Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Алгоритмы и структуры данных

Лабораторная работа №2_3

Графы

Выполнил:

Бараканов Жаргал Мырзабекович

Факультет ИКТ

Группа К3121

Преподаватель:

Харьковская Татьяна Александровна

Санкт-Петербург

2022

Задание 1.

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

- **Формат ввода / входного файла (input.txt).** Неориентированный граф с n вершинами и m ребрами по формату 1. Следующая строка после ввода всего графа содержит две вершины u и v;
- **Ограничения на входные данные.** $2 \leq n \leq 10^3$, $1 \leq m \leq 10^3$, $1 \leq u, v \leq n$, $u \neq v$;
- **Формат вывода / выходного файла (output.txt).** Выведите 1, если есть путь между вершинами u и v; выведите 0, если пути нет;
- **Ограничение по времени.** 5 сек;
- **Ограничение по памяти.** 512 Мб.

Решение:

В начале считываем данные из исходного файла в словарь “sides”, 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 “parent” будет указывать на вершину графа, из которой попали в текущую вершину.

```
with open('input.txt') as f:
    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)
        sides[v2].append(v1)
    visited = [False] * (n+1)
    parent = {}
    u, v = map(int, f.readline().split())
```

Для обхода в глубину нам также понадобятся еще две переменных – “node_found” и “node_completed”. В первой хранится число найденных вершин графа, а во второй – число пройденных. В цикле происходит обход графа в глубину. Если встретится искомая вершина, то программа остановится и выведет ‘1’. Также с каждой новой найденной вершиной увеличивается первый счетчик “node_found”. Если же все ребра вершины посещены, увеличивается счетчик “node_completed” и происходит возврат к вершине, из которой пришли к данной. Если окажется, что эти счетчики равны, то цикл остановится и выведется ‘0’. Это будет означать, что среди вершин, достижимых из начальной, нет искомой.

```
cur_node = u
node_found = 1
node_completed = 0
while True:
    visited[cur_node] = True
    flag = False
    for i in sides[cur_node]:
        if i == v:
            print(1)
            exit()
        if not visited[i]:
            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]
print(0)
```

Задание 3.

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.

Решение:

Считываем граф, учитывая, что он ориентированный.

```
with open('input.txt') as f:
    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)
```

По сути, чтобы обнаружить цикл, нужно найти вершину, которая достижима сама из себя. Для пользуемся обходом в глубину и пробегаемся по всем вершинам графа. Если таковая вершина находится, значит в графе есть цикл и выводится ‘1’. В обратном случае выводится ‘0’.

```
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:
                print(1)
                exit()
            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]
print(0)
```

Задание 5.

Нужно вычислить количество компонентов сильной связности заданного ориентированного графа с n вершинами и m ребрами.

- **Формат ввода / входного файла (input.txt).** Ориентированный граф с n вершинами и m ребрами по формату 1;
- **Ограничения на входные данные.** $1 \leq n \leq 10^4$, $0 \leq m \leq 10^4$;
- **Формат вывода / выходного файла (output.txt).** Выведите число – количество компонентов сильной связности;
- **Ограничение по времени.** 5 сек;
- **Ограничение по памяти.** 512 Мб;

Решение:

Для решения данной задачи также воспользуемся обходом в глубину. В данной задаче нам понадобится обойти исходный и транспонированный

графы в глубину, чтобы определить количество компонентов связности. В первом случае мы параллельно добавляем найденные вершины в массив “order”.

```python
def dfs(u):
    global visited, parent, sides, order
    cur_node = u
    visited[cur_node] = True
    node_found = 1
    node_completed = 0
    while True:
        flag = False
        for i in sides[cur_node]:
            if i not in visited:
                parent[i] = cur_node
                cur_node = i
                visited[cur_node] = True
                order.append(cur_node)
                node_found += 1
                flag = True
                break
        if not flag:
            node_completed += 1
            if node_found == node_completed:
                break
            cur_node = parent[cur_node]
    order.append(u)
```

```python
def dfsTranspon(u):
    global visited, parent, sides_trans, count
    cur_node = u
    visited[cur_node] = True
    node_found = 1
    node_completed = 0
    while True:
        flag = False
        for i in sides_trans[cur_node]:
            if i not in visited:
                parent[i] = cur_node
                cur_node = i
                visited[cur_node] = True
                node_found += 1
                flag = True
                break
        if not flag:
            node_completed += 1
            if node_found == node_completed:
                break
            cur_node = parent[cur_node]
```

В основной части кода считываем ориентированный граф и сразу формируем его транспонированную версию. Далее сначала проходимся по исходному графу, а потом по сформированному массиву “order”. Если текущая

вершина не пройдена, то запускаем обход в глубину и увеличиваем счетчик “count”, значение которого и является ответом.

