Week 2 - Search Algorithms

Machine Intelligence Laboratory

Teaching Assistants : vigneshkamath@43@gmail.com , sarasharish2000@gmail.com

Not a Teaching Assistant : swapnilnair747@gmail.com

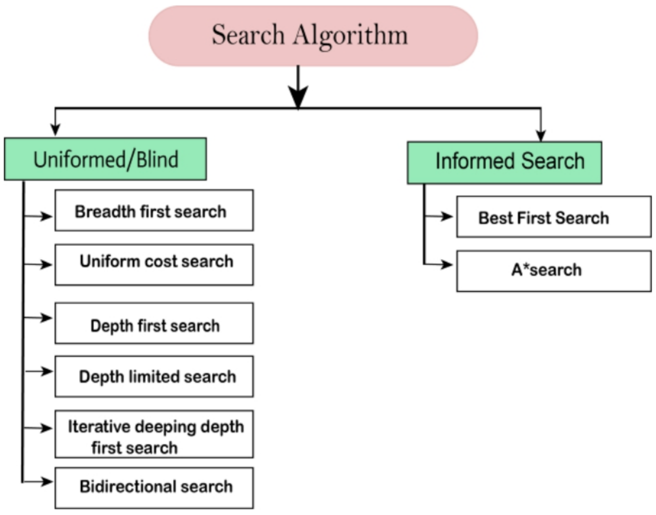
The Introduction :

Search algorithms are algorithms designed to navigate from a start state to a goal state (or states).
In doing so it transitions through multiple intermediate states all of which are a subset of every possible state reachable(called the 'state space').
The set of possible solutions that a system may have is called the search space.

There primarily exist two kinds of search algorithms - Uninformed and Informed

Uninformed Search : Search algorithms that do not have any kind of additional knowledge or intuition about the structure it is working on. This lack of additional information is akin to path-finding in the dark and thus it is called a Blind Search

Informed Search : In informed searches, problem information is available which helps guide them into providing a more solution more efficiently. This additional knowledge is comparable to human intuition and that is why they are also called Heuristic searches .



Here are two excellent resources for learning more about A* searches and its applications in real life:
https://towardsdatascience.com/a-star-a-search-algorithm-eb495fb156bb
https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm

While not mandatory, It is a good idea to know about A* algorithm and its pseudo code before proceeding.

" A Heuristic search does not always find ***the*** most efficient solution, it merely guarantees to find ***a*** solution efficiently "

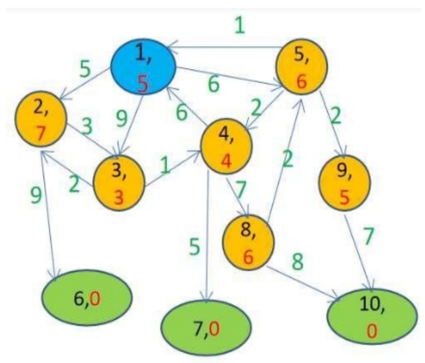
[In layman terms - We choose to find a good solution quickly rather than find the best solution slowly.]

The Brief :

You are provided with the following files :
1. **week2.py**
2. **SampleTest.py**

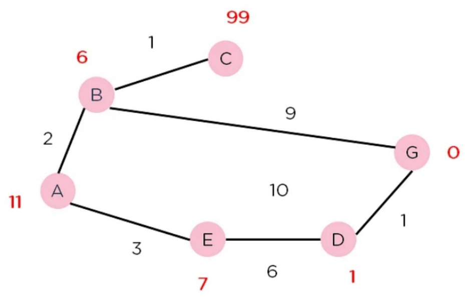
In this assignment you are are required to write two functions **A_star_Traversal** and **DFS_Traversal** which implement the respective algorithms. Verify that they work using the graph given below.

Sample graph for test cases :



Numbers in Black represent Node numbers, numbers in Green represent Cost and numbers in Red represent heuristic values from that node
Start Node is in Blue
Goal Nodes are in Green

Note: This is the same graph that was used in class for A* explanation hence you must be easily able to trace the answer



Letters correspond to their position in the alphabet. A=1, B=2, C=3, D=4, E=5 , however Goal G = 6.

Edge weights are represented along the edge.
Heuristic values are in Red.

Note : This is a very rudimentary graph, should be very simple for you to deduce the path A* picks.

This graph is merely a test case provided to help you check if your implementation is working as it is supposed to, by no means are you to assume that this will be part of the test cases that your assignment will be verified against. Row and Column zero are filled with 0 to make it more convenient check out SampleTest.py for details.

