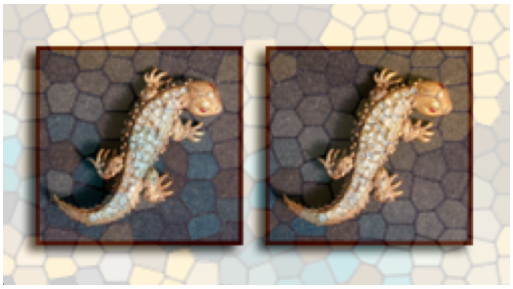
CSCI-561 - Fall 2017 - Foundations of Artificial Intelligence
Homework 1

Due September 20, 2017 23:59:59



Guidelines

This is a programming assignment. You will be provided sample inputs and outputs (see below). Please understand that the goal of the samples is to check that you can correctly parse the problem definitions, and generate a correctly formatted output. The samples are very simple and it should not be assumed that if your program works on the samples it will work on all test cases. There will be more complex test cases and it is your task to make sure that your program will work correctly on any valid input. You are encouraged to try your own test cases to check how your program would behave in some complex special case that you might think of. Since **each homework is checked via an automated A.I. script**, your output should match the specified format *exactly*. Failure to do so will most certainly cost some points. The output format is simple and examples are provided. You should upload and test your code on vocareum.com, and you will submit it there. You may use any of the programming languages provided by vocareum.com.

Grading

Your code will be tested as follows: Your program should not require any command-line argument. It should read a text file called "input.txt" in the current directory that contains a problem definition. It should write a file "output.txt" with your solution to the same current directory. Format for input.txt and output.txt is specified below. End-of-line character is LF (since vocareum is a Unix system and follows the Unix convention).

The grading A.I. script will, 50 times:

- Create an input.txt file, delete any old output.txt file.
- Run your code.
- Check correctness of your program's output.txt file.
- If your outputs for all 50 test cases are correct, you get 100 points.
- If one or more test case fails, you get 50 – N points where N is the number of failed test cases.

Note that if your code does not compile, or somehow fails to load and parse input.txt, or writes an incorrectly formatted output.txt, or no output.txt at all, or OuTpUt.TxT, **you will get zero points**. Anything you write to stdout or stderr will be ignored and is ok to leave in the code you submit (but it will likely slow you down). Please test your program with the provided sample files to avoid this.

Academic Honesty and Integrity

All homework material is checked vigorously for dishonesty using several methods. All detected violations of academic honesty are forwarded to the Office of Student Judicial Affairs. To be safe you are urged to err on the side of caution. Do not copy work from another student or off the web. Keep in mind that sanctions for dishonesty are reflected in *your permanent record* and can negatively impact your future success. As a general guide:

> **Do not copy** code or written material from another student. Even single lines of code should not be copied.
> **Do not collaborate** on this assignment. The assignment is to be solved individually.
> **Do not copy** code off the web. This is easier to detect than you may think.
> **Do not share** any custom test cases you may create to check your program's behavior in more complex scenarios than the simplistic ones considered below.
> **Do not copy** code from past students. We keep copies of past work to check for this.
>
> **Do** ask the professor or TA if you are unsure about whether certain actions constitute dishonesty. It is better to be safe than sorry.

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.

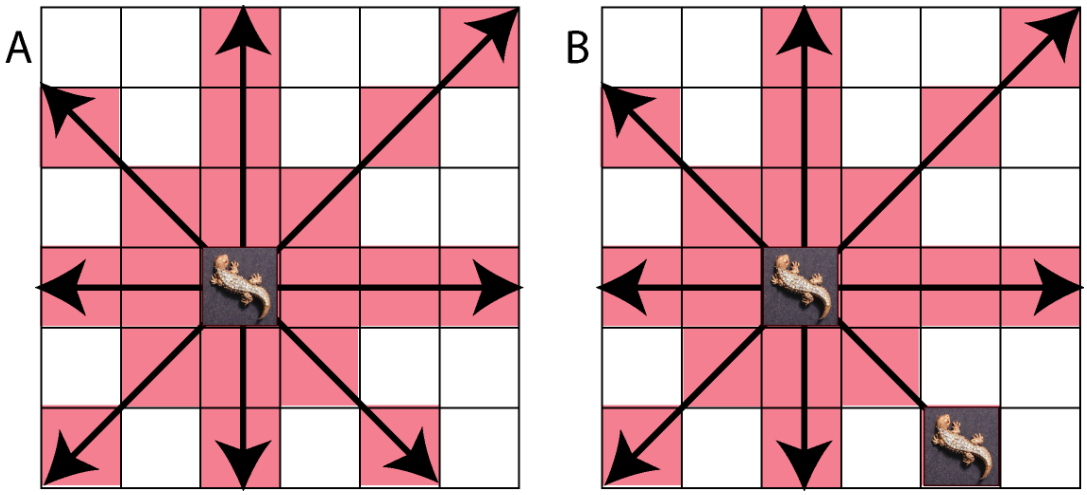


Figure 1 (A) the baby lizard can attack any other lizard in a red square. Thus it can be seen that a baby lizard can eat another lizard to its top, bottom, left right or diagonal. (B) In this example setup, both lizards can eat each other. Your algorithm will try to avoid this.

