# COMS 4701 Artificial Intelligence

Homework 1

Due date: February 9th, 2020

## Question 1: PEAS and environments

Alexa is a virtual assistant from Amazon. Alexa is capable of voice interaction and question answering, and can perform a set of functions such as telling the weather, playing music, searching the web, creating lists, order take-out food, or call a taxi. In this question, we will assume Alexa is not connected to other devices.

1. Do a PEAS analysis of Alexa.
2. Describe Alexa's environment, using the properties seen in class.

## Question 2: Search space formulation

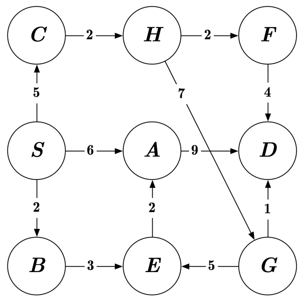
We would like to formulate the following search problem:

"An alien starting from planet Mars wants to collect samples of Earth soil from five different continents North America, Africa, Europe, Asia and Australia. Since he has the very fast flying saucer, he can visit the continents in any order. After completing his mission on earth, he will proceed to any other planet in some other galaxy."

1. What is the initial state?
2. What are the possible states?
3. What are the possible actions?
4. What is the transition model?
5. What is the Goal test?

## Question 3: Search strategies

Consider the following graph. Edges between nodes may only be traversed in the direction indicated by the arrow. We will search the graph with the algorithms we learned, keeping a full explored set as we go. Let $S$ be the start and $G$ the goal. As usual, where an arbitrary choice has to be made, assume that nodes are visited in lexicographical order. The table provides the value of the heuristic function for each node:



| | |
|---|---|
| **A:** | 5 |
| **B:** | 1 |
| **C:** | 3 |
| **D:** | 9 |
| **E:** | 4 |
| **F:** | 0 |
| **G:** | 0 |
| **H:** | 7 |

1. What is the order of visit and path obtained with BFS? Show the queue.
2. What is the order of visit and path obtained with DFS? Show the stack.

3. What is the order of visit and path obtained with UCS? Show the priority queue. If some keys change, show the change.
4. What is the order of visit and path obtained with A*? Show the priority queue. If some keys change, show the change.

## Question 4: Heuristics

Let $h_1$ and $h_2$ be two admissible heuristics. Which of the following heuristics are admissible? Justify your answer.

1. $h(n) = min\{h_1(n), h_2(n)\}$
2. $h(n) = max\{h_1(n), h_2(n)\}$
3. $h(n) = w \cdot h_1(n) + (1 - w) \cdot h_2(n)$ with $0 \leq w \leq 1$

## Question 5: The N-puzzle

The 8-puzzle is a $3 \times 3$ board with 8 tiles (corresponding to $3^2 - 1$), numbered from 1 to 8, and one empty space. The 8-puzzle is a special case of the N-puzzle containing $N = m^2 - 1$ tiles and one empty space. Consider only reachable configurations. Possible reading material (available in courseworks): "Notes on the 15 Puzzle," American Journal of Mathematics, Vol. 2, No. 4 (Dec., 1879), pp. 397-404. Feel free to research the topic in other sources, but cite your references.

1. Provide an upper bound on the size of the state space in the 8-puzzle game. Justify your answer. Examples of justification include drawing a partial search tree, refer to the paper with explanation or provide a more formal justification.

## Programming

### Read Carefully:

- You must name your file `homework1.py`. Any submission that does not follow this naming will not be graded and will receive a zero.
- Do not import any libraries other than `NumPy`.
- A skeleton of each function has been provided to you in `skeleton.py`. You are expected to ONLY write code in the functions and blocks specified. In any case, DO NOT modify the function signatures, or any code that is not specified to be modifiable for any reason. Also, though you may modify the main function to test other cases, do not turn in an assignment with a modified main function. This will be run by our autograder, so any unexpected modifications that make it malfunction will receive a zero. Test cases have been provided in the main function in the skeleton code.
- Only Python 3.x versions will be graded.
- To receive points, make sure your code runs. We recommend using Spyder, Pycharm or Google Colab. They all allow you to download .py files. Be aware that if you write your code in some platforms like Codio and copy and paste it in a text file, there may be spurious characters in your code, and your code will not compile. Always ensure that your .py compiles. Code that does not run will not receive any points.
- **Submission Instructions:** Please submit the `homework1.py` file on Gradescope under the assignment Homework1 Programming.

### Problem 1: Comma Separated Factorials

Write a Python function to return a string of first `k` factorials (starting from `1!`), separated by commas, in reverse order. Use a single for loop to achieve this. You may use a helper function.

