ФИО: Губарева Екатерина Алексеевна

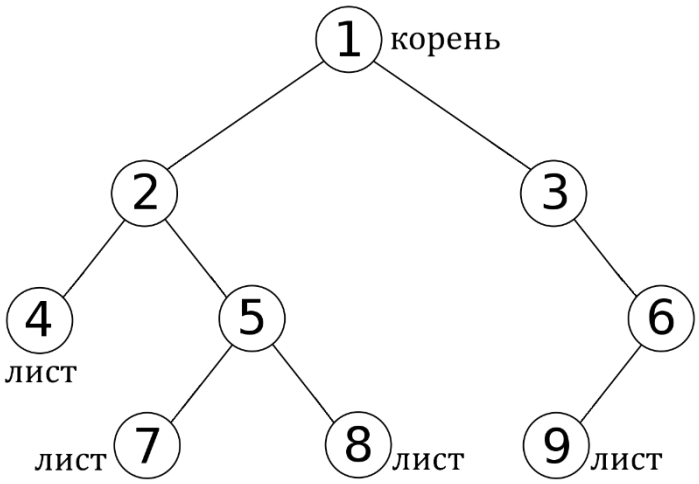
Ссылка на ВК: https://vk.com/id329976472

Номер задания из ЕГЭ – **27**. (средний уровень)

ЗАДАНИЕ И РЕШЕНИЕ НИЖЕ

Задание средней сложности

Пусть G – неориентированный невзвешенный связный ациклический граф без петель и кратных ребер. Вершины в нем пронумерованы, начиная с единицы. Структура G похожа на перевернутое дерево (см. рисунок). У дерева есть корень – самая верхняя вершина (вершина с номером 1). Также у дерева есть листья – вершины, из которых нет ребер в вершины ниже нее (есть только одно ребро в верхнюю вершину, либо вообще нет ребер). Определите номер такой вершины, из которой есть ребра в K листов (ребра в вершины-не-листы нам не важны, они могут как быть, так и не быть), где K – это целое неотрицательное число. Если таких вершин несколько, напишите минимальный из номеров.



Входные данные

Дано два входных файла (файл A_medium и файл B_medium), каждый из которых в первой строке содержит два целых числа N, K (1 < N < 100 000) (-1 < K < 100 000) — количество вершин в G и необходимое количество листьев у искомой вершины соответственно. Далее в N-1 строках перечислены рёбра графа. Каждое ребро задаётся парой чисел — номерами начальной и конечной вершин соответственно. В ответе укажите два числа: сначала значение искомой величины для файла A_medium, затем — для файла B_medium.

Типовой пример организации данных во входном файле

9 2
1 2
2 4
2 5
5 7
8 5
1 3
3 6
9 6

При таких входных данных листьями являются вершины с номерами 4, 7, 8, 9. Из вершины с номером 5 выходит два листа (вершины 7 и 8). Ответом на вопрос задачи является число 5.

Типовой пример имеет иллюстративный характер. Для выполнения задания используйте данные из прилагаемых файлов.

Предупреждение: файл B не следует обрабатывать вручную, поскольку количество вершин в файле велико, и это будет слишком долго.

Решение задания средней сложности

Для решения файла A можно будет нарисовать граф из файла в программе Paint, например. Зная, что корень в вершине 1, можно понять, какая вершина ближе к корню, а какая дальше. Для удобства напишем программу, которая выводит для каждой вершины те вершины, с которыми она соединена ребром (файл A_medium.py).

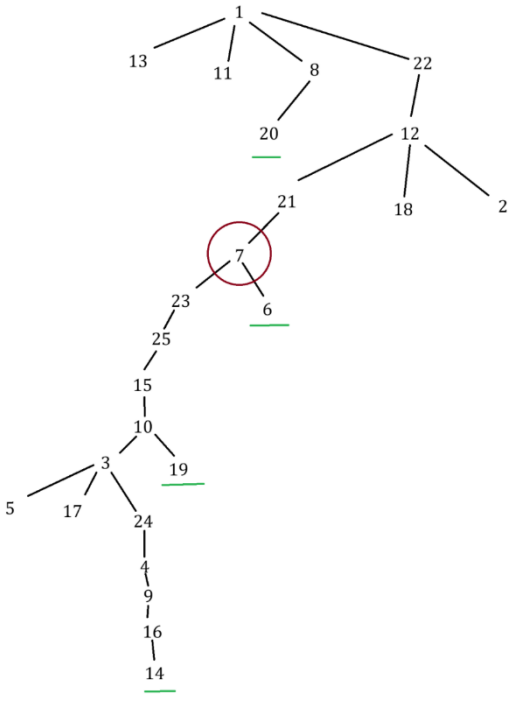
```
with open('A_medium.txt', 'r') as f:
    n, k = map(int, f.readline().split())
    edges = [[] for _ in range(n + 1)]
    for _ in range(n - 1):
        v, u = map(int, f.readline().split())
        edges[v].append(u)
        edges[u].append(v)

for i, neighbours in enumerate(edges):
    if i == 0:
        continue
    print(i, 'is connected to', *neighbours)
```