Function skeleton :

INPUT PARAMETERS			
Name	Description	Type	Example
cost	The cost matrix. Cost of moving from i to j is cost[i][j]. Cost of a node to itself is 0 and to an unreachable node is -1.	list of (list of integers)	[[0,1,2], [1,0,1], [3,0,1]]
heuristic	Heuristic data structure for each node. Heuristic at node i is heuristic[i].	list of integers	[1,2,0]
start_point	Start point of the search algorithm	integer	0
goals	List of goal nodes. Length of this list is equal to the number of goal nodes.	list of integers	[2,3]

OUTPUT PARAMETERS			
Name	Description	Type	Example
path	This contains the path that the algorithm has deduced	list of integers	[1,2,3]

Regarding SampleTest.py :

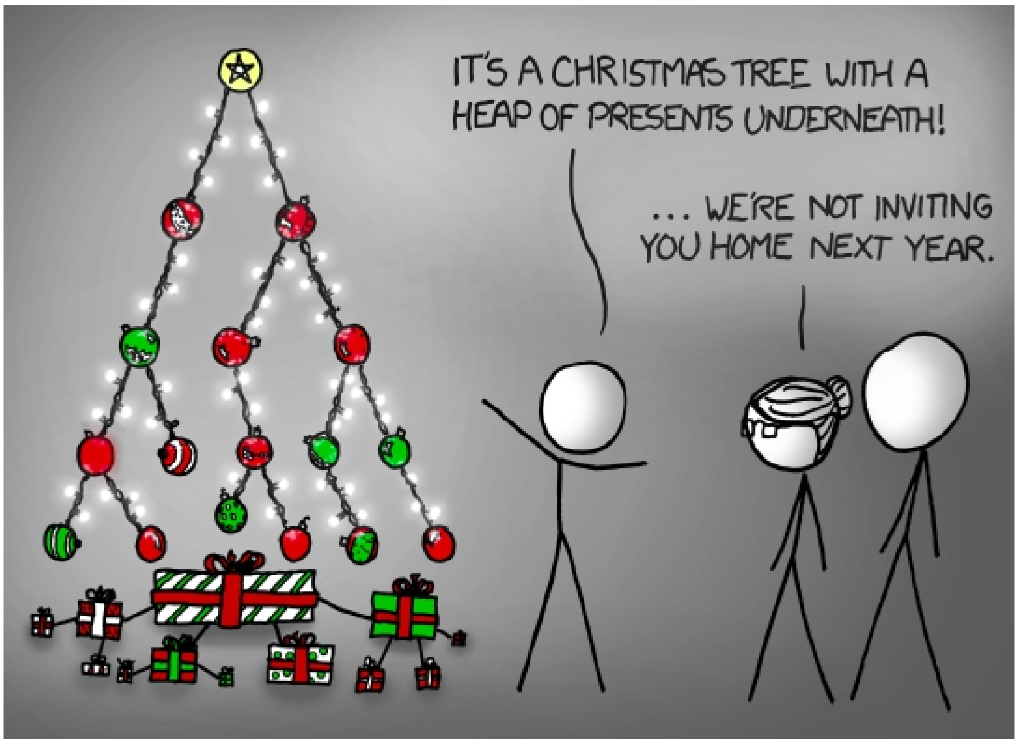
1. This will help you test your code.
2. Note that the test case used here is the same as the graph defined above.
3. Merely passing the test cases does not ensure full marks. Different test cases are present for evaluation.
4. Return [x] if the node which you start on doesn't have any connections.
5. Name your code file as <your_srn>.py

Run the program as :

```
python3 SampleTest.py --SRN <your_srn>
python3.7 SampleTest.py --SRN <your_srn>
```
(in case of any import errors showing up)

Important Points :

1. Please DO NOT alter the function definitions provided to you. Any changes in the skeleton of the function will result in your program failing automatically.
2. You are free to write any helper functions that can be called by the predefined function given.
3. Your code will be auto evaluated by a testing script. Please make sure that your code is well tested and that all edge cases are taken care of.
4. In case of ambiguity, pick nodes in reverse lexicographical order. i.e, Z Y X ... B A
5. The experiment is subjected to **zero tolerance** for plagiarism. Your code will be tested for plagiarism against every code of all the sections and if found plagiarized both the receiver and provider will get zero marks without any scope for explanation.
6. Kindly do not change variables name or other technique to escape from plagiarism, as the plagiarism checker is able to catch such plagiarism.
7. Implement the algorithm in the right way, using appropriate data structures. Inefficient code may lead to Time Limit Exceeded error.
8. Hidden test cases will not be revealed post evaluation.
9. You **may** import libraries that come bundled in with Python3.7.


IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.

Good Luck !