UNIVERSIDADE DA CORUÑA

Grado en Ingeniería Informática
Intelligent Systems

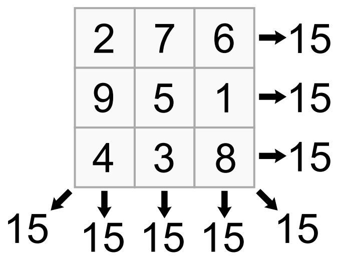
Assignment #1: Search strategies

Course 2021/2022

In this assignment you will implement in Java several search strategies and you will use them to solve a real problem.
You are provided with an IntelliJ IDEA project that has example code, including an implementation of the Basic Search Strategy #4 as presented in previous sessions.

- **Exercise 1 (3 points): Perform the following changes on the example code:**
  a) Create a `Node` class following the description given in class (S45). Adapt `Strategy4` so that it records the explored states as `Nodes`.
  b) Change the solve method of the class `SearchStrategy` so that it returns an array of `Nodes` (`Node[]).` Modify the implementation of `solve` in `Strategy4` so that, when a solution is found, it returns the full list of nodes representing the traversed states to get from the initial state to the found solution. To do so, implement in `Strategy4` a method called `reconstruct_sol` as described in class.
  c) `Strategy4` fails when it reaches a state that has no successors. However, there may be states encountered previously that still have unexplored successors. The strategy `Graph_Search`, described in class, solves this by keeping track of a frontier that stores the successors states encountered while they are not explored. Implement a class called `GraphSearchStrategy` (that extends `Strategy`) that implements this strategy (you may find that using `Strategy4` as a template is helpful).

- **Exercise 2 (7 points): We are trying to solve the following problems using search strategies:**
  A NxN magic square is a matrix that contains the numbers between 1 and N^2 arranged so that the sum of the elements of each row (or each column, or each of its main diagonals) is always the same: $\frac{N(N^2+1)}{2}$



Example 3x3 magic square

Our task is to develop an agent capable of, given a partially complete magic square, fill the remaining positions (if possible) to complete a valid NxN magic square (without changing any of the provided values).

Examples:

Initial state:

4	9	2
3	5	
	1	

Goal state:

4	9	2
3	5	7
8	1	6

Initial state:

2		

Goal state:

2	9	4
7	5	3
6	1	8

or any other 3x3 magic square

2			
	1		

2	8	15	9
14	12	5	3
11	13	4	6
7	1	10	16

or any other 4x4 magic square

PART A (3 points):

- ❑ Formalize the problem so that it can be solved using search strategies. Use simple actions that only change a single cell of the square. Then, implement it by writing a class named `MagicSquareProblem` that extends `SearchProblem` and defines the subclasses of `State` and `Action` that are needed to solve the problem.
- ❑ Implement the breadth-first and depth-first search strategies and use them to solve the problem (use the first provided example or other simple modifications as the initial state).
- ❑ Adapt your implementations to keep track of:
    - ❑ Number of expanded nodes
    - ❑ Number of nodes created
- ❑ Which of the two strategies is better suited for this problem? Verify your intuition using experiments. What is the cause for the difference between the two?

PART B (3 points):

- ❑ Report a suitable heuristic for the Magic Square problem. Create an implementation of said heuristic by extending the class Heuristic, which will calculate its value for this problem. Is your heuristic admissible? Is it consistent? Justify your answer.
- ❑ Implement the A^* search strategy. To do so, create a subclass of `InformedSearchStrategy` and modify the `Node` class so that it stores the cost of the path, the value of the function f and so that it implements the `Comparable` interface. Use this strategy to solve the second example.

PART C (1 point):

- ❑ Can your program solve the third example in a reasonable time? What would you improve to solve it faster? Describe in detail any modifications and implement them. If the implementation requires many changes, you may create a new package called `es.udc.intelligentsystems.gA_xy_2C` with the improved copies of the classes.

- ## Submission

This exercise will be performed in groups of two people as designated in the corresponding Campus Virtual section.
Your submission, delivered via Campus Virtual, must include:

- a small explanatory report document that includes all the answers to the questions and a brief description of the implementations done describing any problems that you may have encountered and justifying the decisions taken.
- the full source code, complete with documentation and comments. All classes must be contained in a package called `es.udc.intelligentsystems.gA_xy` where ga_xy indicates the group that the code belongs to. (Rename the provided `es.udc.intelligentsystems` package to `es.udc.intelligentsystems.gA_xy` and work there) There must be a Main class for each exercise (MainEx1, MainEx2a and MainEx2b, each of which must have a main method).

The due date for this assignment is **March 18th (23:55h)**.