**The Numberlink Problem**

All resources related to this problem are available on *claco and teams.*

The Numberlink problem initially consists of a grid with some cells labeled while the

remaining cells are empty. The goal of this problem is creating disjoint paths connecting

cells with identical labels. Each label appears exactly twice. The path connecting them

must start from one of those cells and end at the other. An example is shown below.

In the picture, the labels are letters and the white cells are empty. On the left we can

see an example of initial state and on the right, we have the final state. It is considered

that an instance is well-designed if it has a unique solution. In this assignment we

consider only well-designed instances. An online version of the game can be found at

<https://numberlinks.puzzlebaron.com/>

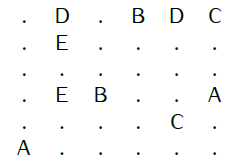


We provide a set of 10 problems. Your program should be able to solve all of them.

Also, don’t forget that we will also use hidden instances for the evaluation. Each problem

is described in a problem file, containing the initial configuration of the grid. For instance,

the file level6s.in corresponding to the sixth problem of the online version of the game (small, 6) contains the following:



We use alphabet letters as labels instead of numbers. Any labels will always be an upper

case letter from ’A’ to ’Z’ and the labels will be consecutive, meaning that if for instance we

use label ’D’ then labels ’A’, ’B’ and ’C’ are also used. The empty positions are represented

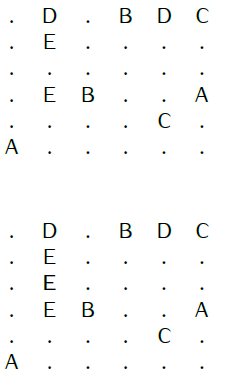
with the character ’**.**’. There are no spaces in the input file. Your program should write a

solution to the standard output. The solution is composed of the successive states to reach

the goal separated by an empty line. The format of the successive states is the same as the

one in the problem files. As example of the format of the solutions, the first two states of the

solution to the problem lelve6s.in might be:



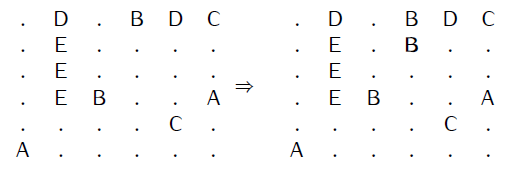
Tip:

You should complete one path at a time. This means that your state has to maintain the

current path under construction and the position of the last extension. Your successors should

consist of extending the last position **Down**, **Up**, **Left** and **Right**, in that order. Then next

figure shows a state and its successors. When starting a new path, you can start in any of its two endpoints.



You will implement at least one class *Numberlink(Problem)* that extends the class *Problem*

such that you will be able to use search algorithms of AIMA. A small template (*numberlink.py*)

is provided in the resources for this problem. In this file you will find a method *pathExists*

that takes as input a matrix of characters *grid* and two positions *(i1, j1)* and *(i1, j1)* and

outputs whether or not there exists a path between those positions that consist only of

empty cells (cells with character ’**.**’). You are not forced to use it.

Your script must be called in this manner:

python numberlink.py pathtoinputfile

And it should print all the states between the initial state and the goal state, like the example code.

Before diving into the code,

we recommend you to first have a look at the questions below.

**Questions**

1. Explain the advantages and weaknesses of the following search strategies **on**

**this problem** (not in general): depth first, breadth first.

2. How can we exploit the uniqueness of solution to reduce the search space?

Why is the method *pathExists* useful?

3. Is the order in which we choose the paths important? How can we use this to

reduce the search space? When starting a new path, we can choose to start

with any of its two endpoint. How should this choice be done?

4. What are the advantages and disadvantages of using the tree and graph search

**for this problem**. Which approach would you choose? Which approach allows

you to avoid expending twice symmetrical states?

5. Implement this problem in Python 3. Extend the *Problem* class and implement

the necessary methods and other class(es) if necessary. Your file must be named

*numberlink.py*. You program must take as only input the path to the init file

of the problem to solve. It must print to the standard output a solution to the

problem satisfying the above format. Your file must be encoded in **utf-8**. Submit

your program on claco.