CS2810 - OOAIA Lab 12
GeoSolitaire: Solitaire for the Geometrically Gifted

Date: 8 April 2025

Link: Hackerrank Link

Problem Description

You are the lead developer at **GeoSolitaire.ai**, a shady casino-backed company building "AI tools" to win at digital solitaire. The idea? Replace number-based cards with **geometric shape cards**, where the **area** of the shape defines the card's strength.

You have been tasked with building a **Solitaire Management System** for a digital card game simulator. The system models a simplified version of solitaire that emphasizes order and strategy. Your goal is to determine how to organize incoming cards into the **minimum number of piles**, based on specific placement rules.

Cards arrive one at a time. The system can place cards in either of the following ways:

- A card can be placed on **top of an existing pile** only if its strength is **atmost** the top card of that pile.
- Create a **new pile** to insert the incoming card.

In order to stay profitable, your team plans to deploy automated players into the system — bots designed to quietly outperform regular users. Much like how gambling platforms stack odds against the player, your system will use subtle algorithmic advantages to optimize every move. The key to this is computing the **Longest Increasing Subsequence (LIS)** of the strengths of the incoming card stream. Your Solitaire Management System must therefore include functionality to extract any valid LIS from the card sequence as part of its strategic backend.

Design Specifications

Implement an abstract base `Card` class. Inherit it with classes for different shapes and override the appropriate methods. Overload the comparator operator and the stream insertion and extraction operators for the `Card` class. The design decisions for `Card` are left up to you. However, design decisions will be evaluated based on good programming practices.

Implement a `CardOrganizer` class that stores the state of the piles. Make sure to include the following methods:

- `void addCard(Card*)` — Finds a suitable pile and inserts the card in it.
- `size_t getPileCount() const` — Returns the number of piles created so far.
- `std::vector<Card*> getLIS() const` — Returns a valid LIS.

You can define additional classes and methods as needed.

Input and Output Format

1. Each test case corresponds to a single instance of the game.
2. The first line of each test case contains **q**, which represents the number of queries.
3. The next `q` lines each represent one of the following query types:
    1. **Add Card Query**
       Format:
       `1 SHAPE param1 <param2>`
       Adds a new card of the specified shape to the game.
        - For a square: `1 SQUARE edge`
        - For a rectangle: `1 RECTANGLE length breadth`
        - For a triangle: `1 TRIANGLE base height`
    2. **Pile Count Query**
       Format:
       `2`
       Prints the current minimum number of piles needed to organize the cards based on the game's placement rules.
    3. **LIS Query**
       Format:
       `3`
       Prints any valid Longest Increasing Subsequence (LIS) of the card sequence encountered so far, using area as the basis for comparison.

Output Format:

NOTE: While printing the output to stdout, please use '\n' instead of std::endl to avoid TLE.

For **queries of type 2**, output a single integer — the current minimum number of piles required to organize the cards according to the placement rules.

For **queries of type 3**, output the following:

- First, print a single integer n, representing the size of a valid Longest Increasing Subsequence (LIS) of the cards seen so far.
- Then, print n lines, each describing one card in the LIS, using the same format as in the input:
    - `SQUARE edge`
    - `RECTANGLE length breadth`
    - `TRIANGLE base height`

The cards in the output must appear in the order they appear in the LIS.

Input Constraints

$1 \leq q \leq 10^5$

The area of each shape is guaranteed to be an **integer** and doesn't exceed 10^9.

It is guaranteed that the number of queries of type 3 doesn't exceed 10^3.

Sample Test Case 1

Input:

```
5
1 TRIANGLE 2 3
1 RECTANGLE 1 2
1 SQUARE 1
```

```
2
3
```

Output:

```
1
1
SQUARE 1
```

Explanation:
Areas: 3, 2, 1. Each card fits on the same pile → 1 pile. One possible LIS is SQUARE 1.

Sample Test Case 2

Input:

```
4
1 SQUARE 1
1 RECTANGLE 2 2
1 TRIANGLE 10 2
2
```

Output:

```
3
```

Explanation:
Areas: 1, 4, 10. Each card goes in a different pile → 3 piles.

Sample Test Case

Input:

```
6
```

```
1 SQUARE 3
1 TRIANGLE 2 3
1 RECTANGLE 2 2

1 TRIANGLE 4 1
2
3
```

Output:

```
2
2
TRIANGLE 2 3
RECTANGLE 2 2
```

Explanation:

Areas: 9, 3, 4, 2. Multiple optimal arrangements exist. One such arrangement: (9, 4) in pile 1, and (3, 2) in pile 2. LIS is 3, 4.