HW 9 Solution

CS/ECE 374 B: Algorithms & Models of Computation, Spring 2020 Version: **1.0**

1 Problem 25:

Solution 25.A:

Algorithm 1 Find all at risk algorithm

Input : Graph G = (V,E)
Output: All people who are at risk

```
1  Function FindAllAtRisk(G):
2      Find the topological ordering of G
3      for all vertices v in V do
4          count [v ] ⟵ []
5      end
6      for all vertices v in topological ordering of G do
7          if len(count[v]) ≥ k or v is marked then
8              mark v
9              for all vertices in v → u do
10                 mark u
11             end
12             continue
13         if C(v) =′ +′ then
14             count [v ].append(v)
15         for all vertices in v → u do
16             if u is marked then
17                 continue
18             else
19                 merge(count[v], count[u])
20             end
21         end
22     end
23     retval ⟵ []
24     for all vertices v in count do
25         if v is marked ∨ C(v) =′ +′ then
26             retval.append(v)
27     end
28     return retval
```

We can perform this algorithm on the same graph as given in the question, that is, G = (V, E), |V| = n and |E| = m, which is a directed acyclic graph. First of all, we need to construct an auxiliary dictionary to store a list of unique vertices of positive patient that can reach this node. And to identify uniqueness, we need to use merge list to form a unique list, using O(length of list).

- First, we need to use the algorithm from textbook to find the topological ordering of this graph by calling ***Depth-first Search*** in $O(m+n)$. Then we would need to traverse the whole topological sequence to find the unique list for each vertex.

- To simplify the algorithm, we define that as long as the list is larger than k, the vertex is marked. Then we will no longer continue this vertex's list since it is already at risk. And we mark all of its neighbor since we do not need to consider these vertices either, along with all the vertices it can reach. In this way, we would be able to mark all the people who are at risk
- Therefore, generally, the most of the computation happens when we merge the lists of unique vertices. In the worst case, we need to merge m times, each with $O(k)$ length. Therefore, except for traversing all the vertices, we would need $O(km)$ to merge all the lists.
- In this case, the total running time of this algorithm would be $O(km + n)$.

Solution 25.B:

We can perform this algorithm on the same graph as given in the question, that is, G = (V, E), |V| = n and |E| = m, which is a general directed graph. With basically the same reasoning, we can first compute all the strongly connected components of G by utilizing ***Tarjan's Algorithm*** in $O(|V| + |E|) = O(m + n)$. After computing all the strongly connected components, we would be able to construct the metagraph of G, which is a directed acyclic graph as proved in the lecture. Therefore, we can use the same algorithm as the last problem to compute the this metagraph, since the unique list for all vertices in a scc are the same.

Therefore, we would have several steps for this algorithm:

- Compute the meta-graph of G using ***Tarjan's Algorithm*** in $O(|V| + |E|) = O(m + n)$.
- Count the unique list of positive vertices in every strongly connected components and store this number into count[scc], as the weight of this scc in $O(m + n)$.
- Find the count for all strongly connected components in this graph using the algorithm described in the last problem, treating each strongly connected components as a single vertex in $O(km + n)$.
- Find all the vertices in the scc's which have count equal to or larger than k in $O(n)$.

Therefore, to solve this problem, the running time of this algorithm is $O(km+n)$. Since we have already justified the correctness of the algorithm in the last problem, this algorithm would be able to find all people who are at risk in this general directed graph.

2 Problem 26:

Solution 26:

We can construct a graph to model this problem G = (V, E), where the vertices and edges are defined as following:

- We construct the vertices as a pair of a bus stop and the time some bus visits this stop. Since for each bus, it stops n times per day, and there are b bus lines in total, we would have $O(nb)$ vertices for each pair of bus stop and the time some bus visits the stop. Then we can construct two additional nodes to represent the stating position Walmart and the ending position home, suppose s and t. Therefore, we have $O(nb)$ vertices in this graph.
- In the construction of this graph, we have different types of edges to represent the transition between bus stops and bus lines.
  - First: During the trip of one bus line, we have no waiting time. Therefore, for every bus line, we have $n-1$ edges between each two of n stops of this bus line. And without waiting time, the weights of these edges are 0.
  - Second: For each transition of bus lines, we need to construct edges between two arriving time in chronological order, i.e., we construct edges for each stop with $\{(stop_1, time_1) \to (stop_1, time_2)\}$, and for each edge, its weight is the time difference between time_1 and time_2. We can easily calculate this weight in $O(nb)$ by traversing the whole schedule list.
  - Apart from these two general edges, we construct edges for the starting point and ending point so that as long as we arrive the bus stop at the destination or starting moving, regardless of bus lines, we automatically switch to the destination or switch to the specific bus line.
  - Therefore, we construct edges from s to all the nodes representing the bus stops at Walmart and edges from all the nodes representing bus stops at home to t. All these edges should have weights 0.
  - Since for these four types of edges, each node can have at most one of every type. Therefore, we have $O(nb)$ edges in this graph.
- To solve this problem, we would want to solve the shortest path starting from s to t, i.e, from Walmart to home. We can compute this problem by calling ***Dijkstra's Algorithm***. And the calculated dist(s, t) represents the total time waiting in the bus stop.
- Therefore, by using this algorithm under the construction of this graph, we have that the running time of this algorithm is $O(|E| + |V| \log |V|) = O(nb \log(nb))$
- *Remark on alternative solution using dynamic programming:* Since G described as previous is a directed acyclic graph, we can compute the shortest path from s to t using dynamic programming. With the same construction of the graph, we can first use the algorithm described in the textbook to find the topological ordering of G in $O(|V| + |E|)$. In this way, to calculate dis(v), which means the shortest path from s to v, as the following recurrence:

