

PRÁCTICA SISTEMAS INTELIGENTES

MAZE SOLVER

INTEGRANTES:

- Alejandro Ruiz Aranda
- Jesús Santiyán Reviriego

GRUPO DE TRABAJO:

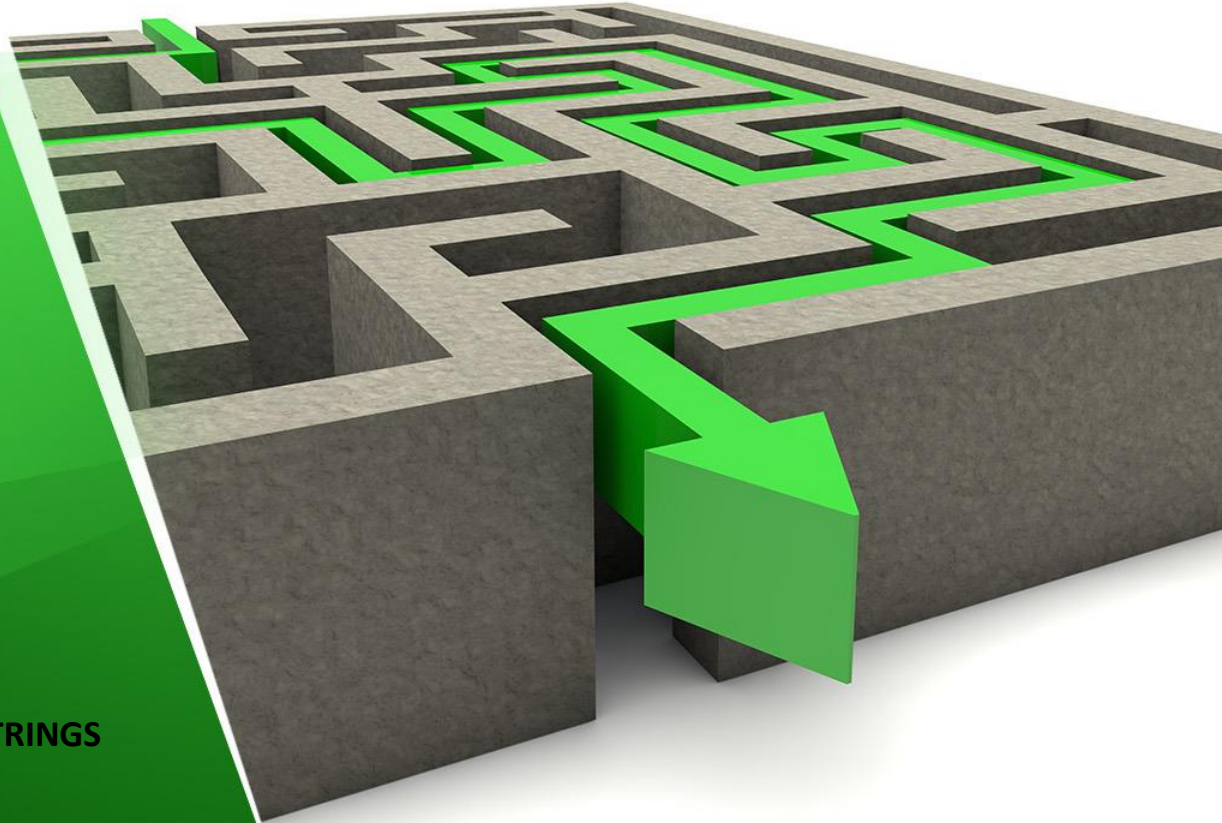
- C1-8

REPOSITORIO DE TRABAJO

- https://github.com/alexra99/SI_C1-8.git

SITIO WEB DE DOCUMENTACIÓN DOCSTRINGS

- https://alexra99.github.io/SI_C1-8/



ÍNDICE

1. Introducción del problema y definición como un problema de búsqueda en el espacio de estados.
2. Desglose del código del programa.
3. Tareas realizadas con la justificación de los criterios y opciones tomadas.
 - TAREA 1
 - TAREA 2
 - TAREA 3
4. Manual de usuario.
5. Valoración personal y del equipo respecto a la práctica.



