Instructions:

- *Your solutions for this tutorial must be TYPE-WRITTEN.*
- *Submit your PRINTED solution(s) to your tutor. You can make another copy for yourself if necessary. Late submission will NOT be entertained.*
- *YOUR SOLUTION TO QUESTION* 4 *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).

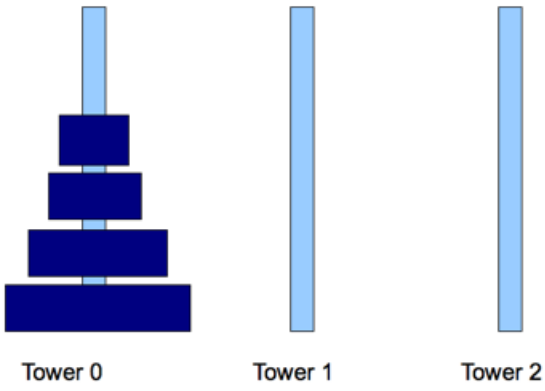

Figure 1: Towers of Hanoi

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?

Solution:

(a) One possible state representation would be to store three stacks, corresponding to which discs are on which peg. If we assume that the N discs are numbered in order of increasing size $1, \ldots, N$, then we can represent each peg as a stack of integers corresponding to which discs are on that peg.

(b) $([1, , N], [\,], [\,])$

(c) We can pop the first integer from any stack (i.e., peg) and push it onto another stack (peg), so long as it is smaller than the integer currently at the top of the stack being pushed to (i.e., peg being moved to).

(d) $([], [], [1, , N])$

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

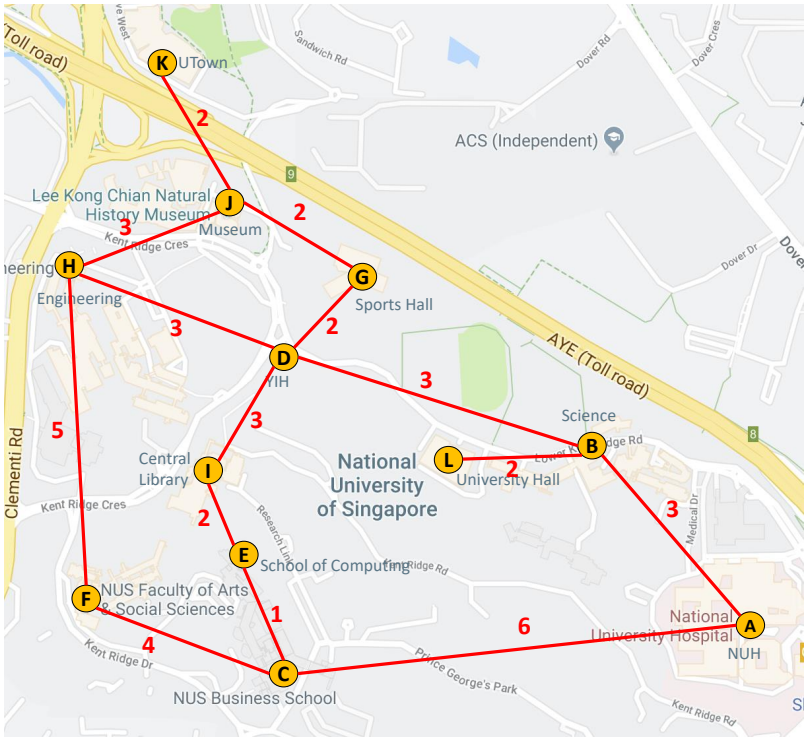

Figure 2: Route map in NUS.

a route from SoC (E) to Utown (K) by different search algorithms. When no other order is specified by the search, assume that the nodes are expanded in alphabetical order. 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.

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

Search Type	List of states
Breadth First	E, . . .
Uniform Cost	E, . . .
Depth First	E, . . .

