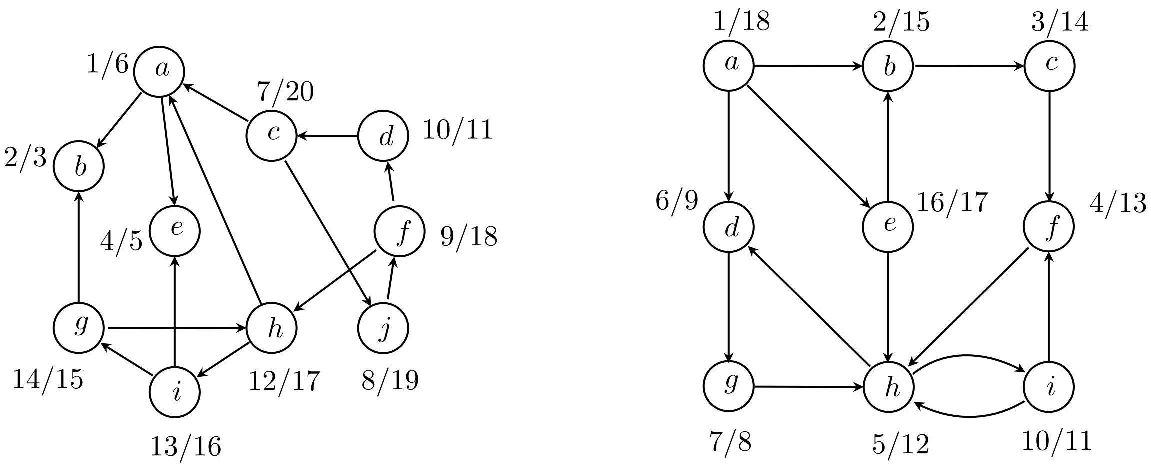
Design and Analysis of Algorithms (2022 Spring)
Solution to Problem Set 2

1.



(a) (b)

Figure 1: Timestamps obtained from DFS on G^R.

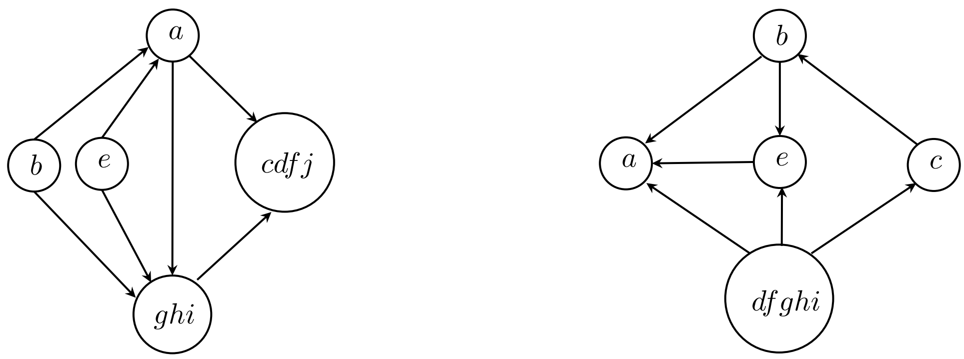
(a) The SCCs are found in the order listed below:

- Graph in (a): $\{c, d, f, j\}$, $\{h, g, i\}$, $\{a\}$, $\{e\}$, $\{b\}$
- Graph in (b): $\{a\}$, $\{e\}$, $\{b\}$, $\{c\}$, $\{d, f, g, h, i\}$

(b) Source SCCs and Sink SCCs:

- Graph in (a): Its source SCCs are $\{b\}$ and $\{e\}$. Its sink SCC is $\{c, d, f, j\}$.
- Graph G in (b): Its source SCC is $\{d, f, g, h, i\}$. Its sink SCC is $\{a\}$.

(c) The metagraphs of graph G in (a) and (b) are as follows:



(a) (b)

Figure 2: Metagraph of G.

(d) To make G strongly connected, the minimum number of edges needed is

- Graph in (a): 2
  (For example, add (u, b) and (v, e) for some u, v in the sink SCC $\{c, d, f, j\}$. Adding 1 edge is not sufficient because to make the two source SCCs $\{b\}$ and $\{e\}$ reachable from any vertex in G, there must be at least one in-degree edge at both b and e.)
- Graph G in (b): 1
  (For example, add (a, u) for some u in the sink SCC $\{d, f, g, h, i\}$.)

2. Let $e = (u, v)$. The algorithm for determining whether G has a cycle containing e is as follows:

   1) Construct G' by removing e from G.
   2) Initialize visited(w)=false for each vertex w in G'.
   3) Run explore(u).
   4) Return visited(v).

   Correctness:

   - If G has a cycle containing e, there is a path from u to v in G'. Thus, visited(v) is set to true during explore(u). The output is true.
   - If G has no cycle containing e, u and v are not connected in G'. Thus, visited(v) remains false during explore(u). The output is false.

