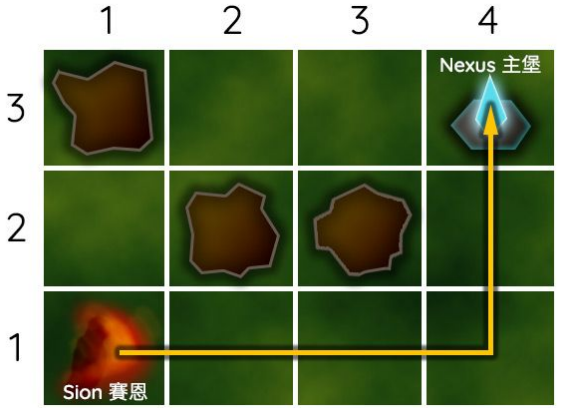
S191 - Unstoppable Onslaught

Percy Wong {percywtc}

Background

Author: percywtc

Setter: percywtc, microtony



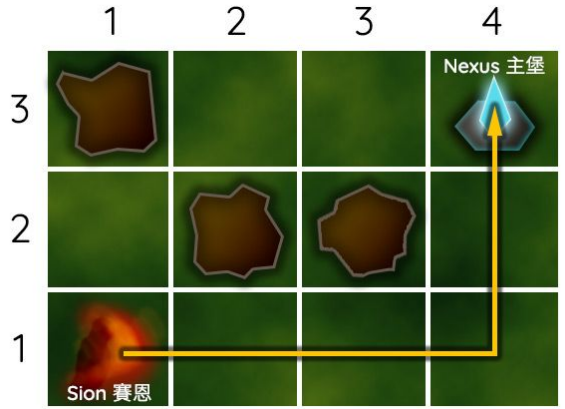
The Problem

Given an **R × C** grid with obstacles

Sion, initially at **(1, 1)**, can only:

- goes upwards by 1
- goes rightwards by 1

You have to block Sion from going to **(R, C)**



SUBTASKS

For all cases:

$1 \le R, C \le 2000$

$R \times C > 2$

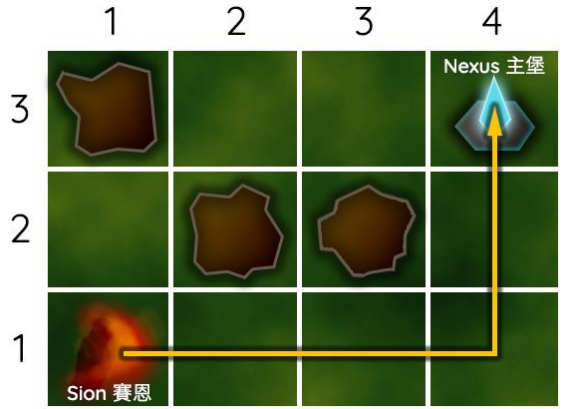
$0 \le N \le \min(500000, R \times C - 2)$

	Points	Constraints
1	9	$N = 0$
2	12	$R = C = 2$
3	20	$1 \le R, C \le 50$
4	25	$1 \le R, C \le 500$
5	34	No additional constraints

Statistics

0points	13 + 0 + 1 + 0 = 14
12points	8 + 0 + 0 + 0 = 8
21points	22 + 15 + 9 + 3 = 49
41points	0 + 0 + 4 + 1 = 5
66points	0 + 1 + 0 + 0 = 1
100points	0 + 0 + 1 + 4 = 5

First solved by **dbsgame** at **1h 1m 17s**



SUBTASKS

For all cases:

$1 \le R, C \le 2000$

$R \times C > 2$

$0 \le N \le \min(500000, R \times C - 2)$

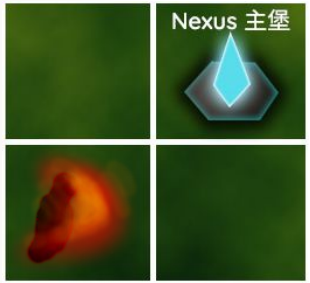
	Points	Constraints
1	9	$N = 0$
2	12	$R = C = 2$
3	20	$1 \le R, C \le 50$
4	25	$1 \le R, C \le 500$
5	34	No additional constraints