Solution:

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

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.

```
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.

(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

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

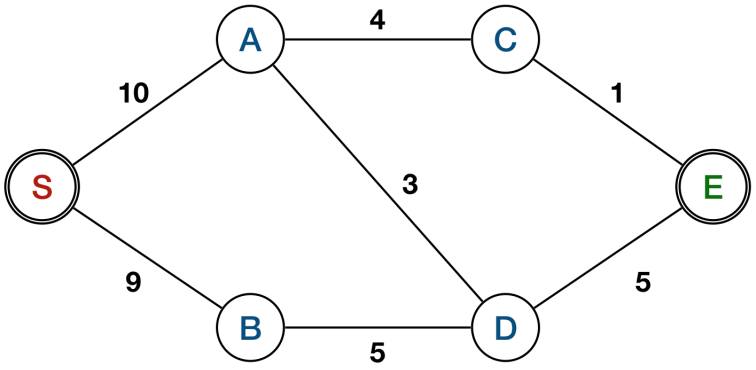

Figure 4: Graph for UCS traversal. Start node is S and goal node is E. Each edge is associated with the cost of traversing that edge.

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

Solution:

Frontier	Explored
$S(0)$	ϕ
$B(9), A(10)$	S
$A(10), D(14)$	S, B
$D(13), C(14)$	S, B, A
$C(14), E(18)$	S, B, A, D
$E(15)$	S, B, A, D, C

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

Solution: **Cost**: 15 , **Path**: $S \rightarrow A \rightarrow C \rightarrow E$

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) *Proof.* We use mathematical induction to prove the given statement.

Base case: It is trivial to show that the statement is true initially, since the initial state is connected to itself.

Inductive hypothesis: We assume that the statement is true for a particular set of explored nodes.

Induction step: Next, we add one new node into this set. Since this node comes from the *frontier*, it must have had a node in the *explored* set that was its parent (in the search tree). Since this parent already satisfies the inductive hypothesis, the new node, thus, has a path to the initial node comprising of explored nodes through this parent. □

(b) *Proof.* We use mathematical induction to prove the given statement.

Base case: In the start of the search, the source node is in the frontier. This satisfies the graph separation property trivially.

Inductive hypothesis: Let us assume that the property holds at the beginning of the n^{th} iteration of `GRAPH-SEARCH`.

Induction step: Our aim is to show that GRAPH-SEPARATION holds at the end of this iteration of `GRAPH-SEARCH` loop. We focus on paths from source node to unexplored nodes which only have n as the frontier node in its path (if other frontier nodes are present, GRAPH-SEPARATION would trivially hold, even if n is removed from the frontier).

As this iteration proceeds, node n is extracted from the frontier and added to the explored set. Consequently, all its children $\{n^{'}\}$ are considered to be put into the frontier. For each child, the following checks are made: 1) If $n^{'}$ is already an explored node, don't add it to frontier. 2) If $n^{'}$ is already in frontier, don't add it to frontier (optionally, update its utility if a shorter path is found) 3) If first two checks fail, add $n^{'}$ to the frontier.

Proof by contradiction: As node n is removed from the frontier, it opens up the possibility of paths through n which violates GRAPH-SEPARATION. We assume such a possibility to be true. Naturally, this path would need to traverse through the child nodes $\{n^{'}\}$ of n. In the `GRAPH-SEARCH` algorithm, we iterate all three conditions and see if our assumption holds:

i. If $n^{'}$ is already an explored node, there can't be paths through it to unexplored nodes directly as it would violate the inductive hypothesis. Thus, we look at the paths that would go next to n [2] and then to unexplored nodes. Since, all the unexplored children of n would be added to the frontier at the end of the iteration, there can't be a path that wouldn't have a frontier node.
ii. If $n^{'}$ is already in frontier, our assumption of existence of a path violating `GRAPH-SEPARATION` fails trivially.
iii. If finally $n^{'}$ is added to frontier, similar to last case, our assumption fails.

Overall, our assumption fails and `GRAPH-SEPARATION` holds.

□

[2] we are only concerned about paths through n.