Mazes (mazes)

Hampton Court Palace (in Richmond, südwestlich von London) ist berühmt für sein Labyrinth, das in den 1690er Jahren von König Wilhelm III. gepflanzt wurde. Da der Palast jetzt als Touristenattraktion für die Öffentlichkeit zugänglich ist, haben die königlichen Behörden beschlossen, das Labyrinth durch eine aktualisierte Version zu ersetzen. Sie haben die Anzahl der Touristen, K, geschätzt, die sie während der Saison erwarten, und möchten, dass das neue Labyrinth genau K mögliche Routen hat, damit jeder Besucher ein einzigartiges Erlebnis haben kann.

Das Labyrinth ist ein Raster mit N Zeilen und M Spalten, wobei jede Zelle entweder leer sein oder eine Hecke enthalten kann. Das Labyrinth ist von einem Zaun umgeben, es gibt einen einzigen Eingang und einen einzigen Ausgang. Der Eingang befindet sich in der Zelle oben links, und der Ausgang befindet sich in der Zelle unten rechts. Der Besucher muss das Labyrinth nur mit Bewegungen nach rechts und nach unten durchqueren. Aufgrund von Platzbeschränkungen kann das Labyrinth **nicht mehr als 200 Zeilen** und **nicht mehr als 200 Spalten** haben.

Deine Aufgabe ist es, ein Labyrinth zu entwerfen, das genau K Routen vom Eingang zum Ausgang hat, wobei nur Bewegungen nach rechts und unten erlaubt sind.

Implementierung

Du musst eine einzelne `.cpp`-Quelldatei einreichen.

> ☞ Unter den Anhängen dieser Aufgabe findest Du eine Vorlage `mazes.cpp` mit einer Beispielimplementierung.

Du musst die folgende Funktion implementieren:

C++
```
vector<vector<char>> solve(long long K);
```

- Der ganzzahlige Wert K stellt die gewünschte Anzahl der Routen durch das Labyrinth dar.
- Die Funktion sollte einen zweidimensionalen Vektor von Zeichen zurückgeben, der das Labyrinth darstellt.
- Das zurückgegebene Labyrinth sollte N Zeilen und M Spalten haben, wobei diese frei gewählt werden können, solange $N, M \leq 200$ sind.
- Jede Zelle des Labyrinths sollte entweder ein `.` (Punkt) für eine leere Zelle oder ein `#` (Raute) für eine Zelle mit einer Hecke sein.
- Der Eingang befindet sich in der Zelle $(0, 0)$ und der Ausgang in der Zelle $(N - 1, M - 1)$.
- Das Labyrinth sollte genau K verschiedene Wege haben, um es vom Eingang zum Ausgang nur durch Bewegungen nach rechts und unten zu durchqueren.

Der Grader wird die Funktion `solve` aufrufen und deren Rückgabe ins Ausgabefile ausgeben.

Beispielgrader

Das Verzeichnis der Aufgabe enthält eine vereinfachte Version des Jury-Graders, die Du verwenden kannst, um Deine Lösung lokal zu testen. Der vereinfachte Grader liest die Eingabedaten aus `stdin`, ruft die Funktion auf, die du implementieren musst, und schreibt schliesslich die Ausgabe in `stdout`.

Die Eingabe besteht aus 1 Zeile, die eine einzelne ganze Zahl K enthält.

Die Ausgabe besteht aus mehreren Zeilen:

- Die erste Zeile enthält zwei ganze Zahlen N und M ($N, M \leq 200$), die die Anzahl der Zeilen bzw. Spalten des Labyrinths darstellen.
- Die nächsten N Zeilen enthalten jeweils M Zeichen, die das Labyrinth darstellen.
- Jedes Zeichen ist entweder ein `.` (Punkt) für eine leere Zelle oder ein `#` (Raute) für eine Zelle mit einer Hecke.

Einschränkungen

$1 \leq K \leq 10^{18}$.

Punktevergabe

Dein Programm wird anhand einer Reihe von Testfällen getestet, die nach Teilaufgaben gruppiert sind. Um die Punktzahl zu erhalten, die einer Teilaufgabe zugeordnet ist, musst Du alle darin enthaltenen Testfälle korrekt lösen.

- **Teilaufgabe 1** [**0 Punkte**]: Beispiel-Testfälle.
- **Teilaufgabe 2** [**9 Punkte**]: $K \leq 10$.
- **Teilaufgabe 3** [**26 Punkte**]: $K \leq 99$.
- **Teilaufgabe 4** [**26 Punkte**]: K ist eine Zweierpotenz.
- **Teilaufgabe 5** [**39 Punkte**]: Keine zusätzlichen Einschränkungen.

Beispiele

stdin	stdout
1	2 2 .# ..
3	8 8 ....######. ..#...#. .#..#.#. ...##... .#...##. .###.##. ....

Erklärung

Im **ersten Beispiel** gibt es offensichtlich nur einen Weg durch das Labyrinth.

Im **zweiten Beispiel** gibt es drei mögliche Wege durch das Labyrinth. Die Startzelle ist grün schattiert, die Endzelle rot, und die Zellen, die die Wege bilden, sind blau schattiert.

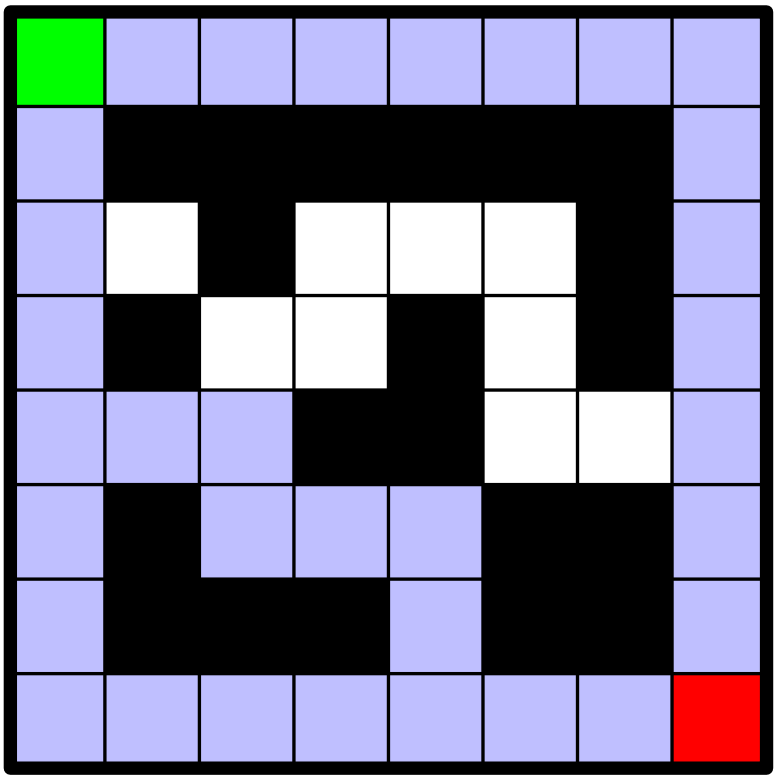

Abbildung 1: Zweites Beispiel