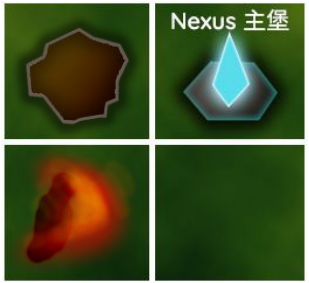
Subtask 2

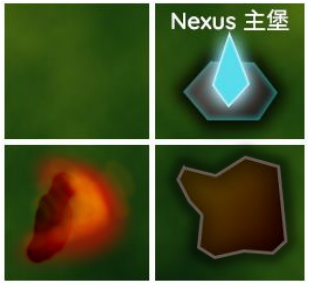
It is given that **R = C = 2**

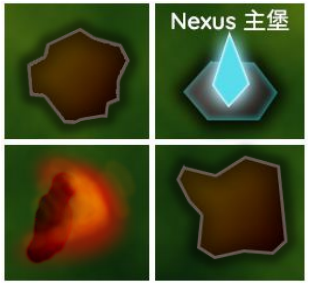
- there are only 4 different cases
- we must place obstacles on the two corners

INPUT









OUTPUT

```
2
1 2
2 1
```

```
1
1 2
```

```
1
2 1
```

```
0
```

Subtask 1

It is given that **N = 0**, meaning that there are no obstacles initially

Key observation

- The answer must be AT MOST 2
- we can always block Sion by placing 2 obstacles around him (or the Nexus)

Subtask 1

It is given that **N = 0**, meaning that there are no obstacles initially

Key observation

- the answer must be AT MOST 2
- we can always block Sion by placing 2 obstacles around him (or the Nexus)

When **N = 0**,

- we should always place 2 obstacles...
- except when **R = 1** or **C = 1**

Subtask 3

With the observation that the answer must not exceed 2, meaning that...

- we can just check if the answer could be 0,
- if not 0... could the answer be 1
- otherwise, the answer must be 2

Subtask 3

With the observation that the answer must not exceed 2, meaning that...

- we can just **check if the answer could be 0**,
- if not 0... could the answer be 1
- otherwise, the answer must be 2

this part is easy, we can just simulate if **(R, C)** is reachable

- breadth first search (BFS)
- nested for-loop from the bottom row
- ...

$O(RC)$

Subtask 3

With the observation that the answer must not exceed 2, meaning that...

- we can just check if the answer could be 0,
- if not 0... could the answer be 1
- otherwise, the answer must be 2

this part is easy as well, we can just output:

```
2
1 2
2 1
```

Subtask 3

With the observation that the answer must not exceed 2, meaning that...

- we can just check if the answer could be 0,
- if not 0... **could the answer be 1**
- otherwise, the answer must be 2

so, we can try to exhaust every non-occupied cell

- try to place an obstacle on that cell
- re-run the simulation (BFS, nested for-loop, ...)

Subtask 3

With the observation that the answer must not exceed 2, meaning that...

- we can just check if the answer could be 0,
- if not 0... **could the answer be 1**
- otherwise, the answer must be 2

so, we can try to exhaust every non-occupied cell ……$O(RC)$

- try to place an obstacle on that cell ……$O(1)$
- re-run the simulation (BFS, nested for-loop, ...) ……$O(RC)$

Overall Time Complexity: $O(R^2C^2)$

Full Solution

- if not 0... **could the answer be 1**

there are several ways to solve the problem...

