



## NAVIGATION

 B4B36ZUI tasks Task 1 - EN - Path Planning Task 1 - CS - Path Planning Task 2 - EN - Nonograms Task 2 - CS - Malované křížovky Task 3 - EN - Game Theory - Gobblet Task 4 - EN - Monkey and banana Task 5 - EN - Grid-world Přednášky / Lectures 2018 Cvičení / Labs 2018 Introduction to Artificial Intelligence (B4B36ZUI,BE4B36ZUI)

## ALL COURSES

Summer 2017 / 2018

Older



## Task 1 - EN - Path Planning

Create an algorithm which finds a path on the road graph of United Kingdom (UK) between two nodes (intersections).

### Assignment

#### Input:

- Graph representing road network in UK
  - The graph edges contain length of the edge and maximal allowed speed
  - The nodes are signed with unique identifier ID.
- Initial node
- Goal node

#### Output:

- A path between the initial and the goal node

**Quality criteria**, listed in descending priority:

- The algorithm fulfills all parts of the assignment (classes name and location, using given structures and the method add() in the open list - see below)
- The path is correct, i.e., leads from origin to destination, the path is connected, i.e., two subsequent edges have a common node and the edges are sorted correctly.
- The found path minimizes transport time between initial and goal state
- The number of expanded nodes is minimal


#### Grading

- Algorithm fulfills the assignment completely
- The path is correct, i.e., leads from origin to destination, the path is connected, i.e., two subsequent edges have a common node and the edges are sorted correctly. [+1b]
- The found path minimizes transport time between initial and goal state [+2b]
- The number of expanded nodes is minimal [+0-3b]

### Submission

- Date/time:** 25/03/2018 23:59:59
- Content:** compressed content of Java package `student` (pack only .java source files, no subfolders please)

### Implementation details

- Use the codebase available  [HERE](#).
- Implement class `Planner` in package `student` (implementation of the interface `PlannerInterface` )
- Create class `OpenList` which extends class `AbstractOpenList` in your package. **For adding the items to OpenList use only the method add(T item)**
- If you modify any item in OpenList (you have added it to the OpenList earlier, you just recompute its cost), then you don't need to call the **add** method (however, you can - the Counter value is going to be increased though).
- The algorithm is going to return **null** if the path does not exist

### Some code you can use

- The class **RoadGraph** allows you to access the graph and contains a set of method which you can use to move along the graph
- The class **GraphEdge** contains the methods getAllowedMaxSpeedInKmph() and getLengthInMetres() which return the maximal allowed speed in km and length in meters respectively.
- The class **Utils** gives you the metric for counting the distance between any two nodes in the graph.
- The class **PlannerExecutor** allows the body of the program where you can test your algorithm.

### Example results

#### Solution 1

- Origin node ID = 13823646
- Destination node ID = 188755778
- Plan length [km]: 932.8542488743733
- Time to travel [hrs]: 12.084881006921961

#### Solution 2

- Origin node ID = 26746953
- Destination node ID = 1037726044
- Plan length [km]: 664.3940259558422
- Time to travel [hrs]: 8.002850863827378

#### Solution 3

- Origin node ID = 243081231
- Destination node ID = 21728749
- Plan length [km]: 560.2105619356079
- Time to travel [hrs]: 7.149357704368445

<b>Table of Contents</b>	–
<a href="#">Task 1 - EN - Path Planning Assignment Submission</a>	
<a href="#">Implementation details</a>	
<a href="#">Some code you can use</a>	
<a href="#">Example results</a>	