Assignment-2

Data Structures and Algorithms
Binary Search Trees

Due Date: 23 Feb, 11:59 PM

Important Notes

- You are not allowed to copy the exact code given in the class. You are allowed to note the logic, but you need to implement the code yourself.
- OPERATION names are case sensitive.
- This problem has 3 parts, each part will be present on OJ as its own problem, but all parts are related.
- Hints for some parts are present at the end of the document.
- Please read the Plagiarism Policy before starting the assignment.

1 Overview

After a legendary career as a Pokémon Master, Ash Ketchum has finally retired from battling. However, a new adventure awaits—the quest to become the ultimate Pokémon card collector!

To begin his journey, Ash returns to Pallet Town to seek guidance from Professor Oak. To Ash's surprise, the Professor was once an avid Pokémon Card Collector himself! Excited to help, Professor Oak gifts Ash an old collection of n Pokémon cards to start his journey. But mastering the art of card collecting requires more than just a good collection—Ash must trade with other collectors, track his cards efficiently, and manage his growing collection.

To assist him, Professor Oak hands over a special device—the PokeCardDex, a digital tool designed to record, manage, and answer queries about Ash's Pokémon cards. But just as Ash eagerly tries to load his collection into the PokeCardDex, he realizes that it is broken! Without it, tracking his collection and handling trades will be nearly impossible.

Determined to continue his journey, Ash turns to you for help in fixing and rebuilding the PokeCardDex. Will you help Ash restore its functionality and embark on his quest to become the greatest Pokémon card collector?

1.1 Problem Statement

Each card is defined by two positive integers: its *attack* and *defense* stats. The strength of a card is determined as follows:

- A card A with stats $(a_{\text{attack}}, a_{\text{defense}})$ is *stronger* than a card B with stats $(b_{\text{attack}}, b_{\text{defense}})$ if

$$a_{\text{attack}} + a_{\text{defense}} > b_{\text{attack}} + b_{\text{defense}}.$$

- If $a_{\text{attack}} + a_{\text{defense}} = b_{\text{attack}} + b_{\text{defense}}$, then A is stronger than B if

$$a_{\text{attack}} > b_{\text{attack}}.$$

Initially, n cards are provided as input (cards may not be distinct). Following this, there are q queries of various types.

The problem is divided into three parts. While Part 1 involves only `TRADE` operations, Parts 2 and 3 allow for both `TRADE` operations *and* an additional query type (either `COMPARE` or `KTH_STRONGEST`).

1.2 Common Input Format and Constraints

Input:

- The first line contains an integer n ($1 \leq n \leq 10^5$), the number of cards in the initial collection.
- The next n lines each contain two space-separated integers, representing the `attack` and `defense` stats of a card ($1 \leq$ `attack`, `defense` $\leq 10^9$). Note that the cards may not be distinct.
    1. The initial n cards are given in a completely random permutation. [**75 PTS**].
    2. The initial n cards may be in any arbitrary order (even sorted). [**25 PTS**].
    3. For each part, 75% of the points are awarded for test cases with the initial n cards in a completely random permutation, and the remaining 25% are awarded for test cases with the initial n cards in an arbitrary order.
- The following line contains an integer q ($1 \leq q \leq 10^5$), the number of queries.
- The subsequent q lines each describe a query. Depending on the problem part, these queries may be of type `TRADE`, `COMPARE`, or `KTH_STRONGEST` (see details in each part).

Common Output Format

For each query that requires an immediate output, print the result as specified for the query (see below for details). After processing all queries, output the final state of Ash's collection:

1. A single line with the total number of *unique* cards.
2. For each unique card, output a line with three space-separated integers: `attack defense count`.
3. All the cards must outputted in order from *strongest to weakest*.

2 Part 1: Trading (45 Points)

Problem Statement

Ash is now ready to trade with fellow collectors. In each trade, a prospective partner offers a card and requests a card from Ash. A trade query is given in the format:

$$\texttt{TRADE}\ a_1\ d_1\ a_2\ d_2,$$

where (a_1, d_1) represent the stats of the *offered* card and (a_2, d_2) represent the stats of the *requested* card. In some cases, no card is requested—in which case the requested card is represented as `-1 -1`.

When processing a trade, Ash follows these rules:

Rule 1: (Trade With a Request) If a card is requested (i.e. $(a_2, d_2) \neq (-1, -1)$):

(a) If Ash does not have the requested card or has only one copy of it, the trade is *declined*.

Rule 2: (No Requested Card) If the requested card is given as `-1 -1`, the trade is automatically *accepted*.

