КСС. Мосты. Конденсации.

1 A. Правильная скорая помощь

Разбор

Существующая по легенде задачи система дорог позволяет проехать к пациенту по встречной полосе движения, но обратно требует движения строго по правилам. То есть, из каждой вершины графа мы должны уметь достигать специально покрашенной вершины, пусть это будет цвет a. Если из вершины достижима вершина такого цвета, то будем красить её в цвет b. Нетрудно заметить, что в цвет a мы можем покрасить только условные "стоки"нашего графа - вершины, из которых не выходит ни одно ребро.

Пусть мы только что покрасили некоторую вершину в цвет a. Тогда все вершины на пути к ней мы можем покрасить в цвет b, т.к. из них достижима станция. Единственная проблема, которая у нас возникает – мы не можем гарантировать порядок обхода графа, поэтому если у нас есть циклы, то мы будем закрашивать лишние точки, т.к. будем думать, что попали в сток – ведь из них мы не можем никуда пойти. Тогда сначала сконденсируем граф, а затем запустим ровно такую же раскраску на нём.

Асимптотика решения: $O(|V| + |E|)$

Решение

```
from collections import defaultdict
from sys import stdin, setrecursionlimit

setrecursionlimit(1000069)
input = stdin.readline

def dfs_sort(u, g, used, res: list):
    used[u] = 1
    for v in g[u]:
        if not used[v]:
            dfs_sort(v, g, used, res)
    res.append(u)

def dfs_condense(u, g, components, c):
    components[u] = c
    for v in g[u]:
        if not components[v]:
            dfs_condense(v, g, components, c)

def dfs_coloring(u, g, colored):
    colored[u] = 1
    visited_any = 0
    for v in g[u]:
        if not colored[v]:
            visited_any = 1
            dfs_coloring(v, g, colored)
        if colored[v] > 1:
            colored[u] = 2
            break
    if not visited_any and colored[u] != 2:
        colored[u] = 3

n, m = int(input()), int(input())
g = defaultdict(list)
for _ in range(m):
    u, v = map(int, input().split())
    g[u - 1].append(v - 1)

top_sort, used = [], [0] * n
for i in range(n):
    if not used[i]:
```

```
        dfs_sort(i, g, used, top_sort)

order = reversed(top_sort)
g_inv = defaultdict(list)
for u in g:
    for v in g[u]:
        g_inv[v].append(u)

components = [0] * n
c = 1
for i in order:
    if not components[i]:
        dfs_condense(i, g_inv, components, c)
        c += 1

g_cond = defaultdict(list)
for u in g:
    for v in g[u]:
        if components[u] == components[v]:
            continue
        g_cond[components[u] - 1].append(components[v] - 1)

n_cond = c - 1
colored = [0] * n_cond
for i in range(n_cond):
    if not colored[i]:
        dfs_coloring(i, g_cond, colored)
ans = len(list(filter(lambda x: x == 3, colored)))
print(ans)
```

2 B. Точки сочленения

Разбор

В данной задаче было необходимо корректно реализовать поиск точек сочленения.
Асимптотика решения: $O(|V| + |E|)$

Решение

```
from collections import defaultdict
from sys import stdin, setrecursionlimit

setrecursionlimit(1000069)
input = stdin.readline

n, m = map(int, input().split())
g = defaultdict(list)
for _ in range(m):
    u, v = map(int, input().split())
    g[u - 1].append(v - 1)
    g[v - 1].append(u - 1)

INF = 10 ** 9 + 7
t_in, dp = [-1] * n, [INF] * n
points = set()

def dfs(u, t=0, p=-1):
    t_in[u] = dp[u] = t
    children = 0
    for v in g[u]:
        if v == p:
            continue
        if t_in[v] < 0:
            t = dfs(v, t + 1, u)
            dp[u] = min(dp[u], dp[v])
            if dp[v] >= t_in[u] and p > -1:
```

```
                points.add(u + 1)
            children += 1
        else:
            dp[u] = min(dp[u], t_in[v])
    if p < 0 and children > 1:
        points.add(u + 1)
    return t

for i in range(n):
    if t_in[i] < 0:
        dfs(i)
print(len(points))
print(*list(sorted(points)))
```

