CSE 101 Homework 1

Fall 2018

This homework is due on gradescope Friday October 12th at 11:59pm. Remember to justify your work even if the problem does not explicitly say so. Writing your solutions in \LaTeX is recommended though not required.

**Question 1** (Time Maze, 30 points). Sheldon is trying to traverse a maze. The maze is only open for 12 hours (noon until midnight). Sheldon has a map of the maze, but unfortunately some of the walls will change locations over time. Additionally, some locations in the maze contain time machines that Sheldon can use to transport himself to a specified hour in the same location. More formally, the maze is represented by an undirected graph $G$. Sheldon starts at a node $s$ and is trying to reach a node $t$. Each edge of the maze is only passible during certain hours (between 1 and 12) that are known. Some vertices of $G$ contain time machines with a specified end time, and Sheldon can also use this machine to transport himself to that time. In traversing the maze, Sheldon can either travel to adjacent vertices if the edge is available at the time (this takes effectively no time), wait around for the next hour to come, or use a time machine in his current location.

Give an algorithm to determine whether or not it is possible for Sheldon to complete the maze. For full credit your algorithm should run in time $O(|V| + |E|)$ or better.

**Solution 1.** The basic idea is to run a DFS to determine reachability. Unfortunately, because of the variable nature of the walls, we can’t do this by just considering the the graph at separate times. Therefore, we will need to create a more complicated graph whose vertices also encode the current time. In order to construct a graph $G = (V, E)$ satisfying the above constraints, we first need to define vertex set $V$ and edge set $E$.

**Graph Construction.** Denoting the set of locations as $V$, the set of undirected edges as $E$, and the set of times as $T$, we define the set of vertices $V'$ as $V \times T$ (i.e. the range $[1, 12]$). Thus, $\forall v \in V'$, $v$ specifies both the physical location and the time at that location.

A directed edge $e \in E'$ exists from vertex $v_i \in V$ to $v_j \in V$ iff we can reach $v_j$ from $v_i$. This can happen in exactly one of three scenarios:

1. $v_i, v_j$ are adjacent locations with no walls between them and $r_i = r_j$.
2. $v_i, v_j$ are the same physical location and there is a time machine at $v_i$ that takes you to specific time $r_j$: $1 \leq r_j \leq 12$ = $T$ and $r_j \neq r_i$.
3. $v_i, v_j$ are the same physical location and we wait an hour (i.e. a special case of 2. where $r_j = r_i + 1$).

Note: Although we are given an undirected graph, we know that undirected graphs are special cases of directed graphs. Case 1 is the trivial decomposition where we replace an undirected edge with symmetric outgoing edges (i.e. undirected edges $\{e = (v_i, v_j)\} \implies$ directed edges $\{e' = (v_i, v_j) \& e'' = (v_j, v_i)\}$).

**Algorithm.** We can now run DFS on our newly constructed graph, starting at $s' = s$ at 12 PM. We can modify it to return True if any $t_i \in t \times T$ is ever visited, and return False otherwise.

**Complexity.** Our total time complexity is $O(|V| + |E|)$. When we constructed our graph, we had $|V'| = |V| \times (T = 12) = 12 |V|$ nodes. By the same logic, this implies adjacent edges $E'_{adj} = (|E| \times 2) \times (T = 12) = 24 |E|$ at most. For time machine edges, since a time machine can only take us to times when the maze is open
As DFS will visit everything reachable from $s$, each physical location can go to at most $12 - 1 = 11$ different times (since self-edges, if they existed, would never be traversed, we can ignore them), so $E_{	ext{time}} = |V'| \times 11 = 12|V| \times 11 = 132|V|$ (as waiting an hour is a special case of a time machine, those edges are included in $E_{	ext{time}}$). Altogether, this implies our edge set $|E'| = 24|E| + 132|V|$. Since we know that DFS will run in $O(|V'| + |E'|)$ time, this implies that our algorithm runs in $O(|V'| + |E'|) = O(12|V| + (24|E| + 132|V|)) = O(144|V| + 24|E|) = O(|V| + |E|)$ time.

**Correctness.** In order to show correctness, we must show two things:

1. If $\exists \text{sol} \in$ valid solutions, the algorithm will find one
2. If the algorithm finds a solution, then $\text{sol} \in$ valid solutions

As DFS will visit everything reachable from $s'$, we know that our algorithm will return True if and only if some $t_i$ is reachable from $s'$. \hfill $\square$

**Question 2** (Edge Activation Maze, 40 points). Penny is trying to escape a mad scientist’s lair. Unfortunately for her, some of the doorways are blocked by security doors. These can be deactivated if she can reach the security panel located somewhere else in the lair. Fortunately, once a security door has been deactivated, it will stay that way indefinitely. Formally, the lair is given by a graph $G$ where Penny starts at vertex $s$. Each edge has an associated control vertex, and Penny cannot traverse the edge until she has reached the associated control vertex.

Provide an algorithm that given $G$, $s$ and the control vertices for each edge, returns the list of vertices that Penny can escape to. For full credit your algorithm should run in linear time.