For example, for `k = 8`, the output would be the following string: `"40320,5040,720,120,24,6,2,1"`

## Problem 2: List Manipulation

Write a single Python function that takes two lists (`x` and `y`) as inputs for each of the following:

a. Return the resulting list after the following operations: (i) sort `y` by descending order, and (ii) delete the last element of this sorted list.

b. Return the reverse list of `x`.

c. Return the list of unique elements that you get by concatenating `x` and `y`, in ascending order.

d. Return a single list consisting of `x` and `y` as its two elements.

## Problem 3: Set Manipulation

Write a single Python function that takes three sets, `x`, `y`, and `z`, as inputs for each of the following:

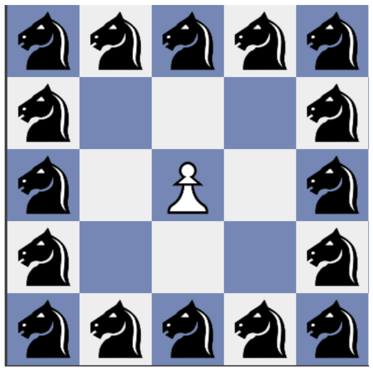
a. Return the union set of all three sets.

b. Return the intersection set of all three sets.

c. Return a set of elements that belong to only a single set out of the three sets.

## Problem 4: Under Attack

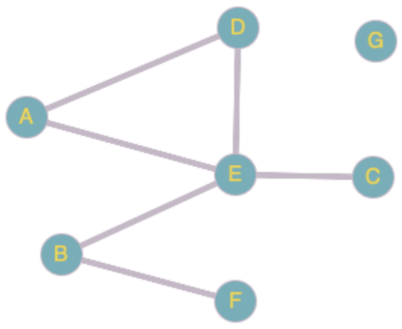
a. Consider the following 5x5 mini-chess board. Assuming that the top left corner is `(0, 0)` and any cell in the board is represented with `(i, j)` where `i` is the row and `j` is column, create a 2D `NumPy` array where black knights are represented with integer `1`, white pawn is represented with `2`, and empty spaces are represented with `0`. Return this array in the associated function.



b. Now suppose you are controlling the white pawn. Write a Python function that takes in **any** such board configuration `x` (i.e. a `NumPy` array) as the only argument and returns a list of indices of the black knights that currently threaten the white pawn (e.g. `[(0, 1), (0, 3), ...]`). This should work for all cases (i.e. any 5x5 board with black knights as `1`s and a single white pawn as `2`). You do not have to worry about the order of tuples in the list. Here is how a knight moves in chess.

## Problem 5: Graphs

Consider the following **undirected graph**.



We can represent this graph with the following Python dictionary, often referred to as an adjacency list:

```
graph = {
   "A":  ["D", "E"],
   "B":  ["E", "F"],
   "C":  ["E"],
   "D":  ["A", "E"],
   "E":  ["A", "B", "C", "D"],
   "F":  ["B"],
   "G":  []
 }
```

Write a single Python function that takes any such dictionary x as input, for each of the following:

a. Return the number of isolated nodes (i.e. nodes without any connections).

b. Return the number of non-isolated nodes (i.e. nodes with connections).

c. Return the unique edges as a list of tuples (e.g. `[("A", "D"), ("A", "E"), ...])`. You do not have to worry about the order of tuples in the list.

d. Return a 2D `NumPy` adjacency matrix representing the same graph. Use a `1` if there is an edge between two nodes, and a `0` if there is not. For example, the `(0, 0)` index of the adjacency matrix will have the value `0` because there is no edge between `A` and `A`, and the `(0, 3)` index will have the value `1` because there is an edge between `A` and `D`.

## Problem 6: Supermarket

Consider the following products and prices in a supermarket:

| | |
|---|---|
| apple | $5.0 |
| banana | $4.5 |
| carrot | $3.3 |
| kiwi | $7.4 |
| orange | $5.0 |
| mango | $9.1 |
| pineapple | $9.1 |

Create a Python class implementing **priority queue** from scratch (i.e. do not use any libraries) with `is_empty()`, `push(element)` and `pop()` methods. Remember that a priority queue is a queue where elements with higher priority are dequeued before elements with lower priority. In case priorities are equal, the elements are dequeued based on their order in the queue. For this problem, assume that the priorities are based on the prices of the products.

For example, let's insert 3 elements into the priority queue, and pop elements until there is none left:

```
pq = PriorityQueue()
pq.push("apple")
pq.push("kiwi")
pq.push("orange")
while not pq.is_empty():
   print(pq.pop())
```