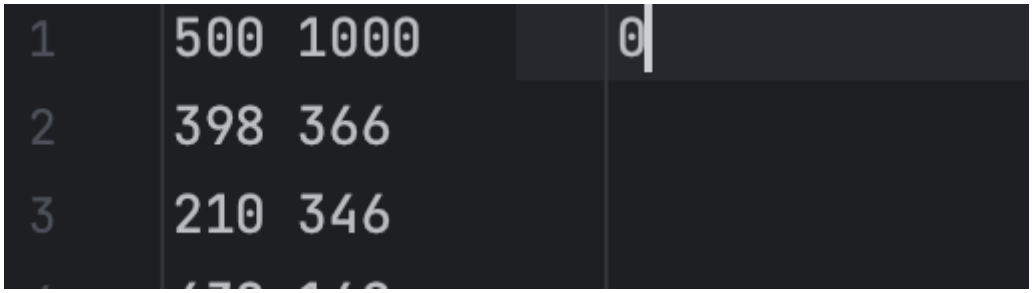
```
f = open("input_1.txt")
a = f.readline().split()
vertexes, ribs = int(a[0]), int(a[1])
gr = [[] for _ in range(vertexes)]
for i in range(ribs):
    s = f.readline().split()
    curr_v1, curr_v2 = int(s[0]), int(s[1])
    gr[curr_v1-1].append(curr_v2)
    gr[curr_v2-1].append(curr_v1)
f1, f2 = [int(x) for x in f.readline().split()]


def are_ribs_together(versh1, versh2, graph, pr):
    if int(versh2) in graph[versh1-1]:
        return 1
    else:
        for j in range(len(graph[versh1 - 1])):
            pr.append(versh1)
            if graph[versh1-1][j] not in pr:
                return are_ribs_together(graph[versh1 - 1][j],
versh2, graph, pr)
        return 0


pr = []
with open('output.txt', 'w') as z:
    z.write(str(are_ribs_together(f1, f2, gr, pr)))
```

Текстовое объяснение решения:
Вершины записываются мной в двумерный список по принципу того, что под каждую вершину отведен свой список, содержащий вершины, смежные ей. Далее я прописала функцию are_ribs_together, принимающую на вход вершины, между которыми ищется путь, двумерный список, о

котором написано выше и список, который будет содержать пройденные вершины. Если вершина конечная содержится в текущем списке для вершины, которую мы рассматриваем сейчас, то возвращается 1. Иначе мы проходимся по этому списку и для каждой вершины в нем проверяем выполнение этой функции. Получается рекурсия. Также задействуется список уже пройденных вершин, чтобы в ходе выполнения функции рекурсия не зацикливалась.

Результат работы кода на примерах из задачи:





	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.0004761219024658203	22.8
Пример из задачи	0.0008678436279296875	23.2
Пример из задачи	0.0008130073547363281	23.2
Верхняя граница диапазона значений входных данных из текста задачи(1000 элеметов)	0.0019190311431884766	23.2

Вывод по задаче: Задача научила меня исправлять ошибки, связанные с рекурсией.

Задание №7. Двудольный граф [1.5 балла]

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.

```
from collections import deque
file = open('input7.txt')
```

```
vert, ribs = file.readline().split()
graph = {}
for i in range(int(ribs)):
    first, second = file.readline().split()
    f, s = int(first), int(second)
    if f in graph:
        graph[f].append(s)
    else:
        graph[f] = [s]
    if s in graph:
        graph[s].append(f)
    else:
        graph[s] = [f]


def is_bipartite(graph):
    colors = {}
    visited = set()
    queue = deque()

    def bfs(node):
        colors[node] = 1
        visited.add(node)
        queue.append(node)
        while queue:
            current = queue.popleft()
            for neighbor in graph[current]:
                if neighbor not in visited:
                    colors[neighbor] = 1 - colors[current]
                    visited.add(neighbor)
                    queue.append(neighbor)
                elif colors[neighbor] == colors[current]:
                    return False
        return True

    for node in graph:
        if node not in visited:
            if not bfs(node):
                return False
    return True


z = open('output.txt', 'w')
if is_bipartite(graph):
    z.write(str(1))
else:
    z.write((str(0)))
```

