ФИО: Губарева Екатерина Алексеевна

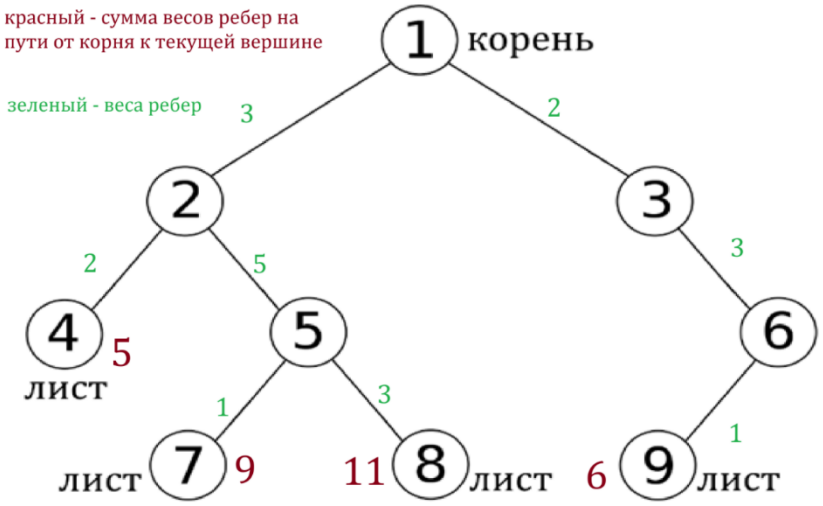
Ссылка на ВК: https://vk.com/id329976472

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

ЗАДАНИЕ И РЕШЕНИЕ НИЖЕ

Задание максимальной сложности

Пусть G – неориентированный взвешенный 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). Веса ребер – целые неотрицательные числа. Определите минимальную сумму весов ребер на пути из корня дерева G в какой-нибудь лист. В ответе укажите эту сумму.



Входные данные

Дано два входных файла (файл A_hard и файл B_hard), каждый из которых в первой строке содержит натуральное число N (1 < N < 100 000)— количество вершин в G. Далее в N-1 строках перечислены рёбра графа. Каждое ребро задаётся тройкой чисел — номерами начальной и конечной вершин соответственно, а также весом ребра. В ответе укажите два числа: сначала значение искомой величины для файла A_hard, затем — для файла B_hard.

Типовой пример организации данных во входном файле

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

При таких входных данных листьями являются вершины с номерами 4, 7, 8, 9. На пути от корня до вершины номер 4 можно собрать минимальную сумму весов ребер, равную пяти. 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 максимальной 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_hard.py).

