Задача A. Подсчет количества ребер неориентированного графа

Вспомним, что ребро между вершинами равносильно тому, что в матрице смежности на пересечении i-й и j-й клеток стоит единица. Значит надо посчитать кол-во единиц. Потом надо поделить ответ на 2, так как мы учли каждое ребро 2 раза.

Код на Python:

```
n = int(input())
ans = 0
for i in range(n):
    ans += sum(map(int, input().split()))
print(ans // 2)
```

Задача B. От списков смежности к матрице смежности

Если для вершины v есть значения $\{u_1, u_2, \ldots, u_k\}$, то надо поставить единички в клетки $g_{v,u_1}, g_{v,u_2}, \ldots, g_{v,u_k}$

Код на Python:

```
n = int(input())
res = [[0 for _ in range(n)] for _ in range(n)]
for i in range(n):
    neighbours = list(map(int, input().split()))[1:]
    for n in neighbours:
        res[i][n - 1] = 1
for i in res:
    print(*i)
```

Задача C. От списка ребер к матрице смежности, ориентированный вариант

Если ребро (u, v), то поставим единичку в клетку $g_{u,v}$.

Код на Python:

```
n, m = map(int, input().split())
g = [[0 for _ in range(n)] for _ in range(n)]
for _ in range(m):
    u, v = map(int, input().split())
    g[u - 1][v - 1] = 1
for i in g:
    print(*i)
```

Задача D. Проверка на неориентированность

Как говорилось на лекции, у неор графа всегда $g_{i,j} = g_{j,i}$. Также надо проверить, что нет ребер из вершины в себя. Проверим же это!

Код на Python:

```
n = int(input())
g = []
for _ in range(n):
    g.append(list(map(int, input().split())))
ok = True
for i in range(n):
    for j in range(n):
        ok &= g[i][j] == g[j][i] and g[i][i] == 0
if ok:
```

```
    print('Yes')
else:
    print('No')
```

Задача E. Проверка на наличие кратных ребер, ориентированный вариант

Когда встречаем ребро (u, v), будем добавлять 1 в соответствующую ячейку (а не присваивать). Потом надо проверить, что все значения не больше 1.

Код на Python:

```
n, m = map(int, input().split())
g = [[0 for _ in range(n)] for _ in range(n)]
for i in range(m):
    u, v = map(int, input().split())
    g[u - 1][v - 1] += 1
ok = True
for i in g:
    for j in i:
        ok &= j <= 1
print('No' if ok else 'Yes')
```

Задача F. Взлом сети

Для каждого алгоритма заведем сет – какие алгоритмы будут взломаны при взломе текущего. Посчитать это можно прямо на этапе ввода.

Код на Python:

```
MAX_ALGO = 200_000

n, m = map(int, input().split())
algo = list(map(int, input().split()))
others = [set() for _ in range(MAX_ALGO)]
for i in range(m):
    u, v = map(int, input().split())
    u -= 1
    v -= 1
    others[algo[u]].add(algo[v])
    others[algo[v]].add(algo[u])
best = -1
for i in range(MAX_ALGO):
    if best == -1 or len(others[best]) < len(others[i]):
        best = i
print(best)
```

Задача G. Связность

Напишем bfs.

Код на Python:

```
from collections import deque


def bfs(start, n, g, used):
    q = deque([start])
    used[start] = True
```

```
    while q:
        v = q.popleft()
        for to in g[v]:
            if not used[to]:
                used[to] = True
                q.append(to)


n, m = map(int, input().split())
g = [[] for _ in range(n)]
for _ in range(m):
    u, v = map(int, input().split())
    u -= 1
    v -= 1
    g[u].append(v)
    g[v].append(u)
used = [0] * n
bfs(0, n, g, used)
print('YES' if 0 not in used else 'NO')
```

Задача H. Обход графа

То же самое, но теперь надо запустить на матрице смежности. Также теперь надо посчитать сумму в массиве used.

Код на Python:

```
from collections import deque


def bfs(start, n, g, used):
    q = deque([start])
    used[start] = True
    while q:
        v = q.popleft()
        for to in range(n):
            if g[v][to] and not used[to]:
                used[to] = True
                q.append(to)


n, start = map(int, input().split())
g = []
for _ in range(n):
    g.append(list(map(int, input().split())))
used = [0] * n
bfs(start - 1, n, g, used)
print(sum(used))
```

Задача I. Компоненты связности

Эта задача тоже разбиралась на лекции.

Код на Python:

```
from collections import deque
```

```
def bfs(start, n, g, used):
    q = deque([start])
    used[start] = True
    while q:
        v = q.popleft()
        for to in g[v]:
            if not used[to]:
                used[to] = True
                q.append(to)


n, m = map(int, input().split())
g = [[] for _ in range(n)]
for _ in range(m):
    u, v = map(int, input().split())
    u -= 1
    v -= 1
    g[u].append(v)
    g[v].append(u)
used = [0] * n
ans = 0
for start in range(n):
    if not used[start]:
        ans += 1
        bfs(start, n, g, used)
print(ans)
```

Задача J. Путь в графе

На лекции упоминалось, что bfs умеет искать кратчайший путь. Понятно, что если u это сосед v, которого еще не добавляли в очередь, то $dist_u = dist_v + 1$ (очень похоже на динамическое программирование

Код на Python:

```
from collections import deque


def bfs(start, to, n, g, used):
    dist, pred = [-1] * n, [-1] * n
    q = deque([start])
    dist[start], used[start] = 0, True
    while q:
        v = q.popleft()
        for to in range(n):
            if g[v][to] and not used[to]:
                dist[to], pred[to] = dist[v] + 1, to
                used[to] = True
                q.append(to)
    return dist[to]


n = int(input())
g = []
```

```
for _ in range(n):
    g.append(list(map(int, input().split())))
start, end = map(int, input().split())
used = [0] * n
print(bfs(start - 1, end - 1, n, g, used))
```