   Running time: $O(|V| + |E|)$, since explore(u) is a sub-procedure of DFS.

3. Denote by $L(u, v)$ the length of edge (u, v). The algorithm for determining whether $G = (V, E)$ has a negative cycle is as follows:

   1) Construct graph $G' = (V', E')$ by adding a source vertex s to V and a directed edge (s, v) with $L(s, v) = 0$ to E for any $v \in V$. Then, every $v \in V$ is reachable from s.
   2) Initialize $\text{dist}(s) = 0$ and $\text{dist}(v) = \infty$ for any $v \in V$.
   3) For $i = 1$ to $|V'| - 1$:
   4)   For each edge $(u, v) \in E$:
   5) $\text{dist}(v) = \min\{\text{dist}(v),\ \text{dist}(u) + L(u, v)\}$
   6) For each edge $(u, v) \in E$:
   7)   If $\text{dist}(v) > \text{dist}(u) + L(u, v)$:
   8)   Return true
   9) Return false

   Correctness: Denote by $d(s, v)$ the shortest path distance from s to v.

   - If G has no negative cycle, $\text{dist}(v, |V'| - 1) = d(s, v)$ for each $v \in V$ after the distance updates in step 3 - 7. For each edge $(u, v) \in E$, $\text{dist}(u) + L(u, v)$ is the distance of the shortest path from s to u union edge (u, v). This distance is no shorter than the shortest path distance from s to v, i.e. $\text{dist}(u) + L(u, v) \geqslant d(s, v) = \text{dist}(v)$. Hence, the algorithm returns false.
   - If G has a negative cycle, the shortest path distance from s to those $v \in V$ involved in the negative cycle is $-\infty$. After the $|V'| - 1$ rounds of distance updates, it must be the case that the distance estimate for some v in the negative cycle drops in the next round.

     Formally, assume for the sake of contradiction that G contains a negative cycle and the algorithm returns false. Let $v_1 \to v_2 \to \cdots \to v_k \to v_{k+1}$ be a negative cycle, where $v_{k+1} = v_1$. Then, $\sum_{i=1}^{k} L(v_i, v_{i+1}) < 0$. Since the algorithm returns false, we have $\text{dist}(v_{i+1}) \leqslant \text{dist}(v_i) +$

$L(v_i, v_{i+1})$ for each i after the distance updates. Then, $\sum_{i=1}^{k} \text{dist}(v_{i+1}) \leqslant \sum_{i=1}^{k} [\text{dist}(v_i) + L(v_i, v_{i+1})]$. Note that $\sum_{i=1}^{k} L(v_i, v_{i+1}) \geqslant 0$ since $\sum_{i=1}^{k} \text{dist}(v_{i+1}) = \sum_{i=1}^{k} \text{dist}(v_i)$. This is a contradiction. Hence, if G contains a negative cycle, the algorithm always returns true.

Running time: $O(|V||E|)$

Remark: To return a negative cycle in G if any, the algorithm can additionally record, for each vertex, its parent in the current path found. Then, backtrack the negative cycle starting from v, which is the endpoint of the edge (u, v) that satisfies $\text{dist}(v) > \text{dist}(u) + L(u, v)$ in step 9.

4. Let $G = (V, E)$ be an undirected graph, where V is the set of cities and $E = \{\{i, j\}$: a highway were already built between city i and city j by the predecessor$\}$. The algorithm for finding a cost minimizing set of highways to be built subject to the choices already made, based on Kruskal's algorithm, is as follows:

   1) Start from $G' = (V, E')$, where $E' = E$.
   2) For all edges $e \notin E$ in ascending order of $c(e)$:
   3)   Add edge e to E' unless doing so would create a cycle
   4) Return $E' - E$ as the set of additional highways to be built

   Correctness: Since the algorithm does not add edges that would create a cycle, each edge added must connect two connected components in G. Thus, the algorithm finds an MST on the meta-graph of G, leading to a cost minimizing set of highways to be built subject to the choices already made by the predecessor.

   Running time: $O(|E| \log |V|)$

5. The Dijkstra Algorithm can be modified slightly to improve the running time for the single-source shortest path problem with positive edge lengths and known diameter D. In particular, D arrays are used to store the vertices whose estimate dist(.) of shortest distance from s is equal to a particular possible value among $1, \cdots, D$. The algorithm is as follows:

   1) Initialize $V' = \{s\}$, $\text{dist}(s) = 0$ and $\text{dist}(x) = \begin{cases} L(s,x) & \text{if } (s,x) \in E \\ \infty & \text{otherwise} \end{cases}$ for other vertices x
   2) Initialize $\text{bin}[i] = \{x : \text{dist}(x) = i\}$ for $i = 1, \cdots, D$
   3) While $V' \neq V$ do
   4)   Remove a node v from the first nonempty bin with respect to the bin index
   5)   Add v to V'
   6)   For all edge $(v, x) \in E$:
   7)   If $\text{dist}(x) > \text{dist}(v) + L(v, x)$:
   8)   If $\text{dist}(x) < \infty$, remove x from bin $[\text{dist}(x)]$
   9)   Update $\text{dist}(x) = \text{dist}(v) + L(v, x)$, then add x to bin $[\text{dist}(x)]$

   Correctness: Same as the argument for the Dijkstra Algorithm

   Running time: Steps 1, 2 and 5 take $O(|V|)$ time. Steps 6 - 9 take $O(|E|)$ time. Step 4 takes $O(D)$ time to locate the first nonempty bin among to D bins in order to find a vertex $v \in V'$ with the smallest $\text{dist}(v)$. Overall running time is $O(|V| + |E| + D)$.

