Dungeons & Dragons

Flodo ist ein kühner Abenteurer, der auf der Suche nach Ruhm und Reichtum in die Welt gezogen ist. Zuletzt erkundete der Halbling eine verlassene Burg, auf der Suche nach vergessenen Schätzen. Dabei er tappte auf eine Falltür und fiel in ein Labyrinth, das sich scheinbar unterhalb der Burg befindet. Um zurück an die Oberfläche zu gelangen muss Flodo nun den Ausgang finden.

Das Labyrinth besteht aus unzähligen Räumen, die untereinander mit Türen verbunden sind. Da das Labyrinth so unvorstellbar groß ist, und die Räume praktisch gleich aussehen, kann sich Flodo niemals merken, in welchen Räumen er bereits war. Glücklicherweise hat unser Held aber ein Stück Kreide dabei, mit dem er sich Notizen an die Wände der einzelnen Räume schreiben kann.

Flodo meint, dass es so kein Problem sein sollte den Ausgang zu finden. Leider muss er jedoch feststellen, dass feuerspeiende Drachen in verschiedenen Räumen des Labyrinths ihr Unwesen treiben. Da unser Held nicht besonders geübt im Kampf ist, töten ihn diese Drachen sofort, wenn er einen solchen Raum betritt. Er hat jedoch erneut Glück: Da er ja doch nur ein Charakter in einem Spiel ist, respawned er nach seinem Tod direkt wieder in jedem Raum, in den er ursprünglich hineinfiel. Seine Notizen bleiben dabei erhalten!

Kannst du Flodo helfen und einen Strategie entwickeln, mit der er in absehbarer Zeit den Ausgang findet?

Implementierungsdetails

Dies ist eine interaktive Aufgabe. Das bedeutet, dass dir bereits ein Programm vorgegeben ist, mit dem du interagieren musst. Dieses findest du am Server unter `Statement` → `Attachments` zum Download. Die Datei `grader.cpp` übernimmt das Einlesen. Du musst lediglich die Funktion `void findExit()` in `dnd.cpp` implementieren. Diese soll deine Strategie implementieren. Lade bei der Abgabe die Datei `dnd.cpp` hoch.

Folgende Funktionen werden dir vom Grader (`grader.cpp`) bereitgestellt:

- `int numOfDoors()` gibt die Anzahl an Türen des aktuellen Raums zurück. Diese sind beginnend von eins durchnummeriert.
- `int takeDoor(int door)` lässt Flodo durch Tür Nummer `door` schreiten. Dabei muss `door` zwischen 1 und `numOfDoors()` liegen. Der Rückgabewert dieser Funktion ist die Nummer der Tür, mit der man wieder zurückgehen kann.
  Sollte hinter der Tür ein Drache sein, so stirbst du. In diesem Fall wechselst du wieder zum ersten Raum und `findExit()` wird erneut aufgerufen.
  Falls du jenen Raum betrittst, der aus dem Labyrinth hinaus führt, wird das Programm automatisch beendet und du hast gewonnen.
- `void takeNote(std::string note)` schreibt die Notiz `note` an die Wand des aktuellen Raums. Falls in diesem Raum bereits eine Notiz steht, so wird diese überschrieben. `note` darf maximal 42 Zeichen lang sein.
- `std::string readNote()` gibt die Notiz zurück, die sich an der Wand des aktuellen

Raums befindet. Fall noch keine Notiz vorhanden ist gibt `readNote()` einen leeren String zurück.

Deine Strategie darf insgesamt *maximal 4000000 Aufrufe* von `takeDoor` machen. Anderenfalls wird dein Programm beendet und als falsch gewertet, da Flodo nach so vielen Schritt endgültig am Ende ist :)

Achtung: Es ist *nicht* erlaubt, sich Informationen „über den Tod hinweg" zu merken. Deine Strategie darf einzig und allein auf den Notizen in den Räumen basieren.

Einschränkungen

- Es gibt genau einen Raum, in dem sich der Ausgang befindet. Dieser ist vom Start erreichbar, ohne dabei auf einen Drachen zu treffen.
- Weder im Startraum, noch beim Ausgang, befindet sich ein Drache.
- Es gibt maximal 1000 verschiedene Räume und Türen.