Текстовое объяснение решения:
Граф считывается из файла и записывается в формате словаря смежности. Далее при помощи алгоритма раскраски графа проверяется его двудольность. Начинаем с выбранной вершины и помечаем ее цветом 1. Добавляем вершину в очередь. В цикле вытаскиваем соседей вершины. Если соседняя еще не посещена мы присваиваем ей цвет противоположный цвету текущей. Если сосед уже был посещен и его цвет совпадает с цветом текущей то значит граф не двудольный.

Результат работы кода на примерах из задачи:

	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.00536372377328323	22.8
Пример из задачи	0.006372632763281263	23.2

Пример из задачи	0.0072362168737267 16	23.2
Верхняя граница диапазона значений входных данных из текста задачи(1000 элеметов)	0.27362173897289312239	23.2

Вывод по задаче: Задача научила меня раскрашивать граф.

Задание №13. Грядки[3 балла]

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:

- излюбогоквадратаэтойгрядкиможнопопастьвлюбойдругойквадратэтойжегрядки,последовательнопереходя по грядке из квадрата в квадрат через их общую сторону;
- никакиедвегрядкинепересекаютсяинекасаютсядругдруганиповертикальной,нипогоризонтальнойсторонам квадратов (касание грядок углами квадратов допускается).
  Подсчитайте количество грядок на садовом участке.

```
from collections import deque


def bfs(grid, visited, row, col):
    rows = len(grid)
    cols = len(grid[0])
    queue = deque([(row, col)])
    visited[row][col] = True
    while queue:
        r, c = queue.popleft()
        for dr, dc in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
            nr, nc = r + dr, c + dc
            if 0 <= nr < rows and 0 <= nc < cols and grid[nr][nc] ==
'#' and not visited[nr][nc]:
                visited[nr][nc] = True
                queue.append((nr, nc))


def count_gardens(grid):
    rows = len(grid)
    cols = len(grid[0])
    visited = [[False] * cols for _ in range(rows)]
    count = 0
    for i in range(rows):
        for j in range(cols):
            if grid[i][j] == '#' and not visited[i][j]:
                bfs(grid, visited, i, j)
```

```
                count += 1
    return count


with open('input.txt', 'r') as file:
    n, m = map(int, file.readline().split())
    garden = [list(file.readline().strip()) for _ in range(n)]
garden_count = count_gardens(garden)
with open('output.txt', 'w') as file:
    file.write(str(garden_count))
```

Текстовое объяснение решения:
Алгоритм поиска в ширину позволяет проходить по всем клеткам в списке и помечает их как пройденные. И определяет какие будут пройдены дальше. Далее считается количество грядок помеченных #. И если она не посещена, то для нее запускается BFS.

	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.28378273812930812	9.855
Пример из задачи	0.237812739081239323	9.855
Пример из задачи	0.232839485392337293	9.855
Верхняя граница диапазона значений входных данных из текста задачи(1000 элеметов)	0.346278192372837192	10.85

Вывод по задаче: Задача реализуется при помощи поиска в ширину.

