Uninformed Search - Prac Sheet SOLUTION

Last modified on Friday, 8 March 2019 by f.maire@qut.edu.au
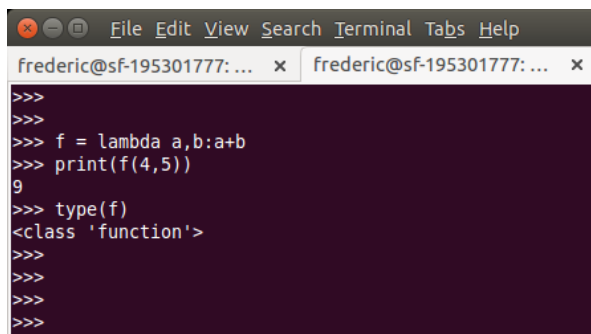
In this prac, you will implement and test some of the search functions seen during the Week 02 lecture on *uninformed search.*

The first exercise introduces *lambda function,* a commonly used tool in Python software libraries.

The remaining exercises dive into search algorihms. It will take you some time to get your head around the different classes defined in the provided files. But it is worth the effort! We build a generic search module that can be applied to a wide range of problems without having to change any line of the generic search functions. This library will be exploited in your first assignment. You should focus on the interface between the generic search functions and the methods of the class *Problem*. We will see several examples of classes derived from the *Problem* class to help you understand how the derived classes should be defined. When you are faced with a new search problem, your task will boil down to derived a new Problem subclass.

Exercise 1 - Lambda functions and list comprehension

Python *lambda functions* are similar to anonymous functions in Matlab. They accept inputs and return outputs, just as standard functions do. However, they can contain only a single executable statement. Small anonymous functions can be created with the *lambda* keyword. For example, the function *lambda a, b: a+b* returns the sum of its two arguments.

For more information on lambda expressions and their applications, visit
https://docs.python.org/3/tutorial/controlflow.html#lambda-expressions

- Create a lambda expression to compute the square of its argument

>>> f = lambda x: x2**

- Using lambda-expression and list comprehension, generate the list of the squares of the even integers in the interval [30,50].

L = [f(x) for x in range(30,50,2)]
or
L = [f(x) for x in range(30,50) if x%2==0]

Exercise 2

- Open the file *search.py,* and browse through the code. Ask yourself how the classes are related. Map them to the slides of Week 02 that contain pseudo code. In particular, check the attributes of the *Node* and *Problem* classes.

- Complete the functions marked with "INSERT YOUR CODE HERE"

See file *search_SOLUTION .py*

Exercise 3

- By editing the *__main__* function of *sliding_puzzle.py* , experiment with the size of the grid and the number of random moves N used for scrambling the puzzle.
  That is, explore the behaviour of *breadth_first_tree_search* by varying the arguments of Sliding_puzzle(nr=3, nc=5, N=15)

- Try to use depth first tree search. What happens? Why?

- Compare the running time of depth limited search and iterative deepening search. Make sure you set the max depth to an appropriate value for your problem!

You should have observed that *BFS tree search* always terminates. However, *DFS tree search* might run forever if it gets stuck in a cycle.