Höhlenforscher

Herr Dr. Rainer ist ein renommierter Wissenschaftler auf dem Gebiet der Speläologie (Höhlenforschung). Sein Team und er sind gerade dabei ein neu erschlossenes Höhlensystem zu erkunden. Da dieses erst vor kurzem entdeckt wurde, ist nur wenig über die Höhlen bekannt. Fasziniert von seiner Arbeit ist Dr. Rainer an jenem Tag besonders Tief in den Höhlenkomplex vorgedrungen. Erst viel zu spät wurde ihm bewusst, dass er ganz vergessen hatte, sich den Weg zurück zu merken. Wie er nun je wieder hinaus finden soll, ist ihm ein Rätsel.

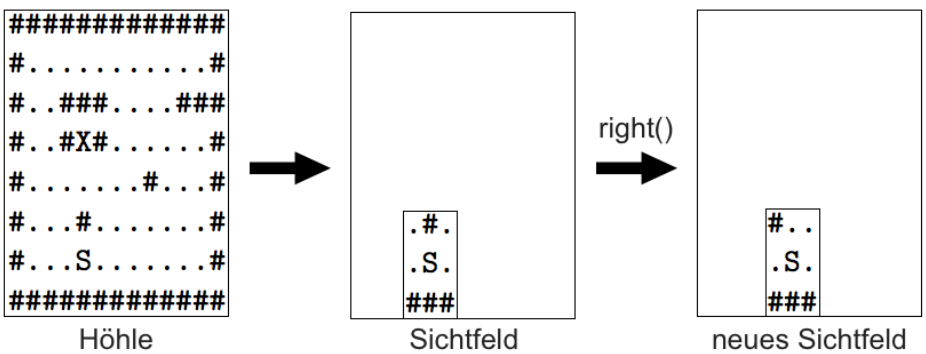
Was die ganze Situation noch schlimmer macht ist, dass seine Forschungsgelder knapp bemessen sind. Deshalb besitzt Dr. Rainer anstatt einer professionellen Stirnlampe (oder Funkgeräten, um seine Kollegen zu kontaktieren) nur eine kleine Taschenlampe, deren Batterie schon dem Ende zugeht. Mehr ist im Budget einfach nicht drinnen.

Für den Zweck dieser Aufgabe repräsentieren wir die Höhle, in der sich Dr. Rainer befindet, vereinfacht als $n \times m$ Raster. Jede Zelle dieses Rasters ist entweder begehbar (gekennzeichnet durch einen Punkt "."), oder durch Gestein ("#") geblockt. Der Rand ist dabei nie begehbar, da dieser die "Außenwände" der Höhle darstellt. Weiteres befindet sich bei genau eine (unbekannten) Zelle der Ausgang ("X").

Dr. Rainer ist sich leider Gottes nicht sicher, wo er sich gerade befindet. Seine Taschenlampe liefert ihm aber zumindest ein 3×3 großes Sichtfeld rund um seine Position (welche als "S" gekennzeichnet ist). Ausgehend von dieser Information muss er nun den Ausgang finden, wenn er nicht in der Höhle erfrieren möchte.

Implementiere dazu die Funktion `findExit(subtask, fieldOfView)`. Die Zahl `subtask` gibt an, um welchen Subtask es sich bei diesem Test handelt. Der Parameter `fieldOfView` ist ein 3×3 Array, das sein aktuelles Sichtfeld beschreibt. Dein Program kann Dr. Rainer kontrollieren, indem es über die Funktionen `left()`, `right()`, `up()` und `down()` seine Bewegung steuert. Diese bewegen ihn einen Schritt nach Links, Rechts, Oben bzw. Unten und aktualisieren das `fieldOfView` Array mit dem neuen Sichtfeld. Sollte ein Schritt in diese Richtung nicht möglich sein (da z.B. Gestein im Weg ist), so bleibt er auf der Stelle stehen. Sobald er die Zelle mit dem Ausgang betritt, wird das Program automatisch beendet und Dr. Rainer damit gerettet.

