Practice 1: Navigation

Part 1: Uninformed search methods

Artificial Intelligence

2023-2024

Computer Science Department
Universitat Autònoma de Barcelona

1 Introduction

In this practice we will solve a simple navigation problem on a map, given the coordinates of the starting and ending points, we will search for the best route between them. For the sake of simplicity, we will only consider the possible routes between two points on a city metro map. We will try to find the optimal number of stops as the best route. For this purpose, we will use four search methods explained in theory:

1. Depth-first search;
2. Breadth-first search;
3. Uniform-cost search;
4. A* search.

In this first part of the practice we will focus on un-informed and non-optimal search methods.

Warning! In this practice we will save the paths in reverse order from that we did in theory. We will save them in a list where the first element is the root (initial state) and the last element is the leaf node (current state of the path).

2 Needed files

To do the practice, we need to download the following folders:

1. `cityInformation`: it contains the files that represent the city map for the whole city (Lyon_bigCity) and a simplified one (Lyon_smallCity). In this folder you will have the following files:

(a) `InfoVelocity.txt` contains the information of the speed at which each metro line travels.
(b) `Stations.txt` contains information of stations, for each station, you have ID, name, line number and coordinates where it is located.
(c) `Time.txt`, is a table that shows how long it takes to get from one station to another. If the table value is zero, it means that there is no connection between two stations.
(d) `Lyon_city.jpg`: the Lyon city metro map.

2. `Code`: it contains some python files with useful functions, and the files you need to code your functions. In this folder you will have the following files:

   (a) `utils.py`: contains some functions hat can be helpful to understand the code.
   (b) `SubwayMap.py`: contains the two main classes to work with: Map (contains all the information about the city) and Path (stores the information about a route or set of stops).
   (c) `TestCases.py`: contains the code you can use to test whether your coded functions are given the expected results.
   (d) `SearchAlgorithm.py`: this is the file where you will program all the functions for this practice.

3 Preliminaries

Before starting to code it is highly recommended to understand the two classes we will be working with: Map and Path. To properly perform your work, you need to understand the variables you can get from each class and how to call them. Thus, we recommend you to open the `SubwayMap.py` file before you start programming and identify which variables and functions you will need to call during practice. Therefore, to confirm that you have understood these classes and before to move on, you should be able to:

1. Access all the information of a specific stop (Line number, coordinates and speed)
2. Access the connections of a metro stop and their value.
3. Understand and be able to create a Path.

4 What do we have to code?

In the first part of this practice, you will program the `Breadth-first` and `Depth-first` search algorithms (BFS, DFS) that have a very similar structure. Only the order in which the nodes are processed varies. To code them you will need to create two previous functions.

4.1 Preliminary functions: `Expand` and `Remove_cycles`

`Expand`**.** function that has as input parameter a parent Path and the Map. It returns a list of Paths, where each one is the parent Path with an additional node. These additional nodes are those connected to the parent Path in the Map.

Example

Input: `expand([14,13,8,12], MAP)`

Output: `[[14,13,8,12,8], [14,13,8,12,11], [14,13,8,12,13]]`

`Remove_cycles`**.** function that has as input parameter a list of Paths. It deletes the Paths in which the last station has been previously visited in the same path, i.e. a cycle. In this way we make sure that we do not fall into endless loops where we always move through the same seasons. It returns the entry list without the Paths with cycles.

Example

Input: `remove_cycles([[14,8,12,8], [14,8,12,11], [14,8,12,14]])`

Output: `[[14,8,12,11]]`

4.2 Depth-first Search Algorithm (DFS)

We have to implement the following functions

`Insert_depth_first_search`**:** function that have as input parameter the list of child Paths that we have calculated and the Queue of the list of Paths that we are exploring and returns the union of these two lists according to the principle of in-depth search.

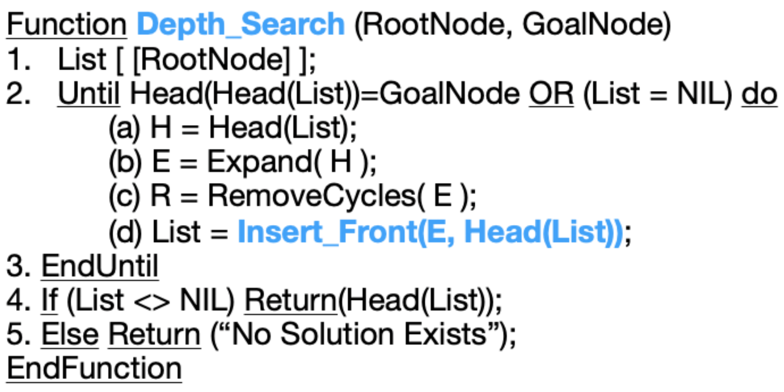
`Depth_first_search`**:** Given a starting station, a final station, and the city map, this function is able to find a route between the two stations using the algorithm shown in Figure 1 but watching the order in which the roads are kept.

