Яндекс. Тренировки по алгоритмам 2.0, занятие 8 (A)

A. Города-1

Ограничение времени	1 секунда
Ограничение памяти	256Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Дорожная сеть в Байтландии обладает следующими свойствами:

- Неориентированность: На всех дорогах движение является двусторонним.
- Связность: Из любого города Байтландии можно проехать в любой другой по сети дорог.
- Отсутствие циклов: Между любыми двумя городами Байтландии существует ровно один путь.

Назовём *удалённостью* города максимум из расстояний от него до других городов. Требуется найти все города с минимальной удалённостью.

Формат ввода

Первая строка входа содержит целое число N — количество городов ($1 \le N \le 10^5$). Каждая из последующих $N-1$ строк содержит по два целых числа — номера городов, соединённых очередной дорогой. Города занумерованы последовательными целыми числами от 1 до N.

Формат вывода

Выведите в одной строке через пробел минимальную удалённость, количество городов, для которых она достигается, а также список этих городов, отсортированный по возрастанию номеров.

Пример

Ввод	Вывод
2 1 2	1 2 1 2

Язык Python 3.9 (PyPy 7.3.11)

Набрать здесь | Отправить файл

```python
from collections import deque

def bfs(graph, v):  # функция поиска в ширину
    distances = [(-1, -1) for _ in range(len(graph))]  # массив расстояний до вершины i вида: (предыдущая вершина на пути, расст
    queue = deque()  # очередь
    distances[v] = (v, 0)  # расстояние начальной вершины
    queue.append(v)
    while queue:  # пока есть очередь
        cur_v = queue.popleft()  # берем из очереди вершину
        for next_v in graph[cur_v]:  # для каждого ее соседа
            if distances[next_v][1] == -1:  # если еще не обходили
                queue.append(next_v)  # добавляем в очередь
                distances[next_v] = (cur_v, distances[cur_v][1] + 1)  # сохраняем предыдущую вершину и расстояние до исходной
    return distances

# считываем данные
N = int(input().strip())  # число вершин
graph = [[] for _ in range(N)]  # граф
for _ in range(N - 1):
    a, b = map(int, input().split())
    a -= 1
    b -= 1
    graph[a].append(b)
    graph[b].append(a)


distances = bfs(graph, 0)  # находим начальные расстояния
first_max = max(range(len(distances)), key=lambda i: distances[i][1])  # выбираем самую дальнюю виршину
distances = bfs(graph, first_max)  # находим расстояния от уже выбранной вершины
second_max = max(range(len(distances)), key=lambda i: distances[i][1])  # находим вторую вершину самого длинного пути
# расстоянием ответа - расстояние от вершин на середине самого длинного пути
answer_dist = (distances[second_max][1] + distances[second_max][1] % 2) // 2
answer = []  # массив ответа
while second_max != first_max:  # восстанавливаем путь между полученными first_max и second_max вершинами
    answer.append(second_max)
    second_max = distances[second_max][0]
answer.append(first_max)
```

Отправить

Предыдущая

Следующая