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

order = []
visited = {}
parent = {}
for u in sides:
    if u not in visited:
        dfs(u)

visited = {}
parent = {}
order = order[::-1]
count = 0
for u in order:
    if u not in visited:
        dfsTranspon(u)
        count += 1
print(count)
```

Задание 6.

Дан *неориентированный* 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).

- **Формат ввода / входного файла (input.txt).** Неориентированный граф задан по формату 1. Следующая строка содержит две вершины u и v;
- **Ограничения на входные данные.** $2 \le n \le 10^5$, $0 \le m \le 10^5$, $1 \le u, v \le n$, $u \neq v$;
- **Формат вывода / выходного файла (output.txt).** Выведите минимальное количество ребер в пути из u в v. Выведите -1, если пути нет;
- Ограничение по времени. 10 сек;
- Ограничение по памяти. 512 Мб;

Решение:

Для решения данной задачи нам понадобится обход графа в ширину. Для этого используется очередь. В очередь добавляются все еще не посещенные вершины, доступные из текущей, вместе с количеством ребер, которое надо пройти, чтобы достичь данной вершины из исходной.

```python
from collections import deque

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
```

Считываем неориентированный граф и кладем результат функции “shortPath” в переменную “result”, значение которой и есть ответ.

```python
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())

result = shortPath(u, v)
print(result)
```

Задание 7.

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.** $1 \leq n \leq 10^5$, $0 \leq m \leq 10^5$;

- **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.

Решение:

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 записанным в вышеупомянутый массив. Если они различаются, то возвращаем ‘0’. Иначе в конце возвращается ‘1’.

```
from collections import deque

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
```

В основном коде считываем неориентированный граф, запускаем нашу функцию и выводим ее результат.

```
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)

total_colours = [None] * (n+1)
result = halfGraph(1)
print(result)
```

Задание 8.

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).

- **Формат ввода / входного файла (input.txt).** Ориентированный взвешенный граф задан по формату 1. Следующая строка содержит две вершины u и v;
- **Ограничения на входные данные.** $1 \leq n \leq 10^4$, $0 \leq m \leq 10^5$, $1 \leq u, v \leq n$, $u \neq v$, вес каждого ребра – неотрицательное целое число, не превосходящее 10^8;
- **Формат вывода / выходного файла (output.txt).** Выведите минимальный вес пути из u в v. Выведите -1, если пути нет;
- **Ограничение по времени.** 10 сек;
- **Ограничение по памяти.** 512 Мб.

Решение:

В этой задаче также воспользуемся обходом в ширину. В очереди будем хранить путь до текущей вершину, а также ее ‘родителя’ – вершину, из которой мы пришли. 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, ибо ставить метки прохождения вершин мы не можем, из-за того что граф не ориентирован.

```python
from collections import deque

def shortPath(u, v):
    global sides
    min_path = float('inf')
    search_queue = deque()
    search_queue.append((u, 0, None))
    while search_queue:
        cur_node, path, parent = search_queue.popleft()
        if cur_node == v:
            if path < min_path:
                min_path = path
        else:
            for node in sides[cur_node]:
                if node[0] != parent:
                    search_queue.append((node[0], path + node[1], cur_node))
    if min_path == float('inf'):
        return -1
    return min_path
```

При считывании графа убираем все ‘лишние’ ребра (то есть ребра между вершинами, длина которых больше, чем длина минимального ребра между ними). Далее запускаем функцию и выводим ответ.

```python
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, l = map(int, f.readline().split())
        flag = False
        for node in sides[v1]:
            if node[0] == v2 and l < node[1]:
                node[1] = l
                flag = True
        if not flag:
            sides[v1].append([v2, l])
        for node in sides[v1]:
            if node[0] == v2 and l < node[1]:
                node[1] = l
                flag = False
        if flag:
            sides[v2].append([v1, l])
    u, v = map(int, f.readline().split())

result = shortPath(u, v)
print(result)
```

Задание 11.

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 ($0 \le m \le 1000$) – 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.

Решение:

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

```python
from collections import deque

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
```

```python
with open('input.txt') as f:
    m = int(f.readline())
    sides = {}
    for i in range(m):
        v1, sign, 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 = shortPath(u, v)
with open('output.txt', 'w') as f:
    f.write(str(result))
```

17186508	21.05.2022 19:03:45	Бараканов Жаргал Мырзабекович	0743	Python	Accepted		0,046	802 Кб

Задание 12.

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 $c_1 \ldots c_k$. 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.

Решение:

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’.

```python
def labirint(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)
```

Граф считывается с добавлением цвета ребер между вершинами.

