Week 02 Prac – Zebra Puzzle

Last modified on Friday, 25 February 2022 by f.maire@qut.edu.au

The first half of the unit is devoted to search algorithms. In AI, search problems can often be reduced to the exploration of a graph (often a tree). For non trivial problems, the associated search trees can grow very large. These graphs sometimes can have an infinite number of nodes. However, these graphs or trees do not have to be computed explicitly to be helpful. For example, you can associate a tree with the nested loops of the zebra puzzle Python code. Each level of the tree corresponds to the assignment of a value to a variable of the CSP.

In Exercises 2 and 4 of this prac, you will see two tricks to efficiently explore the search tree. The first trick is to test the constraints as early as possible in order to prune large parts of the search tree (Exercise 2). The second trick is to use Python ***generators*** instead of ***lists***.

The last two optional exercises are about graph traversal and problem encoding. These exercises let you consider a data structure for the representation of paths in a road network. Graph exploration is a standard tool of AI search. Trees and graphs are key data structures in AI. You should make sure that they become your friends!

Getting started

- If you have not done it already, read the Python "Getting Started" tutorial https://docs.python.org/3/tutorial/index.html

- In particular, before attempting the exercies below, read first about **iterators**, **generators** and **generator expressions** at https://docs.python.org/3/tutorial/classes.html#iterators
  Generators are useful because they can save time and memory compared to the explicit creation of list in scenarios where you may not need all the elements of the list. In other situations, you may need to create on-demand objects. Generators are quite common in machine learning libraries.

Exercise 1

- Download the file *zebra_puzzle.py*
- Relax the problem by commenting out some of the constraints in the function *zebra_puzzle_naive*. How does this affect the running time? What can you say about the return solution?

Exercise 2

The shortcoming of the function "zebra_puzzle_naive" is that it waits until all variables have been assigned to check whether a candidate solution satisfies the constraints. If the *Englishman* variable is not assigned the same value as the *red* variable, we should know that the partially constructed candidate solution is doomed when we enter the body of the second *for* loop. At that point, we have already assigned *red* and *Englishman* to some houses (possibly the same). If we didn't assign them to the same house, we should not bother considering the other attributes. Therefore, we should test each constraint as early as possible.

With this insight, copy and modify the provided code (create a new function) so that the search time is improved. Compare the running time of the two different versions of the search.

Exercise 3

- Consider the *generator expression* g = (i*i for i in range(5,10))
- What are the first 3 values generated? hint: try next(g)

Exercise 4

- What is returned by the function *zebra_puzzle_gen*?
- How do you use the object returned by the function *zebra_puzzle* ?
  hint: see previous exercise
- Leveraging the same idea as for Exercise 2, make your code more efficient.

The last two exercises are optional modelling exercises, (not programming exercises).

These exercises are challenging. Feel free to ignore them (not examinable).

Optional Exercise 1

Most current navigation systems generate global, metric representation of the environment with either obstacle-based grid maps or feature-based metric maps. While suitable for small areas, global metric maps have inefficiencies of scale. Arguably, topological mapping is more efficient as it concisely represents a partial view of the world as a graph. In this Exercise, you will show that when the graph is without cycles, a map is superfluous for navigation purpose.

When following the instructions of a navigation GPS device, the driver receives instructions of the form "*at the next roundabout, take the third exit*". This command format is well suited for logging the itinerary followed by a mobile agent in an environment that has a graphical topology like a road-network or the corridors of a building.

