# PRACTICA 1 – Algoritmos de Búsqueda en: PACMAN

**Título: Memoria Practica 1 Grupo 2301 Pareja nº 15**

**Autores: Saul Almazan de Pie & Guillermo Hoyo Bravo.**

## Sección 1 – Búsqueda Primero en Profundidad (depthFirstSearch())

* 1. **Comentario personal en el enfoque y decisiones de la solución propuesta (1pt)**

Para implementar el algoritmo de Búsqueda primero en profundidad, nos hemos decidido por usar una pila. Introducimos el nodo inicial hacemos un bucle que termina cuando la pila esta vacías, es decir que hemos llegado al último de los nodos hoja. Para implementar la eliminación de estados repetidos guardamos el nodo en una lista llamada Visitados, y si no ha sido visitado lo introducimos a la pila.

* 1. **Lista & explicación de las funciones del framework usadas (1pt)**
* Class Stack: 🡪 util.Stack()
* def push(self,item): 🡪 Pila.push() 🡪 Pila.push((problem.getStartState(),[],[]))
* def getStartState(self): 🡪 problem.getStartState(),[],[]
* def isEmpty(self): 🡪 Pila.isEmpty()
* def pop(self): 🡪 Pila.pop()
* def isGoalState(self, state): 🡪 problema.isGoalState(nodo)
* def getSuccessors(self, state): 🡪 problema.getSuccessors(nodo)
  1. **Incluye el código añadido (0.25 pts)**

Acciones = []

Aux = []

Nodos = util.Stack()

* 1. **Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados (1pt)**

\***Recordad** imprimir capturas de pantalla de estos tests y explicar lo que habéis observado para la memoria de la práctica.

* 1. **Conclusiones en el comportamiento de pacman, es optimo (s/n), llega a la solución (s/n), nodos que expande, etc (1pt)**

Si, nodos que expande, etc

* 1. **Respuesta a pregunta 1.1 (1pt)**

**¿El orden de exploración es el que esperabais?**

Si

* 1. **Respuesta a pregunta 1.2 (1pt)**

**¿Pacman realmente va a todas las casillas exploradas en su camino hacia la meta?**

Si

* 1. **Respuesta a pregunta 2 (1pt)**

**¿Es esta una solución de menor coste? Si no es así, pensad qué está haciendo mal la búsqueda en profundidad.**

**asdadsasd**

## Sección 2 – Búsqueda en Anchura (breadthFirstSearch(problem))

* 1. **Comentario personal en el enfoque y decisiones de la solución propuesta (1pt)**

Para este algoritmo hemos utilizado el enfoque usado en la Busqueda del primero en profundidad, ya que son muy parecidos. La diferencia aquí es que nos decidimos a usar una cola en vez de continuar con la pila. Este cambio lo propone el mismo algoritmo, que se basa en investigar los nodos por el orden en el que se descubren. De este modo, vamos investigando cada nodo en una misma profundidad, hasta llegar a la solución o al final del árbol.

* 1. **Lista & explicación de las funciones del framework usadas (1pt)**
* Class Queue: 🡪 util.Queue ()
* def push(self,item): 🡪 Queue.push() 🡪 Queue.push((problem.getStartState(),[],[]))
* def getStartState(self): 🡪 problem.getStartState(),[],[]
* def isEmpty(self): 🡪 Queue.isEmpty()
* def pop(self): 🡪 Queue.pop()
* def isGoalState(self, state): 🡪 problema.isGoalState(nodo)
* def getSuccessors(self, state): 🡪 problema.getSuccessors(nodo)
  1. **Incluye el código añadido (0.25 pts)**

**sdfsfsdf**

* 1. **Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados (1pt)**

1. \***Recordad** imprimir capturas de pantalla de estos tests y explicar lo que habéis observado para la memoria de la práctica.
   1. **Conclusiones en el comportamiento de pacman, es optimo (s/n), llega a la solución (s/n), nodos que expande, etc (1pt)**

Sí, nodos que expande, etc??

* 1. **Respuesta a pregunta 3 (1pt)**

**¿BA encuentra una solución de menor coste? Si no es así, verificad vuestra implementación.**

**asdasdads**

## Sección 3 – Variar la Función de Coste (uniformCostSearch(problem))

* 1. **Comentario personal en el enfoque y decisiones de la solución propuesta (1pt)**

Este algoritmo es muy parecido a los dos anteriores que hemos explicado. En este caso nos hemos decidido por usar la estructura de la cola de prioridad, ya que este algoritmo en vez de darle la prioridad de exploración al último nodo encontrado para investigar una rama hasta su profundidad máxima y continuar por la siguiente, o investigar cada nivel de profundidad por completo antes de pasar al siguiente, expande primero el nodo de menor coste, obteniendo la solución óptima.

