# Intelligent Systems

ASSIGNMENT 1.

DAVID GARCÍA RAMALLAL
ALFREDO JAVIER FREIRE BOUZAS

# Exercise 1.

We have created a class *Node* (with three attributes: its state, its parent and the action that results on such node).

As we want the method *solve* to return an array of Nodes, we have changed the declaration of this method at the Interface *SearchStrategy*.

The class *Strategy4* has been modified in order to work using Nodes, making a call to a method that reconstructs the solution.

As it is said on the pdfs, the Strategy4 fails when it reaches a state that has no successors. For avoiding that, we have implemented a graph search (*GraphSearch*).

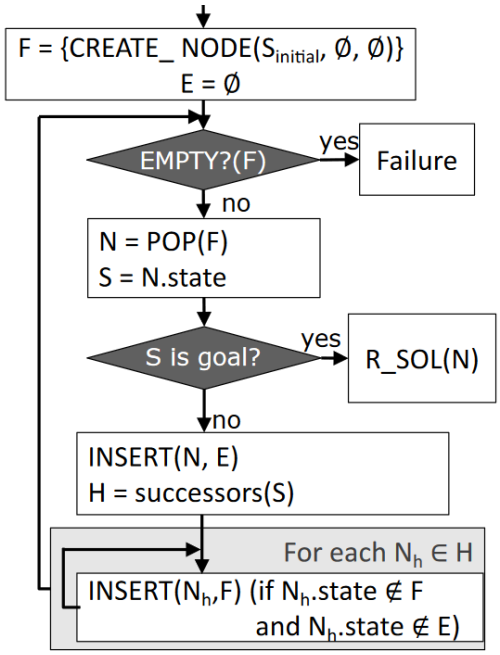



*Figure 1. Unit2. Slide 64.*

# Exercise 2. Part A.

## Formalization of the problem:

- States:
  - N x N matrix *A* of elements $a_{i,j}$ with i,j ∈ $[0,N-1]$
  - i represents the column number starting on the left and j represent the row number starting on the bottom.
  - Each cell a must contain $a_{i,j}$ number between 0 and $N^2$ (0 represents a blank cell).
  - No two cell share the same value.
- Initial state: Any valid state.

- Actions/Transition model:

| Action | Precondition | Result |
|---|---|---|
| Insert number n in cell $A_{i,j}$ | Let $A_{i,j} = 0$ be the element that is empty.<br>Let *n* be a number between 1 and $N^2$, not in the matrix at the moment of the insertion. | Same matrix, but inserting the number n at position $A_{i,j}$. |

- Goal test:
  For every i,j ∈ $[0,N-1]$:

$$\sum a_{ij} + a_{i(j+1)} + \dots + a_{i(j+N-1)} = \sum a_{ij} + a_{(i+1)j} + \dots + a_{(i+N-1)j}$$

- Path cost: number of actions (cost of action = 1).

## 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?

The main difference between Depth-First and Breadth-First is the way in which the data is stored. Breadth-First uses a FIFO queue for the frontier (the Node that entered the longest time ago is expanded) while Depth-First is based on a LIFO queue (the most recently entered Node is the one expanded).

To check which of the algorithms has a better a performance, we use as an example the matrix: [[0,0,2],[0,5,7],[0,0,0]].

To measure the times, we have used the method ***System.currentTimeMillis()***.

With the breadth-first algorithm it takes 8154.0 ms.

With the depth-first algorithm it takes 914.0 ms.

The depth-first algorithm is obviously faster than the breadth-first one. The main reason for this is that each path is followed before changing to another one, which means that the most recently entered is the one expanded.

## Exercise 2. Part B.

### 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.

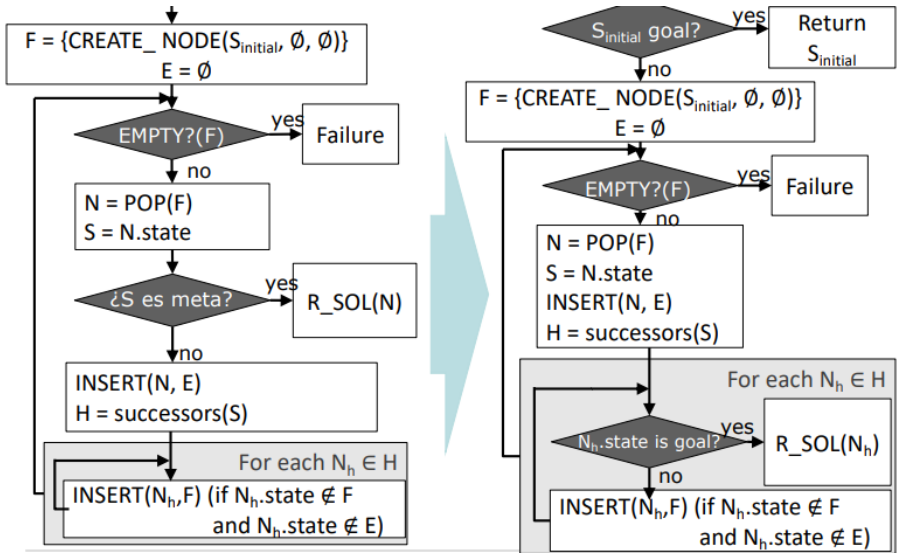
The value of the heuristic in our implementation is based on the relation between the number of empty cells and the total number of cells. It also takes into account the sum of the values of each row and each column, comparing them with the value that they are supposed to have.

It is **admissible** since it never overestimates the cost of reaching the target. This is due to the fact that we would not empty a cell that was already occupied by a number (obviously different from zero).

On the other hand, it is also **consistent** as long as f(n) is never increasing (f(n) = h(n) + g(n)). The closer to the goal, the smaller the value of h(n). When reaching the goal, the value of h(n) turns into 0.

# Exercise 2. Part C.

## Can your program solve the third example in a reasonable time? What would you improve to solve it faster?




No, it can't. First of all, we try to improve the implementations made in section A. To do so, we follow the Breadth-First algorithm provided in the slides, from the left one that we have used in the exercise 2A, to the one on the right. However, in spite of the fact that the performance of the algorithm was improved notably (the number of expanded and created nodes is smaller), it is not strong enough to perform the third example in a reasonable time.

Even using the heuristic implementation of the Exercise 2B, the program is not able to solve this third example in a reasonable time (it works correctly, but expends too much time).