

**Министерство науки и высшего образования Российской Федерации  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО  
ITMO University**

**Отчет по лабораторной работе № 2-3**

**По дисциплине** Алгоритмы и структуры данных

**Обучающийся** Овсянкин Даниил Витальевич

**Преподаватель:** Ромакина О.М

**Факультет** Инфокоммуникационных технологий

**Группа** K3244

**Направление подготовки** 45.03.04 Интеллектуальные системы в гуманитарной сфере

**Образовательная программа** Интеллектуальные системы в гуманитарной сфере

Санкт-Петербург

2025 г.

## 12 Задача. Цветной лабиринт [1 s, 16 Mb, 2 балла]

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  $k$  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, сообщаящей 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.

```
def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        data = list(map(int, fin.read().split()))
    if not data:
        with open('output.txt', 'w', encoding='utf-8') as fout:
            fout.write('INCORRECT')
        return

    it = iter(data)
    n = next(it)
    m = next(it)

    adj = [dict() for _ in range(n + 1)]
    for _ in range(m):
        u = next(it); v = next(it); c = next(it)
        adj[u][c] = v
        adj[v][c] = u

    k = next(it)
    colors = [next(it) for _ in range(k)] if k > 0 else []

    cur = 1
    for col in colors:
        nxt = adj[cur].get(col)
        if nxt is None:
            with open('output.txt', 'w', encoding='utf-8') as fout:
                fout.write('INCORRECT')
            return
        cur = nxt

    with open('output.txt', 'w', encoding='utf-8') as fout:
        fout.write(str(cur))
```

### Текстовое объяснение решения:

В задаче было реализовано последовательное прохождение цветного лабиринта. Граф читается как неориентированный, для каждой комнаты строится словарь от цвета к соседней комнате. Затем, начиная из комнаты 1, для каждого цвета из заданной последовательности выбирается единственный коридор этого цвета; если такого нет - путь некорректен и

выводится INCORRECT. Если все переходы возможны, печатается номер конечной комнаты.

**Результат на примере:**

Пример №1

```
n.py  input.txt x
3 2
1 2 10
1 3 5
5
10 10 10 10 5
n.py  input.txt  output.txt x
3
```

Пример №2

```
n.py  input.txt x
3 2
1 2 10
2 3 5
5
5 10 10 10 10
n.py  input.txt  output.txt x
INCORRECT
```

Вывод: задача решалась прямой симуляцией на словарях «цвет → сосед» для каждой комнаты; это даёт простой и быстрый  $O(m+k)$

## 14 Задача. Автобусы [1 s, 16 Mb, 3 балла]

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  $d$ ).

```

def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        tokens = fin.read().split()
        it = iter(tokens)
        n = int(next(it))
        d = int(next(it))
        v = int(next(it))
        r = int(next(it))

        trips = []
        for _ in range(r):
            u = int(next(it))
            t_dep = int(next(it))
            w = int(next(it))
            t_arr = int(next(it))
            trips.append((t_dep, u, w, t_arr))

        trips.sort(key=lambda x: x[0])

        INF = 10**15
        earliest = [INF] * (n + 1)
        earliest[d] = 0

        for t_dep, u, w, t_arr in trips:
            if earliest[u] <= t_dep and t_arr < earliest[w]:
                earliest[w] = t_arr

        ans = earliest[v] if earliest[v] < INF else -1

        with open('output.txt', 'w', encoding='utf-8') as fout:
            fout.write(str(ans))

```

### Текстовое объяснение решения:

В задаче было реализовано вычисление кратчайшего по времени прибытия маршрута в расписании: все рейсы сортируются по времени отправления, затем для каждого рейса проверяется достижимость его исходной деревни к этому моменту и, при успехе, обновляется минимально возможное время прибытия в деревню назначения. Сложность  $O(R \log R)$  на сортировку и  $O(R)$  на проход, память  $O(N)$ .

### Результат на примере:

```
h.py  input.txt x
3
1 3
4
1 0 2 5
1 1 2 3
2 3 3 5
1 1 3 10|

h.py  input.txt  output.txt x
5|
```

Вывод: задача решалась по отсортированному расписанию; один линейный проход после сортировки даёт минимальное время прибытия или  $-1$ , если добраться нельзя.

## 15 Задача. Герои [1 s, 16 Мб, 3 балла]

Коварный кардинал Ришелье вновь организовал похищение подвесок королевы Анны; вновь спасти 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.

```

from collections import deque

def bfs(grid, qx, qy):
    n, m = len(grid), len(grid[0])
    INF = 10**9
    dist = [[INF] * m for _ in range(n)]
    if grid[qx][qy] == '1':
        return dist
    dq = deque([(qx, qy)])
    dist[qx][qy] = 0
    while dq:
        x, y = dq.popleft()
        d = dist[x][y] + 1
        for nx, ny in ((x-1,y), (x+1,y), (x,y-1), (x,y+1)):
            if 0 <= nx < n and 0 <= ny < m and grid[nx][ny] == '0' and dist[nx][ny] > d:
                dist[nx][ny] = d
                dq.append((nx, ny))
    return dist