```python
with open('input.txt') as f:
    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()))

with open('output.txt', 'w') as f:
    f.write(labirint(k, path, sides))
```

17171158	19.05.2022 2:14:20	Бараканов Жаргал Мырзабекович	0601	Python	Accepted		0,14	13 Мб

Задание 13.

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.

Решение:

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

```
with open('input.txt') as f:
    N, M = map(int, f.readline().split())
    field = []
    for i in range(N):
        field.append(f.readline()[:M])

visited = [[False for _ in range(M)] for _ in range(N)]
```

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

```
count = 0
for i in range(N):
    for j in range(M):
        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 != N-1:
                            stack.append((x+1, y))
                        if y != 0:
                            stack.append((x, y-1))
                        if y != M-1:
                            stack.append((x, y+1))

with open('output.txt', 'w') as f:
    f.write(str(count))
```

17169878	18.05.2022 18:35:36	Бараканов Жаргал Мырзабекович	0432	Python	Accepted		0,156	4814 Кб

Задание 14.

Между некоторыми деревнями края Власюки ходят автобусы. Поскольку пассажиропотоки здесь не очень большие, то автобусы ходят всего несколько раз в день.

Марии Ивановне требуется добраться из деревни *d* в деревню *v* как можно быстрее (считается, что в момент времени 0 она находится в деревне *d*).

- **Формат входных данных (input.txt) и ограничения.** Во входном файле INPUT.TXT записано число N - общее число деревень ($1 \le N \le 100$), номера деревень *d* и *v*, затем количество автобусных рейсов R ($0 \le R \le 10000$). Затем идут описания автобусных рейсов. Каждый рейс задается номером деревни отправления, временем отправления, деревней назначения и временем прибытия (все времена - целые от 0 до 10000). Если в момент *t* пассажир приезжает в деревню, то уехать из нее он может в любой момент времени, начиная с *t*;
- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT вывести минимальное время, когда Мария Ивановна может оказаться в деревне *v*. Если она не сможет с помощью указанных автобусных рейсов добраться из *d* в *v*, вывести -1;
- **Ограничение по времени.** 1 сек;
- **Ограничение по памяти.** 16 Мб.

Решение:

Считываем описания рейсов в массив “buses”, пункты отправления и прибытия.

```
with open('input.txt') as f:
    N = int(f.readline())
    buses = [[] for _ in range(N+1)]
    d, v = map(int, f.readline().split())
    R = int(f.readline())
    for i in range(R):
        n1, t1, n2, t2 = map(int, f.readline().split())
        buses[n1].append((t1, n2, t2))
```

Далее формируем массив “Time”? который мы заполняем бесконечностями, а ячейку пункта отправления обнуляем. Данный массив

будет показывать минимальное время прибытия в соответствующую деревню. Также понадобится массив “visited” для обозначения того, проверили ли мы все рейсы из данной деревни или нет. В цикле мы ищем деревню с минимальным временем прибытия и отмечаем ее, как проверенную на все рейсы. Затем проверяем рейсы из данной деревни. Если время его отправления из нее меньше, чем время прибытия в нее, а время прибытия в другую деревню меньше той, что была, то заменяем время прибытия в деревню по соответствующему рейсу. Таким образом мы получим массив прибытий в деревни за минимально возможное время. Если же значение времени осталось бесконечным, то в данную деревню невозможно попасть с данными рейсами.

```
INF = float('inf')
Time = [INF] * (N+1)
Time[d] = 0
visited = [False] * (N+1)
while True:
    min_time = INF
    for i in range(1, N+1):
        if not visited[i] and Time[i] < min_time:
            min_time = Time[i]
            min_village = i
    if min_time == INF:
        break
    n1 = min_village
    visited[n1] = True
    for t1, n2, t2 in buses[n1]:
        if Time[n1] <= t1 and t2 <= Time[n2]:
            Time[n2] = t2
```

Записываем результат в файл.

```
with open('output.txt', 'w') as f:
    if Time[v] == INF:
        f.write('-1')
    else:
        f.write(str((Time[v])))
```

17170747	18.05.2022 21:35:56	Бараканов Жаргал Мырзабекович	0134	Python	Accepted		0,062	1542 Кб

Задание 15.

Коварный кардинал Ришелье вновь организовал похищение подвесок королевы Анны; вновь спасать королеву приходится 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. Дворцовый 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.

- **Формат входных данных (input.txt) и ограничения.** Первая строка входного файла INPUT.TXT содержит целые числа N и M ($1 \le N, M \le 20$) – размеры сада. Далее идут N строк по M символов в каждом; символы ’0’ соответствуют участкам, на которых нет засады, символы ’1’ – участкам, на которых разместились гвардейцы. В N + 2-ой строке теста записано три целых числа: координаты участка, на котором королева будет ждать мушкетёров (Q_x, Q_y) ($1 < Q_x < N$, $1 < Q_y < M$) и время в минутах до начала балета ($1 \le L \le 1000$). В N + 3-ей строке записаны через пробел целые числа координаты участка, с которого стартует Атос (A_x, A_y) ($1 < A_x < N$, $1 < A_y < M$) и количество подвесок, хранящихся у него ($1 \le P_a \le 1000$). В N + 4, N + 5 и N + 6-ой строках аналогично записаны стартовые координаты и количество подвесок у Портоса, Арамиса и д’Артаньяна;
- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите количество подвесок, которое королева успеет получить у мушкетеров до начала балета;
- Ограничение по времени. 1 сек;
- Ограничение по памяти. 16 Мб.

Решение:

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.

