Aufgabe: TRI
Dreiecke (Triangles)

german

CEOI 2018 Warsaw POLAND

CEOI 2018, Tag 2. Speicherlimit: 256 MB. *16.08.2018*

Byteland ist ein schönes Land mit n ($n \geq 3$) Städten, repräsentiert durch n unterschiedliche Punkte in einer Ebene. Die Städte sind von 1 bis n durchnummeriert. Als Tourist kennst du zwar nicht die exakten Positionen der Städte in Byteland, aber du hast in einem Fremdenführer gelesen, dass keine drei Städte kollinear zueinander liegen.

Die konvexe Hülle einer Menge von n Punkten ist ein konvexes Polygon mit der kleinstmöglichen Fläche, sodass alle n Punkte entweder innerhalb oder auf dem Rand des Polygons liegen. Ein konvexes Polygon hat nur Innenwinkel mit weniger als 180 Grad und darf sich nicht selbst schneiden.

Deine Aufgabe ist es nun, die Anzahl der Ecken auf dem Rand der konvexen Hülle der Städte von Byteland zu berechnen. Du darfst hierbei nur Fragen für Tripel von unterschiedlichen Städten i, j, k ($1 \leq i, j, k \leq n$) stellen. Solch eine Frage betrifft das Dreieck mit den Städten i, j und k als Ecken. Die Antwort auf die Frage gibt an, ob du das Dreieck im oder gegen den Uhrzeigersinn abläufst, wenn du die Städte in der Reihenfolge i, j, k besuchst.

Kommunikation

Dein Programm muss eine Bibliothek verwenden, die es erlaubt, Fragen zu stellen und die endgültige Antwort abzugeben.

Die Bibliothek (`trilib.h` für C und C++) stellt die folgenden Funktionen zur Verfügung:

- `int get_n();`
  Gibt die Anzahl an Städten zurück.
- `bool is_clockwise(int a, int b, int c);`
  Gibt `true` zurück, wenn die Ecken des Dreiecks a, b, c ($1 \leq a, b, c \leq n, a \neq b \neq c \neq a$) im Uhrzeigersinn übergeben wurden und `false`, falls sie gegen den Uhrzeigersinn übergeben wurden.
- `void give_answer(int s);`

In Java bietet die Klasse `trilib` die folgenden Methoden an:

- `static public int get_n();`
- `static public boolean is_clockwise(int a, int b, int c);`
- `static public void give_answer(int s);`

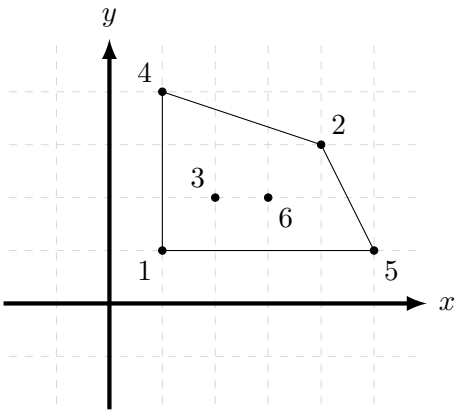
Nachdem dein Programm `give_answer` aufgerufen hat, darf es keine weiteren Fragen mehr stellen. Es muss `give_answer` genau einmal aufrufen.

Deinem Programm ist es nicht erlaubt, von der Standardeingabe zu lesen oder auf die Standardausgabe zu schreiben. Nachdem es `give_answer` aufgerufen hat, sollte sich dein Programm sofort beenden.

Du darfst annehmen, dass die Positionen der Punkte im Voraus festgelegt sind und sich im Laufe der Ausführung des Programmes nicht verändern (das bedeutet, dass sich die Bibliothek vollständig deterministisch verhält). Zum Beispiel würde es im Beispieltestfall (siehe unten) ausreichen, `give_answer(4)` aufzurufen und sich sofort zu beenden, um den Test zu bestehen. Deinem Programm ist es erlaubt, die Antwort zu raten, ohne sich sicher zu sein.

