Project description

You are a zookeeper in the reptile house. One of your rare South Pacific Tufted Wizzo lizards (*Tufticus Wizzocus*) has just had several babies. Your job is to find a place to put each baby lizard in a nursery.

However, there is a catch, the baby lizards have very long tongues. A baby lizard can shoot out its tongue and eat any other baby lizard before you have time to save it. As such, you want to make sure that no baby lizard can eat another baby lizard in the nursery (burp).

For each baby lizard, you can place them in one spot on a grid. From there, they can shoot out their tongue up, down, left, right and diagonally as well. Their tongues are very long and can reach to the edge of the nursery from any location.

Figure 1 shows in what ways a baby lizard can eat another.

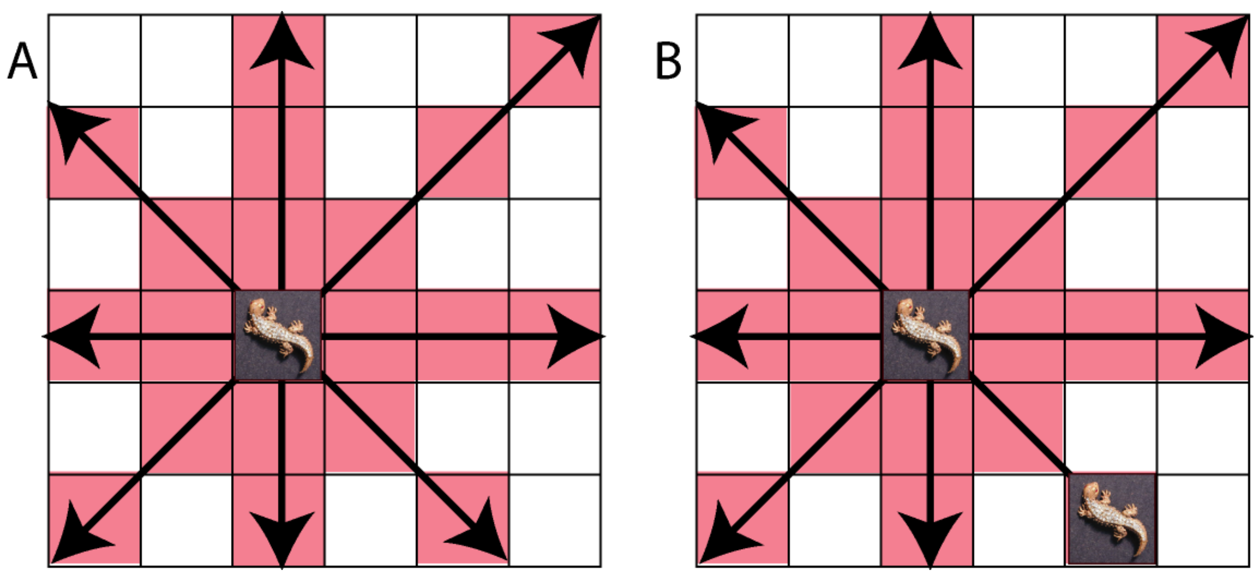

Figure 1 (A) the baby lizard can attack any other lizard in a red square. Thus it can be seen that a baby lizard can eat another lizard to its top, bottom, left right or diagonal. (B) In this example setup, both lizards can eat each other. Your algorithm will try to avoid this.

In addition to baby lizards, your nursery may have some trees planted in it. Your lizards cannot shoot their tongues through the trees nor can you move a lizard into the same place as a tree. As such, a tree will block any lizard from eating another lizard if it is in the path. Additionally, the tree will block you from moving the lizard to that location.

Figure 2 shows some different valid arrangements of lizards.