```python
from collections import deque

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
```

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.

```python
with open('input.txt') as f:
    N, M = map(int, f.readline().split())
    garden = []
    for _ in range(N):
        garden.append(f.readline()[:M])
    x_queen, y_queen, L = map(int, f.readline().split())
    x_atos, y_atos, p_atos = map(int, f.readline().split())
    x_portos, y_portos, p_portos = map(int, f.readline().split())
    x_aramis, y_aramis, p_aramis = map(int, f.readline().split())
    x_dartan, y_dartan, p_dartan = map(int, f.readline().split())

result = toQueen(x_atos, y_atos, p_atos) + toQueen(x_portos, y_portos, p_portos)\
         + toQueen(x_aramis, y_aramis, p_aramis) + toQueen(x_dartan, y_dartan, p_dartan)
with open('output.txt', 'w') as f:
    f.write(str(result))
```

17171144	19.05.2022 1:30:31	Бараканов Жаргал Мырзабекович	0846	Python	Accepted		0,062	814 Кб

Задание 16.

Рассмотрим программу, состоящую из n процедур $P_1, P_2, \ldots, P_n$. Пусть для каждой процедуры известны процедуры, которые она может вызывать.

Процедура P называется потенциально рекурсивной, если существует такая последовательность процедур Q_0, Q_1, . . . , Q_k, что $Q_0 = Q_k = P$ и для $i = 1...k$ процедура Q_{i-1} может вызвать процедуру Q_i. В этом случае задача будет заключаться в определении для каждой из заданных процедур, является ли она потенциально рекурсивной.

Требуется написать программу, которая позволит решить названную задачу.

- **Формат входных данных (input.txt) и ограничения.** Первая строка входного файла INPUT.TXT содержит целое число n – количество процедур в программе ($1 \le n \le 100$). Далее следуют n блоков, описывающих процедуры. После каждого блока следует строка, которая содержит 5 символов «*».

Описание процедуры начинается со строки, содержащий ее идентификатор, состоящий только из маленьких букв английского алфавита и цифр. Идентификатор не пуст, и его длина не превосходит 100 символов. Далее идет строка, содержащая число k ($k \le n$) – количество процедур, которые могут быть вызваны описываемой процедурой. Последующие k строк содержат идентификаторы этих процедур – по одному идентификатору на строке.

Различные процедуры имеют различные идентификаторы. При этом ни одна процедура не может вызвать процедуру, которая не описана во входном файле;

- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT для каждой процедуры, присутствую щей во входных данных, необходимо вывести слово YES, если она является потенциально рекурсивной, и слово NO – в противном случае, в том же порядке, в каком они перечислены во входных данных;
- **Ограничение по времени.** 1 сек;
- **Ограничение по памяти.** 16 Мб.

Решение:

Для чтобы функция была потенциально рекурсивной, нужно, чтобы в графе существовал цикл, включающий данную функцию. Для этого с помощью обхода в глубину ищем из вершины графа в саму себя, как и в задании №3.

```python
with open('input.txt') as f:
    n = int(f.readline())
    sides = {}
    for i in range(n):
        v1 = f.readline()
        sides[v1] = []
        k = int(f.readline())
        for j in range(k):
            v2 = f.readline()
            sides[v1].append(v2)
        edge = f.readline()
```

```python
    for u in sides:
        visited = []
        parent = {}
        cur_node = u
        node_found = 1
        node_completed = 0
        while True:
            visited.append(cur_node)
            flag = False
            found = False
            for i in sides[cur_node]:
                if i == u:
                    f.write('YES\n')
                    found = True
                    break
                if i not in visited:
                    parent[i] = cur_node
                    cur_node = i
                    node_found += 1
                    flag = True
                    break
            if found:
                break
            if not flag:
                node_completed += 1
                if node_found == node_completed:
                    break
                cur_node = parent[cur_node]
        if not found:
            f.write('NO\n')
```

17186586	21.05.2022 19:32:46	Бараканов Жаргал Мырзабекович	0345	Python	Accepted		0,062	2138 Кб

Вывод:

В ходе данной лабораторной работы была изучена и применена на практике такая структура данных, как графы. Рассмотрены различные виды графов, типы их обходов и другие алгоритмы, связанные с графами.