3 C. Магнитные подушки

Разбор

Основная проблема этой задачи – подушки представлены тройками, поэтому чтобы перебрать мосты, нам нужно учитывать, что мы одновременно должны удалить несколько рёбер. Давайте переформулируем задачу так, чтобы вместо мостов работать с точками сочленения. Представим тройку небоскрёбов, соединённых рёбрами, в виде четвёрки: по вершине на каждый небоскрёб и дополнительно вершина, задающая центр масс между небоскрёбами. Рёбра подведём к этой вершине. Если мы удалим такую вершину, то соединённые ранее подушкой небоскрёбы также распадутся. Заменим каждую заданную тройку подобным образом, а затем просто найдём точки сочленения в получившемся графе.

Асимптотика решения: $O(|V| + |E|)$

Решение

```
from collections import defaultdict
from sys import stdin, setrecursionlimit

setrecursionlimit(1000069)
input = stdin.readline
INF = 10 ** 9 + 7
int_v = lambda v: int(v) - 1

def dfs(u, t=0, p=-1):
    t_in[u] = dp[u] = t
    children = 0
    for v in g[u]:
        if v == p:
            continue
        if t_in[v] < 0:
            t = dfs(v, t + 1, u)
            dp[u] = min(dp[u], dp[v])
            if dp[v] >= t_in[u] and p > -1 and u >= n:
                cutpts.add(u + 1 - n)
            children += 1
        else:
            dp[u] = min(dp[u], t_in[v])
    if p < 0 and children > 1:
        cutpts.add(u + 1)
    return t

n, m = map(int, input().split())
g = defaultdict(list)
for k in range(m):
    x, y, z = map(int_v, input().split())
    cutpt = n + k
    g[x].append(cutpt)
    g[y].append(cutpt)
    g[z].append(cutpt)
    g[cutpt].append(x)
```

```
    g[cutpt].append(y)
    g[cutpt].append(z)

q = n + m
t_in, dp = [-1] * q, [INF] * q
cutpts = set()
for i in range(q):
    if t_in[i] < 1:
        dfs(i)
print(len(cutpts))
print(*list(cutpts))
```

4 D. Минимизация мостов

Разбор

Для начала пройдёмся по каждой компоненте связности и найдём в ней мосты. Затем, используя знания о посчитанной динамике, перестроим граф – будем считать КСС такой набор вершин, у которых величина dp_k одинаковая. Теперь у нас получилось дерево. Как нам посчитать, сколько максимум мостов мы можем удалить? Ответом будет являться сумма длин двух самых больших ветвей полученного дерева. Пройдёмся по всем компонентам связности, посчитаем для каждой, сколько максимум мостов из неё можно удалить, и вычтем этот максимум из общего числа мостов в графе.

Асимптотика решения: $O(|V| + |E|)$

Решение

```
from collections import defaultdict
from sys import stdin, setrecursionlimit

setrecursionlimit(1000069)
input = stdin.readline
INF = 10 ** 9 + 7
int_v = lambda v: int(v) - 1

n, m = map(int, input().split())
g = {}
for _ in range(m):
    u, v = map(int_v, input().split())
    if u not in g:
        g[u] = defaultdict(int)
    g[u][v] += 1
    if v not in g:
        g[v] = defaultdict(int)
    g[v][u] += 1

t_in = [-1] * n
used, bridges = [0] * n, [0] * n
dp = [INF] * n

def dfs_bridges(u, t=0, p=-1):
    t_in[u] = dp[u] = t
    if u not in g:
        return
    for v in g[u]:
        if v == p or not g[u][v]:
            continue
        if t_in[v] < 0:
            t = dfs_bridges(v, t + 1, u)
            dp[u] = min(dp[u], dp[v])
            if dp[v] > t_in[u] and g[u][v] == 1:
                bridges[v] = 1
        else:
            dp[u] = min(dp[u], t_in[v])
    return t
```

```
def dfs_rebuild(u, prev_t, p=-1):
    global t
    used[u] = 1
    dp[u] = prev_t
    if u not in g:
        return
    for v in g[u]:
        if used[v] or v == p:
            continue
        if bridges[v]:
            t += 1
            dfs_rebuild(v, t, u)
        else:
            dfs_rebuild(v, prev_t, u)
        if dp[u] != dp[v]:
            g_time[dp[u]].append(dp[v])
            g_time[dp[v]].append(dp[u])

def dfs_solve(u_time, p_time=-1):
    best_b, best_t = 0, u_time
    for v_time in g_time[u_time]:
        if v_time == p_time:
            continue
        best_b_tmp, best_t_tmp = dfs_solve(v_time, u_time)
        if best_b_tmp + 1 > best_b:
            best_b, best_t = best_b_tmp + 1, best_t_tmp
    return best_b, best_t

for i in range(n):
    if t_in[i] < 0:
        dfs_bridges(i)

total_bridges = sum(bridges)
ans = 0
for i in range(n):
    if used[i]:
        continue
    t = 0
    g_time = defaultdict(list)
    dfs_rebuild(i, t)
    _, best_t = dfs_solve(dp[i])
    best_b, best_t = dfs_solve(best_t)
    ans = max(ans, best_b)

print(total_bridges - ans)
```