$$\text{dist(v)} = \begin{cases} 0, & \text{if v = s} \\ min_{u \to v} dist(u) + w(u \to v), & \text{Otherwise} \end{cases}$$

  We can solve this recurrence through dynamic programming, where pseudocode is described in the textbook. For the analysis of this algorithm, there are |V| vertices in total. In the worst case, we would need to go over every incoming edges for every vertex. Therefore, the runtime is $O(|V| + \sum_{v \in V} \text{in-degree}(v)) = O(V + E)$ since the sum of degrees of all vertices is 2|E| by Handshaking Theorem. In this case, the running time of this algorithm is $O(nb)$.

3 Problem 27:

Solution 27.A:

Suppose a cycle C, which can be represented as $e_1e_2...e_le_1$, have the minimum fluctuation f(C). If there are multiple cycles with minimum fluctuation, we can choose the one with the smallest number of edges. We will prove the statement by contradiction. Suppose that one vertex or edge is repeated since more repeated vertices or edges can only have more fluctuations than the one-repeated situation.

Then for some edge e_i, we know that its end vertex is the same with the start vertex of some edge e_j where $j \geq i+3$ since we need two more edges in between to return to this vertex. Therefore, we can have the following equation to represent the fluctuations between e_i and e_j:

$|w(e_i) - w(e_{i+1})| + |w(e_{i+1}) - w(e_{i+2})| + ... + |w(e_{j-1}) - w(e_j)|$
By the application of triangle inequality, where $|a-c| + |b-c| \geq |a-b|$
We know that: $|w(e_i) - w(e_{i+1})| + |w(e_{i+1}) - w(e_{i+2})| + ... + |w(e_{j-1}) - w(e_j)| \geq |w(e_i) - w(e_j)|$

By the assumption of edge e_i and e_j, we know that e_i and e_j are connected. Therefore, we have the fluctuations of C be:

$$\begin{aligned} f(C) &= |w(e_1) - w(e_2)| + |w(e_2) - w(e_3)| + ... + |w(e_i) - w(e_{i+1})| + ... + |w(e_{j-1}) - w(e_j)| + ... + |w(e_l) - w(e_1)| \\ &\geq |w(e_1) - w(e_2)| + |w(e_2) - w(e_3)| + ... + |w(e_i) - w(e_j)| + |w(e_j) - w(e_{j+1})| + ... + |w(e_l) - w(e_1)| \end{aligned}$$

In this case, we know that $e_1e_2...e_ie_j...e_l$ is also a cycle with the same or smaller fluctuations than the supposed cycle C. By the assumption, cycle C has the minimum fluctuation, but the new cycle has at most the same fluctuation with less edges. Therefore, we have a contradiction with our assumption. By the property of contradiction, we proved that the cycle with minimum fluctuation cannot have repeated vertices, i.e., it must be a simple cycle.

Solution 27.B:

Let the given directed graph be G = (V, E) where |V| = n and |E| = m. Then to solve the problem we can construct a graph G' = (V', E')defined as following:

- V' = {e | e ∈ E}, that is, we make every edge in the original graph as a vertex. Therefore, there are $O(m)$ vertices in this graph.
- E' = $\{(u,v) \to (v,k) | (u,v),(v,k) \in E\}$ for every pair of edges in E. In other words, we make an edge between two pairs of edges where the start of the second edge is the end of the first edge. Since for every edge in E, there are at most n choices for k for the second edge. Thus, there are $O(mn)$ edges in this graph.
- We can define the weights of each edge $\{(u,v) \to (v,k)\}$ to be $|w((u,v)) - w((v,k))|$
- To find the cycle, suppose starting at vertex v and ending at vertex u with the minimum fluctuation, we can convert the problem into the finding the shortest path between u and v + the weights of (u, v), i.e., finding the optimal dist(u, v) + w((u, v)).
- To solve this problem, we can call ***Dijkstra's Algorithm*** on every vertex to find the optimal cycle with minimum fluctuation. Therefore, the running time of this algorithm is $O(|E| + |V|' \log |V'|) * O(|V'|) = O(m^2n + m^2 \log m)$.