

ФИО: Губарева Екатерина Алексеевна

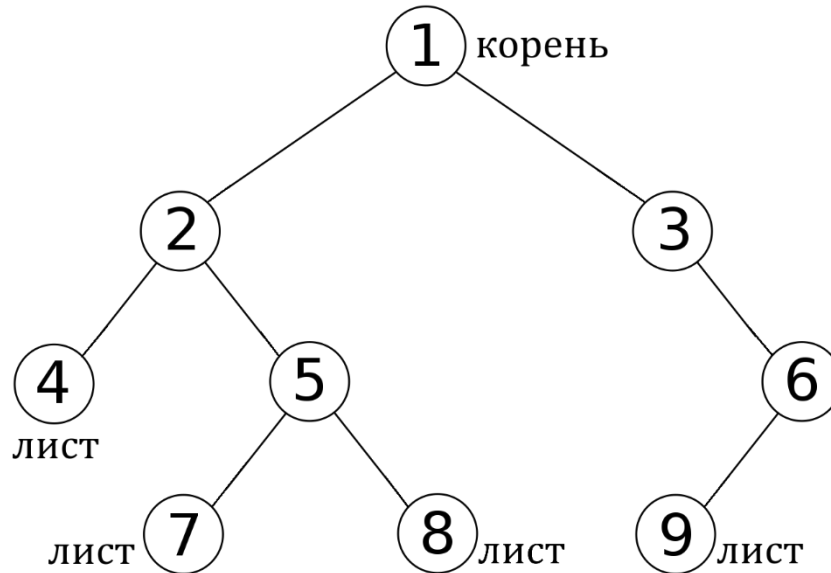
Ссылка на ВК: <https://vk.com/id329976472>

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

ЗАДАНИЕ И РЕШЕНИЕ НИЖЕ

Задание легкой 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, сколько в G листьев.



Входные данные

Дано два входных файла (файл A_easy и файл B_easy), каждый из которых в первой строке содержит целое число ($1 < N < 100\,000$) — количество вершин в G . 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_easy, затем — для файла B_easy.

Типовой пример организации данных во входном файле

```
9
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.

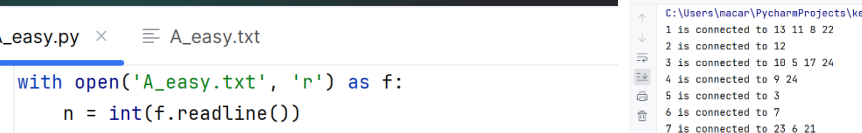
Ответом на вопрос задачи является число 4.

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 легкой сложности

Для решения файла А 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_easy.py).



The screenshot shows a code editor with a dark theme. The top bar includes icons for a file explorer, a search icon, and a 'Run' button. The editor has two tabs: 'A_easy.py' (active) and 'A_easy.txt'. The Python code in 'A_easy.py' reads a graph from 'A_easy.txt' and prints the neighbors for each node. The output in 'A_easy.txt' shows the neighbors for each node from 1 to 25. The code is as follows:

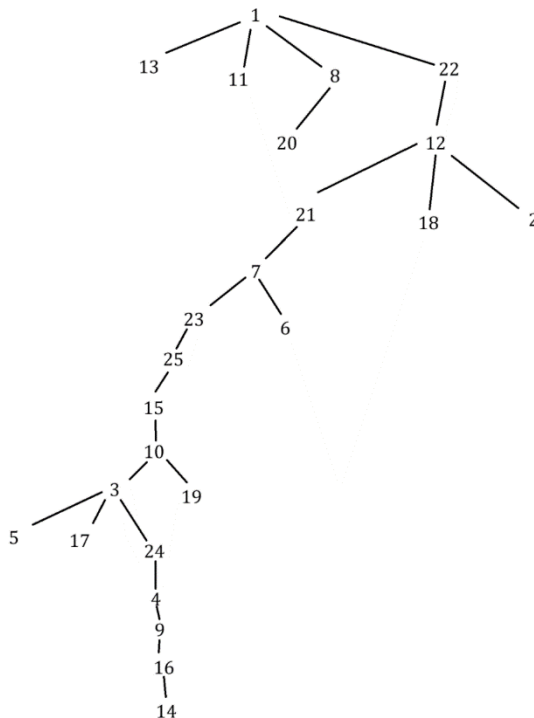
```
1 with open('A_easy.txt', 'r') as f:
2     n = int(f.readline())
3     edges = [[] for _ in range(n + 1)]
4     for _ in range(n - 1):
5         v, u = map(int, f.readline().split())
6         edges[v].append(u)
7         edges[u].append(v)
8
9 for i, neighbours in enumerate(edges):
10     if i == 0:
11         continue
12     print(i, 'is connected to', *neighbours)
```

The output in 'A_easy.txt' is:

```
1 is connected to 13 11 8 22
2 is connected to 12
3 is connected to 10 5 17 24
4 is connected to 9 24
5 is connected to 3
6 is connected to 7
7 is connected to 23 6 21
8 is connected to 20 1
9 is connected to 4 16
10 is connected to 3 15 19
11 is connected to 1
12 is connected to 21 18 2 22
13 is connected to 1
14 is connected to 16
15 is connected to 10 25
16 is connected to 14 9
17 is connected to 3
18 is connected to 12
19 is connected to 10
20 is connected to 8
21 is connected to 12 7
22 is connected to 1 12
23 is connected to 7 25
24 is connected to 4 3
25 is connected to 23 15
```

At the bottom, it says 'Process finished with exit code 0'.

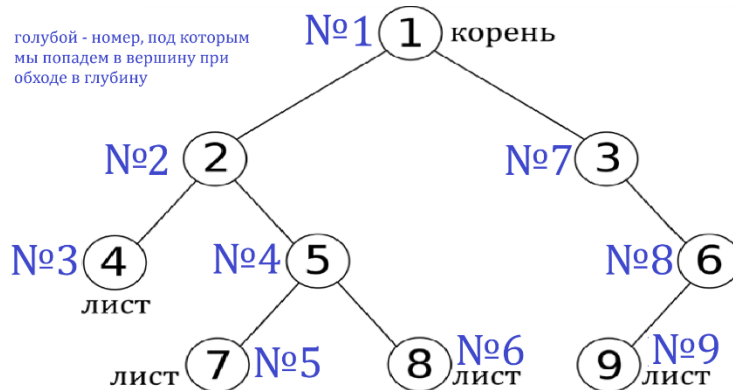
По полученным результатам легко нарисовать граф. Вот он.



В данном случае листьями являются вершины с номерами 13, 11, 20, 18, 2, 6, 19, 5, 17, 14. Всего листьев 10.

Ответ: 10

Для решения файла В нужно написать обход графа в глубину и подсчет количества листьев (файл В_easy.py). Суть обхода в глубину в том, чтобы пройти по каждой вершине дерева, начиная от корня, до листьев. Будем делать это рекурсивно, что даст нам возможность сначала упереться из корня в лист, а затем пойти дальше по дереву. Порядок обхода вершин из примера при обходе в глубину можно увидеть на рисунке ниже.



Хранить граф будем списком смежности, то есть для i -той вершины $edges[i]$ = список соседних вершин. Заведем булевый массив $used$ для того, чтобы пометить посещенные вершины, чтобы не посещать вершины несколько раз и обход в глубину работал за линейное время.

Обходя граф, будем определять, является вершина листом или нет. Тут есть два случая. Если вершина является корнем, то она лист, если у нее нет никаких соседей по ребрам. Если же вершина не корень, то она лист, если у нее только один сосед (вершина, выше нее).

```
import sys

sys.setrecursionlimit(20000)

with open('B_easy.txt', 'r') as f: # открываем файл на чтение
    n = int(f.readline()) # считываем количество вершин
    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)

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

    s = 0 # тут храним количество листьев из поддерева текущей вершины
    for neighbour in edges[cur]: # обходим всех соседей текущей вершины
        if used[neighbour]: # если уже посещали этого соседа
            continue # то второй раз в него не идем, пропускаем

        s += dfs(neighbour) # прибавляем количество листьев из поддерева соседа

    return s # возвращаем количество листьев в текущем поддереве

print(dfs(0)) # функция вернет ответ для всего дерева
```

Ответ: 1515