Rule 3: (Comparing Cards) Otherwise (i.e., Ash has at least two copies of the requested card):

(a) If Ash does not already have the offered card, the trade is *accepted*.

(b) If Ash already owns the offered card, compare the two cards:

- If the offered card is *weaker* than the requested card (according to the strength rules above), the trade is *declined*.
- Otherwise, the trade is *accepted*.

Trade Operation

When a trade is **accepted**:

- If a card is requested (i.e., $(a_2, d_2) \neq (-1, -1)$), Ash gives away one copy of that card.
- Ash receives one copy of the offered card.

Input Format

```
n
a_1 d_1
a_2 d_2
...
a_n d_n
q
<q queries, each in the format: TRADE a1 d1 a2 d2>
```

Note: All cards—including those offered in the queries—are provided in an almost random order. Therefore, you may assume that the BST will remain reasonably balanced, and you do not need to worry about degenerate cases resulting in a skewed tree.

Output Format

- For each `TRADE` query, output a single line containing:
  - `1` if the trade is accepted.
  - `0` if the trade is declined.
- After processing all queries, output the final state of the collection:
  1. A single line with the total number of *unique* cards.
  2. For each unique card, a line with three space-separated integers: `attack defense count`.
  3. All the cards must outputted in order from *strongest to weakest*.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq a_1, d_1 \leq 10^9$, $a_2 = d_2 = -1$ **[20 PTS]**
- $1 \leq a_1, d_1, a_2, d_2 \leq 10^9$ **[25 PTS]**
- $1 \leq q \leq 10^5$

Example

Input:

```
5
100 200
150 150
200 100
100 200
120 180
3
TRADE 130 170 100 200
TRADE 110 190 -1 -1
TRADE 150 150 120 180
```

Output:

```
1
1
0
6
200 100 1
150 150 1
130 170 1
120 180 1
110 190 1
100 200 1
```

Explanation:

1. *First Trade:* The partner offers a card $(130, 170)$ and requests a copy of $(100, 200)$. Assuming Ash has more than one copy of $(100, 200)$, the trade is accepted.

2. *Second Trade:* Since the requested card is `-1 -1`, the trade is automatically accepted.

3. *Third Trade:* Ash is asked to give away one copy of $(120, 180)$ in exchange for $(150, 150)$. Although he has at least two copies of $(120, 180)$, if he already owns $(150, 150)$ and the offered card is determined to be weaker than the requested card, the trade would be declined. (In this example, the conditions lead to a decline.)

3 Part 2: Card Insights (20 Points)

Problem Statement

Ash wishes to gain insights into his collection. In this part, the queries may be of two types:

1. `COMPARE attack defense`: For the given card, perform the following:
   (a) Report the number of copies of the specified card in Ash's collection.
   (b) Find the *strongest* card in the collection that is *strictly weaker* than the given card, and output its stats along with its count. If no such card exists, output `-1`.
   (c) Find the *weakest* card in the collection that is *strictly stronger* than the given card, and output its stats along with its count. If no such card exists, output `-1`.
2. `TRADE a1 d1 a2 d2`: Process a trade operation as described in Part 1.

Input Format

```
n
a_1 d_1
a_2 d_2
...
a_n d_n
q
<q queries, each either in the format:
   COMPARE attack defense
   OR
   TRADE a1 d1 a2 d2>
```

Output Format

For each query, output as follows:

- For a `COMPARE` query, output three lines:
  1. The number of copies of the specified card.
  2. Either the stats and count of the strongest card strictly weaker than the specified card (formatted as `attack defense count`), or `-1` if no such card exists.
  3. Either the stats and count of the weakest card strictly stronger than the specified card (formatted as `attack defense count`), or `-1` if no such card exists.
- For a `TRADE` query, output a single line:
  - `1` if the trade is accepted.
  - `0` if the trade is declined.

After processing all queries, output the final state of the collection:

1. A single line with the total number of *unique* cards.
2. For each unique card, a line with: `attack defense count`.
3. All the cards must outputted in order from *strongest to weakest*.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq \texttt{attack}, \texttt{defense} \leq 10^9$
- $1 \leq q \leq 10^5$

Example

Input:

```
4
100 200
150 150
100 200
120 180
3
COMPARE 100 200
TRADE 130 170 100 200
COMPARE 100 200
```

Output:

```
2
-1
120 180 1
1
1
-1
120 180 1
4
150 150 1
130 170 1
120 180 1
100 200 1
```

Explanation:

