CIS 675 (Fall 2018) Disclosure Sheet

Name: Wentan Bai
HW # 4

Yes **No** Did you consult with anyone on parts of this assignment, including other students, TAs, or the instructor?

Yes **No** Did you consult an outside source (such as an Internet forum or a book other than the course textbook) on parts of this assignment?

If you answered **Yes** to one or more questions, please give the details here:

I consulted question 5 with teaching assistant, Siddhartha Roy Nandi, to confirm my understanding. I also consult all questions and extra question with another student, Wentian Bai. For all questions, we discussed our ideas and I finish my algorithms independently.

By submitting this sheet through my Blackboard account, I assert that the information on this sheet is true.

This disclosure sheet was based on one originally designed by Profs. Royer and Older.

Question 1:

My algorithm is.

- Place one guard at $l_1 + 1$ location, where 1 represent the 1 unit of distance.
- move along the hallway until there is a painting l_m , which is not protected. Place one guard at $l_m + 1$ location.
- Repeat the second step until all paintings are protected.

---

guard $\leftarrow l_1 + 1$
num $\leftarrow 1$
 For i = 2 to k :
  If guard $< l_i$
   guard $\leftarrow l_i + 1$
   num $\leftarrow num + 1$

---

running time:

The running time of Modified Dijkstras shortest-path algorithm is the same as the time of original algorithm because the change part costs the same time as replaced part.

Thus, the final running time is double of running time of Dijkstras shortest-path algorithm. The final time is $O((|V| + |E|) \log |V|)$

Question 2:

My goal is to maximize the amount of payment. Thus, my idea is trying to assign the high-paying jobs to the slot just before their deadline. My algorithm is:

- Sort all jobs by their payments in decreasing order. Denote the elements of sorted list as $J_1, J_2, ..., J_m$, and for each job J_i, $i \leftarrow 1...m$, $p_i \leq p_{i+1}$.
- Schedule J_1 to time slot from time $d_1 - 1$ to time d_1.
- Iterate through the list of remaining jobs in order, and at each step i, check whether we can schedule current job J_i from time $d_i - 1$ to time d_i.
  If this time slot is already scheduled with other job, traverse the schedule forward from time $d_i - 1$ and find whether there is a empty slot to schedule current job.
  If there is not a empty slot after traversing, we will abandon this job.
- After going through list, the schedule is expected result.

Sort all jobs by their payments in decreasing order.
schedule[m] // m-size array
schedule[d_1] $\leftarrow J_1$
For i = 2 to m :
    If schedule[d_i] **is** empty
        schedule[d_i] $\leftarrow J_i$
    Else :
        For k = $d_i - 1$ to 1
            If schedule[k] **is** empty
                schedule[k] $\leftarrow J_i$
Return schedule

---

running time:

The sorting all jobs uses $O(n \log n)$. When we iterate through sorted jobs, for each job, we may need to traverse the schedule forward which uses $O(n)$ time. Therefore, whole iteration need $O(n^2)$ time.

The final time is $O(n \log n) + O(n^2) = O(n^2)$

Question 3:

Based on description of question, I only need to give a **reasonable greedy** algorithm.

For current city, I will find another city which is not visited and has the shortest distance between it and current city. Then mark current city as visited and move to newly found city. Repeat this step.

My algorithm will not always find the correct answer. For example:

1. In this graph, we begin at C_0. The shortest distance is $C_0 \to C_1$. Move to C_1 and mark C_0 as visited.
2. Then at C_1, the shortest distacne is $C_1 \to C_2$ without visited cities. Move to C_2 and mark C_1 as visited.
3. Then at C_2, the shortest distacne is $C_2 \to C_3$ without visited cities. Move to C_3 and mark C_2 as visited.
4. Then at C_3, the shortest distacne is $C_3 \to C_4$ without visited cities. Move to C_4 and mark C_3 as visited.
5. All cites visited and back to C_0.

The total distacne is $2+2+11+5 = 20$. However, if we follow the path $C_0 \to C_2 \to C_1 \to C_3 \to C_0$, the total distance will be $3+2+3+5 = 13$. Thus, my algorithm does works but does not find the correct answer.

running time:

My algorithm visits all cities exactly once. Thus the time is $O(|V|)$, where $|V|$ means the number of all cities.

Question 4:

Based on description of question, I think this procedure does not give me the correct shortest path from s to t. For example:

In this example, we parallely use Dijkstras algorithm for node t and node s. Since the length $l(s, A)$ is shorter than $l(s, C)$ and $l(t, B)$ is shorter than $l(t, F)$, the $dist(A)$ will be smaller than $dist(C)$ and $dist(B)$ will be smaller than $dist(B)$. Then there will be a overlap between node A and node B without finding the real shortest path. Based on description, $d_1 = 5 + 11$, $d_2 = 5$, $d_1 + d_2 = 21$. This is incorrect.

Thus, this algorithm does not find the correct answer.

Question 5:

(1):

Based on description of question, I think the directed graph G seems as below.

The infinite path p is infinite sequence $s, a, b, c, d, b, c, d, b, c, ..., b, c, d, e, f$, where "..." is infinite loop. The $Inf(p)$ will be a set $\{b, c, d\}$.

Claim: If p is an infinite path of G, then the $Inf(p)$ is a subset of a single strongly connected component of G.

Proof: By contradiction. Suppose p is an infinite path of G, and the $Inf(p)$ is not a subset of a single strongly connected component of G. Based on description and my example, the vertices in an infinite path p are visited infinitely often because there is a cycle. The $Inf(p)$ is the set of vertices that occur infinitely many times in p. Therefore, the vertices in this set are connected by some directed path.

The defination of SCC is: in a directed graph, SCC is a set of nodes such that there is a (directed) path between every pair in both directions. Based on defination of SCC, for each set of a single strongly connected component of G, their nodes are connected by a path. Since the vertices in $Inf(p)$ are all connected, the $Inf(p)$ must be a subset of a single strongly connected component of G. There is a contradiction and my suppose is incorrect. The claim holds.

Question 5:

(2):

If graph G has an infinite path, then there will be a cycle in G. Since G has a finite number of vertices, only in a cycle, some vertices of G are able to be visited infinitely often.

Thus, my algorithm is to determine if G has a cycle. I use DFS on G. If depth-first search reveals a back edge of a directed graph G, G has a cycle and there is a infinite path. If we explore all nodes and there does not exist a back edge, then G does not has an infinite path.

Claim: If the DFS reveals a back edge, the graph has a cycle.

Proof: let (c, s) be a back edge from node c to node s. By definition of back edge, s is an ancestor of c. In the DFS tree, there is a path from s to c. The path $s \rightarrow c$ and back edge (c, s) form a cycle. The claim holds.

running time:

My algorithm is the same as basic DFS. Thus my running time is $O(|V| + |E|)$.

Extra:

For each binary tree, the number of its edges $|E|$ equals to the number of its nodes minus one, $|E| = |V| - 1$. If the a binary tree has a perfect matching, the $|V|$ must be even. Since there is only one root node at the first level. Thus, for a binary tree T, we need to check every node:

- For current node, if the number of all descendants in left is odd and the number of all descendants in right is even or zero, this partition has a perfect matching.
- For current node, if the number of all descendants in right is odd and the number of all descendants in left is even or zero, this partition has a perfect matching.
- If current node is a leaf, ignore it and return back to check its parent.
- For current node, if the number of all descendants in right is and the number of all descendants in left are both even or odd, this whole tree does not have a perfect matching.

I recursively go to deepest level and recusively back with check each node.

running time:

My algorithm is check all nodes one time. Thus my running time is $O(|V|)$.