Puzzle: Wine Distribution at Apan Vineyards

A large group of friends from the town of **Nocillis** is visiting the renowned vineyards of **Apan** to sample and purchase fine wines. Each friend is allowed to buy **up to 3 bottles** of wine, **provided the wines are available**.

However, there's a special rule at the vineyards:

Only one bottle of each type of wine can be sold — no duplicates.

To manage this fairly, the vineyard asks each guest to submit a list of **up to 10 wines** they enjoyed and would be happy to purchase.

Your task is to help the vineyard **maximize the number of wine bottles sold** to the group, under the following constraints:

Input

A **tab-separated values (TSV)** file with two columns:

- **Column 1:** `person_id` – unique identifier of a person
- **Column 2:** `wine_id` – unique identifier of a wine the person likes

Each row represents one person liking one wine. A person may appear in up to 10 rows (10 liked wines).

Example files (in increasing size and complexity):

- person_wine_3.txt
- person_wine_4.txt.zip
- person_wine_5.txt.zip

Output

- The **first line** should be a single integer: the **total number of wine bottles sold**.
- Each **subsequent line** should contain:
  - `person_id` (who will receive the wine)
  - `wine_id` (the wine assigned to that person)
  - **Tab-separated**

Constraints

- Each **wine** can be sold to **only one person**.
- Each **person** can receive **up to 3 wines** from their list of preferences.
- A person's ID may appear **at most 3 times** in the output.
- A wine's ID must appear **exactly once or not at all** in the output.

Goal

Write a program that reads the input TSV file and produces the required output while maximizing the number of bottles sold. The solution should:

- Be implemented using **any mainstream programming language** (e.g., **Python**, **PHP**, **Node.js**, **Java**, etc.)
- Follow **clear and consistent naming conventions**
- Include **concise and meaningful comments**
- Follow **best coding practices** such as:
    1. Proper error handling
    2. Logical modular structure
    3. Efficient algorithmic design
- The final submission should:
    1. Be uploaded to **GitHub** as a **public repository**
    2. Include a clear and concise `README.md` file with:
        - Setup instructions
        - How to run the code
        - Example input/output usage
        - Any dependencies or assumptions