Die Batterie seiner Taschenlampe hat noch Energie für maximal 2000000 Schritte (`left()`, `right()`, `up()` und `down()`). Beachte dabei, dass Schritte "gegen" die Wand genauso mitzählen. Sollte es ihm bis dahin nicht gelingen den Ausgang zu finden, dann schaut es für Herrn Dr. Rainer wohl schlecht aus - und du bekommst "Wrong Answer". Kannst du ihm helfen?



Eingabe

Das Einlesen wird bereits vom Grader übernommen. Dieser liest die Daten in folgendem Format ein (dies kann für dich beim lokalen Testen hilfreich sein): Die erste Zeile enthält Subtask Nummer, n (Anzahl Zeilen), m (Anzahl Spalten) - in dieser Reihenfolge. Die folgenden n Zeilen beschreiben die Höhle. "#" steht dabei für Gestein, "." für eine leere Zelle, "S" für die Startposition und "X" für den Ausgang.

Beispiel

Eingabe	Interaktion

`5 8 13`
`#############`
`#...........#`
`#..###....###`
`#..#X#......#`
`#.......#...#`
`#...#.......#`
`#...S.......#`
`#############`
	`right()`
	`up()`
	`up()`
	`left()`
	`up()`

Diese 5 Schritte führen vom Start zum Ausgang. Natürlich ist auch jeder andere Weg, sofern er die maximale Anzahl an Schritte (2000000) nicht überschreitet, korrekt.
Die erste Zahl bedeutet, dass dieser Test Teil von Subtask 5 ist. 8 und 13 geben die Größe der Höhle an.

Implementierungsdetails

Für diese Aufgabe musst du nur die Funktion `findExit(subtask, fieldOfView)` in der Datei *cave.cpp* bzw. *cave.java* implementieren. Alles andere, also das Einlesen, die Ausgabe sowie das Aufrufen dieser Funktion, wird von einem (bereits fertigem) Grader (*grader.cpp* bzw. *grader.java*) übernommen. Diese beiden Dateien, sowie für C++ *cave.h*, kannst du vom Grading-System herunterladen (CAVE → Statement → Attachments). Einsenden musst du nur *cave.cpp* bzw. *cave.java*. Achte darauf, dass du *nicht* mit dem Standardoutput interagierst. Das kann zu einen falschen Ergebnis führen.

Subtasks

Allgemein gilt:

- $4 \leq n, m \leq 1000$
- Maximal $2 \cdot 10^6 = 2000000$ Schritte sind erlaubt
- Der Ausgang kann vom Startpunkt immer erreicht werden
- Der Rand des Rasters besteht aus Gestein

Subtask 1 (10 Punkte): Kein Gestein im inneren der Höhle. Der Ausgang befindet sich am Rand. Z. B.:

```
##############
#............#
#X...........#
#............#
#......S.....#
#............#
##############
```

Subtask 2 (12 Punkte): Kein Gestein im inneren der Höhle. Z. B.:

```
##############
#............#
#............#
#...X........#
#......S.....#
#............#
##############
```

Subtask 3 (19 Punkte): Alles Gestein im inneren der Höhle hängt (direkt oder indirekt) mit dem Rand zusammen. Der Ausgang befindet am Gestein anliegend. Z. B.:

```
##############
#.......#...#
#.......#####
#...##......#
#....#.S....#
#.X..#......#
##############
```

Subtask 4 (16 Punkte): Alles Gestein im inneren der Höhle hängt (direkt oder indirekt) mit dem Rand zusammen. Z. B.:

```
##############
#............#
#..X....#####
#....#..#...#
#....#.S#...#
#....#..#...#
##############
```

Subtask 5 (43 Punkte): Keine Einschränkungen. Z. B.:

```
##############
#............#
#..###....###
#..#X#......#
#.......#...#
#...#.......#
#...S.......#
##############
```

Beschränkungen

Zeitlimit: 2 s **Speicherlimit:** 256 MB