

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 27

Выполнил:

Филиппов А.Э.

К3139

Проверила:

Артамонова В.Е.

Санкт-Петербург

2023 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3-13
Задача №2. Компоненты [1 балл]	3-6
Задача №8. Стоимость полета [1.5 балла]	6-9
Задача №11. Алхимия [3 балла]	10-13
Вывод	14

Задачи по варианту

Задача №2. Компоненты [1 балл]

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 связности в нем.

- **Формат ввода / входного файла (input.txt).** Неориентированный граф с n вершинами и m ребрами по формату 1.
- **Ограничения на входные данные.** $1 \leq n \leq 103$, $0 \leq m \leq 103$.
- **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.

```

import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

def dfs(v):
    if not visited[v]:
        visited[v] = True
        for i in range(len(adj_list[v])):
            el = adj_list[v][i]
            if not visited[el]:
                dfs(el)

n, m = map(int, lines[0].split())
adj_list = [[] for i in range(n)]
visited = [False] * n
count = 0

for i in range(m):
    a, b = map(int, lines[i+1].split())
    a -= 1
    b -= 1
    adj_list[a].append(b)
    adj_list[b].append(a)

for i in range(n):
    if not visited[i]:
        count += 1
        dfs(i)
out.write(str(count))

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Для нахождения компонент связности в графе используется обход в глубину. Сам граф хранится в списке связности.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```

4 2
1 2
3 2

```

RMAL input

Результат работы кода на максимальных и минимальных значениях:
(скрины input output файлов)

```

1 0

```

RMAL input.txt
put.txt" 1L, 4

```

1000 999
1 1 2
2 2 3
3 3 4
4 4 5
5 5 6
6 6 7
7 7 8
8 8 9
9 9 10
10 10 11
11 11 12
12 12 13
13 13 14

```

RMAL input.txt

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00018111300050804857	13890
Пример из задачи	0.00019918500038329512	13996
Верхняя граница диапазона значений входных данных из текста задачи	0.026576685999316396	701464

Вывод по задаче: Количество компонент связности можно найти разными способами. Один из них — обход в глубину

Задача №8. Стоимость полета [1.5 балла]

Теперь вас интересует минимизация не количества пересадок, а общей стоимости полета. Для этого строится взвешенный граф: вес ребра из одного города в другой – это стоимость соответствующего перелета. 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.** $1 \leq n \leq 104$, $0 \leq m \leq 105$, $1 \leq u, v \leq n$, $u \neq v$, вес каждого ребра – неотрицательное целое число, не превосходящее 108.
- **Формат вывода / выходного файла (output.txt).** Выведите минимальный вес пути из u в v . Введите -1, если пути нет.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 512 мб.
- Примеры:

input t	output t	input t	output t	input t	output t
4 4	3	5	6	3	-1
1 2		9		3	
1		1 2		1 2	
4 1		4		7	
2		1 3		1 3	
2 3		2		5	
2		2 3		2 3	

		2			
		3 2			
		1			
		2 4			
		2			
1 3		3 5			
5		4		2	
1 3		5 4		3 2	
		1			
		2 5			
		3			
		3 4			
		4			
		1 5			

• Объяснения:

Пример 1 – В этом графе существует единственный кратчайший путь из вершины 1 в вершину 3 ($1 \rightarrow 2 \rightarrow 3$), и он имеет вес 3. Пример 2 – Есть два пути от 1 до 5 общего веса 6: $1 \rightarrow 3 \rightarrow 5$ и $1 \rightarrow 3 \rightarrow 2 \rightarrow 5$. Пример 3 – Нет пути от вершины 3 до 2.

```
import queue
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

def dijkstra(zu):
    while not q.empty():
        w, v = q.get()
        visited[v] = True
        if v == zu:
            return w
        for i in range(len(adj_matrix[v])):
            if i != v and adj_matrix[v][i] != -1:
                el = adj_matrix[v][i]
                if not visited[i]:
                    q.put((w+el, i))
```

```

        return dijkstra(zu)
    return -1

n, m = map(int, lines[0].split())
visited = [False] * n
adj_matrix = [[-1 for j in range(n)] for i in range(n)]
q = queue.PriorityQueue(maxsize=0)

for i in range(m):
    u, v, w = map(int, lines[i+1].split())
    u -= 1
    v -= 1
    adj_matrix[u][v] = w

von, zu = map(int, lines[m+1].split())

q.put((0, von-1))
visited[von-1] = True
out.write(str(dijkstra(zu-1)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Для решения данной задачи был использован алгоритм Дейкстры. Обход останавливается тогда, когда мы приходим к искомой вершине

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

```

4 4
1 2 1
4 1 2
2 3 2
1 3 5
1 3

RMAL input.txt
projects/labs/se
3

```

```

5 9
1 2 4
1 3 2
2 3 2
3 2 1
2 4 2
3 5 4
5 4 1
2 5 3
3 4 4
1 5

RMAL input.tx
out.txt" 11L,
6

```


Результат работы кода на максимальных и минимальных значениях:
(скрины input output файлов)