1. Introducción del problema y definición como un problema de búsqueda en el espacio de estados.

DEFINICIÓN DEL PROBLEMA

Tenemos un tablero en el que hay dibujado un laberinto, este laberinto ha sido generado de forma aleatoria con el algoritmo de Wilson y todas sus casillas tienen conexiones.

Además, como es lógico cuenta con una casilla de entrada y una casilla de salida. La misión consiste en encontrar el camino más óptimo entre la entrada y la salida para poder establecerla como la mejor ruta de escape.

Este camino se podrá realizar siguiendo distintas estrategias. Teniendo en cuenta que hay diferentes tipos de casillas que pueden ser de tierra, agua, hierba y asfalto, según su tipo tendremos más o menos dificultades para atravesar y siempre deberíamos elegir la que más nos convenga siguiendo un criterio, este criterio consiste en elegir la casilla con menor coste.

Si varias casillas tuvieran el mismo coste habría que elegir la que tuviera un número de fila menor. Si coincidiera nos guiaríamos por el número de columna y si todo esto no sirviera elegiríamos la que tuviera un menor ID de casilla.

1. Introducción del problema y definición como un problema de búsqueda en el espacio de estados.

- **Variables:** Las variables son las casillas que tenga el laberinto, representadas por su número de fila y de columna. Dominio: El dominio serían todas las casillas (FC) que tenga el laberinto (L). $\forall FC \subseteq L$.
- **Estado:** La casilla en la que nos encontramos, con su movimiento (M), número de fila (F) y de columna (C) y el coste del movimiento (MC). Un estado E sería: $E = (M, F, C, MC) \mid (M \in FC), (F \in FC), (C \in FC) \text{ y } (MC \in FC)$
- **Función sucesor:** Estado inicial: Casilla de entrada del laberinto Sucesor: dada una casilla, su sucesores serán las casillas a la que pueda acceder desde la misma, es decir, no hay una pared que lo impida. Desde una casilla solo podrán hacer lo siguientes movimientos en el siguiente orden:
 - **NORTE:** decrementar columna
 - **ESTE:** incrementar fila
 - **SUR:** incrementar columna
 - **OESTE:** decrementar fila
- **Objetivo:** Casilla de salida el laberinto

2. Desglose del código del programa.

A través de los docstrings del código hemos generado un sitio web donde se puede ver un desglose del código del programa comentado.

El enlace del sitio web es el siguiente:

https://alexra99.github.io/SI_C1-8/

3. Tareas realizadas con la justificación de los criterios y opciones tomadas

TAREA 1

A) Elección del lenguaje de programación:

Para la resolución de este problema hemos optado por usar el lenguaje de programación Python.

Tras haber estado investigando y barajar otras opciones como Java, definitivamente escogimos Python por las siguientes razones

- **Sencillez:** se puede conseguir lo mismo que en otros lenguajes como Java en muchas menos líneas de código.
- **Documentación:** hemos encontrado mucha más documentación sobre este tema y ejemplos en los que apoyarnos escritos en Python.
- **Nuevas funciones:** nunca habíamos trabajado con ficheros json, sin embargo hemos encontrado una forma muy fácil y sencilla de trabajar con ellos usando Python.
- **Ampliar conocimientos:** hasta ahora, la mayoría de las prácticas de las demás asignaturas las habíamos realizado en Java, excepto en redes 2 dónde ya empezamos a dar los primeros pasos en lenguaje Python y nos dio muy buenas sensaciones. Por ello queremos seguir ampliando conocimientos en este lenguaje y consideramos que el hecho de realizar esta práctica en Python nos ayudará a conseguirlo.

3. Tareas realizadas con la justificación de los criterios y opciones tomadas

TAREA 1

B) Generar json del laberinto:

