Artificial Intelligence
Practical 3: Uninformed Search

Dr. Manuel Pita

Some Python constructs

Consider the following Python function, and (1) determine what will it return if you run it passing the value 10 for the parameter *max*; (2) what does this function do?

```
def myFunction(max):
    for x in range(1,max):
        if x % 2 == 0:
            return x
```

Now replace the keyword *return* with the keyword *yield*. (1) What does the modified function return for the same parameter *max*= 10? (2) Investigate how to print the contents of the generator object returned by the yield construct. What is the difference between yield and return, and how could you use it in the context of writing code to perform search in a rational agent?

Lists and Sets

Consider the following two lists in Python. How can you obtain a third list c that contains all the elements of a that are not in b?

```
a = [1,2,3,4]
b = [2,3,4,5]
```

Now do a little research on Python sets and how they work. How could you simplify the problem above using sets? And how would you use this when implementing your search algorithm? In addition spend some time with the tutorial at the following URL:
`http://simeonvisser.com/posts/python-3-using-yield-from-in-generators-part-1.html`

Uninformed Search

Consider the following search space representation:

```
s =     {'A': set(['B', 'C']),
         'B': set(['A', 'D', 'E']),
         'C': set(['A', 'F']),
         'D': set(['B', 'G']),
         'E': set(['B', 'F']),
         'F': set(['C', 'E']),
         'G': set(['D'])}
```

If you look at it carefully (I suggest you draw the graph structure) you will realise that it is an adjacency list representation. This was one of the two types of graph (network) representation we used in class to formally capture the structure of a search space. Use the concepts studied in this tutorial to write for code for finding *all possible paths* between some *initial* and *goal* nodes in a search graph like the one represented above. To do this you need to implement the basic search algorithm I described in our lesson.

One property of sets in Python that you will want to explore in detail is called *pop*(). You can do, for example *s.pop*() What does that do? Using sets and the pop() function will help you in this tutorial. You can start working on your function from the definition below. But you can use a variation of this definition. Here you can use an extended representation of q that contains at first the current node, but also the current path to update your paths throughout the execution of the algorithm. It is not strictly required that you use this extended definition of q, but for many of you it will be easier to think about the algorithm having this data structure. Also, you may want to modify the arguments to the function, if necessary for your implementation. The code below is only a starting point. What is important here is that you perform uninformed search, DFS in this case, using an Adjacency List representation of the search space, and that in addition, you store the different paths between initial state and goal state.

```
def dfs(s, init, goal):
    q = [(init, [init])]
    .
    .
    .
```