Дополнительные задачи

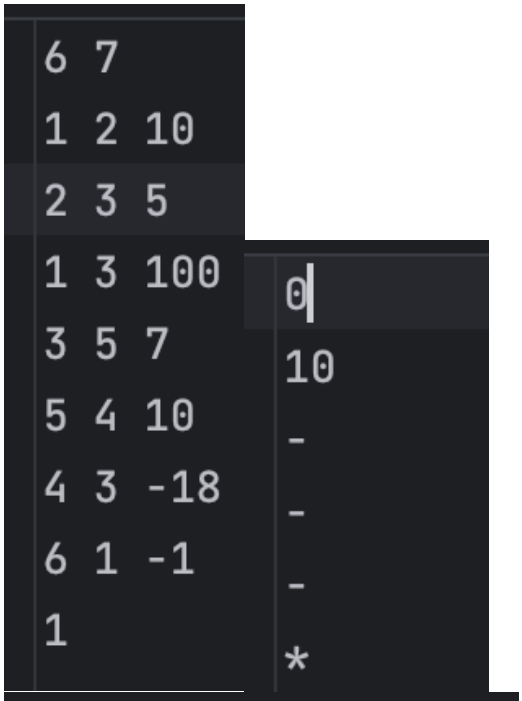
Задание №10. Оптимальный обмен валют [2 балла]

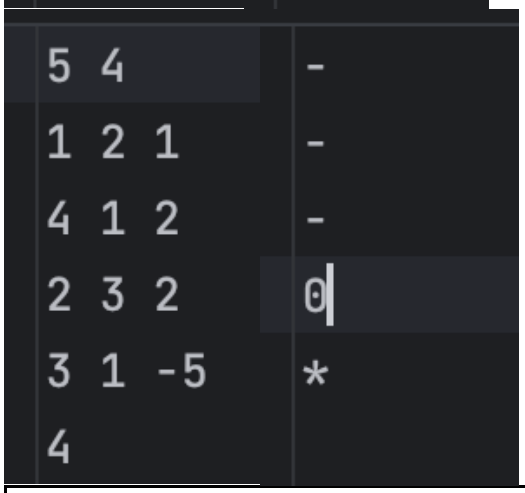
Теперь вы хотите вычислить оптимальный способ обмена данной вам валюты ci на *все* другие валюты. Для этого вы находите кратчайшие пути из вершины ci во все остальные вершины.

Дан ориентированный граф с возможными отрицательными весами ребер, у которого n вершин и m ребер, а также задана одна его вершина s. Вычислите длину кратчайших путей из s во все остальные вершины графа.

```
def shortest_dist(graph, n, m, s):
    path = [float('inf')] * (n + 1)
    path[s] = 0
    for _ in range(n - 1):
        for u, v, w in graph:
            if path[u] != float('inf') and path[u] + w < path[v]:
                path[v] = path[u] + w

    for u, v, w in graph:
        if path[u] != float('inf') and path[u] + w < path[v]:
            path[v] = float('-inf')
    return path


with open('input10.txt', 'r') as file:
    n, m = map(int, file.readline().split())
    graph = [list(map(int, file.readline().split())) for _ in
range(m)]
    s = int(file.readline())

distances = shortest_dist(graph, n, m, s)
with open('output.txt', 'w') as file:
    for i in range(1, n + 1):
        if distances[i] == float('-inf'):
            file.write("-\n")
        elif distances[i] == float('inf'):
            file.write("*\n")
        else:
            file.write(str(distances[i]) + "\n")
```

Текстовое объяснение решения:
функция shortest_dist вычисляет кратчайшее расстояние от стартовой поисковой вершины до всех остальных в графе. Создается path содержащий эти самые расстояния. Расстояние от поисковой равно нулю. Далее цикл пробегает по всем вершинам и ищет расстояния.





	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.026237689273987291	8.855
Пример из задачи	0.002739202839292832	9.845
Пример из задачи	0.0023829319827398172	9.985
Верхняя граница диапазона значений входных данных из текста задачи(1000 элеметов)	0.026372638712687316 23	10.95

Вывод по задаче: Задача реализуется благодаря алгоритму поиска кратчайших расстояний.