Para generar el json hemos utilizado la librería json de Python y le hemos dado el formato que se pedía. Pedimos la filas y las columnas por teclado, y para saber los vecinos de cada celda recorreremos el tablero y en cada celda con el método islinked() vemos si tienen celda unidas en el norte, sur, este y oeste, esto indicará si hay vecino en esa coordenada o no.

```
def save_json(self):
    """Generar el json con el formato dado"""
    output = {
        "rows": 4,
        "cols": 4,
        "max_n": 4,
        "mov": [[-1, 0], [0, 1], [1, 0], [0, -1]],
        "id_mov": ["N", "E", "S", "O"],
        "cells": {}
    }
    for r in range(self.rows):
        for column in range(self.columns):
            key_cell = f'({r}, {column})'
            output["cells"][key_cell] = {
                "value": 0,
                "neighbors": [self.grid[r][column].islinked(self.grid[r][column].cellNorth),
                             self.grid[r][column].islinked(self.grid[r][column].cellEast),
                             self.grid[r][column].islinked(self.grid[r][column].cellSouth),
                             self.grid[r][column].islinked(self.grid[r][column].cellWest)]
            }

    with open(f'Lab_{self.rows}_{self.columns}.json', 'w') as outfile:
        dump(output, outfile)
```

3. Tareas realizadas con la justificación de los criterios y opciones tomadas

TAREA 1

C) Leer laberinto de un json:

Para leer el json en el constructor del grid se crea el grid correspondiente con los datos que se toman del json.. Para crear el resto del laberinto se van leyendo lo vecinos que pueda tener una celda y con el método link().

```
with open(filename) as fdata:
    jsondata = load(fdata)
self.rows = jsondata['rows']
self.columns = jsondata['cols']
self.grid = self.prepare_grid()
for cell in self.each_cell():
    key = f'{cell}'
    row, col = cell.row, cell.column
    cell.setValue(int(jsondata['cells'][key]['value']))
    cell.cellNorth = self[row - 1, col]
    cell.cellSouth = self[row + 1, col]
    cell.cellWest = self[row, col - 1]
    cell.cellEast = self[row, col + 1]

    if jsondata['cells'][key]['neighbors'][0]:
        cell.link(self[row - 1, col])
    if jsondata['cells'][key]['neighbors'][1]:
        cell.link(self[row, col + 1])
    if jsondata['cells'][key]['neighbors'][2]:
        cell.link(self[row + 1, col])
    if jsondata['cells'][key]['neighbors'][3]:
        cell.link(self[row, col - 1])
```


3. Tareas realizadas con la justificación de los criterios y opciones tomadas

TAREA 2

A) Artefacto software nodo:

Para la creación del artefacto nodo hemos hecho una clase nodo donde se le definen todos sus atributos y se imprime en un string todos los atributos del nodo en el mismo orden.

```
class Node:
    def __init__(self, ID, cost, state, father_id, action, depth, h, value):
        self.ID = int(ID)
        self.cost = int(cost)
        self.state = state
        self.father_id = int(father_id)
        self.action = str(action)
        self.depth = int(depth)
        self.h = int(h)
        self.value = float(value)
```

3. Tareas realizadas con la justificación de los criterios y opciones tomadas

TAREA 2

B) Artefacto software frontera:

La frontera es una lista ordenada de artefactos nodos. También contiene una serie de métodos para trabajar con ella: sacar elementos, insertar, etc

```
class Frontier:
    def __init__(self):
        self.items=[]
    def is_empty(self):
        return self.items==[]

    def push(self, data):
        self.items.append(data)

    def insert(self, pos, data):
        self.items.insert(pos, data)

    def pop(self):
        return self.items.pop()

    def remove(self):
        self.items.remove()

    def get_value(self, pos):
        return self.items[pos].value

    def get_size(self):
        return len(self.items)

    def pop_order(self):
        return self.items.pop(0)

    def remove_by_pos(self, pos):
        for index in range(len(self.items)):
            if self.items[index].state.get_tuple() == __pos:
                self.items.pop(index)
                break
```

3. Tareas realizadas con la justificación de los criterios y opciones tomadas

B) Artefacto software frontera:

En la frontera podemos destacar el método `push_frontier` que hace un push ordenado de nodos en la misma. Con este método ordenamos los nodos que entran en la frontera y los ordenamos para que a la hora de sacar la solución ésta venga dada en el orden correcto. Para la ordenación nos servimos de cuatro criterios: el primero, sería mirar el campo "value".

Si este campo es igual, el siguiente criterio a mirar sería el número de fila. A continuación, en el caso de coincidir, se miraría el número de columna. Por último, si este también fuera el mismo, se ordenaría por menor ID.