**Solution 2.** The basic idea is to run DFS and keep track of which vertices are in the same connected component as $s$. Unfortunately, because of the locked doors, we can’t do this by just running DFS once from $s$.

Therefore, we will need to modify our algorithm to accommodate this constraint.

**Graph Construction.** We are given:

1. $G = (V, E)$ where $v \in V$ are rooms of the lair and $e \in E$ are doors connecting rooms together.
2. $s \in V$ where $s$ is the starting room
3. Control vertices denoted as $VC_e : VC_e = v_i$ for each edge $e \in E$ : visiting $v_i$ unlocks $e$. Without loss of generality, we can assume that $VC_e = s$, $\forall e \in E$ : $e$ is already unlocked and that $VC_e$ is stored directly in $e$ (i.e. $e$ knows its control vertex).

**Algorithm.** Our basic algorithm is as follows:

For each vertex $v \in V$, we keep a list of reachable vertices that $v$ unlocks, denoted as $U_v$

We modify our DFS so that when it visits $v$, for each edge $e = (v, w)$, it checks $V_c$ for an associated control vertex $c$. We have two cases to consider:

1. **c has been visited:** This means that the door is unlocked. If $w$ has not been visited, we recursively search $w$.
2. **c has not been visited:** This means that the door is locked. Put $w$ in $U_c$.

Now, $\forall u \in U_v$, if $u$ has not been visited, search $u$. At the end of the algorithm, we return the vertices marked as visited. These are all the rooms to which Penny can escape.

**Complexity.** Our algorithm runs in $O(|V| + |E|)$ time.

In the main part of our algorithm, we see that we only explore vertices that have not been visited before, which implies we search at most $|V|$ vertices. We check edges at most $2|E|$ times (once for each incident vertex). By extension, we only check control vertices at most $2|E|$ times. Finally, we check unlock lists no more than $|E|$ times. This implies that our algorithm runs in $O(|V| + 2|E| + 2|E| + |E|) = O(|V| + 5|E|) = O(|V| + |E|)$ time, linear in the number of vertices and the number of edges.
Algorithm 1 Q2

procedure FIND_ESCAPABLE(v) ▷ Finds the vertices to which Penny can escape
v.visited ← True
for (e ∈ v.edges) do
    w ← e.target
    c ← e.control_vertex
    if (c.visited) then
        c.unlock_list.add(w)
    else
        if (c.visited && !w.visited) then
            find_escapable(w)
for (u ∈ u.unlock_list do
    if (!u.visited) then
        find_escapable(w)

Correctness. In order to show correctness, we must show two things:

1. If ∃ sol ∈ valid solutions, the algorithm will find one
2. If the algorithm finds a solution, then sol ∈ valid solutions

Proof:

1. Any vertex explored by the algorithm is one that Penny can reach, because it must be either s, or adjacent to another vertex Penny can reach via an edge whose control vertex Penny can also reach. Since we always explore reachable vertices, either immediately when we discover the edge leading the vertex and the edge is unlocked, or later during the procedure call that unlocked that edge, then we see that the algorithm will find a valid solution.

2. Penny cannot reach any unexplored vertices because to get to the first unexplored vertex, she would need to cross an edge whose control vertex she hasn’t reached yet. This is impossible given how we have defined our algorithm.

Question 3 (Pre- and Post- Order Computations, 30 points). Given the set of vertices of an undirected graph G and the pre- and post-order numbers of depth first traversal of G, show how to compute the following properties of G:

(a) The number of connected components of G. [20 points]
(b) The number of leaves in the depth first search tree (that is the number of nodes from which DFS discovered no new nodes). [10 points]

Solution 3.

(a) Algorithm. We first sort the list of vertices based on their pre-order number. We initialize a variable count to keep track of number of components. The first vertex will be part of some component, and we will find all the other vertices of this component. We now do a linear traversal on the list of sorted vertices, and find the last vertex where it’s pre-order is less than post-order of the first vertex. All the vertices traversed before and including this will be part of the same connected component. We end our first connected component here and increment the count of connected components by 1. We move to the next vertex(if exists) and find the vertices of it’s component. We keep repeating this routine, till we exhaust all the vertices.

Correctness. We know that if we start from vertex v, it will explore all the vertices in it’s component before ending the exploration of vertex v. Therefore, for all the other vertices in the component, the pre-order and post-order will lie in the range of pre-order and post-order of vertex v.
(b)

Algorithm. We iterate over all the vertices, and check if the post-order of a vertex is exactly 1 more than the pre-order of the same vertex. If it's same, it means that it is a leaf node and we can keep a counter which will be incremented when we encounter such a vertex. The final value of count will store the number of leaf nodes.

Correctness. Pre-order number represents when we start exploring a node, and post-order number represents when the node is completely explored. If post-order number of a node is exactly 1 step away from pre-order it means the exploration of the node ended right after it started, which means there is no other to explore from that node, making it a leaf node.

Complexity. The time complexity for this is $O(|V|)$, as we check the pre-order and post-order of every vertex exactly once.

Question 4 (Extra credit, 1 point). Approximately how much time did you spend on this homework?