Задание №11. Алхимия[3 балла]

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

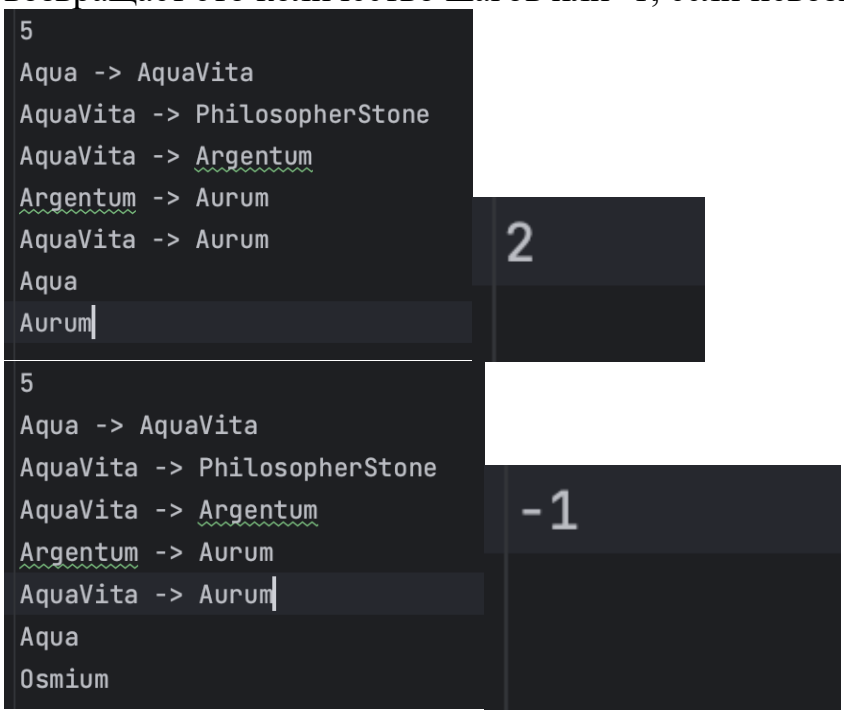
В ходе археологических раскопок было обнаружено m глиняных табличек, каждая из которых была покрыта непонятными на первый взгляд символами. В результате расшифровки выяснилось, что каждая из табличек описывает одну алхимическую реакцию, которую умели проводить алхимики.

Результатом алхимической реакции является превращение одного вещества в другое. Задан набор алхимических реакций, описанных на найденных глиняных табличках, исходное вещество и требуемое вещество. Необходимо вы- яснить: 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.

```
from collections import defaultdict, deque


def min_reactions(reactions, start, target):
    graph = defaultdict(list)
    for reaction in reactions:
        reactant, product = reaction.split(' -> ')
        graph[reactant].append(product)
    visited = set()
    queue = deque([(start, 0)])
    while queue:
        substance, steps = queue.popleft()
        if substance == target:
            return steps
        visited.add(substance)
        for product in graph[substance]:
            if product not in visited:
                queue.append((product, steps + 1))
    return -1


with open("input11.txt", "r") as file:
    m = int(file.readline().strip())
    reactions = [file.readline().strip() for _ in range(m)]
    start = file.readline().strip()
    target = file.readline().strip()
result = min_reactions(reactions, start, target)
with open("output.txt", "w") as file:
```

```
    file.write(str(result))
```

Текстовое объяснение решения:
Определение функции min_reactions(reactions, start, target):
Функция принимает список реакций, стартовое вещество и целевое вещество. Строится граф реакций, где ключ -то из чего может появиться реакция, а значение - продукт реакции. С помощью обхода в ширину (BFS) находится минимальное количество шагов для реакции. Функция возвращает это количество шагов или -1, если невозможно.



	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.26237689273987291	10.855
Пример из задачи	0.032671621273981732	10.845
Пример из задачи	0.02637821392873183	9.985
Верхняя граница диапазона значений входных данных из текста задачи(1000 элеметов)	0.05732738723923293	10.95

Вывод по задаче: Задача реализуется при помощи поиска в ширину.