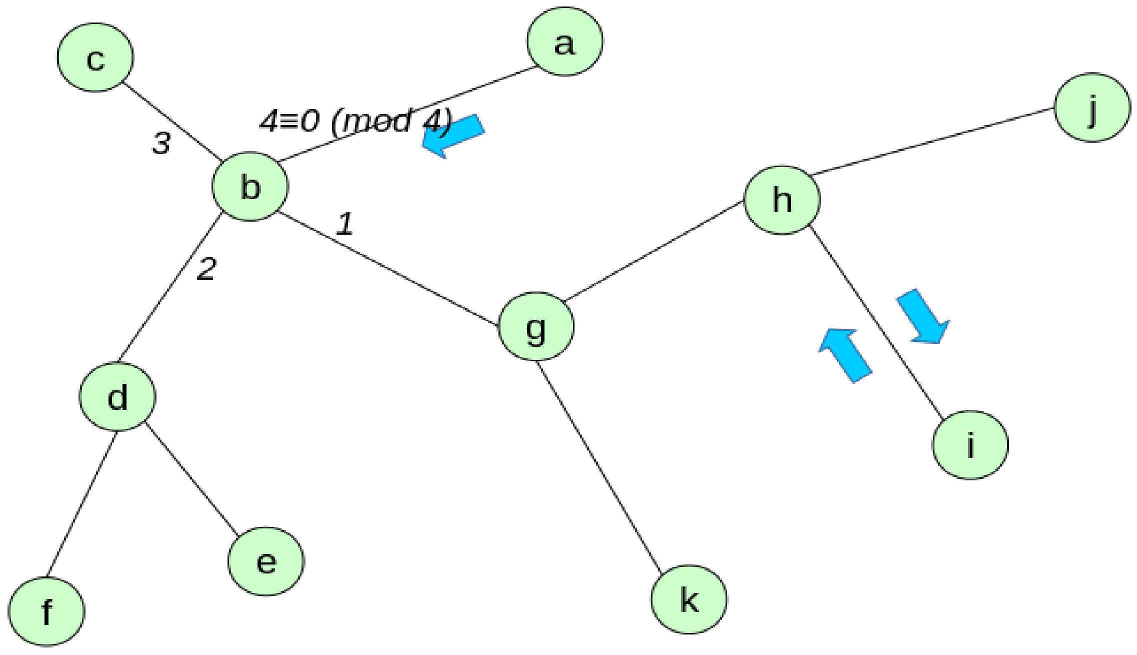
The figure below illustrates such an environment. The agent/robot traverses the graph following its edges and using its vertices like roundabouts. In order to indicate the direction the robot is facing on an edge, we specify the location of the robot by an arc. We adhere to the standard terminology of Graph Theory, and reserve the term **arc** for an edge that has been given an orientation.

The blue arrow on the left of arc **ab** represents the position of the robot going from Node **a** to Node **b**.

The navigational route instructions from the arc $a \rightarrow b$ as the starting position, to the arc labeled $h \rightarrow i$ as the destination, would sound as follows if told by a GPS device:

1. "drive to the next the intersection"
2. "turn first left"
3. "drive to the next the intersection"
4. "turn first left"
5. "drive to the next the intersection"
6. "turn second left"
7. "drive to the next the intersection"
8. "arrive at your destination"

A less verbose representation would code the route into the integer sequence (1, 1, 2)