* 1. **Lista & explicación de las funciones del framework usadas (1pt)**
* Class Queue: 🡪 util.PriorityQueue()
* def push(self,item): 🡪 PriorityQueue.push() 🡪 PriorityQueue.push((problem.getStartState(),[],0),0)
* def getStartState(self): 🡪 problem.getStartState(),[],[]
* def isEmpty(self): 🡪 PriorityQueue.isEmpty()
* def pop(self): 🡪 PriorityQueue.pop()
* def isGoalState(self, state): 🡪 problema.isGoalState(nodo)
* def getSuccessors(self, state): 🡪 problema.getSuccessors(nodo)
  1. **Incluye el código añadido (0.25 pts)**

**asdads**

* 1. **Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados (1pt)**

**CAPTURAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAS**

* 1. **Conclusiones en el comportamiento de pacman, es óptimo (s/n), llega a la solución (s/n), nodos que expande, etc (1pt)**

Sí

Nodos que explande, etc??

## Sección 4 – Busqueda A\* (aStarSearch(problema, heuristic=nullHeuristic))

* 1. **Comentario personal en el enfoque y decisiones de la solución propuesta (1pt)**

Este algoritmo es la suma del algoritmo desarrollado en el punto 3 (Coste Uniforme) con una heurística. Literalmente este algoritmo es guiado por el coste mínimo de la siguiente función:

**F(n) = G(n) + H(n)**

El coste mínimo es la suma de el coste hasta el nodo n (G(n)) y el valor de la heurística para el nodo n (H(n))

Por lo que hemos usado el código implementado en el punto 3, sumando en las partes necesarias el coste adicional que proporciona la función nullHeuristic.

* 1. **Lista & explicación de las funciones del framework usadas (1pt)**
* Class Queue: 🡪 util.PriorityQueue()
* def push(self,item): 🡪 PriorityQueue.push() 🡪 PriorityQueue.push((problem.getStartState(),[],0), heuristic(problema.getStartState(), problem))
* def nullHeuristic(state, problema=None): 🡪 heuristic=nullHeuristic 🡪 heuristic(problema.getStartState(), problem)
* def getStartState(self): 🡪 problem.getStartState(),[],[]
* def isEmpty(self): 🡪 PriorityQueue.isEmpty()
* def pop(self): 🡪 PriorityQueue.pop()
* def isGoalState(self, state): 🡪 problema.isGoalState(nodo)
* def getSuccessors(self, state): 🡪 problema.getSuccessors(nodo)
  1. **Incluye el código añadido (0.25 pts)**

**CODIFOOOO**

* 1. **Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados (1pt)**

**CAPTURAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA**

* 1. **Conclusiones en el comportamiento de pacman, es optimo (s/n), llega a la solución (s/n), nodos que expande, etc (1pt)**

Sí, A\* sin repetición de estados siempre encuentra la solución Óptima.

Nodos que expande, etc??

* 1. **Respuesta a pregunta 4 (1pt)**

**¿Qué sucede en openMaze para las diversas estrategias de búsqueda?**

**ojoijoi**

## Sección 5 – Encontrar todas las Esquinas ()

* 1. **Comentario personal en el enfoque y decisiones de la solución propuesta (1pt)**
  2. **Lista & explicación de las funciones del framework usadas (1pt)**
  3. **Incluye el código añadido (0.25 pts)**
  4. **Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados (1pt)**
  5. **Conclusiones en el comportamiento de pacman, es optimo (s/n), llega a la solución (s/n), nodos que expande, etc (1pt)**

## Sección 6 – Problema de las Esquinas Heurística ()

* 1. **Comentario personal en el enfoque y decisiones de la solución propuesta (1pt)**
  2. **Lista & explicación de las funciones del framework usadas (1pt)**
  3. **Incluye el código añadido (0.25 pts)**
  4. **Capturas de pantalla de los resultados de ejecución y pruebas analizando los resultados (1pt)**
  5. **Conclusiones en el comportamiento de pacman, es optimo (s/n), llega a la solución (s/n), nodos que expande, etc (1pt)**
  6. **Respuesta a pregunta 5: heurística (1pt)**

## Sección 7

**Comentarios personales de la realización de esta práctica**

Hemos encontrado alguna dificultad con el lenguaje de Python y la memorización o compresión del extenso código del programa de Pacman.

A demás de esto hemos hecho brainstorming para encontrar alguna heurística plausible en el punto 4 antes de pensar en la de manhattan y utilizarla, así como para la heurística del ejercicio 6.

El ejercicio 6 nos ha traído también dificultados…

## Nota de la memoria (40% de la práctica)

**Total de puntos (X/31.5)**