CSE 101 Homework 1: Solutions

Spring 2017

Question 1 (Video Game, 30 points). In a video game you control two characters, John and Rose. They are each located on a square of an $n \times n$ grid. They begin at two specified locations and your objective is to move each of them to specific ending locations. On a turn you can move either character to an adjacent (vertically or horizontally) square, however, there are some restrictions. Certain squares are only traversable by one character or the other. Furthermore, certain doorway squares can only be entered or exited by one character if the other character is currently on a designated unlocking square.

Design a polynomial time algorithm that given a description of a particular level of this game (i.e. is given $n$, the initial and ending locations for John and Rose, a list of squares traversable only by John, a list of squares traversable only by Rose, a list of squares traversable by neither, and a list of pairs of doorway squares and the corresponding unlocking squares), determines whether or not it is possible to win the level.

Solution:
The basic idea is to run a DFS to determine reachability. Unfortunately, because of the doors, we cannot figure out what the legal moves are just by considering the positions of Rose and John individually. Therefore, we will need to create a more complicated graph whose vertices encode the locations of both characters. 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:
Let $S$ denote the set of all squares of the $n \times n$ grid. Let $J \subseteq S$ denote the set of squares traverse by John and $R \subseteq S$ denote the set of squares traverse by Rose.

We define the vertex set $V = (J \times R)$. The intuition of defining the vertex set in such a way is to keep track of the movement of both John and Rose simultaneously. The value of a single vertex in our graph $G$ would give us the positions of both John and Rose.

A directed edge $e \in E$ exists from vertex $v_1 \in V$ to $v_2 \in V$ if we can reach $v_2$ from $v_1$ by moving one of the characters in horizontal or vertical direction. Furthermore, if one of the characters moved onto or off of a doorway square, this edge will only be included if the other character is on the corresponding unlocking square. Here we assume that the initial position of John and Rose forms the vertex $v_1$. The intuition of defining edge in such a way is to track the movement of both John and Rose at each time step.

Depth First Search(DFS) Algorithm:
Let $S_J \in S, S_R \in S$ denote the starting position of John and Rose, and $E_J \in S, E_R \in S$ denote the ending position of John and Rose. Thus, we would run explore on $G$ taking $(S_J, S_R)$ as the starting vertex. If we are able to reach $(E_J, E_R)$, then we would conclude it is possible to win the level else it is not possible.

Complexity:
Explore take $O(|V| + |E|)$ time of a graph $G=(V,E)$. There are total $O(n^4)$ vertices. There are $n^2$ positions for John in $S$, and $n^2$ positions for Rose in $S$. Thus, total possible combination of tuples are $n^2 \times n^2 = n^4$. There are a total of $O(8n^4)$ edges since at each time step John can make maximum 4 moves and Rose can also make maximum of 4 moves. Hence, the overall running time would be $O(n^4)$. 
**Correctness:** In order to show correctness of the algorithm, we would have to show two things:

1. If a solution exists, the algorithm will find it.
2. If the algorithm finds a solution, then it’s a valid solution.

**Completeness of graph** $G$: The way we have defined the graph vertex set $V$ in $G = (V, E)$ ensures that we only take set of traversable squares by both John and Rose as well as the set of unlocking square tuples. Thus, the vertex set $V$ is complete in the sense all the tuples in $S \times S \setminus V$ cannot be visited by John and Rose. The edge set $E$ defined during the graph construction ensures that exactly one characters takes a step at a time. Thus, the edge set $E$ is complete in the sense that it covers all possible transitions that John and Rose could take.

**Proof (1):** Since the graph is complete, if a path exists from starting vertex to ending vertex, it should lie on same connected component of the graph (from completeness of $E$). As our explore will discover the entire connected component, this will find the ending square.

**Proof (2):** If explore is able to find a path from starting vertex to ending vertex, then it should be present in a single connected component. Moreover, since all edges of $G$ correspond to legal moves in the game, the path from $(S_R, S_J)$ to $(E_R, E_J)$ will correspond to a sequence of legal moves that take Rose and John from their starting squares to their ending squares.

**Question 2** (Vacation Planning, 35 points). Dave is planning a vacation to Graphlandia. He has a road map given by an undirected graph $G = (V, E)$, where the vertices denote cities and the edges denote roads connecting them. At each of the cities is one of $k$ different types of attractions. Dave wants to plan a trip whereby he flies into some city, drives around along the roads visiting a number of cities containing at least one city containing each type of attraction, and flies out of a (potentially different) city.

(a) Give an $O(|V| + |E|)$ time algorithm to determine whether or not it is possible for Dave to do this. [10 points]

(b) It turns out that there is a holiday in Graphlandia when Dave is planning to arrive, and many of the roads are blocked off in one direction (making $G$ into a directed graph). Does the algorithm from part (a) still work? Why or why not? [5 points]

(c) Give an $O((|V| + |E|)^2k)$ time algorithm solving the problem in the case of directed graphs. Hint: you will need to create new vertices to keep track of which attractions Dave has visited so far. [20 points]

**Solution:**

(a) Let the undirected graph representing the road map of cities be $G(V, E)$. We note that Dave can visit only cities in a single connected component, but that he can always visit as many cities in the connected component he flies into as he likes. Therefore, we need to know if $G$ has any connected components with one attraction of each type.