Eingabe

Die Eingabe wird bereits vom Grader übernommen. Dieser liest das Labyrinth in folgendem Format ein: Die erste Zeile enthält die Anzahl an Räume n, sowieso die Anzahl an Türen m. Die folgenden m Zeilen enthalten jeweils zwei Zahlen a, b ($1 \leq a, b \leq n$, $a \neq b$), die beschreiben, dass es eine Tür zwischen Raum a und Raum b gibt. Die nächste Zeile enthält eine Zahl d, die Anzahl an Drachen. In der folgenden Zeile befinden sich d Raumnummern, die angeben wo sich die Drachen befinden.

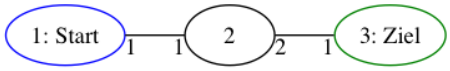
Es wird davon ausgegangen, dass du in Raum 1 startest, und der Ausgang sich in Raum n befindet.

Beispiele

Eingabe	Beispielaufrufe

`3 2`
`1 2`
`2 3`
`0`
	`findExit()` wird vom Grader aufgerufen. Wir befinden uns in Raum 1.
	`numOfDoors()` gibt 1 zurück.
	Mit `takeDoor(1)` gehen wir durch Tür 1 und sind somit in Raum 2.
	`takeDoor(1)` gibt 1 zurück, da wir mit Tür Nr. 1 von Raum 2 in Raum 1 zurückkommen.
	`numOfDoors()` gibt 2 zurück.
	`takeDoor(2)` geht in den dritten Raum, findet damit den Ausgang und beendet das Programm.

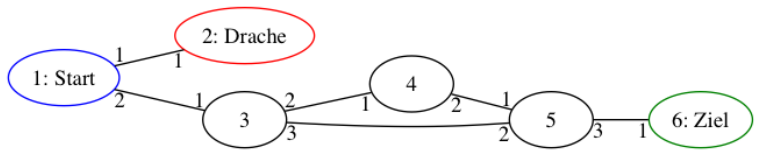
In diesem Fall gibt es keine Drachen. Man startet in Raum 1, der Ausgang befindet sich in Raum 3. Das Labyrinth sieht so aus:



Eingabe	Beispielaufrufe

`6 6`
`1 2`
`1 3`
`3 4`
`4 5`
`5 3`
`5 6`
`1`
`2`
	`findExit()` wird aufgerufen.
	`numOfDoors()` gibt 2 zurück.
	`readNote()` gibt einen Leerstring zurück.
	Flodo schreibt mit `takeNote("AOI")` "AOI" an die Wand.
	`takeDoor(1)` führt zu Raum 2, wo sich ein Drache befindet. Du stirbst.
	`findExit()` wird erneut aufgerufen (respawn). Du befindest dich in Raum 1.
	`readNote()` liefert "AOI".
	`takeDoor(2)` führt zu Raum 3 und gibt 1 zurück.
	`takeDoor(2)` führt zu Raum 4.
	`takeDoor(2)` führt zu Raum 5.
	`takeDoor(3)` führt zum Ausgang (Raum 6).

Bei diesem Beispiel gibt es einen Drachen in Raum 2. Das Labyrinth sieht so aus:



Subtasks

Subtask 1 (13 Punkte): Die Räume bilden einen Gang. Am einen Ende des Gangs befindet sich ein Drache, am anderen Ende der Ausgang. Der Eingangsraum ist irgendwo dazwischen. Mit Ausnahme des Ausganges und des Raumes mit dem Drachen haben alle Räume also genau zwei Türen.

Subtask 2 (20 Punkte): Keine Drachen, und jeder Raum ist über genau einen Weg erreichbar. Es gibt keine Wege, die im Kreis führen.

Subtask 3 (23 Punkte): Wie Subtask 2, aber mit Drachen

Subtask 4 (19 Punkte): Keine Drachen

Subtask 5 (25 Punkte): Keine Einschränkungen

Limits

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