Editorial for Warm-up - Pairing Cows Problem

Description of Problem

1. Functionality

   You are given N cow types. For each type i you know c_i (the number of cows of this type) and an ID value v_i. You are also given two integers A and B. You can form a pair from any two cows if the sum of their ID values is exactly A or exactly B. Each cow can be used in at most one pair. *A pair can be formed between two distinct cows of the same or different types*. Compute the maximum number of pairs that can be formed.

2. Input and Output

   (a) Input:

   - The first line contains N, A, B
   - The next N lines each contain c_i and v_i

   (b) Output:

   - maximum number of pairs that can be formed.

Solution of Problem

We can model the cows and their pairings as an undirected graph. Each distinct ID (*along with cow count*) is a node. We add an edge between x and y if x + y = A or x + y = B. If 2x = A or 2x = B, then x has a self-loop. In each connected component we greedily take as many pairs as possible while walking along the component and reducing the cow counts. It is optimal to start from a node of degree 1 (a leaf).

Key Observations:

1. There is no simple cycle of length ≥ 3.

   Suppose a, b, c are distinct and form a triangle with:

   a + b = A

   b + c = B

   a + c = A

   From the first two:

   (a + b) − (b + c) = A − B ⇒ a − c = A − B

   Subtract this from a + c = A:

(a + c) − (a − c) = A − (A − B) ⇒ 2c = B ⇒ c = B/2

Then b + c = B ⇒ b = B/2, so b = c, contradicting that all IDs are distinct.

So each connected component is either:

– a path (possibly with a self-loop on an endpoint), or

– a single node with a self-loop.

It's suboptimal to take the self loop first since it leaves you with 1 pair instead of 2, it's more optimal to handle outgoing neighbors first.

2. Self-loops should be selected last.

Consider a node a with:

a + b = A and 2a = B.

If you choose the self-loop on a first, you consume 2 copies of a and get 1 pair.
But if you first pair a with b as much as possible, you may be able to form 2 pairs total (using both a–b edges and possibly the self-loop later). So taking the self-loop first can reduce the total number of pairs, hence it is suboptimal. We always prioritize edges to other nodes over self-loops.

3. Starting from leaves is optimal.
Since components are paths (no long cycles), processing from leaves inward lets us always push pairs along the path, never "wasting" cows that could have paired with others.

Complexity:
N = total number of (count, id) pairs you read
M = number of distinct ids
Time: O(N +M) Space: O(M)

Implementation of Solution

The solution is implemented in Python. The file is pairs.py under the directory compatible_pairs

Test Cases

All the test cases are stored under the directory compatible_pairs/pairs_tc