Задание №15. Герои [3 балла]

Коварный кардинал Ришелье вновь организовал похищение подвесок королевы Анны; вновь спасать королеву при- ходится героическим мушкетерам. Атос, Портос, Арамис и д'Артаньян уже перехватили агентов кардинала и вернули украденное; осталось лишь передать подвески королеве Анне. Королева ждет мушкетеров в дворцовом саду. Дворцо- вый сад имеет форму прямоугольника и разбит на участки, представляющие собой небольшие садики, содержащие коллекции растений из разных климатических зон. К сожалению, на некоторых участках, в том числе на всех участках, расположенных на границах сада, уже притаились в засаде гвардейцы кардинала; на бой с ними времени у мушкетеров нет. Мушкетерам удалось добыть карту сада с отмеченными местами засад; теперь им предстоит выбрать наиболее оптимальные пути к королеве. Для надежности друзья разделили между собой спасенные подвески и проникли в сад поодиночке, поэтому начинают свой путь к королеве с разных участков сада. Двигаются герои по максимально короткой возможной траектории.

Марлезонский балет вот-вот начнется; королева не в состоянии ждать героев больше L минут; ровно в начале L + 1- ой 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.

```
from collections import deque


def bfs(garden, start, destination, time_limit):
    rows, cols = len(garden), len(garden[0])
    queue = deque([(start, 0)])
    visited = set([start])
    while queue:
        (x, y), time = queue.popleft()
        if (x, y) == destination:
            return time <= time_limit
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            nx, ny = x + dx, y + dy
            if 1 <= nx < rows - 1 and 1 <= ny < cols - 1 and
garden[nx][ny] == '0' and (nx, ny) not in visited:
                visited.add((nx, ny))
                queue.append(((nx, ny), time + 1))
    return False


with open('input.txt', 'r') as file:
```

```
    n, m = map(int, file.readline().split())
    garden = [file.readline().strip() for _ in range(n)]
    qx, qy, l = map(int, file.readline().split())
    musketeers = [tuple(map(int, file.readline().split())) for _ in
range(4)]
total= 0
destination = (qx - 1, qy - 1)
for mx, my, pendants in musketeers:
    if bfs(garden, (mx - 1, my - 1), destination, l):
        total += pendants
with open('output.txt', 'w') as file:
    file.write(str(total))
```

Текстовое объяснение решения:
Функция bfs принимает на вход garden: двумерный массив (список списков) представляющий собой карту сада. start: координаты начальной позиции поиска.
destination: координаты целевой позиции, где находится сундук. time_limit: ограничение на время для достижения цели. Создается очередь queue с начальным элементом (начальная позиция, текущее время). Создается множество visited для отслеживания посещенных клеток. В цикле while извлекается элемент из очереди. Если текущая позиция совпадает с целевой позицией, проверяется, не превышено ли ограничение времени, и возвращается True, если условие выполнено.
Для каждого из соседних положений (сверху, снизу, слева, справа) проверяется, находится ли оно в пределах карты, не содержит ли препятствия и не было ли уже посещено. Если все условия выполнены, соседнее положение добавляется в очередь. Если целевая позиция не была достигнута до превышения ограничения времени, возвращается False. Для каждого мушкетера вызывается функция bfs с его координатами и координатами целевой позиции. Если цель достижима в рамках ограничения времени, количество сундуков увеличивается на количество сундуков в этом саду.

```
5 5
11111
10001
10001
10001
11111
4 4 10
2 2 1
2 3 2
3 2 3
3 3 4
```

```
10
```

	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.00536372377328323	22.7
Пример из задачи	0.082736827169287321	22.7
Пример из задачи	0.072736781729873298	22.8
Верхняя граница диапазона значений входных данных из текста задачи(1000 элеметов)	0.4373987298372897323	23.2

Вывод по задаче: Задача очень интересная.

Вывод

Лабораторная обучила меня работе с графами.