Zentralmatura

Damit bei der Zentralmatura auch wirklich alles nach Vorschrift verläuft, hat das Ministerium angeordnet, dass die Prüfungsräumlichkeiten genauestens überwacht werden. Für dieses Problem betrachten wir den Raum vereinfacht als $n \cdot n$ Grid. Die Zeilen sind von oben nach unten von 1 bis n nummeriert, die Spalten von links nach rechts.

n Lehrer wurden eingeteilt die Deutschmatura zu beaufsichtigen. Damit ihnen auch wirklich nichts entgeht, haben sie beschlossen, dass jeder von ihnen sich um eine Spalte und Zeile des Grids kümmern sollt. D.h. wenn ein Lehrer in Zelle (r, c) des Grids steht, dann kontrolliert dieser sowohl Zeile r als auch Spalte c. Es dürfen keine zwei Lehrer in derselben Zeile oder Spalte stehen. Wenn sich die n Lehrer also entlang einer der beiden Diagonalen aufstellen wird das gesamte Grid kontrolliert.

Leider Gottes ist es aber nicht so einfach: Die Lehrer möchten, dass sie möglichst nahe bei ihren Klasse sind (im Prüfungsraum sitzen Schüler von verschiedenen Klassen). Genauer gesagt gibt es für jeden Lehrer ein Rechteck, wo er gerne stehen würde.

Da es sich aber wie gesagt um die Deutschmatura handelt, haben die zuständigen Lehrer mehr Ahnung von Beistrichsetzung als vom Lösen solcher Aufgaben. Kannst du ihnen helfen und eine mögliche Anordnung der Lehrer finden, sodass jeder von ihnen glücklich ist?

Implementierungsdetails

Implementiere die folgende Funktion:

```
Position[] solve(Preference[] preferences)
```

- *preferences* ist ein Array in dem Präferenz von jedem Lehrer gespeichert ist.
- Diese Funktion soll ein Array mit den Positionen der Lehrer zurückgeben.Wenn es mehrere Lösungen gibt kannst du ein beliebige Lösung zurückgeben. Wenn es keine Lösung gibt, gib ein leeres Array zurück. In Java soll *null* zurück gegeben werden.

```
Preference {
  int a
  int b
  int c
  int d
}
```

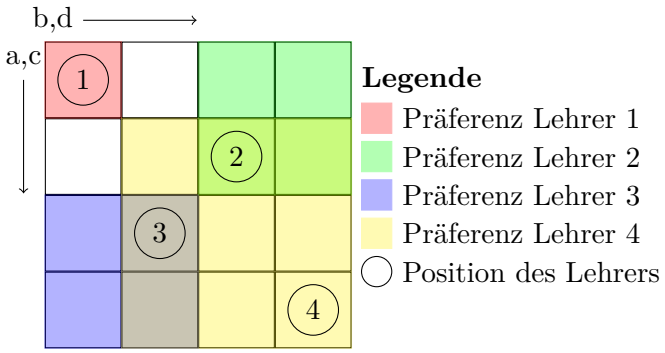
- *Preference* ist eine oben genutzte Datenstruktur, um die Präferenz eines Lehrers anzugeben.
- (a, b) gibt die linke obere Ecke des Rechtecks an, (c,d) die rechte unter Ecke.

```
Position {
  int r
  int c
}
```

- *Position* ist eine oben genutzte Datenstruktur, um die Position eines Lehrers anzugeben.
- (r,c) gibt die Position von einem Lehrer an.

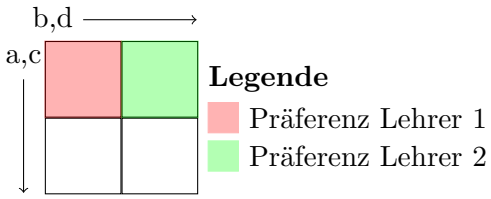
Beispiel

Beispiel 1



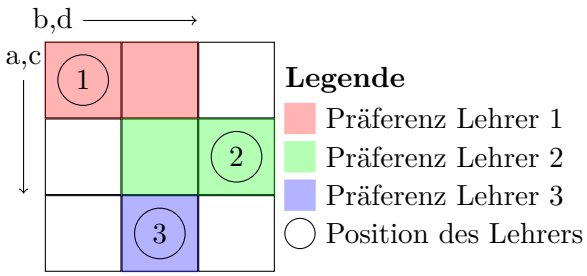
In der oberen Graphik ist eine Beispiel für eine Aufgabenstellung und eine Mögliche Lösung dargestellt. Es handelt sich hier um eine gültige Lösung, da in jeder Zeile und Spalte *genau* ein Lehrer steht. Beachte, dass die im oberen Beispiel dargestellte Lösung ist nicht die einzige Lösung ist.

Beispiel 2



In diesem Beispiel gibt es keine Lösung, da wir in der zweiten Zeile keinen Lehrer positionieren können.

Beispiel 3



In diesem Beispiel möchte Lehrer i in Zeile i stehen, somit entspricht dieses Beispiel den Einschränkungen in Subtask 2 bzw. 3.

Subtasks

Allgemein gilt:

- $1 \leq n \leq 10^5$

Subtask 1 (21 Punkte): $n \leq 10$

Subtask 2 (26 Punkte): Wie Subtask 3, aber zusätzlich mit $n \leq 3000$

Subtask 3 (41 Punkte): Lehrer i möchte in Zeile i stehen ($a_i = c_i = i$)

Subtask 4 (12 Punkte): Keine Einschränkungen

Beispielgrader

Damit du nicht selbst Ein- und Ausgabe implementieren musst haben wir dass in einem Beispielgrader für dich übernommen. Um dein Programm zu testen kannst du das folgende Format für die Ein- und Ausgabe verwenden.

Eingabe

Die erste Zeile des Inputs enthält eine einzige Zahl n, die Seitenlänge des Grids. In den folgenden n Zeilen befinden sich jeweils 4 Zahlen, die das Rechteck beschreiben, indem der i-te Lehrer sein will: a_i, b_i, c_i, d_i. (a_i, b_i) repräsentiert die links-obere Ecke (Zeile/Spalte) des Rechtecks, (c_i, d_i) die rechts-untere Ecke.

Ausgabe

Der Beispielgrader gibt n Zeilen mit jeweils zwei Zahlen aus. Die i-te Zeile beschreibt die Position von Lehrer i, zuerst Zeile, dann Spalte. Wenn du ein leeres Array zurückgibst (also sagst, dass es keine Lösung gibt) dann gibt der Beispielgrader -1 zurück.

Zur besseren Verständlichkeit ist hier noch die Ein und Ausgabe für Beispiel 1.

Eingabe	**Ausgabe**
4 1 1 1 1 1 3 2 4 3 1 4 2 2 2 4 4	1 1 2 3 3 2 4 4