Mandatory Exercise 02105

The mandatory exercise is part of the overall assessment. The exercise is handed in using Gradescope and CodeJudge. See the course webpage for how to sign up for Gradescope. The exercise is done in small groups of 3, 2, or 1 students.

How To Prepare the Hand-in

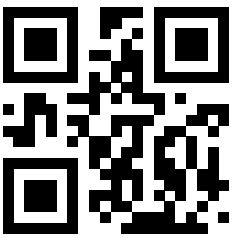
- For the written exercise first prepare a single `pdf`-file with your solutions to all the written exercises (those not marked with [†]). Write your solutions to the exercises directly in the marked areas of the template (this file). Hence, your final submission should be a single `pdf`-file identical to the template (this file) but with your information and solutions filled in the designated areas. Do not change or modify the template in any other way. To produce the file we recommend that you print out the document, fill it in by hand, and scan it. Make sure that your handwriting and scan is of good quality and easily readable. Alternatively, you may use a pdf-editor to fill in the template directly.
- Then, submit your solution in Gradescope. To do so first upload the submission and then invite the other group members within Gradescope to join the group. Make sure that the other members accept the invitation.
- For the implementation exercises, you submit your code to the programming exercises (those marked with [†]) through CodeJudge. Make sure to create the group.

Written Exercises For the written exercises note the following:

- Asymptotic bounds should be as tight as possible. Unless otherwise specified, the base of all logarithms is 2 and $\log^k n$ means $(\log n)^k$.
- "Give an algorithm" means that you should describe your solution in a short, precise, and unambiguous manner and analyze the complexity of your solution. Unless specified otherwise, the description should be in natural language and not pseudocode. The analysis should explain how you derived the complexity bound.
- "Argue correctness of your algorithm" means that you should provide a short argument for correctness of your algorithm.
- "Analyze the running time of your algorithm in the relevant parameters (parameters x, y, ..) of the problem" means that you should analyze the running time using the explicitly stated parameters of the problem (parameters x, y, ..).

Collaboration Policy The mandatory exercise is subject to the following collaboration policy. The exercise should be done in groups of 3, 2, or 1 students. It is not allowed to collaborate with other groups on the exercises, except for discussing the text of the exercise with teachers and fellow students enrolled on the course in the same semester. Under no circumstances is it allowed to exchange, hand-over or in any other way communicate solutions or parts of solutions to the exercises to other people. It is not allowed to use solutions from previous years, solutions from similar courses, or solutions found on the internet or elsewhere.

Deadlines The deadline for handing in the exercise is 18.04.2021, at 20:00 on Gradescope and CodeJudge.



Pacman

In your new job as chief map designer for a new version of the classic Pacman game you need to be able to efficiently evaluate different map designs from your developers. Luckily, all map designs are constructed in a simple and well-defined format, so you can automate your evaluation of them.

The format is illustrated in Figure 1. We define a *map* as a quadratic grid of size $N \times N$. In Figure 1 N is 8, 8, and 7, respectively. Each map consists of *walls* (marked with `#`), one or more *ghosts* (marked with `G`), and exactly one *Pacman* (marked with `P`) (except in exercise 4). Pacman and the ghosts can move/stand on all fields that are not walls and are inside the grid.

```
########          ########
# #    #          #G#    #          #######
# # ## #          # ####G#          #G    #
# #  #G#          #  G  P#          ##### #
#   P###          #### G #          #     #
#### # #          #G #   #          # #####
#G   #G#          # G# G #          #    P#
########          ########          #######
```

Figure 1: Three maps.

1 Counting Ghosts

We want to compute the number of ghosts on a given map. For example, on the three maps in Figure 1 there are 3,7, and 1 ghosts, respectively.

1.1 Give an algorithm, that given a map computes the number of ghosts on the map. Analyze the running time of your algorithm in terms of parameter N.

Solution:

1.2 [†] Implement your algorithm.

2 Reachable Ghosts

Now we want to compute which ghost that are an actual threat to Pacman. More precisely, we want to know how many ghosts that can reach Pacman on a given map, that is, that has a path to Pacman. For example, on the three maps in Figure 1 there are 2, 5, and 1 ghosts that can reach Pacman, respectively.

2.1 Give an algorithm, that given a map computes the number of ghosts that can reach Pacman. Analyze the running time of your algorithm in terms of parameter N.

Solution:

2.2 [†] Implement your algorithm.

3 Closest Ghost

When we compute the number of ghosts that can reach Pacman, we would like to know which ghost that is the biggest immediate threat and where it is in relation to Pacman. More precisely, we want to compute the shortest path from Pacman to the closest ghost. Pacman can only move one field up, down, left, or right – abbreviated `N`, `S`, `W`, and `E`. For example, the shortest paths on the three maps in Figure 1 are `SSWWW`, `N`, and `WWWWNNEEEENNWWWW`, respectively.

3.1 Give an algorithm, that given a map computes the shortest path from Pacman to a ghost. If there are more than one possible ghost choose any of them. You can assume that there is always at least one ghost that can reach Pacman. Output should be the sequence of `N`, `S`, `W`, and `E` as a string. Analyze the running time of your algorithm in terms of parameter N.

Solution:

3.2 [†] Implement your algorithm.

4 Multiplayer Closest Ghost

A few of the maps also have a multiplayer version with multiple Pacman players that helps each other. The maps are as before, but now there can be more Pacmen, marked with P. See figure 2.

```
########
#P#   P#
# # ## #
# #  #G#
#   P###
#### # #
#G   #G#
########
```

Figure 2: Multiplayer pacman.

In this situation we want to compute the shortest path from a Pacman to a ghost, that is, we want to find the shortest path among all possible pairs of Pacmen and ghosts. For example, in Figure 2 the shortest path is the path from Pacman in the upper right corner to the ghost below it (given by SS). All other pairs consisting of a Pacman and a ghost have a longer shortest path.

4.1 Give an algorithm, that given a map computes the shortest path from a Pacman to a ghost. If there are more than one possible pair you can choose any of them. You can assume that there is always at least one ghost that can reach at least one Pacman. Output should be the length of the path. Analyze the running time of your algorithm in terms of parameter N.

Solution:

4.2 [†] Implement your algorithm.