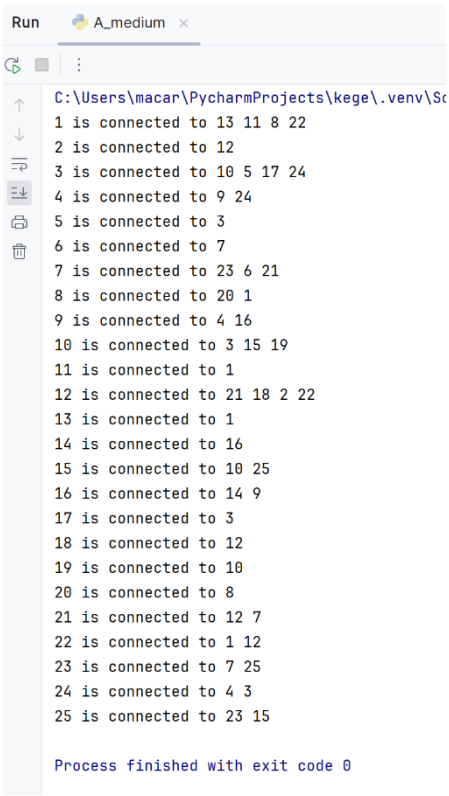
По полученным результатам легко нарисовать граф.

Рисунок ниже.

В данном случае кандидатами на ответ являются вершины 8, 7, 10, 16. С минимальным номером вершина 7.

Ответ: 7





Для решения файла B нужно написать обход графа в глубину и подсчет количества листьев для конкретной вершины (файл B_medium.py). Суть обхода, описание структуры данных для хранения графа, способы определения листьев были описаны в задании легкой сложности.

```
import sys

sys.setrecursionlimit(20000) # увеличиваем глубину рекурсии, потому что дерево может быть высоким

with open('B_medium.txt', 'r') as f: # открываем файл на чтение
    n, k = map(int, f.readline().split()) # считываем количество вершин и число k
    edges = [[] for _ in range(n)] # заводим список смежности
    for _ in range(n - 1): # добавляем элементы в список смежности
        v, u = map(int, f.readline().split())
        v -= 1
        u -= 1
        edges[v].append(u) # говорим, что вершина v связана с вершиной u ребром
        edges[u].append(v) # и наоборот

used = [False for _ in range(n)] # массив посещенных вершин (изначально никакие не посетили)
potential = [] # заводим список вершин, которые потенциально могут являться ответом


def dfs(cur):
    used[cur] = True # посетили текущую вершину
    # проверка на то, что текущая вершина - лист
    if cur == 0: # если это корень, то условия одни
        if len(edges[cur]) == 0: # если нет соседей вообще
            return 1 # возвращаем единичку, тк текущая вершина - лист
    else:
        if len(edges[cur]) == 1: # если соседом является только вершина выше
            return 1 # это лист, возвращаем единичку


    q_leaves = 0 # здесь будем хранить количество листьев в поддереве текущей вершины
    for neighbour in edges[cur]: # проходимся по всем соседям текущей вершины
        if used[neighbour]: # если уже были в этом соседе
            continue # то еще раз в него не заходим, пропускаем

        q_leaves += dfs(neighbour) # прибавляем единичку, если сосед является листом, иначе 0

    if q_leaves == k: # если из вершины идет столько листьев, сколько нам надо
        potential.append(cur) # то в теории она может быть ответом

    return 0  # сама эта вершина не лист, значит, на своего предка она не влияет


dfs(0)
print(min(potential) + 1) # все действия были в 0-индексации, поэтому +1 в конце
```

1) Устанавливаем большую глубину рекурсии, так как данные большие.
2) Считываем данные из файла. Будем граф хранить в списке смежности, где edges[i] = список соседних вершин.
3) Создаем массив, где будем хранить информацию о том, были мы в этой вершине или нет (обращение по индексу).
4) Пишем обход в глубину (помечаем, что побывали в вершине, возвращаем единичку, если сейчас находимся в листе, обходим соседей и считаем общее количество листов, в которые можно попасть из этой вершины; если количество равно k, то добавляем текущую вершину в список потенциальных ответов)
5) Вызываем обход в глубину от корня, выводим минимальный номер вершины из потенциальных.

Ответ: 43