Beispielinteraktion

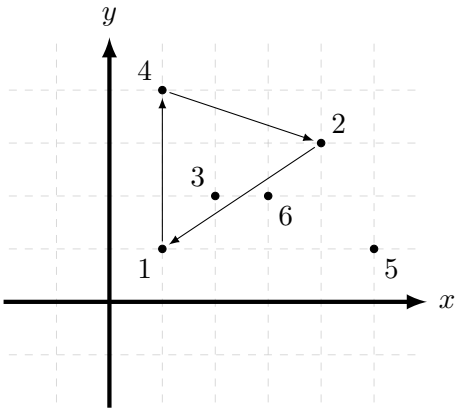
Nehmen wir $n = 6$ mit Städten an den Positionen $(1, 1)$, $(4, 3)$, $(2, 2)$, $(1, 4)$, $(5, 1)$, $(3, 2)$ an, wie im unteren Diagramm dargestellt. Die konvexe Hülle wird durch die eingezeichneten Linien veranschaulicht. Sie enthält vier Ecken auf ihrem Rand, somit ist das Resultat 4.



Die folgende Tabelle enthält eine Beispielinteraktion mit der Bibliothek, die diesem Beispiel entspricht.

Aufruf	**Rückgabewert**
`get_n()`	6
`is_clockwise(1, 4, 2)`	`true`
`is_clockwise(4, 2, 1)`	`true`
`is_clockwise(1, 2, 4)`	`false`
`is_clockwise(3, 6, 5)`	`true`
`give_answer(4)`	–

Das nachfolgende Diagramm zeigt das Dreieck der ersten Abfrage. Die Städte 1, 4, 2 sind im Uhrzeigersinn angeordnet, deshalb ist der Rückgabewert `true`.



Bewertung

Die Testfälle sind in die folgenden Teilaufgaben mit zusätzlichen Beschränkungen gegliedert. Jede dieser Teilaufgaben besteht aus einer oder mehreren Testfallgruppen. Jede Testfallgruppe enthält einen oder mehrere Testfälle.

In allen Testfällen gilt $3 \leq n \leq 40\,000$. Du darfst `is_clockwise` höchstens 1 000 000 Mal pro Testfall aufrufen.

Teilaufgabe	**Beschränkungen**	**Punkte**
1	$n \leq 50$	15
2	$n \leq 500$	20
3	$n \leq 15\,000$	20
4	maximal ein Punkt liegt nicht auf dem Rand der konvexen Hülle	20
5	keine zusätzlichen Beschränkungen	25

Lokales Testen

Im Ordner `public` gibt es eine Beispielbibliothek, die es dir erlaubt, die formale Korrektheit deines Programmes zu testen. Die Bibliothek liest eine Beschreibung von Byteland von der Standardeingabe im folgenden Format ein:

- In der ersten Zeile eine Ganzzahl n, die Anzahl an Städten,
- in den nächsten n Zeilen: Jeweils zwei Ganzzahlen, die Koordinaten der i-ten Stadt.

Die zur Verfügung gestellte Bibliothek überprüft deine Lösung **nicht**. Sie prüft auch nicht die Korrektheit der Eingabe. Sie ist nicht dieselbe wie die (geheime) Bibliothek auf dem Server.

Eine Beispieleingabe für die Bibliothek ist in der Datei `tri0.in` gegeben.

Nachdem `give_answer` aufgerufen wurde, gibt die Bibliothek die übergebene Antwort und die Anzahl an Aufrufen von `is_clockwise` auf der Standardausgabe aus.

Um die Lösungen mit der Beispielbibliothek zu kompilieren, kannst du die folgenden Befehle verwenden:

- **C**: `gcc -O2 -static trilib.c tri.c -lm -std=gnu99`
- **C++**: `g++ -O2 -static trilib.c tri.cpp -lm -std=c++11`

In Java brauchst du keinen besonderen Befehl, um die Lösung mit der Bibliothek zu kompilieren.

Die Dateien der Lösung und der Bibliothek sollten im gleichen Ordner sein.