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.

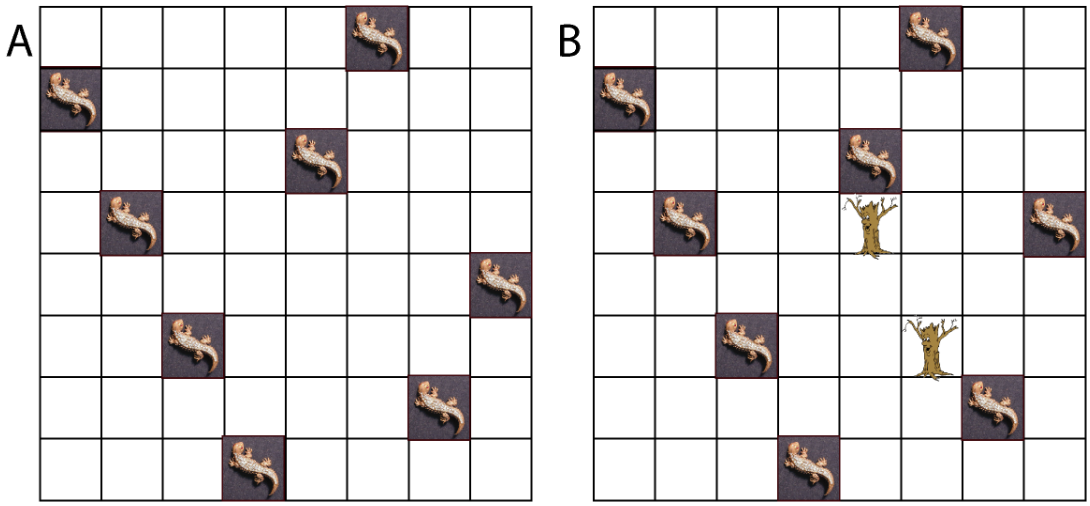


Figure 2 Both nurseries have valid arrangements of baby lizards such that they cannot eat one another. (A) with no trees, no lizard is in a position to eat another lizard. (B) Two trees are introduced such that the lizard in the last column cannot eat the lizard in the second or fourth column.

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

Notes and hints:

- Please name your program "**homework.xxx**" where 'xxx' is the extension for the programming language you choose ("py" for python, "cpp" for C++, and "java" for Java). If you are using C++11, then the name of your file should be "homework11.cpp" and if you are using python3 then the name of your file should be "homework3.py".
- n (width and height of nursery) and p (number of lizards) will be 32-bit integers with values 1 or more, and they may take different values ($n \neq p$) or not.
- If your output.txt does not contain exactly p lizards, that test case will fail irrespective of whether your output.txt says "OK" on the first line.
- If your output.txt is such that some lizards can eat each other, that test case will fail irrespective of whether your output.txt says "OK" on the first line.
- We recommend that you first start with the BFS and DFS implementations, and that you first focus on the simpler situation where there are no trees. You may want to consider the following questions:
    - Is the problem solvable for p>n when there are no trees?
    - Is the problem solvable for p<=n when there are no trees?
    - In particular, what happens for low values of n?
    - What is a good representation of states and operators? (see next bullet for further hint that may help you with this one).
    - How about a simple strategy to get started in which an operator at depth d in the search tree places a lizard in one of the empty cells in column d+1 of the nursery (depth starts at 0 for the root, columns are numbered starting with 1 for the leftmost)? Do you think that would work when there are no trees?
    - How would you modify the operators when there are trees, using the definition of the problem given to you above?
- For simulated annealing:
    - How about starting with an initial random set of lizard locations (which does not conflict with any trees)?
    - You will need to play with the temperature schedule and see what seems to work well on test cases that you create yourself.
- Likely (but no guarantee) we will create 15 DFS, 15 BFS, and 20 SA text cases.
- Your program will be killed after 5 minutes of execution on a given text case, to allow us to grade the whole class in a reasonable amount of time.

Extra credit:

Among the programs that get 100% correct on all 50 test cases,

- the fastest 10% on the DFS test cases will get an extra 5% credit on this homework
- the fastest 10% on the BFS test cases will get an extra 5% credit on this homework
- the fastest 10% on the SA test cases will get an extra 5% credit on this homework

So, if you are in the top 10% speed on all 3 algorithms (and 100% correct), there is opportunity for up to 15% extra points on this homework (score of 115 for this homework).

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
```