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).

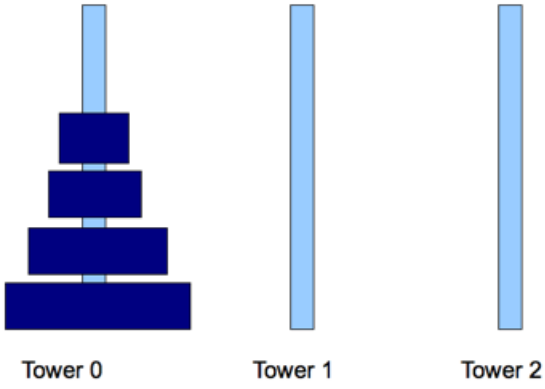

Figure 1: Towers of Hanoi

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?

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

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

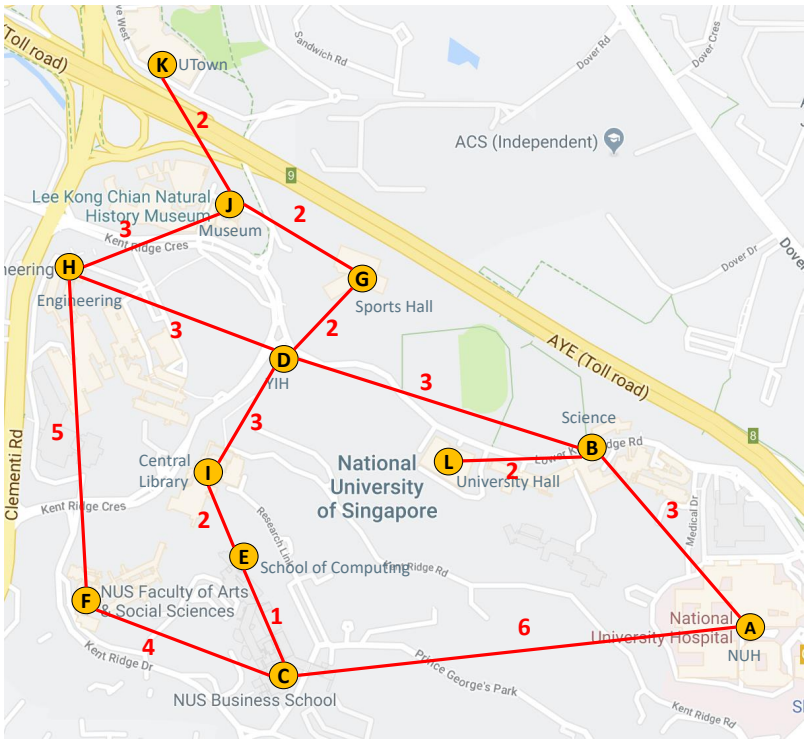

Figure 2: Route map in NUS.

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, . . .
Uniform Cost	E, . . .
Depth First	E, . . .

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.

   (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 graph presented in Figure 4, trace the contents in the *frontier* and *explored* sets as the

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

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.

UCS algorithm proceeds (use the algorithm in Figure 3). `Note:` the frontier should maintain the ordering imposed on it.

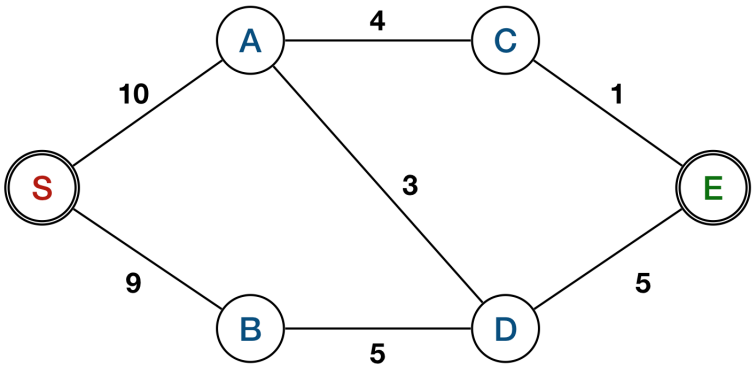

Figure 4: Graph for UCS traversal. Start node is S and goal node is E. Each edge is associated with the cost of traversing that edge.

Frontier	Explored
$S(0)$ $B(9), A(10)$	ϕ S

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

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.