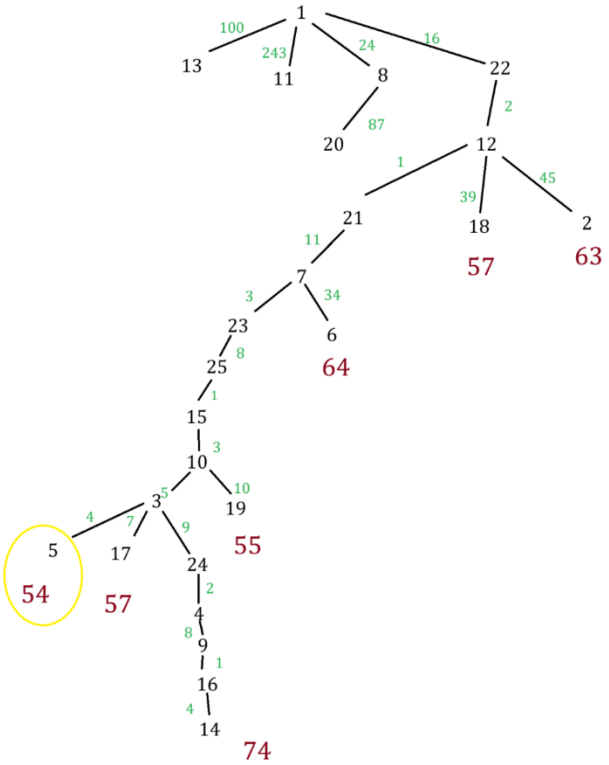
```
with open('A_hard.txt', 'r') as f:
    n = int(f.readline())
    edges = [[] for _ in range(n + 1)]
    for _ in range(n - 1):
        v, u, cost = map(int, f.readline().split())
        edges[v].append((u, cost))
        edges[u].append((v, cost))

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

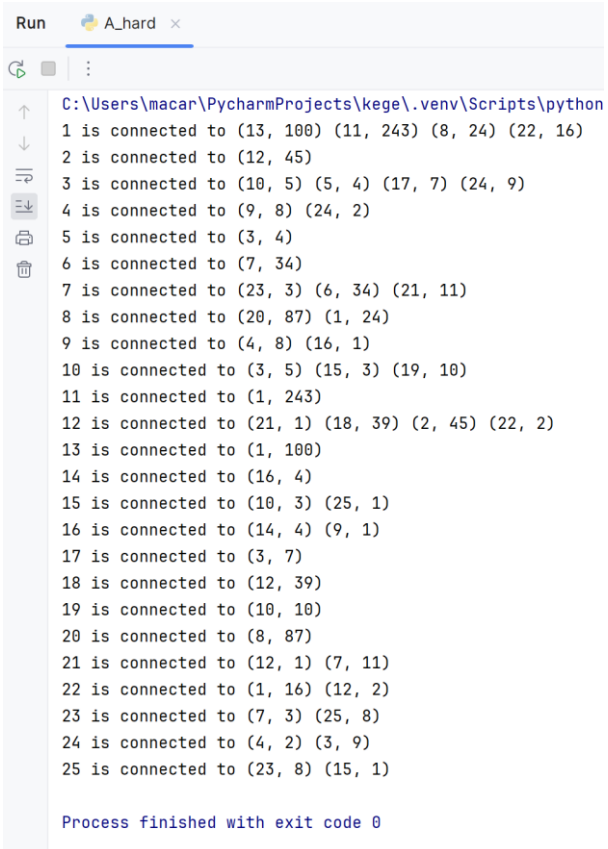
По полученным результатам легко нарисовать граф.

Рисунок ниже.

В данном случае из всех листов минимальная сумма по ребрам от корня до вершины равна 54, что соответствует пути от вершины 1 до вершины 5.

Ответ: 54





Для решения файла B нужно написать обход графа в глубину (ширину) и подсчет минимальной суммы весов на ребрах (файл B_hard.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_hard.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, cost = map(int, f.readline().split())
        v -= 1 # числа даны в 1-индексации, а мы будем работать в 0-индексации, поэтому уменьшаем
        u -= 1
        edges[v].append((u, cost)) # 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

    result = 10**18 # здесь храним минимальную сумму
    # весов на пути от какого-то листа до текущей вершины
    has_kids = False # заводим флаг о наличии соседей ниже текущей вершины
    for neighbour, weight in edges[cur]: # проходимся по соседям и весам на ребрах
    # между текущей вершиной и соседом
        if used[neighbour]: # если уже были в этом соседе
            continue # то еще раз в него не заходим, пропускаем

        has_kids = True # помечаем, что есть какие-то соседи у текущей вершины

        result = min(result, dfs(neighbour) + weight) # обновляем мин. Сумму из того, что было, и
#       того, что удалось получить из соседа)

    if has_kids: # если соседи есть
        return result # то возвращаем то, что смогли насчитать
    return 0 # если это лист, то сумма весов ребер на пути от какого-то листа до листа = 0


print(dfs(0)) # функция вернет мин. Сумму весов ребер на пути от какого-то листа до корня
```

1) Устанавливаем большую глубину рекурсии, так как данные большие.
2) Считываем данные из файла. Будем граф хранить в списке смежности, где edges[i] = список кортежей (вершина, цена) 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, попутно проверяем, находимся ли мы в листе, обходим соседей и считаем минимальную сумму по ребрам, в которые можно попасть из этой вершины; в итоге если мы в листе, то возвращаем 0, иначе возвращаем то, что насчитали)
5) Вызываем обход в глубину от корня, который возвращает ответ.

Ответ: 2037