TAREA 2

```
def push_frontier(frontier, node):
    if (frontier.is_empty()):
        frontier.push(node)
    else:
        x = frontier.get_size() - 1
        pos = frontier.items[x].state.get_tuple()[0]
        while x >= 0:
            if ((frontier.get_value(x) < node.value)):
                frontier.insert(x+1,node)
                break
            elif(frontier.get_value(x) == node.value):
                if(frontier.items[x].state.get_tuple()[0] < node.state.get_tuple()[0]):
                    frontier.insert(x+1,node)
                    break
                else:
                    if(frontier.items[x].state.get_tuple()[0] == node.state.get_tuple()[0]):
                        if(frontier.items[x].state.get_tuple()[1] < node.state.get_tuple()[1]):
                            frontier.insert(x+1,node)
                            break
                        elif(frontier.items[x].state.get_tuple()[1] == node.state.get_tuple()[1]):
                            if (frontier.items[x].state.ID < node.ID):
                                frontier.insert(x+1,node)
                                break
            if (x==0):
                frontier.insert(0,node)
        x = x-1
```

3. Tareas realizadas con la justificación de los criterios y opciones tomadas

TAREA 3

A) Función de expansión:

Para expandir un nodo, a través de la función expandir primero, obtenemos los vecinos de un estado con la función neighbors_format. Una vez tenemos los vecinos, los metemos en una lista llamada sucesores, que está formada por los nodos vecinos. Procedemos a comprobar si fueron visitados o no. Para ellos utilizamos un diccionario que contiene todo las posiciones del laberinto inicializadas a false. Si un estado fue visitado se añade la frontera. Por el contrario si no lo fue, no se hace pues estaríamos tratando un estado que nunca ofrecerá una solución prometedora.

```
def neighbors_format(self):
    """ Return a list of all cells neighboring this cell
    """
    n = []
    if self.isLinked(self.cellNorth):
        n.append(('N', self.cellNorth))
    if self.isLinked(self.cellEast):
        n.append(('E', self.cellEast))
    if self.isLinked(self.cellSouth):
        n.append(('S', self.cellSouth))
    if self.isLinked(self.cellWest):
        n.append(('O', self.cellWest))
    return n
```

```
def expandir(current_node, g, visited, expanded, frontier, all_sucesor, strategy, end):
    list_sucesores = get_sucesor_nodes(current_node, g, current_node.state.neighbors_format(), all_sucesor, strategy, end)
    if strategy == 2:
        for sucesor in list_sucesores:
            if not visited[f'{sucesor.state}']:
                push_frontier(frontier, sucesor)
                visited[f'{sucesor.state}'] = True
            elif not expanded[f'{sucesor.state}']:
                frontier.remove_by_pos(sucesor.state.get_tuple())
                push_frontier(frontier, sucesor)
                visited[f'{sucesor.state}'] = True
        else:
            for sucesor in list_sucesores:
                if not visited[f'{sucesor.state}']:
                    push_frontier(frontier, sucesor)
                    visited[f'{sucesor.state}'] = True
```

3. Tareas realizadas con la justificación de los criterios y opciones tomadas

TAREA 3

A) Función de expansión:

Una vez obtenidos los sucesores procedemos a asignarle los valores necesarios en la función `get_sucesor_nodes`. En esta función podemos destacar que en función de la estrategia que se elija el `value` se calculará de una forma u otra. (La estrategia las representamos con un número del 1 al 5 que se introduce por teclado)

```
def get_sucesor_nodes(current_node, g, list_sucessors, all_sucessor, strategy, end):
    list_nodes = []
    value = 0
    for successor in list_sucessors:
        all_sucessor.append(successor)
        ID = len(all_sucessor)
        cost = current_node.cost + g[sucessor[ID].get_tuple()].value + 1
        state = sucessor[ID]
        father_id = current_node.ID
        depth = current_node.depth + 1
        action = sucessor[0]
        h = abs(sucessor[ID].row - end[0]) + abs(sucessor[ID].column - end[1])
        node = Node(ID, cost, state, father_id, action, depth, h, value)
        if strategy == 1:
            value = depth
        elif strategy == 2:
            value = 1.0/(depth+1)
        elif strategy == 3:
            value = cost
        elif strategy == 4:
            value = abs(sucessor[ID].row - end[0]) + abs(sucessor[ID].column - end[1])
        elif strategy == 5:
            value = cost + h
        node.value = value
        output = f'{[node.ID]}({node.cost},{node.state},{node.father_id},{node.action},{node.depth},{node.h},{(round(node.value))})'
        print(f'+ {output}')
        list_nodes.append(node)
    return list_nodes
```