5 E. Обновление дата-центров

Разбор

Изначально нам не дан граф датацентров. Построим его сами: проверим, будет ли одинаковое время для двух связанных датацентров после сдвига времени на 1 час у одного из них. Если да, то считаем, что перевод часов в зоне одного датацентра повлияет на общую доступность данных для i-го клиента – значит, надо провести ребро от одной вершины к другой. Что нам хочется узнать? Мы должны выбрать такой набор вершин, чтобы

- все вершины образовывали **замкнутый** цикл – иначе говоря, вне зависимости от того, в каком датацентре сдвигается время на час, все остальные датацентры продолжают быть доступны
- в наборе была только одна вершина – то есть, сдвиг времени на час никак в одном датацентре никак не пересекается с другими

В обоих случаях важно, чтобы степень получившейся КСС была равна нулю. Тогда просто сожмём граф и выберем такую КСС степень 0, чтобы в ней лежало минимально возможное число вершин.

Асимптотика решения: $O(|V| + |E|)$

Решение

```
from collections import defaultdict
from sys import stdin, setrecursionlimit

setrecursionlimit(1000069)
input = stdin.readline
INF = 10 ** 9 + 7
int_v = lambda v: int(v) - 1

def dfs_sort(u, g, used, res: list):
    used[u] = 1
    for v in g[u]:
        if not used[v]:
            dfs_sort(v, g, used, res)
    res.append(u)

def dfs_condense(u, g, components, c):
    components[u] = c
    for v in g[u]:
        if not components[v]:
            dfs_condense(v, g, components, c)

def solve():
    n, m, h = map(int, input().split())
    upd_t = list(map(int, input().split()))
    g, g_inv = defaultdict(list), defaultdict(list)

    for _ in range(m):
        u, v = map(int_v, input().split())
        if (upd_t[u] + 1) % h == upd_t[v]:
            g[u].append(v)
            g_inv[v].append(u)
        if (upd_t[v] + 1) % h == upd_t[u]:
            g[v].append(u)
            g_inv[u].append(v)

    top_sort, used = [], [0] * n
    for i in range(n):
        if not used[i]:
            dfs_sort(i, g, used, top_sort)

    order = reversed(top_sort)
    components = [0] * n
    c = 0
    for i in order:
        if not components[i]:
            c += 1
            dfs_condense(i, g_inv, components, c)

    cmp_to_node = defaultdict(list)
    for i in range(n):
        cmp_to_node[components[i] - 1].append(i + 1)
    deg = [0] * c
    for u in g_inv:
        for v in g_inv[u]:
            if components[u] != components[v]:
                deg[components[v] - 1] += 1

    min_c, min_size = -1, INF
    for comp in range(c):
        d, cmp_size = deg[comp], len(cmp_to_node[comp])
        if d == 0 and cmp_size < min_size:
            min_c, min_size = comp, cmp_size
    return cmp_to_node[min_c]

cluster = solve()
print(len(cluster))
print(*cluster)
```