# sheet03-programming

May 11, 2024

Exercises for the course Artifical Intelligence Summer Semester 2024

G. Montavon Institute of Computer Science Department of Mathematics and Computer Science Freie Universität Berlin

Exercise Sheet 3 (programming part)

[1]:
```
%matplotlib inline
import numpy
import utils
```

## 0.1 Exercise 3: Depth-First-Search (10 + 20 P)

In this exercise, we build on the results from last week's programming exercise, where our goal was to travel from Arad to Urziceni in seven days. The map depicting the cities and the roads is shown here (from the book by Russell & Norvig):

The method we have used last week consisted of first reducing the associated CSP to an arc-consistent one, and to perform exhaustive search on all 2400 possible combinations. While the enumeration could run quickly on this simple problem, the solution is not scalable for larger problems.

In this exercise, we start from the arc-consistent CSP from last week, with the reduced domain for each variable (in temporal order) given below.

[2]:
```
domains = utils.domains

nbdays = len(domains)
```

This time, we would like to find all possible solutions using a tree search procedure. We consider depth-first-search with variables ordered from day 1 to day 7 and use backtracking to ensure we only visit nodes that satisfy local constraints (i.e. we only move to a city on the next day if the latter is reachable from the current location). Because there are no further constraints than local connectivity in the CSP, we are certain that all explored leaves of the search tree are actual solutions of the CSP.

**Task:**

- Complete the backtracking depth-first-search implementation below. The function receives a node as input. The node is a string representing the partial assignment, e.g. 'AAS' for Arad -> Arad -> Sibiu -> ? -> ? -> ? -> ?. When the search reaches a leaf of the tree (i.e. when

we get a complete assignment), the full variable assignment should be printed. If the node is not a leaf, one should proceed further into the tree by calling DFS recursively.

[36]:
```
names = utils.names
edges = utils.edges

def DFS(node):
    depth = len(node)
    if depth == nbdays:
        print(' -> '.join([names[city] for city in node]))

    else:
        neighbors = edges[node[-1]]
        valid_cities = [city for city in domains[depth]]
        for city in valid_cities:
            if city in neighbors:
                DFS(node + city)
```

The implementation can now be tested by calling DFS with the partial assignment corresponding to starting in Arad.

[37]:
```
DFS('A')
```

```
Arad -> Arad -> Arad -> Sibiu -> Fagaras -> Bucharest -> Urziceni
Arad -> Arad -> Sibiu -> Fagaras -> Bucharest -> Bucharest -> Urziceni
Arad -> Arad -> Sibiu -> Fagaras -> Bucharest -> Urziceni -> Urziceni
Arad -> Arad -> Sibiu -> Fagaras -> Fagaras -> Bucharest -> Urziceni
Arad -> Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest -> Urziceni
Arad -> Arad -> Sibiu -> Sibiu -> Fagaras -> Bucharest -> Urziceni
Arad -> Sibiu -> Arad -> Sibiu -> Fagaras -> Bucharest -> Urziceni
Arad -> Sibiu -> Fagaras -> Bucharest -> Bucharest -> Bucharest -> Urziceni
Arad -> Sibiu -> Fagaras -> Bucharest -> Bucharest -> Urziceni -> Urziceni
Arad -> Sibiu -> Fagaras -> Bucharest -> Fagaras -> Bucharest -> Urziceni
Arad -> Sibiu -> Fagaras -> Bucharest -> Giurgiu -> Bucharest -> Urziceni
Arad -> Sibiu -> Fagaras -> Bucharest -> Pitesti -> Bucharest -> Urziceni
Arad -> Sibiu -> Fagaras -> Bucharest -> Urziceni -> Bucharest -> Urziceni
Arad -> Sibiu -> Fagaras -> Bucharest -> Urziceni -> Hirsova -> Urziceni
Arad -> Sibiu -> Fagaras -> Bucharest -> Urziceni -> Urziceni -> Urziceni
Arad -> Sibiu -> Fagaras -> Bucharest -> Urziceni -> Vaslui -> Urziceni
Arad -> Sibiu -> Fagaras -> Fagaras -> Bucharest -> Bucharest -> Urziceni
Arad -> Sibiu -> Fagaras -> Fagaras -> Bucharest -> Urziceni -> Urziceni
Arad -> Sibiu -> Fagaras -> Fagaras -> Fagaras -> Bucharest -> Urziceni
Arad -> Sibiu -> Fagaras -> Sibiu -> Fagaras -> Bucharest -> Urziceni
Arad -> Sibiu -> Oradea -> Sibiu -> Fagaras -> Bucharest -> Urziceni
Arad -> Sibiu -> Rimnicu Vilcea -> Craiova -> Pitesti -> Bucharest -> Urziceni
Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest -> Bucharest -> Urziceni
Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest -> Urziceni -> Urziceni
Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Pitesti -> Bucharest -> Urziceni
```

```
Arad -> Sibiu -> Rimnicu Vilcea -> Rimnicu Vilcea -> Pitesti -> Bucharest ->
Urziceni
Arad -> Sibiu -> Rimnicu Vilcea -> Sibiu -> Fagaras -> Bucharest -> Urziceni
Arad -> Sibiu -> Sibiu -> Fagaras -> Bucharest -> Bucharest -> Urziceni
Arad -> Sibiu -> Sibiu -> Fagaras -> Bucharest -> Urziceni -> Urziceni
Arad -> Sibiu -> Sibiu -> Fagaras -> Fagaras -> Bucharest -> Urziceni
Arad -> Sibiu -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest -> Urziceni
Arad -> Sibiu -> Sibiu -> Sibiu -> Fagaras -> Bucharest -> Urziceni
Arad -> Timisoara -> Arad -> Sibiu -> Fagaras -> Bucharest -> Urziceni
Arad -> Zerind -> Arad -> Sibiu -> Fagaras -> Bucharest -> Urziceni
Arad -> Zerind -> Oradea -> Sibiu -> Fagaras -> Bucharest -> Urziceni
```

We can observe that the DFS algorithm returns the same 35 solutions as those found by enumeration in last week's programming exercise. With this search algorithm, we did not have to enumerate all possible 2400 solutions, and therefore, the procedure is much more scalable than enumeration.