

Mazes (mazes)


Hampton Court Palace (in Richmond, just southwest of London) is famous for its maze, which was planted by King William III in the 1690s. Since the palace is now open to the public as a tourist attraction, the royal authorities have decided to replace the maze with an updated version. They have estimated the number of tourists K they expect to visit during the season, and they would like the new maze to have exactly K possible routes through it, allowing each visitor to have a unique experience.

The maze is a grid of N rows and M columns, where each cell can be either empty or contain a hedge. The maze is surrounded by a fence, and there is a single entrance and a single exit. The entrance is in the top-left cell, and the exit is in the bottom-right cell. The visitor must complete the maze by making only right and down moves. Due to space constraints, the maze can have **no more than 200 rows** and **no more than 200 columns**.

Your task is to design a maze with exactly K routes from the entrance to the exit involving only right and down moves.

Implementation

You will have to submit a single `.cpp` source file.

 Among this task's attachments you will find a template `mazes.cpp` with a sample implementation.

You have to implement the following function:

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

- Integer K represents the desired number of routes through the maze.
- The function should return a two dimensional vector of characters, representing the maze.
- The returned maze should have N rows and M columns, for some $N, M \leq 200$.
- Each cell of the maze should be either a `.` (dot) for an empty cell or a `#` (hash) for a cell containing a hedge.
- The entrance is in the cell $(0, 0)$ and the exit is in the cell $(N - 1, M - 1)$.
- The maze should have exactly K different ways to go through it from the entrance to the exit by making only right and down moves.

The grader will call the function `solve` and will print its return value to the output file.

Sample Grader

The task's directory contains a simplified version of the jury grader, which you can use to test your solution locally. The simplified grader reads the input data from `stdin`, calls the function that you must implement, and finally writes the output to `stdout`.

The input is made up of 1 line, containing a single integer K .

The output is made up of several lines, containing:

- The first line contains two integers N and M ($N, M \leq 200$), the number of rows and columns of the maze respectively.
- The next N lines contain M characters each, representing the maze.
- Each character is either a `.` (dot) for an empty cell or a `#` (hash) for a cell containing a hedge.

Constraints

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

Scoring

Your program will be tested on a set of test cases grouped by subtask. To obtain the score associated to a subtask, you need to correctly solve all the test cases it contains.

- **Subtask 1** [0 points]: Sample test cases.
- **Subtask 2** [9 points]: $K \leq 10$.
- **Subtask 3** [26 points]: $K \leq 99$.
- **Subtask 4** [26 points]: K is a power of two.
- **Subtask 5** [39 points]: No additional constraints.

Examples

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

Explanation

In the **first sample case**, there is obviously just one way to go through the maze.

In the **second sample case**, there are three possible ways to go through the maze. The start cell is shaded in green, the end cell in red, and the cells that form the paths are shaded in blue.

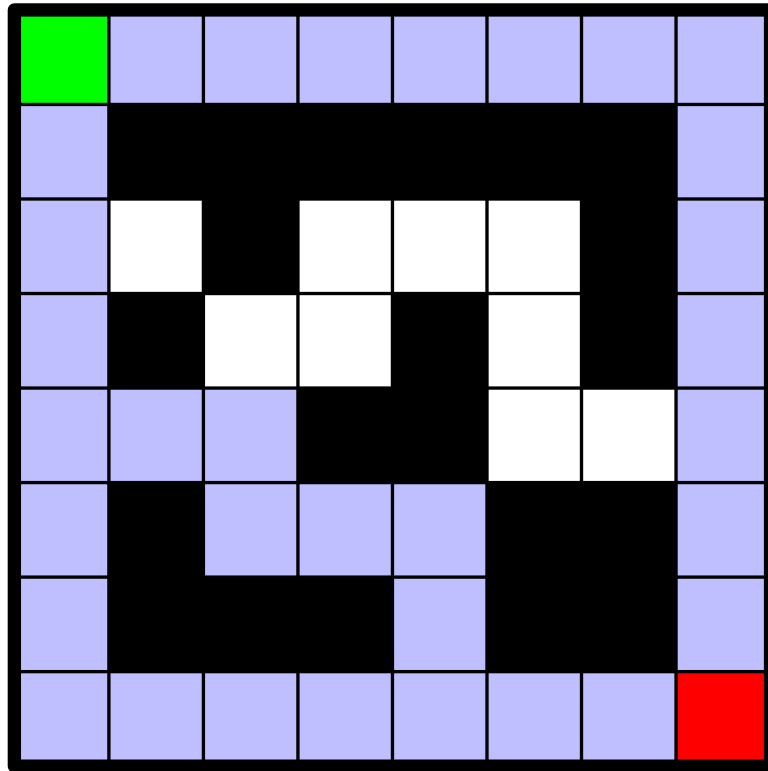


Figure 1: Second sample case