```

```

def solve():
    with open('input.txt', 'r', encoding='utf-8') as fin:
        N, M = map(int, fin.readline().split())
        grid = [fin.readline().strip() for _ in range(N)]

        Qx, Qy, L = map(int, fin.readline().split()) # 1-based
        musk = []
        for _ in range(4):
            line = fin.readline()
            while line is not None and line.strip() == '':
                line = fin.readline()
            if not line:
                break
            ax, ay, p = map(int, line.split())
            musk.append((ax, ay, p))

        dist = bfs(grid, Qx - 1, Qy - 1)

```

```

total = 0
n, m = N, M
for ax, ay, p in musk:
    x, y = ax - 1, ay - 1
    if 0 <= x < n and 0 <= y < m and grid[x][y] == '0' and dist[x][y] <= L:
        total += p

with open('output.txt', 'w', encoding='utf-8') as fout:
    fout.write(str(total))

```

## Текстовое объяснение решения:

В задаче было реализовано BFS по прямоугольной решётке с препятствиями: из клетки королевы вычисляется минимальное число шагов до всех доступных клеток. Затем для каждого мушкетёра проверяется достижимость и укладывание в лимит времени  $L$ ; если условие выполняется, его количество подвесок прибавляется к сумме. Сложность  $O(NM)$  по времени и  $O(NM)$  по памяти

## Результат на примере:

```
.py  input.txt x
5 5
11111
10001
10001
10001
11111
4 4 10
2 2 1
2 3 2
3 2 3
3 3 4

1.py  input.txt  output.txt x
10|
```

Вывод: задача решалась поиском в ширину от королевы и проверкой, кто из четырёх мушкетёров успевает прийти за  $\leq L$  минут: итог - сумма их подвесок.

## 16 Задача. Рекурсия [1 s, 16 Mb, 3 балла]

Одним из важных понятий, используемых в теории алгоритмов, является рекурсия. Неформально ее можно определить как использование в описании объекта самого себя. Если речь идет о процедуре, то в процессе исполнения эта процедура напрямую или косвенно (через другие процедуры) вызывает сама себя.

Рекурсия является очень «мощным» методом построения алгоритмов, но таит в себе некоторые опасности. Например, неаккуратно написанная рекурсивная процедура может войти в бесконечную рекурсию, то есть, никогда не закончить свое выполнение (на самом деле, выполнение закончится с переполнением стека).

Поскольку рекурсия может быть косвенной (процедура вызывает сама себя через другие процедуры), то задача определения того факта, является ли данная процедура рекурсивной, достаточно сложна. Попробуем решить более простую задачу.

Рассмотрим программу, состоящую из  $n$  процедур  $P_1, P_2, \dots, P_n$ . Пусть для каждой процедуры известны процедуры, которые она может вызывать. Процедура  $P$  называется потенциально рекурсивной, если существует такая последовательность процедур  $Q_0, Q_1, \dots, Q_k$ , что  $Q_0 = Q_k = P$  и для  $i = 1 \dots k$  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.

```

import sys
sys.setrecursionlimit(1 << 20)

def solve(): 1 usage
    lines = [ln.rstrip('\n') for ln in open('input.txt', 'r', encoding='utf-8')]
    it = iter(lines)
    n = int(next(it))

    names, calls = [], []
    for _ in range(n):
        name = next(it).strip()
        names.append(name)
        k = int(next(it))
        calls.append([next(it).strip() for __ in range(k)])
        _ = next(it)

    idx = {name: i for i, name in enumerate(names)}
    g = [[] for _ in range(n)]
    gt = [[] for _ in range(n)]
    selfloop = [False]*n

```

```

    for u in range(n):
        for nm in calls[u]:
            v = idx[nm]
            g[u].append(v)
            gt[v].append(u)
            if v == u:
                selfloop[u] = True

```

```

    used, order = [False]*n, []
    def dfs(v):
        used[v] = True
        for w in g[v]:
            if not used[w]: dfs(w)
        order.append(v)

```

```

    for v in range(n):
        if not used[v]: dfs(v)

```

```

    comp = [-1]*n; cid = 0

```



```
def rdfs(v, cid):
    comp[v] = cid
    for w in gt[v]:
        if comp[w] == -1: rdfs(w, cid)

    for v in reversed(order):
        if comp[v] == -1:
            rdfs(v, cid); cid += 1

sz = [0]*cid
for v in range(n): sz[comp[v]] += 1

ans = ["YES" if (sz[comp[v]] > 1 or selfloop[v]) else "NO" for v in range(n)]
open('output.txt', 'w', encoding='utf-8').write("\n".join(ans))
```

### Текстовое объяснение решения:

читаем  $n$ , затем  $n$  блоков вида  $id, k, k$  имён, \*\*\*\*\*. По ним строим отображение имя - индекс и ориентированный граф. Запускаем Косарайю/Тарьяна, считаем размер каждой КСС; отмечаем самопетли. Для каждой процедуры в порядке входа печатаем YES, если  $size(SCC) > 1$  или есть ребро  $u \rightarrow u$ , иначе NO.