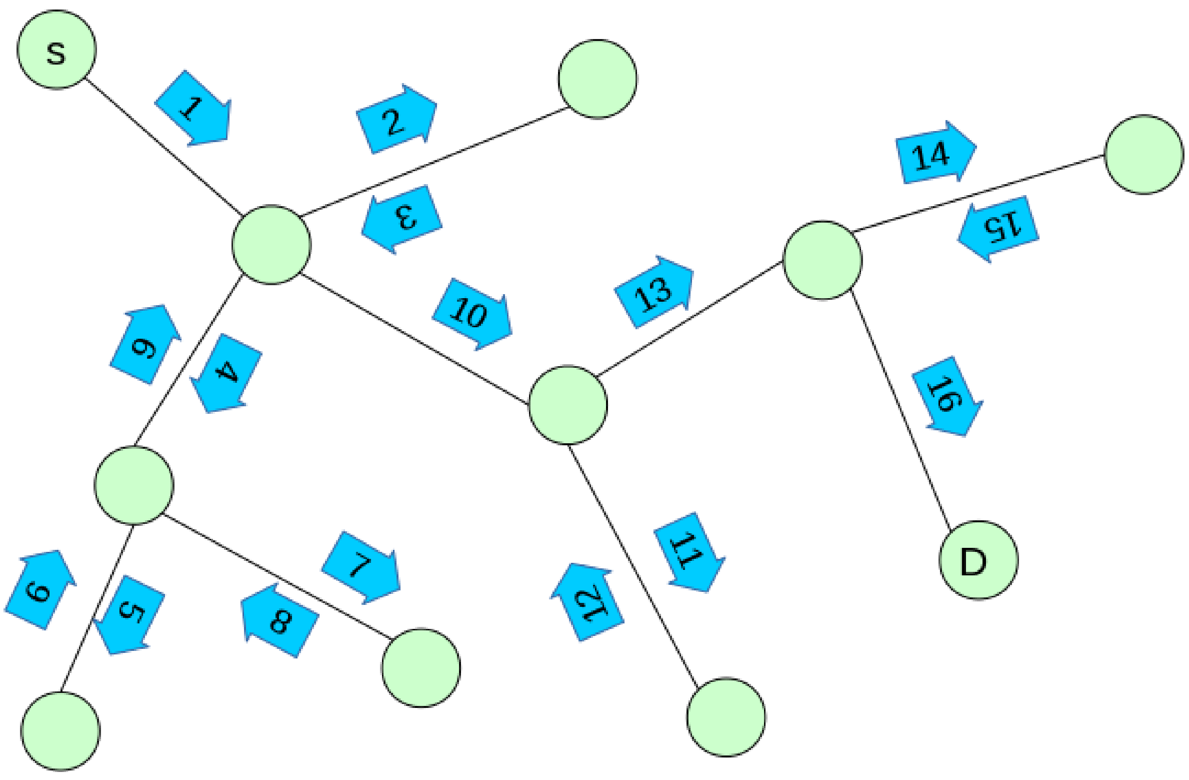


- Observe that when arriving at a node with n edges, the commands c and c' have the same effect if and only the difference c-c' is a multiple of n.
- What is the effect of the command c=0 ?
- Compare the effect of the commands -c and n-c
- What are the possible commands for a U-turn ?
- What happens at a leaf node (node with exactly one incident edge), does the value of c matters at a leaf?
- Given a forward route (c1 , c2 , . . . , ck) from arc alpha to arc beta, provide a formula for a return route from beta to alpha.

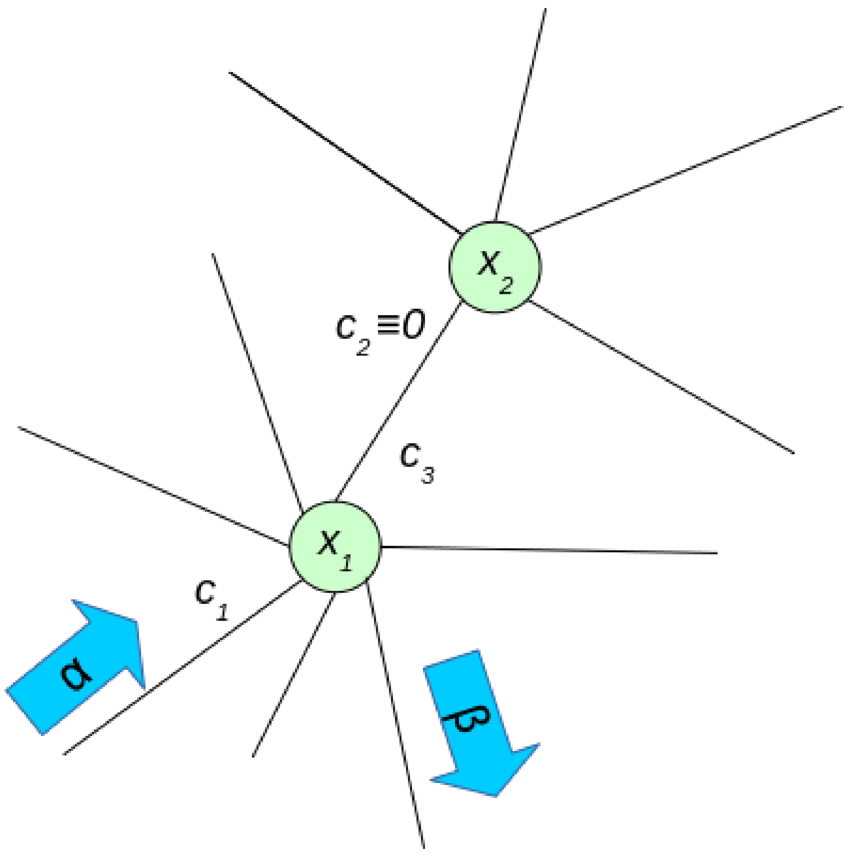
Optional Exercise 2

This exercise continues Optional Exercise 1 but is much harder. Consider it as a brain-teaser! Skip it if you are short on time.

When two routes c and c' start with the same arc α and end with the same destination arc β, we say that the routes are ***equivalent***. For example, in the Figure below, the route from Node S to Node D (1, 0, 2, 2, 0, 2, 0, 2, 3, 2, 0, 2, 1, 0, 1) is equivalent to the direct route (2,1,2).



Design an algorithm to reduce a route to its shortest equivalent route. Hint: Look at the figure next page. Observe that whenever c2 is a multiple of n2 where n2 is the degree of the node x2 , the sequence (c1 , c2 , c3) has the same effect as the singleton sequence (c1 + c3).



x_2
$c_2 \equiv 0$
c_3
x_1
c_1
α
β