ensia The National School of Artificial Intelligence المدرسة الوطنية العليا للذكاء الاصطناعي

Introduction to Artificial Intelligence

Lab 5 (Week 5) -Search Algorithms: Part 1 - Problem Formulation and Uninformed Search

2023 - 2024

02 March 2024

Objectives

- Using OOP to formulate the given problem according to the 5 aspects seen in CHAPTER 3.
- Implementation of the uninformed search strategies using the graph search general algorithm.

Overview:

The aim of this lab is to implement the graph search general procedure outlined in Algorithm 1. The idea is to have a powerful general algorithm able to receive a problem and specified strategy as inputs for solving the given problem and produce a solution to the problem. The algorithm dynamically determines the appropriate data structure (`LIFO, etc.`) based on the specified strategy.

Algorithm 1: Graph search general algorithm

```
Input: Problem, Extra ;        /* Extra for other parameters; data_struct, depth... */
Output: Solution;
Initialize the frontier using the initial state of the problem and data structure;
Initialize the explored set to be empty;
while True do
    if frontier is empty then
        return failure;
    else
        choose a leaf node and remove it from the frontier;
        if node contains a goal state then
            return node ;                                          /* Solution */
        end
        add the node to the explored set;
        children ← expand the chosen node;
        if child does not exist in frontier and explored set then
            Add child to frontier ;
        end
    end
end
```

Two distinct problems will be explored: The 8-puzzle and Travel planning. For the `8-puzzle` problem, a corresponding Python class is available in a separate file (EightPuzzle.py). This class is represented according to the standard formulation principle (5 aspects: initial_state, goal_test, State_transition_model, actions and path_cost).
Additionally, a Node class is provided to facilitate the representation of the problem through a graph architecture. As explained in CHAPTER 3, a node represents a specific state within the given problem.

```
from collections import deque
class Node:
    def __init__(self, state, parent=None, action=None, cost=0):
        self.state = state
        self.parent = parent # node
        self.action = action # action performed to get to this node
        self.cost = cost     # (incremented with each newly expanded node)
        if parent is None:   # root node
            self.depth = 0   # level in the graph 0 for the root node
        else:
            self.depth = parent.depth + 1 # parent level + 1

    def __hash__(self):
        ..... # Fill in the gap

    def __eq__(self, other):
        ..... # Fill in the gap
```

Your mission

Your mission in this lab is to explore the code snippets and the attached Python file to accomplish the following two tasks:

Task 1

- Implement the general Graph search algorithm outlined in `Algorithm` 1.
- Initialize an 8-puzzle `object` and shuffle it.
- Solve the 8-puzzle problem using the general search algorithm employing the **Breadth first search (BFS)**
- Solve the same problem (with the same initial configuration) using **Depth first search (DFS)**.
- What do you notice?

Task 2

The objective of the Travel Planning problem is to find a path from an initial position or state to a goal state (a city in this context). Let us consider the following example where the goal is to find the path from `Algiers` to `Guelma` respecting the map given in Figure 1.

Figure 1: Map of Algeria.

Your travel planning problem is defined by the following components: **the initial state**, which is `Algiers`; **the goal state** which is *Guelma*; and **the State Transition Model** containing the set of connections between the cities of Algeria. For the purpose of this task, let us consider the following connections (although you have the flexibility to enhance your model with additional connections).

- "Algiers" : "Oran", "Constantine", "Tizi Ouzou", "Bechar", "Setif".
- "Oran": "Algiers", "Tlemcen", "Mascara", "Skikda", "Bejaia".
- "Constantine": "Algiers", "Annaba", "Setif", "Ghardaia".
- "Tizi Ouzou": "Algiers", "Bejaia", "Bouira", "Adrar".
- "Tlemcen": "Oran", "Sidi Bel Abbes", "Mascara", "Guelma".
- "Annaba": "Constantine", "Guelma", "Skikda", "El Oued".
- "Bechar": "Algiers", "Adrar", "Tindouf".
- "Setif": "Algiers", "Constantine", "Ghardaia", "El Oued".
- "Bejaia": "Oran", "Tizi Ouzou", "Bouira", "Tindouf".
- "Mascara": "Oran", "Tlemcen", "Guelma", "El Oued".
- "Guelma": "Tlemcen", "Annaba", "Mascara", "Tindouf".
- "Skikda": "Oran", "Annaba", "El Oued", "Tindouf".

- ”Ghardaia”: ”Constantine”, ”Setif”, ”Adrar”.
- ”Bouira”: ”Tizi Ouzou”, ”Bejaia”, ”Tindouf”.
- ”Adrar”: ”Bechar”, ”Tizi Ouzou”, ”Ghardaia”.
- ”Tindouf”: ”Bechar”, ”Bejaia”, ”Guelma”, ”Skikda”, ”Bouira”.

Considering the constraints above, perform the following tasks:

- Create a class that gives the well-formulated Travel planning problem.
- Initialize a travel planning object.
- Solve the travel-planning problem using the same general search algorithm ***implemented for the previous task***, employing both Breadth-First Search (**BFS**) and Depth-First Search(**DFS**).
- What do you notice ?

Hint

Keep in mind that the Breadth-First Search(BFS) and Depth-First Search(DFS) strategies primarily differ in the data structure employed (`FIFO...`) for exploring frontier nodes (leaf nodes). Certain versions of the Depth-First Search method might involve additional parameters such as depth limit.

Perhaps you should create a customized data structures that utilize consistent methods, such as ”push” for addition and ”pop” for removal, to perform specific tasks uniformly, ensuring the general search algorithm remains truly **GENERAL!** 🤔 (or search for a library that already did that 🙃).