1. *First Query (COMPARE 100 200):* Initially, Ash has 2 copies of $(100, 200)$, no card strictly weaker than $(100, 200)$, and the weakest card strictly stronger than $(100, 200)$ is $(120, 180)$ (count 1).
2. *Second Query (TRADE 130 170 100 200):* Ash has 2 copies of $(100, 200)$, so the trade is processed. Since he does not have $(130, 170)$, the trade is accepted. After the trade, one copy of $(100, 200)$ is given away and $(130, 170)$ is added.
3. *Third Query (COMPARE 100 200):* Now, Ash has 1 copy of $(100, 200)$. As before, no card is strictly weaker than $(100, 200)$, and the weakest card strictly stronger than $(100, 200)$ remains $(120, 180)$ (count 1).

Finally, the collection has 4 unique cards:

- $(150, 150)$ with count 1
- $(130, 170)$ with count 1
- $(120, 180)$ with count 1
- $(100, 200)$ with count 1

4 Part 3: Kth Strongest Card (35 Points)

Problem Statement

Ash now wants to rank his cards by strength. In this part, queries may be of two types:

1. `KTH_STRONGEST k`: For a given integer k ($1 \leq k \leq 2 \times 10^5$), determine the k^{th} strongest card from among all the cards (a card with multiple copies will count multiple times) in Ash's collection along with its count. The ranking follows the card strength ordering described in the Overview. If k exceeds the total number of cards in the collection, output `-1`.
2. `TRADE a1 d1 a2 d2`: Process a trade operation as described in Part 1.

Input Format

```
n
a_1 d_1
a_2 d_2
...
a_n d_n
q
<q queries, each either in the format:
   KTH_STRONGEST k
   OR
   TRADE a1 d1 a2 d2>
```

Output Format

For each query, output as follows:

- For a `KTH_STRONGEST` query, output a single line:
  - If the k^{th} strongest card exists, print its stats and count as: `attack defense count`.
  - Otherwise, print `-1`.
- For a `TRADE` query, output a single line:
  - `1` if the trade is accepted.
  - `0` if the trade is declined.

After processing all queries, output the final state of the collection:

1. A single line with the total number of *unique* cards.
2. For each unique card, a line with: `attack defense count`.
3. All the cards must outputted in order from *strongest to weakest.*

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq \texttt{attack}, \texttt{defense} \leq 10^9$
- $1 \leq q \leq 10^5$
- $1 \leq k \leq 2 \times 10^5$

Example

Input:

```
5
100 200
150 150
120 180
100 200
130 170
```

```
4
KTH_STRONGEST 2
TRADE 140 160 120 180
KTH_STRONGEST 3
KTH_STRONGEST 7
```

Output:

```
130 170 1
0
120 180 1
-1
4
150 150 1
130 170 1
120 180 1
100 200 2
```

Explanation:

1. Initially, the collection consists of:
   - $(100, 200)$ with count 2
   - $(150, 150)$ with count 1
   - $(120, 180)$ with count 1
   - $(130, 170)$ with count 1

   The overall ordering (if expanded by duplicates) is:

   (a) $(150, 150)$ (strongest)
   (b) $(130, 170)$
   (c) $(120, 180)$
   (d) $(100, 200)$ [first copy]
   (e) $(100, 200)$ [second copy]
2. *First Query (KTH_STRONGEST 2):* The second strongest card is $(130, 170)$ with count 1.
3. *Second Query (TRADE 140 160 120 180):* Ash is asked to trade for $(120, 180)$. However, since he has only one copy of $(120, 180)$, the trade is declined, so output `0`.
4. *Third Query (KTH_STRONGEST 3):* The third strongest card is $(120, 180)$ with count 1.
5. *Fourth Query (KTH_STRONGEST 7):* There are only 5 cards in total, so output `-1`.

Finally, the collection remains unchanged with 4 unique cards:

- $(150, 150)$ with count 1
- $(130, 170)$ with count 1
- $(120, 180)$ with count 1
- $(100, 200)$ with count 2

5 Submission Details

- All parts are to be submitted to OJ only.
- **All solutions submitted in OJ will be considered for plagiarism**

All the best and happy coding and collecting!

Hints

- **Initial Input Subpart 2:** In Subpart 1, where the input is already given in random order is not a problem when inserting into a BST. Can you insert elements in random order perhaps by iterating over the indices using a modulo. Or any other technique you can think of.
- **Part 1:** Managing duplicate elements in a BST can be challenging. How can you maintain only unique elements in a BST?
- **Part 3:** For each node in the BST, store and maintain the number of nodes in its subtree.