## Section 0

Summary of the concepts reviewed in the practice

## Section 1

* 1. **Personal comment on the approach and decisions of the proposed solution** 
     1. **List & explanation of the framework functions used**
     2. **Includes code written by students**
     3. **Screenshots of executions and test carried out analyzing the results**
  2. **Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc** 
     1. **Answer to question 1.1**
     2. **Answer to question 1.2**
     3. **Answer to question 2**

## Section 2

* 1. **Personal comment on the approach and decisions of the proposed solution** 
     1. **List & explanation of the framework functions used**
     2. **Includes code written by students**
     3. **Screenshots of executions and test carried out analyzing the results**
  2. **Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc** 
     1. **Answer to question 3**

## Section 3

* 1. **Personal comment on the approach and decisions of the proposed solution** 
     1. **List & explanation of the framework functions used**
     2. **Includes code written by students**
     3. **Screenshots of executions and test carried out analyzing the results**
  2. **Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc**

## Section 4

* 1. **Personal comment on the approach and decisions of the proposed solution** 
     1. **List & explanation of the framework functions used**
     2. **Includes code written by students**
     3. **Screenshots of executions and test carried out analyzing the results**
  2. **Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc** 
     1. **Answer to question 4**

## Section 5

* 1. **Personal comment on the approach and decisions of the proposed solution** 
     1. **List & explanation of the framework functions used**
     2. **Includes code written by students**
     3. **Screenshots of executions and test carried out analyzing the results**
  2. **Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc**

## Section 6

* 1. **Personal comment on the approach and decisions of the proposed solution** 
     1. **List & explanation of the framework functions used**
     2. **Includes code written by students**
     3. **Screenshots of executions and test carried out analyzing the results**
  2. **Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc** 
     1. **Answer to question 5: heuristics**

## Section 7

Personal comments on the development of this practice

## Section 1: Depth First Search

Texto

Descripción generada automáticamente

For the first four exercises we created the GraphSearch function: given an open list structure as an argument, it carries out the search algorithm with elimination of repeated states (using a list as the closed list). We found it very useful to create a separate Node class to facilitate working with the state space. The class works as follows:

Texto

Descripción generada automáticamente

A node object only stores three pieces of information: the game state itself it refers to, a complete list of operators from the root node to the current node, and the total cost required to get from the root node to the current node. The last two pieces are assembled using the parent node’s information (the parent is passed as an argument in the node constructor) plus the operator and cost of the current node. The root node is the only one without a parent. In GraphSearch, we can see how useful the node structure is, since each node carries the list of operators already in case they contain the goal state.

Texto

Descripción generada automáticamente

All four search algorithms are almost as simple as this one: we use a stack as the open list and execute the GraphSearch algorithm. A stack for depth search is appropriate since stacks have a LIFO (Last In First Out) policy. This means the nodes generated from the latest expanded node will be expanded first, thus prioritising depth.

Imagen que contiene Icono

Descripción generada automáticamentePatrón de fondo, Escala de tiempo

Descripción generada automáticamenteCódigo QR

Descripción generada automáticamenteTexto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Pacman does reach the goal on all three attempts since elimination of repeated states guarantees it. However, if we were to implement tree search with a depth first search, Pacman might follow a path of infinite length and never finish the search.

**Question to answer: Is the exploration order what you would have expected? Does Pacman go to all the explored squares on his way to the goal?**

As we can see, more than half the state space in medium and big mazes was not ever generated. This is to be expected from a depth first search, since the “current path” followed is not ever changed unless there are no more nodes to generate. Of course, not all expanded nodes will be on the final solution: that is the nature of search algorithm.

**Question to answer: Is this a least cost solution? If not, think about what depth-first search is doing wrong.**

Depth first search rarely offers an optimal solution, by virtue of not considering all possibilities in the state space and committing to a path until it must rectify. This case is no different.

## Section 2: Breadth First Search

Texto

Descripción generada automáticamente

We used a queue as the open list, which has a FIFO (First In First Out) policy, which is appropriate because we ensure a node of higher depth never gets expanded before a lower depth node.

Imagen que contiene Icono

Descripción generada automáticamentePatrón de fondo

Descripción generada automáticamenteImagen que contiene Código QR

Descripción generada automáticamenteTexto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

**Question to answer: Does BFS find a least cost solution?**

It is not surprising that Pacman finds a solution on all experiments, since breadth first search guarantees to find the shallowest solution that exists. Since in the current problem all actions have the same cost of 1, the shallowest path is the most optimal by default. Thus, Pacman’s solution will always be optimal under these circumstances.

What is not optimal, though, is the number of search nodes expanded: almost all the game states were generated while finding the least-cost path (since, as we established, a lower depth path will always be explored earlier than another path of higher depth). This is an inefficient use of time and space.

## Section 3: Uniform Cost Search

Texto

Descripción generada automáticamente

We will use the priority queue with function as the open list: this queue pops the element with the lowest priority score. Uniform cost search expands the node with the lowest cost path from root node to current node, so our priority function will return exactly that: the cumulative cost from root to the current node that every node has already stored (as explained in the Node class in Section 1). We defined an internal function that did as described and used it as argument for the queue.

Patrón de fondo

Descripción generada automáticamenteTexto

Descripción generada automáticamente

In this first test we can notice that the algorithm behaves the same as in the breadth first search algorithm. This is because all actions cost the same, so all nodes of the same depth have the same priority; the priority queue then behaves like a regular queue. Thus, Pacman does find a solution and it is optimal.

Next two tests are done with two different agents, respectively: the first agent heavily favours staying on the East side of the map, while the second agent heavily favours the West side. This is done by varying the cost function.

Imagen que contiene Patrón de fondo

Descripción generada automáticamenteImagen de la pantalla de un video juego

Descripción generada automáticamente con confianza bajaTexto

Descripción generada automáticamente

In these cases, uniform cost search is very different from breadth first search since the cost function does no longer increase monotonously with depth. The behaviour is definitely optimal, successors of nodes will always have a higher cost path than the node itself, while breadth first search would not consider path costs and end up with a highly inefficient solution, even though it is the lowest-depth solution.

## Section 4: A\* Search

Texto

Descripción generada automáticamente

The structure of this function is almost identical to the uniform cost search function, with a difference being the priority function: in a\* the function returns the sum of the cumulative path cost and the result of the heuristic given as an argument.

Imagen que contiene Código QR

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Pacman does find the solution, which is optimal. Since the Manhattan heuristic is admissible and monotonic, it will always find the optimal solution for graph search. It is noticeable that the a\* solution needed less node expansion than the uniform cost search algorithm: the a\* algorithm is also optimally efficient.

**Question to answer: What happens on openMaze for the various search strategies?**

## Section 5: Corners Problem