Full Solution - 1st idea

First run BFS (or something similar) to find the reachability of each cell from **(1, 1)**

We can perform the same thing starting from **(R, C)**

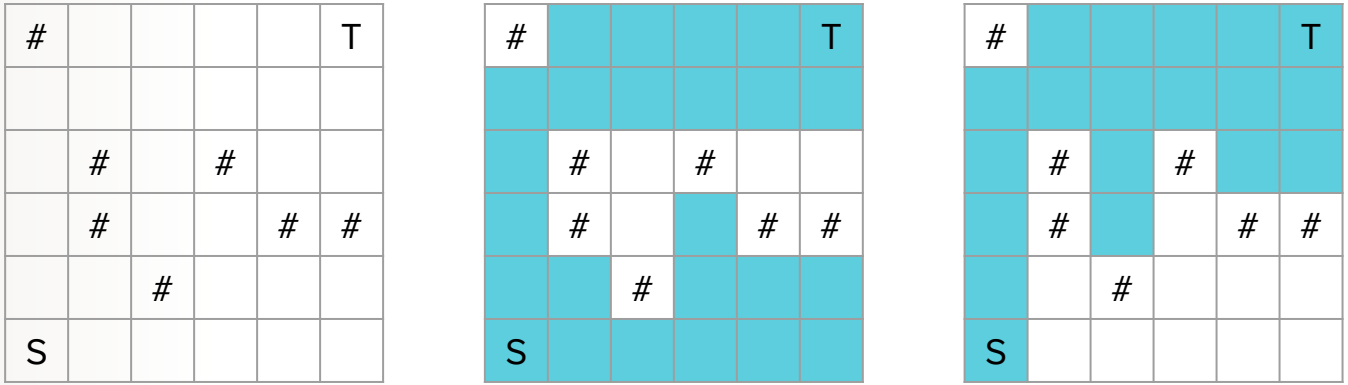
- the directions are reversed too... so they are “downwards” and “leftwards”
- this reachability of **(R, C)** to **(i, j)** is quivalent to
  - reachability of **(i, j)** to **(R, C)** by moving “upwards” and “rightwards”

Full Solution - 1st idea

First run BFS (or something similar) to find the reachability of each cell from **(1, 1)**

We can perform the same thing starting from **(R, C)**

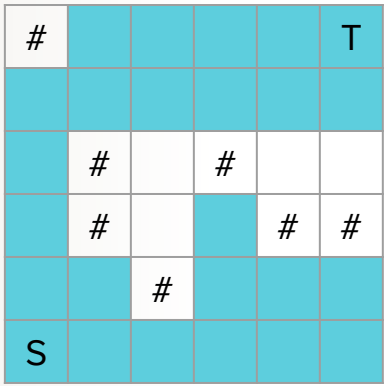
- the directions are reversed too... so they are “downwards” and “leftwards”
- this reachability of **(R, C)** to **(i, j)** is quivalent to
  - reachability of **(i, j)** to **(R, C)** by moving “upwards” and “rightwards”

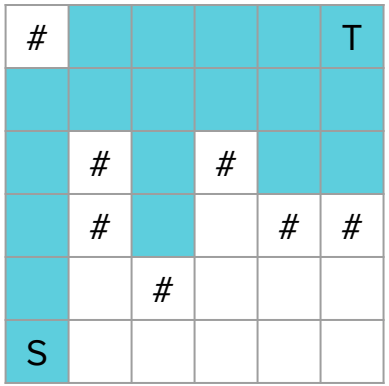


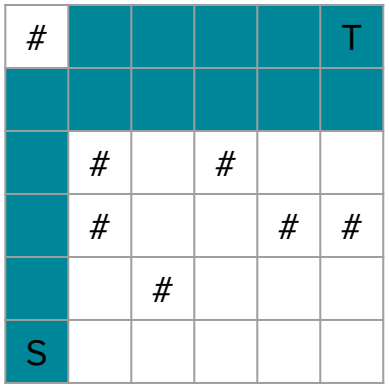
Full Solution - 1st idea

We can do **AND** operation on the two “reachable” arrays

- the resultant cell is **TRUE** only if
  - it’s reachable from **(1, 1)**
  - it can reach **(R, C)**
  - we can pass through this cell while going from **(1, 1)** to **(R, C)**
- when is it a MUST?