4. Manual de Usuario



Al ejecutar el programa nos aparecerá el siguiente menú:

1. Generar laberinto
2. Cargar laberinto
3. Generar Problema
4. Resolver Problema
5. Salir

Introduzca opción en el rango [1-5]:

Nos pide que introduzcamos un número del 1 al 5 dependiendo de la opción de programa que queramos usar.

4. Manual de Usuario

OPCIÓN 1

Si introducimos un 1 se muestra lo siguiente:

Introduzca opción en el rango [1-5]:

1

Generando laberinto...

Introduzca el número de filas: 5

Introduzca el número de columnas: 5

Nos pide que introduzcamos las filas y las columnas del laberinto que queremos generar. Una vez introducidas se genera el laberinto y se guarda el archivo .json generado en el directorio raíz.

4. Manual de Usuario



OPCIÓN 2

Introduzca opción en el rango [1-5]:

2

1.Cargar puzzle

2.Cargar problema

Introduzca opción en el rango[1-2]

Nos pide que introduzcamos un número del 1 al 2 dependiendo de si lo que queremos cargar es un puzle o un problema.

Si introducimos en este caso la opción 1:

Introduzca opción en el rango[1-2]

(1) Nos pide que escribamos la ruta donde se encuentra el laberinto a cargar, una vez especificado se pinta y el resultado se guarda en el directorio actual (puzzles).

Si introducimos en este caso la opción 2:

Introduzca opción en el rango[1-2]

(2) Nos pide que escribamos la ruta del archivo .json que contiene el problema, al proporcionársela se carga el problema y se guarda en el directorio actual (problemas).

4. Manual de Usuario

OPCIÓN 3

Introduzca opción en el rango [1-5]:

3

Creando problema en laberinto

Introduzca ruta del fichero:puzzle_5_5.json

Problema generado, json guardada en el directorio actual.

Nos pide que introduzcamos la ruta del fichero que contiene el puzle para poder así generar el problema relacionado con este laberinto.

Este problema generado se guarda en el directorio raíz.

4. Manual de Usuario

OPCIÓN 4

Si introducimos un 4 se muestra lo siguiente:

Resolviendo laberinto...

Introduzca ruta del fichero:problema_5x5.json

ESTRATEGIAS DISPONIBLES PARA RESOLVER LABERINTO:

1. Anchura
2. Profundidad
3. Coste uniforme
4. Voraz
5. A estrella


Introduzca opción en el rango [1-5]

Lo primero que se nos pide es que indiquemos la ruta del fichero donde se encuentra el problema a resolver, una vez introducido y comprobado que el archivo existe, se nos muestran las distintas estrategias para resolver dicho problema. Si introducimos un número del 1 al 5 se nos resolverá el laberinto con la estrategia indicada.

OPCIÓN 5

Finalmente, si introducimos un 5 el programa se cerrará.

4. VALORACIÓN



Alejandro Ruiz Aranda: al principio me parecía una práctica compleja ya que no teníamos mucha idea de como empezar y estábamos un poco perdidos, pero al final creo que tras ir comprendiendo los conceptos que se enseñan en la asignatura todo fue cobrando sentido. También destacar que me ha parecido interesante trabajar con los ficheros json ya que no los había utilizado hasta ahora y me parece algo muy útil. En cuanto al equipo creo que ambos hemos hecho un buen trabajo y le hemos dedicado bastantes horas.

Jesús Santiyán Reviriego: Es una práctica difícil y en la que en muchos momentos no sabíamos cómo avanzar, quizá no sea la mejor metodología la que se está usando, pero desde luego creo que ha sido efectiva ya que hemos podido aprender todo lo que se enseña en la asignatura tanto a la hora de plantear los problemas como en resolverlos. En cuanto al equipo, los dos hemos trabajado bastante pero quiero resaltar a mi compañero Alejandro que ha tenido más constancia y sabido seguir adelante cuando el trabajo se complicaba.