6. It suffices to show that G is a tree. Then, there exists a unique tree in G that includes all nodes of G, meaning that both the DFS tree rooted at u and the BFS tree rooted at u obtained are exactly G.

   Suppose to the contrary that G is not a tree. Since G is undirected and connected, there exists a cycle $v_1 \to v_2 \to \cdots \to v_\ell \to v_1$ in G.

   Let v_i be the first node in the cycle visited by the DFS. Then, all other nodes in the cycle will be visited at some point when v_i is explored, so $v_1, v_2 \cdots, v_\ell$ will all be on the same path from the root.

   However, $v_1, v_2 \cdots, v_\ell$ will form at least two branches in the BFS tree. Suppose it is not the case. Then similar to the situation in the DFS tree, v_1,$v_2, \cdots, v_\ell$ will all be on the same path from the root. Let v_j and v_k be the first and the last node in the cycle visited by the BFS. Since v_j and v_k are adjacent in the cycle, BFS should have added v_k to the queue when v_j is visited. Thus, v_k should be a child of v_j in the BFS tree instead. This is a contradiction.

   Hence, the DFS tree will be different from the BFS tree. The result follows.

7. Denote by $w(v)$ the weight of vertex v. The algorithm for the variant of the single-source shortest path problem is as follows:

   1) Initialize $\text{dist}(s) = w(s)$, and $\text{dist}(x) = \infty$ for other vertices x
   2) $V' = \{\}$
   3) While $V' \neq V$ do
   4)   Pick the node $v \notin V'$ with the smallest $\text{dist}(v)$
   5)   Add v to V'
   6)   For all edges $(v, x) \in E$:
   7)   If $\text{dist}(x) > \text{dist}(v) + w(x)$, update $\text{dist}(x) = \text{dist}(v) + w(x)$

   Correctness: Similar to the argument for Dijkstra Algorithm, except for the computation of the shortest distance because of the length of a path is defined to be the sum of vertex weights instead of the sum of edge weights on the path.

   Running time: $O((|E| + |V|) \log |V|)$

8. (a)
   - "Only if": Assume graph $G = (V, E)$ is bipartite. Let (V_1, V_2) be a bipartition of V. Consider a cycle $v_1 \to v_2 \to \cdots \to v_k \to v_1$ in the graph. Suppose WLOG that $v_1 \in V_1$. Then $v_i \in V_1$ for odd i and $v_i \in V_2$ for even i. Then, k must be even because G is bipartite.
   - "If": Assume to the contrary that G contains no odd cycle and yet G is not bipartite. Then we pick an arbitrary vertex s in V and run $\text{BFS}(G, s)$ to compute the shortest path distance from s to every $v \in V$. After that, we color all vertices at even distance from s red, and color all vertices at odd distance from s blue. Since G is not bipartite, there exists some edge (u, v) whose endpoints receive the same color. Hence, there exists a path from s to u and a path from s and v such that the parity of the two path distances is the same. Then, the two paths, together with the edge (u, v), forms an odd cycle. This is a contradiction.

   (b) The algorithm for determining whether an undirected graph $G = (V, E)$ is bipartite is as follows:

   1) Pick an arbitrary vertex s in V. Run $\text{BFS}(G, s)$ to compute the shortest path distance from s to every $v \in V$.
   2) Color all vertices at even distance from s red, and color all vertices at odd distance from s blue.
   3) Return true if and only if for every edge, both of its endpoints are of different colors.

Correctness:

- Suppose G is bipartite. Let (V_1, V_2) be a bipartition of V. Without loss of generality, let $s \in V_1$. The algorithm colors s red. Then all neighbors of s are in V_2. The algorithm colors them blue. Then all neighbors of these blue vertices must be in V_1. The algorithm colors them red. Since there is no odd cycle in G, there will not be an edge with both endpoints of the same color, which implies that the algorithm must return true.
- Suppose G is not bipartite. Then G contains an odd cycle, so it is not possible to produce a coloring such that for every edge, both of its endpoints are of different colors. Hence, the algorithm must return false.

Running time: Step 1 takes $O(|V| + |E|)$ time. Step 2 takes $O(|V|)$ time. Step 3 takes $O(|E|)$ time. Overall running time is $O(|V| + |E|)$.