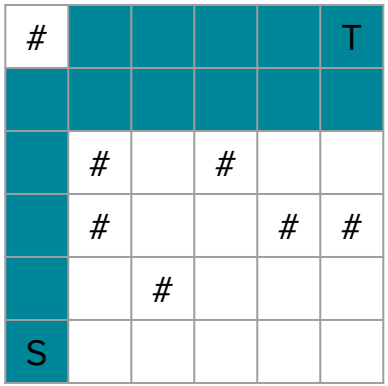






Full Solution - 1st idea

- the resultant cell is **TRUE** only if
  - it's reachable from **(1, 1)**
  - it can reach **(R, C)**
  - we can pass through this cell while going from **(1, 1)** to **(R, C)**
- when is it a MUST?
  - if at some diagonal ↘, there is exactly <u>one **TRUE** cell</u> (except S and T)
  - we must have to pass through this cell
- why?
  - as on the path from (1, 1) to (N, M)
  - we reach exactly one cell on each diagonal ↘



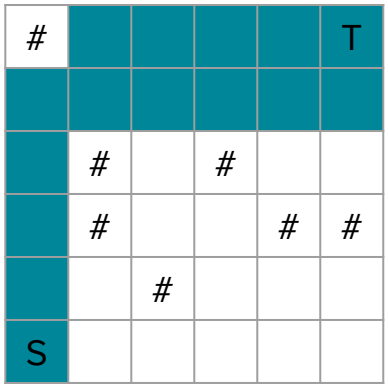
Full Solution - 1st idea

- if at some diagonal ↘, there is exactly <u>one **TRUE** cell</u> (except S and T)
- we must have to pass through this cell

so we can just block this cell :)

what if there does not exist such cell? must the answer be 2?

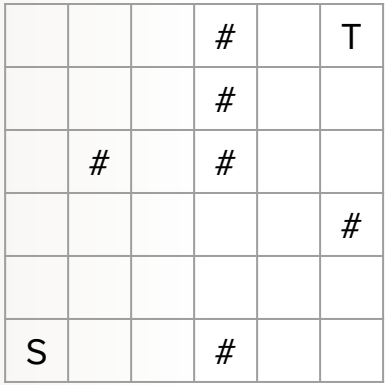
- YES

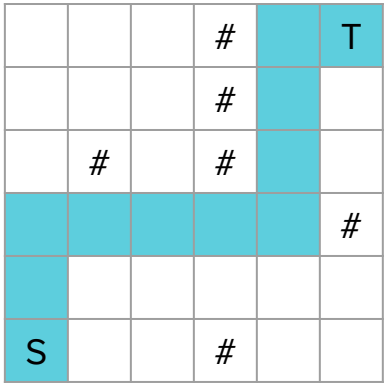


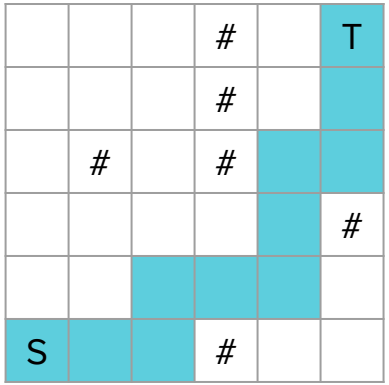
Full Solution - 2nd idea

Find the uppest and lowest possible path that can reach **(R, C)** from **(1, 1)**

- how to find these paths?
- we still have to build the “reachable” arrays
- greedily choose the next step







Full Solution - 2nd idea

Find the uppest and lowest possible path that can reach **(R, C)** from **(1, 1)**

if they intersect at some cell (except S and T)

- that's a MUST cell
- that's the answer too