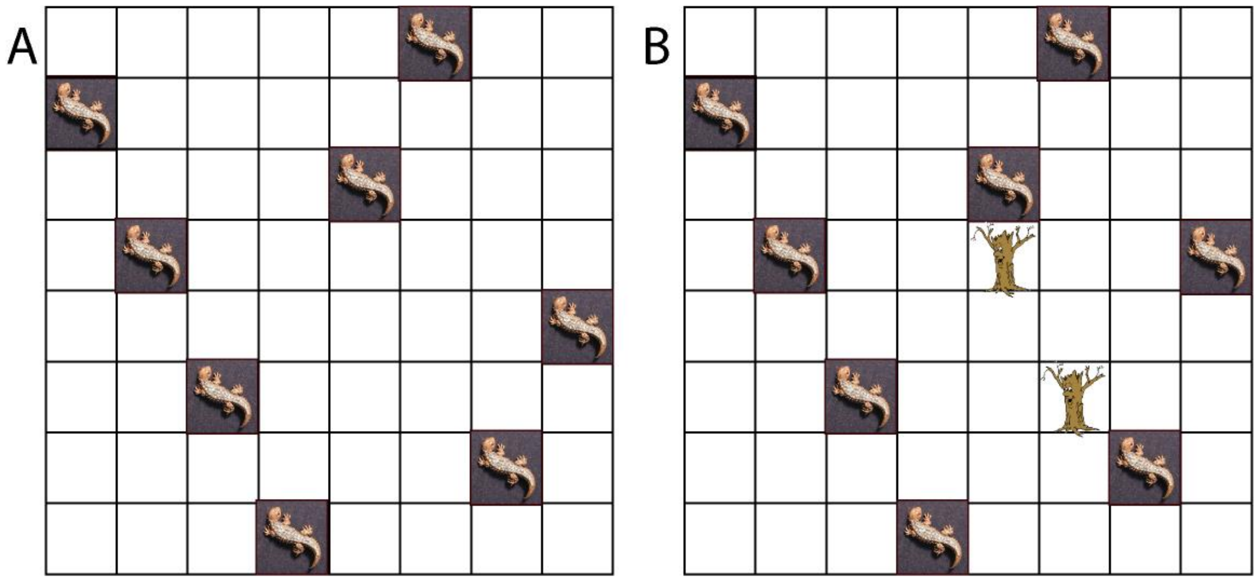

Figure 2 Both nurseries have valid arrangements of baby lizards such that they cannot eat one another. (A) with no trees, no lizard is in a position to eat another lizard. (B) Two trees are introduced such that the lizard in the last column cannot eat the lizard in the second or fourth column.

You will write a program that will take an input file that has an arrangement of trees and will output a new arrangement of lizards (and trees; as already mentioned, you can't move the trees) such that no baby lizard can eat another one. You will be required to create a program that finds the solution.

To find the solution you will use the following algorithms:
- Breadth-first search (BFS)
- Depth-first search (DFS)
- Simulated annealing (SA).

Input: The file `input.txt` in the current directory of your program will be formatted as follows:

First line: instruction of which algorithm to use: BFS, DFS or SA

Second line: strictly positive 32-bit integer n, the width and height of the square nursery

Third line: strictly positive 32-bit integer p, the number of baby lizards

Next n lines: the n x n nursery, one file line per nursery row (to show you where the trees are). It will have a 0 where there is nothing, and a 2 where there is a tree.
So for instance, an input file arranged like figure 2B (but with no lizards yet) and requesting you to use the DFS algorithm to place 8 lizards in the 8x8 nursery would look like:

DFS
8
8
00000000
00000000
00000000
00002000
00000000
00000200
00000000
00000000

Output: The file `output.txt` which your program creates in the current directory should be formatted as follows:

First line: `OK` or `FAIL`, indicating whether a solution was found or not.

If FAIL, any following lines are ignored.

Next n lines: the n x n nursery, one line in the file per nursery row, including the baby lizards and trees. It will have a 0 where there is nothing, a 1 where you placed a baby lizard, and a 2 where there is a tree.

For example, a correct output.txt for the above sample input.txt (and matching Figure 2B) is:

OK
00000100
10000000
00001000
01002001
00000000
00100200
00000010
00010000

Example 1:

For this input.txt:

```
BFS
2
2
00
00
```

one possible correct output.txt is:

```
FAIL
```

Example 2:

For this input.txt:

```
DFS
4
4
0000
0000
```

```
0000
0000
```

one possible correct output.txt is:

```
OK
0010
1000
0001
0100
```

Example 3:

For this input.txt (same layout of trees as in Figure 2B, but we now want to place 9 lizards in this 8x8 nursery with 2 trees):

```
SA
8
9
00000000
00000000
00000000
00002000
00000000
00000200
00000000
00000000
```

one possible correct output.txt is:

```
OK
00000100
10000000
00001000
01002001
00001000
00100200
00000010
00010000
```