4.3 Breadth-first Search Algorithm (BFS)

We have to implement the following functions

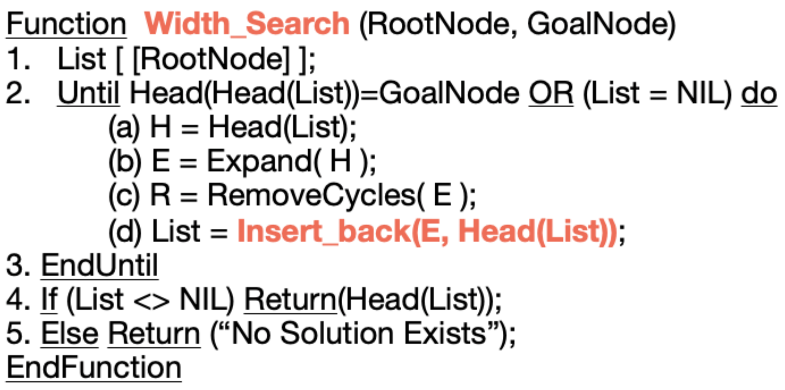
`Insert_breadth_first_search`**:** Function that takes as input the list of Paths children that we have calculated and the Queue of the list of Paths that we are exploring and returns the union of these two lists according to the principle of search in width.

`Breadth_first_search`**:** Function that given an origin station, a final station, and the city map is able to find a route between the two stations using the algorithm shown in Figure 2 but watching the order in which the roads are stored.



```
Function Depth_Search (RootNode, GoalNode)
1.  List [ [RootNode] ];
2.  Until Head(Head(List))=GoalNode OR (List = NIL) do
        (a) H = Head(List);
        (b) E = Expand( H );
        (c) R = RemoveCycles( E );
        (d) List = Insert_Front(E, Head(List));
3. EndUntil
4. If (List <> NIL) Return(Head(List));
5. Else Return ("No Solution Exists");
EndFunction
```

Figure 1: Pseudocode of the In-depth Search Algorithm



```
Function  Width_Search (RootNode, GoalNode)
1.  List [ [RootNode] ];
2.  Until Head(Head(List))=GoalNode OR (List = NIL) do
        (a) H = Head(List);
        (b) E = Expand( H );
        (c) R = RemoveCycles( E );
        (d) List = Insert_back(E, Head(List));
3. EndUntil
4. If (List <> NIL) Return(Head(List));
5. Else Return ("No Solution Exists");
EndFunction
```

Figure 2: Pseudocode of the Width Search Algorithm

4.4 A last function: `distance_to_stations`

How do we find the best route if the user is not right next to a subway stop? This last function will help us to compute the distance between a point in the map and all the metro stations of the map.

`distance_to_stations`. Function that takes as input a list with two values $[X, Y]$ indicating where the user is in the Map, and returns a dictionary with all the metro stations IDs as keys, and the distances between the position of the user and all the stations as values. This dictionary is sorted firstly by the distance (ascendent order), and secondly by the station ID (ascendent order).

Example:

Call: `distance_to_stations([100,200],MAP)`

Output: `{8:10.0, 12:10.0, 13:10.0, 9:24.76, 7:58.73, 14:60.03, 11:66.48, 6:93.94, 1:125.42, 2:149.45, 5:149.45, 10:149.45, 3:151.61, 4:177.56}`.

5 Submission of Part 1

To evaluate this first part of the practice you must upload your `SearchAlgorithm.py` file to the moodle room (cv.uab.cat). The file should contain your NIA in the authors variable and your group in the variable group (at the beginning of the file).

Delivery must be made before 03/February/2023 at 23:55.

Warning! It is important that you consider the following points:

1. Code correction is done automatically, so be sure to upload the files with the correct nomenclature and format.

2. The code is subject to automatic plagiarism detection during correction.

3. Any part of the code that is in the functions of the `SearchAlgorithm.py` file cannot be evaluated, so do not modify anything outside of it.

4. To prevent code from looping, there is a time limit for each exercise, so if your functions take too long it will count as an error.