This can be done fairly easily in two steps. Firstly, we run DFS to compute the connected components of $G$. Next, for each connected component, we need to determine whether it contains all attractions. We can do this by simply scanning through the vertices of each component one at a time, checking off on a list which attractions are present and seeing if we find all of them.

The runtime of this algorithm is $O(|V| + |E|)$ for the DFS and $O(|V|)$ for the final scan, so the total runtime is $O(|V| + |E|)$.

(b) Algorithm in part (a) works only for undirected graphs. When the road map is represented as a directed graph, the algorithm needs changes as the fact that all the $k$ attractions lying in the same connected component as the starting city ensures that all of them can be covered is no longer true. Because a directed graph can be weakly connected and thus we cannot simply assume that every vertex is reachable from the starting vertex.
(c) Let the graph $G$ be a directed graph representing the road map of cities. The problem is same as in part (a). To determine if Dave can visit all the $k$ attractions starting from some city $s$. In order to do this, we want to create a more complicated graph whose vertices take into account not just Dave’s current location, but also which attractions he has visited so far. The vertices of this new graph $G'$ are indexed by a pair of a vertex $v \in V$ and an array $A$ of $k$ 0’s and 1’s. Thus, there are $2^k|V|$ many vertices. A vertex $(v, A)$ will have edges to vertices of the form $(w, A')$ when there is an edge in $G$ from $v$ to $w$ and $A'$ is $A$ but with the entry corresponding to the attraction type of $w$ set to 1. This denotes how if Dave is in $v$ having visited the attractions listed in $A$, he can move to $w$, and will then have also visited the attraction there.

We need to determine if Dave can, starting from a city $v$ and having visited only that city’s attraction move until he is in some different city and has visited all of the attractions. We can figure this out by running explore in $G'$ on all the vertices of the form $(v, e_v)$ (here $e_v$ denotes the array corresponding to having visited only the attraction in city $v$) without resetting the visited tokens, and seeing if any vertex of the form $(w, (1, 1, \ldots, 1))$ is reached. If so, Dave can succeed and otherwise not.

To understand the runtime, the explores performed are cheaper than running a full DFS and thus run in time at most $O(|V'| + |E'|)$. It’s not hard to see that $|V'| = 2^k|V|$ and $|E'| = 2^k|E|$. Constructing $G'$ runs in similar time, and thus the total runtime is $O(2^k(|V| + |E|))$.

**Question 3** (Learning from Pre- and Post- Orders, 35 points). Suppose that when depth first search is run on an undirected graph $G$ the pre- and post- orders computed for the vertices are as follows:

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Pre-</th>
<th>Post-</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>F</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>G</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>H</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>I</td>
<td>16</td>
<td>19</td>
</tr>
<tr>
<td>J</td>
<td>17</td>
<td>18</td>
</tr>
</tbody>
</table>

(a) What are the connected components of $G$? [10 points]

(b) Give an example of a graph that is consistent with this DFS execution. [10 points]

(c) What is the greatest possible number of edges that $G$ could have? [15 points]

**Solution:** DFS tree formed using the above traversal is shown in Figure 1.

(a) Note that since $A$ has preorder number 1, it is the first explored vertex. It is explored through time step 12, and in the meantime discovers $B, C, D, E, F$. Therefore, these vertices form a connected component. Next the algorithm explores $G$, finding no new vertices, so this is a second component. Finally, it explores $H$ from time steps 15 through 20, discovering the third component $H, I, J$. These are therefore the only three components.

(b) The graph in Figure 1 is easily seen to be consistent with this DFS execution.

(c) If we count the number of edges in Figure 2, we can see the greatest possible number of edges that $G$ could have is 11. Figure 2 shows that this is achievable. To show that there cannot be any more edges, note that two vertices cannot have edges between them if their pre-post intervals are disjoint. Since if they were connected, you would discover one vertex while exploring the other. Firstly, by our discussion in part (a), we know that the graph splits into three components. Clearly the component of $G$ has no edges, and the component of $H, I, J$ has at most 3. For the component of $A, B, C, D, E, F$, we note that since $B, C, D$ all has postorder number at most 7, which is less than the preorder numbers of $E, F$, none of $B, C, D$ can connect to $E$ or $F$. Similarly, you can show that $C$ and $D$ cannot be connected. This shows that only the 11 edges in 2 can be in $G$. 

3
Figure 1: DFS Tree from the given Preorder and Postorder traversal.

Figure 2: Graph constructed by drawing all possible edges. Here we should only edges from a node to all its ancestors.