```
1 0
1 1
NORMAL input.txt
out.txt" 2L,
```

```
13 9987 9988 83050717
12 9988 9989 33453181
11 9989 9990 67055904
10 9990 9991 20232216
9 9991 9992 16625289
8 9992 9993 77385889
7 9993 9994 78842158
6 9994 9995 42412638
5 9995 9996 68523308
4 9996 9997 57490037
3 9997 9998 91259172
2 9998 9999 16781226
1 9999 10000 98588098
001 9470 5657
NORMAL input.txt
input.txt" 10001L, 186647B
1
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00023144400074670557	19140
Пример из задачи	0.0003126209994661622	20422
Пример из задачи	0.00039897599890537094	23307
Верхняя граница диапазона значений входных данных из текста задачи	5.09698841900172	853275887

Вывод по задаче:

Для вычисления расстояния в задаче можно использовать алгоритм Дейкстры. Причем не важно ориентирован он или нет

Задача №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 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 изначально, $m + 3$ -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	input.txt	output.txt
5 Aqua -> AquaVita AquaVita -> PhilosopherStone AquaVita -> Argentum Argentum -> Aurum AquaVita -> Aurum Aqua Aurum	2	5 Aqua -> AquaVita AquaVita -> PhilosopherStone AquaVita -> Argentum Argentum -> Aurum AquaVita -> Aurum Aqua Osmium	-1

- Проверяем **обязательно** – [на астр](#).

```
import queue
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()

file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

def dijkstra(zu):
    while not q.empty():
        w, v = q.get()
        visited[v] = True
        if v == zu:
            return w
        for i in range(len(adj_matrix[v])):
            if i != v and adj_matrix[v][i] != -1:
                el = adj_matrix[v][i]
                if not visited[i]:
                    q.put((w+el, i))
    return dijkstra(zu)
return -1
```

```

n, m = map(int, lines[0].split())
visited = [False] * n
adj_matrix = [[-1 for j in range(n)] for i in range(n)]
q = queue.PriorityQueue(maxsize=0)

for i in range(m):
    u, v, w = map(int, lines[i+1].split())
    u -= 1
    v -= 1
    adj_matrix[u][v] = w

von, zu = map(int, lines[m+1].split())

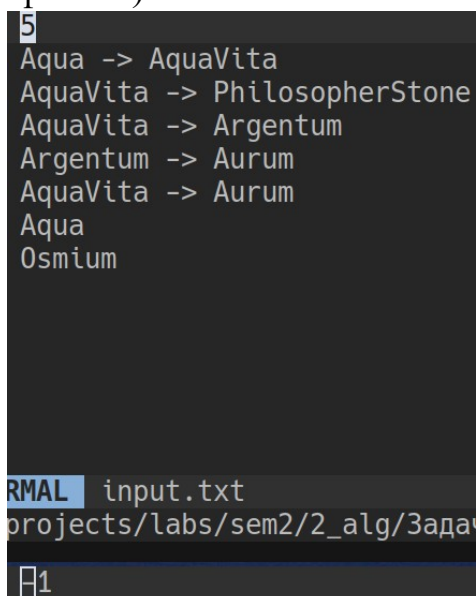
q.put((0, von-1))
visited[von-1] = True
out.write(str(dijkstra(zu-1)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Сперва определяем индексы для веществ. После этого находим расстояние используя алгоритм Дейкстры

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

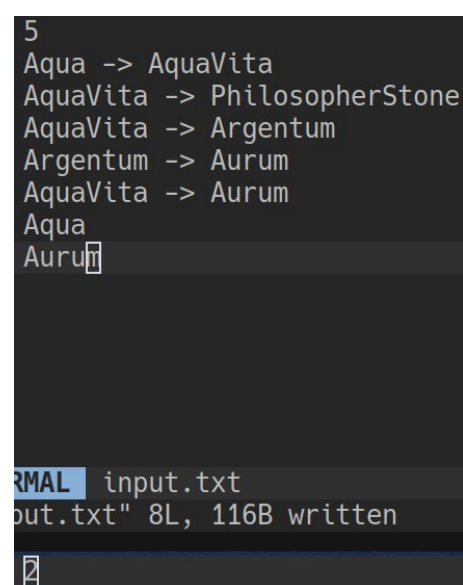


```

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

```

RMAL input.txt
projects/labs/sem2/2_alg/Зада



```

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

```

RMAL input.txt
out.txt" 8L, 116B written

Результат работы кода на максимальных и минимальных значениях:
(скрины input output файлов)

```
0
Aqua
Aqua

RMAL input.txt
out.txt" 3L, 12B
```

Проверка задачи на (openedu, астр и тд при наличии в задаче). (скрин)

19298972	25.04.2023 15:31:01	Филиппов Артём Эдуардович	0743	Python	Accepted	0,062	870 Кб
----------	---------------------	---------------------------	------	--------	----------	-------	--------

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00021823399947606958	16145
Пример из задачи	0.00023134199909691233	18136
Пример из задачи	0.00030197300111467484	21173
Верхняя граница диапазона значений входных данных из текста задачи	~0.062	~870000

Вывод по задаче: Алгоритм дейкстры лучше всего реализовывать с целью найти путь до всех вершин, так проще. Для оптимизации можно использовать priority queue.

Вывод

Раздел «Графы» предмета «Алгоритмы и структуры данных» показал как с помощью абстракций можно решать прикладные задачи на расстояние, связность. Например как различными способами можно представить в коде граф тем самым упростить себе задачу в дальнейшей поддержке кода или оптимизации. Кроме того в разделе были затронуты безусловно важные алгоритмы такие как обход в глубину и алгоритм Дейкстры.