### Результат на примере:

```
input.txt
3
p1
2
p1
p2
*****
p2
1
p1
*****
p3
1
p1
*****

output.txt
YES
YES
NO
```

**Вывод:** задача сведена к поиску циклов через КСС: решение линейное  $O(n+m)$

## 17 Задача. Слабая K-связность [1 s, 16 Мб, 4 балла]

Ане, как будущей чемпионке мира по программированию, поручили очень ответственное задание. Правительство вручает ей план постройки дорог между  $N$  городами. По плану все дороги односторонние, но между двумя городами может быть больше одной дороги, возможно, в разных направлениях. Ане необходимо вычислить минимальное такое  $K$ , что данный ей план является слабо  $K$ -связным.

Правительство называет план слабо  $K$ -связным, если выполнено следующее условие: для любых двух различных городов можно проехать от одного до другого, нарушая правила движения не более  $K$  раз. Нарушение правил - это проезд по существующей дороге в обратном направлении. Гарантируется, что между любыми двумя городами можно проехать, возможно, несколько раз нарушив правила.

```
from collections import deque
def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        data = list(map(int, fin.read().split()))
        it = iter(data)

        n = next(it)
        m = next(it)

        outs = [[] for _ in range(n + 1)]
        ins = [[] for _ in range(n + 1)]
        for _ in range(m):
            u = next(it); v = next(it)
            outs[u].append(v)
            ins[v].append(u)

        INF = 10**9
        answer = 0
```

```

for s in range(1, n + 1):
    dist = [INF] * (n + 1)
    dist[s] = 0
    dq = deque([s])

    while dq:
        u = dq.popleft()
        du = dist[u]

        for v in outs[u]:
            if dist[v] > du:
                dist[v] = du
                dq.appendleft(v)
        for v in ins[u]:
            if dist[v] > du + 1:
                dist[v] = du + 1
                dq.append(v)

    local_max = 0
    for i in range(1, n + 1):
        if dist[i] > local_max:
            local_max = dist[i]

```

```

if local_max > answer:
    answer = local_max

with open('output.txt', 'w', encoding='utf-8') as fout:
    fout.write(str(answer))

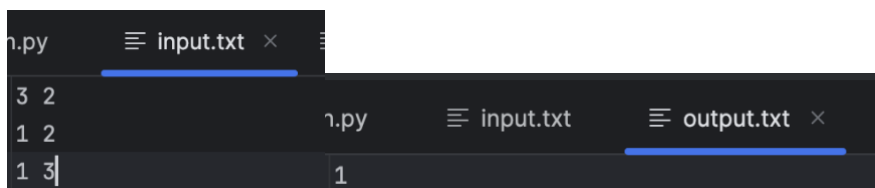
```

### Текстовое объяснение решения:

программа ищет минимальное  $K$  для слабой  $K$ -связности. Из входа читаются  $N, M$  дороги; каждую  $u \rightarrow v$  заменяем двумя рёбрами:  $u \rightarrow v$  с весом 0 (по правилу) и  $v \rightarrow u$  с весом 1 (нарушение). Для каждого города выполняется 0–1 BFS на деке: переходы с 0 кладём в голову, с 1 - в хвост, поэтому расстояние равно минимальному числу нарушений. Берём максимум из полученных расстояний - это ответ. Сложность  $O(N \cdot (N+M))$  память -  $O(N+M)$ .

### Результат на примере:

#### Пример №1

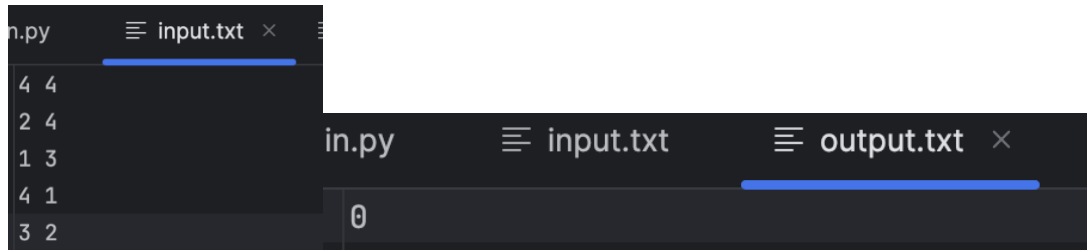


```

n.py  input.txt  output.txt
3 2
1 2
1 3
1

```

## Пример №2



The screenshot shows a code editor with two tabs: 'input.txt' and 'output.txt'. The 'input.txt' tab is active and displays a 5x5 grid of numbers: 4 4, 2 4, 1 3, 4 1, 3 2. The 'output.txt' tab is also active and displays the number 0.

**Вывод:** реализован 0–1 BFS и моделирование обратного проезда весами 0/1; получаем минимальное K

### Вывод по лабораторной работе

В процессе выполнения лабораторной работы были рассмотрены и реализованы основные алгоритмы обработки графов. Были изучены методы обхода в ширину и глубину, а также алгоритмы поиска кратчайшего пути, включая алгоритм Дейкстры, Флойда–Уоршелла и другие. Практическая реализация данных алгоритмов позволила закрепить теоретические знания и освоить их применение на практике.