Задача A. Правильная скорая помощь

Построим конденсацию алгоритмом с лекции. Достаточно расположить по одной станции скорой помощи в каждой компоненте сильной связности, так как из любой вершины достижимы все вершины её компоненты. Можно ли обойтись меньшим количеством станций? Иногда можно, например, если из компоненты c_i достижима компонента c_j, в компоненте c_i можно не размещать станцию. Более общее рассуждение можно провести так: пусть из какой-то компоненты не выходит ни одного ребра, тогда очевидно, в ней необходимо расположить станцию. Так как граф конденсации ацикличен, из любой другой компоненты получится добраться до какой-то компоненты со станцией.

```
for node in range(1, n+1):
    if not visited[node]:
        dfs1(node)
order = order[::-1]
color = [0] * (n+1)
cnt = 1
for v in order:
    if color[v] == 0:
        dfs2(v, cnt)
        cnt += 1
out_degree = [0] * (cnt)
for a, b in g_res:
    out_degree[a] += 1
answer = 0
for i in range(1, cnt):
    if out_degree[i] == 0:
        answer += 1
print(answer)
```

Задача B. Конденсация графа

Реализуем алгоритм с лекции. Выполним «топсорт» графа, затем запустим DFS от всех вершин в порядке этого «топсорта» на графе с инвертированной ориентацией рёбер. Если мы должны запустить DFS из ещё не помеченной вершины, пометим её и увеличим число компонент сильной связности. Асимптотика решения $O(n + m)$.

```
def dfs_2(v, color, g, c):
    color[v] = c
    for u in g[v]:
        if not color[u]:
            dfs_2(u, color, g, c)


def dfs(v, color, g, ans):
    color[v] = 1
    for u in g[v]:
        if not color[u]:
            dfs(u, color, g, ans)
    ans.append(v)


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

ans.reverse()
```

```
used = [0] * n
cnt = 0
for v in ans:
    if not used[v]:
        cnt += 1
        dfs_2(v, used, gr, cnt)
```

Задача C. Производство деталей

Построим граф, в котором каждой детали соответствует вершина, а ребро $v \to u$ означает зависимость детали v от u. По условию этот граф ацикличен, значит достаточно найти количество вершин, достижимых из вершины 1. Их можно изготавливать в обратном порядке топсорта. Асимптотика решения $O(n+m)$.

```
def dfs (val):
    global x,y
    used [val] = True
    y += 1
    x += q[val]
    for i in f[val]:
        if not used [i]:
            dfs(i)
    result.append (str(val))

x = 0
y = 0
result = []
used = [False] * (z+1)
dfs(1)
print(x,y)
print(' '.join(result))
```

Задача D. 2-SAT

Заметим, что $x \vee y = (\neg x \to y) = (\neg y \to x)$. Создадим для каждой переменной две вершины (под неё и её отрицание). Проведём рёбра, соответствующие выполненным импликациям. Найдём в этом графе компоненты сильной связности, вершина x_i будет лежать в компоненте c_{x_i}, вершина $\neg x_i$ будет лежать в компоненте $c_{\neg x_i}$. Если из $\neg x$ достижим x и из x достижим $\neg x$, решения не существует. В противном случае возможно присвоить переменным значения $x_i = (c_{x_i} > c_{\neg x_i})$, чтобы все нули не были достижимы из единиц. Асимптотика решения $O(n)$.

```
def slv(n, graph):
    comp = unite(2 * n, graph)
    res = []
    for i in range(n):
        if comp[2 * i] > comp[2 * i + 1]:
            res.append('1')
        else:
            res.append('0')

    return ''.join(res)


for i in range(n):
    graph[2 * i].append(2 * i)
    graph[2 * i + 1].append(2 * i + 1)
```

```
for i in range(m):
    i, e1, j, e2 = map(int, lines.pop().split())
    u, v = 2 * i + (1 - e1), 2 * j + e2
    graph[u].append(v)
    graph[v ^ 1].append(u ^ 1)

res = slv(n, graph)
print(res)
```