Name	Prithivi Raj	Matric number	A0173060R
Collaborator		Collaborator's matric number	

Instructions:

- *Your solutions for this tutorial must be TYPE-WRITTEN.*
- *Print and submit your solution(s) to your tutor. You can make another copy for yourself if necessary. Late submission will NOT be entertained.*
- *YOUR SOLUTION TO QUESTION* 1 *will be GRADED for this tutorial.*
- *You can work in pairs, but each of you should submit the solution(s) individually.*
- *Include the name of your collaborator in your submission.*

1. The Towers of Hanoi is a famous problem for studying recursion in computer science and recurrence equations in discrete mathematics. We start with N discs of varying sizes on a peg (stacked in order according to size), and two empty pegs. We can move a disc from one peg to another, but we are never allowed to move a larger disc on top of a smaller disc. The goal is to move all the discs to the rightmost peg (see Figure 1).

images/towersofhanoi.pdf

Figure 1: Towers of Hanoi

In this problem, we will formulate the Towers of Hanoi as a search problem.

(a) Propose a state representation for the problem.
(b) What is the start state?
(c) From a given state, what actions are legal?
(d) What is the goal test?

Solution:

(a) Each state for the problem could consist of the current position (tower) that each disk is on as well as the arrangement of the disks in each tower.
(b) All disks are located in the left-most tower and they are stacked in an ascending order of size (from the top).

(c) From each state we can move the top disk (of size k of each tower to one of the other towers IF the top disk of the other tower is $\leq k$.
(d) The goal state would be all the disks stacked on the right-most tower in an ascending order of size (from the top).

2. After taking class in SoC, you are going to UTown for meal. The routes you can choose are shown in Figure 2[1]. The time consumed for each path is shown in the figure and each place (node) is represented by alphabet, e.g., E is for SoC. Now, you are going to search for a

images/map.pdf

Figure 2: Route map in NUS.

route from SoC (E) to Utown (K) by different search algorithms. Assume that the nodes are expanded in alphabetical order when no other order is specified by the search. What order would the states be expanded by each type of search? Stop when you expand K. Write only the sequence of states expanded by each search.

Search Type	List of states
Breadth First	E, I, C, D, F, A, H, G, B, J, L, K
Uniform Cost	E, C, I, D, G, J, K. . .
Depth First	E, I, D, G, J, K . . .

Solution: Filled up table above...

3. The *Uniform-Cost Search* (UCS) algorithm comes under the *best-first* family of search strategies which selects a node n in the *frontier* with respect to a utility function. For UCS, the utility function $g(n)$ is set to be the shortest path from the source node to n.

Figure 3: Algorithm for UCS. It follows the `GRAPH-SEARCH` format comprising of a *frontier* and *explored* sets. The *frontier* is implemented as a priority queue where each element in the queue is ordered as per the utility function $g(n)$ from the source node.

(a) UCS uses a priority queue (*min-heap*) as the data-structure to implement the *frontier*. It also maintains an *explored* set which contains nodes that are already visited. For the

[1] `https://www.google.com/maps`

graph presented in Figure 4, trace the contents in the *frontier* and *explored* sets as the UCS algorithm proceeds (use the algorithm in Figure 3). `Note:` the frontier should maintain the ordering imposed on it.

`images/UCS_graph.png`

Figure 4: Graph for UCS traversal. Start node is S and goal node is E. Each edge is associated with the cost of traversing that edge.

Frontier	Explored
$S(0)$	ϕ
$B(9), A(10)$	S
$D(5), A(10)$	S, B
$A(3), E(5), A(10)$	S, B, D
$C(4), E(5), A(10)$	S, B, D, A
$E(1), E(5), A(10)$	S, B, D, A, C
$E(5), A(10)$	S, B, D, A, C, E

(b) What is the cost of the shortest path from S to **E**? Also mention the path as a sequence of the nodes.

Solution: The path from S to E would be $\underline{[S \rightarrow B \rightarrow D \rightarrow A \rightarrow C \rightarrow E]}$ with a total cost of $\underline{22}$.

4. `GRAPH-SEARCH` is an important traversal framework that removes redundant paths in the search tree.

   (a) Prove that every explored node in GRAPH-SEARCH is connected to the initial state in the search tree by a path of explored nodes.

   (b) The *graph-separation* property states that every path from the source state to an unexplored set has to pass through a state in the frontier. In other words, the *frontier* separates the states of the graph into explored and unexplored regions. Prove that GRAPH-SEARCH holds this property. Give clear explanations to the hypotheses used in your proof.

Solution:

(a) Lets assume that a node N that belongs in the explored set maintained by the GRAPH-SEARCH algorithm, is NOT connected to a the initial state in the search tree by a path of explored nodes.

As per the the algorithm in Figure 3, which follows the GRAPH-SEARCH format, a node can only be added to the set of explored states when selected from the $frontier$. To be added to the $frontier$, a node must be selected from a parent node through the set of possible actions (i.e. A node k must be connected to a parent node which itself must have been in the $frontier$ in order for k to qualify for the $frontier$).

Hence if node N exists within the explored set, it must have been selected from the the $frontier$ and to be in the $frontier$ it must have been connected to a node $N - 1$. Following path of connected-ness backward we arrive at node 1 which would have been in the $frontier$ due to its connection with node 0 (the initial state).

Given that that each state of the application varies with every action taken from every explored node, we can definitively state that any node that exists in the path of explored nodes is connected to the initial state through said path, hence contradicting with our initial assumption.

(b) Suppose that GRAPH-SEARCH does not hold the property of *graph-separation*.

If our initial assumption is true, there exists a node K within the explored region (bounded by the $frontier$) that has in-fact not been explored. Therefore there is a possibility that the goal state that is represented by the exploration of node "E" could be impossible to reach since there is a chance that E exists within the explored set but has not actually been visited.

This would violate the fact that GRAPH-SEARCH is a *complete* algorithm (when the number of nodes are not infinite), as the inability to "reach" node E would mean that there is no solution. Therefore the assumption that GRAPH-SEARCH does not hold the property of *graph-separation* is not true.