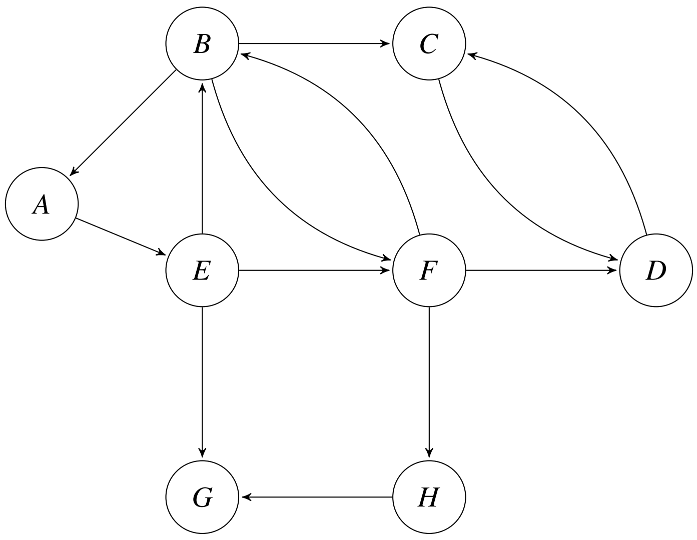
CMPSC 465 Data Structures & Algorithms
Fall 2025 Ke Chen and Yana Safonova

Worksheet 5

Monday, Oct 06, 2025

1. **SCC.** Run the strongly connected components algorithm on the following directed graph G. When doing DFS on G^R: whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.



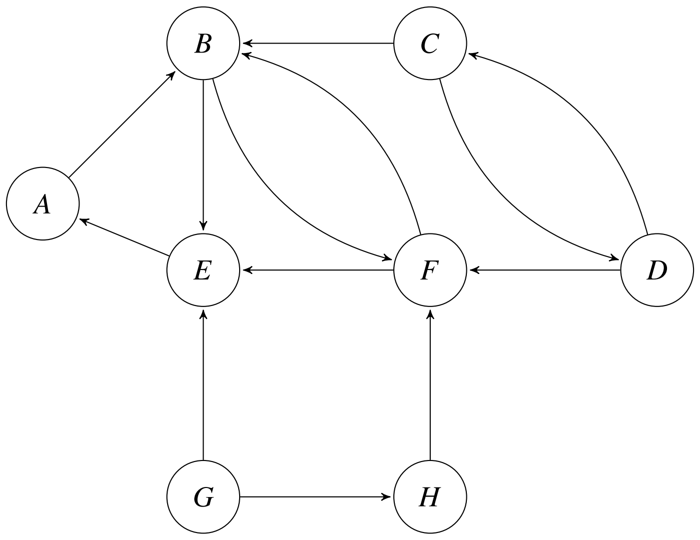
   (a) Give the pre and post number of each vertex in the reverse graph G^R.

   (b) In what order are the connected components found?

   (c) Which are source connected components and which are sink connected components?

   (d) Draw the "metagraph" (each meta-node is a connected component of G).

Solution: Reverse graph G^R is as follows,

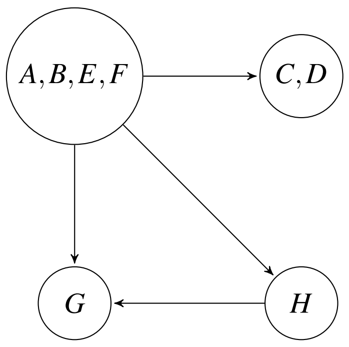


(a) pre and post numbers in G^R
A : 1, 8
B : 2, 7
C : 9, 12
D : 10, 11
E : 3, 4
F : 5, 6
G : 13, 16
H : 14, 15

(b) arranging the vertices in the decreasing order of post numbers,
G, H, I, J, C, D, A, B, F, E
Performing DFS on the given graph with above order will give us the connected components as follows,
{G}, {H}, {C, D}, {A, B, E, F}

(c) Source connected components: {A, B, E, F}
Sink connected components: {G}

(d) Meta graph of the given graph is as follows,



2. **Edges for SCC.** Let $G = (V,E)$ be a finite directed graph. Determine the minimum number of edges that must be added to G so that the resulting graph becomes strongly connected.

   Solution. First, we need to find the SCCs using the algorithm described in the class. In the process we can generate the meta graph G_{meta} by contracting each strongly connected component (SCC) of G into a single vertex. G_{meta} is a DAG. Let s be the number of sources (indegree 0) and t the number of sinks (outdegree 0) in G_{meta}. If G is already strongly connected, then G_{meta} has one vertex and no edges are needed. Otherwise, at least $\max(s,t)$ new edges are necessary, since each source needs an incoming edge and each sink needs an outgoing edge.

   To see that $\max(s,t)$ edges are also sufficient, consider pairing sources and sinks. If $s = t$, we can connect each sink to a distinct source, producing a directed cycle that passes through all SCCs, hence strong connectivity. If $s \neq t$, say $s > t$, we first connect each sink to a distinct source, covering all sinks. The remaining $s - t$ sources can be chained into the cycle by directing edges from already connected components, ensuring all components lie on a single strongly connected structure. A symmetric argument holds when $t > s$. Thus exactly $\max(s,t)$ edges always suffice.

   The algorithm runs in $O(|V| + |E|)$: SCCs are found in linear time , the meta graph is built in linear time, and counting sources/sinks is also linear. Hence,

$$\boxed{\text{Minimum edges required } = \max(s,t)}$$

3. **Multi Source Shortest Path** Let $G = (V,E)$ be an unweighted graph (directed or undirected) and let $S \subseteq V$ be a set of source vertices. For every vertex $v \in V$ compute

$$d(v) = \min_{s \in S} \text{dist}_G(s,v),$$

   the minimum number of edges from v to the closest source.

   Solution: Initialize a queue with all sources S; set $d(s) = 0$ for every $s \in S$ and $d(v) = \infty$ for all other v. Perform a standard BFS: repeatedly dequeue a vertex u, and for each neighbor w with $d(w) = \infty$ set $d(w) = d(u) + 1$, record a parent pointer $\text{parent}(w) = u$ if path reconstruction is desired, and enqueue w. When the queue is exhausted every vertex reachable from S has been assigned its distance to the nearest source.

   Correctness BFS explores vertices in nondecreasing order of distance from the set of sources: the initial layer is exactly the sources (distance 0), and when a vertex w is first discovered it is reached by a path of length $d(u) + 1$ where u is a previously discovered vertex at distance $d(u)$. By induction on layers, no shorter path from any source to w can exist when w is first discovered, so $d(w)$ is the minimum over all sources. (Equivalently, adding a temporary super-source with zero-cost edges to every $s \in S$ and running single-source BFS from that super-source yields the same distances.)

   Complexity. Each vertex is enqueued and dequeued at most once and each edge is examined at most once, so the running time is $O(|V| + |E|)$. Space is $O(|V|)$ for distances, parent/root arrays and the queue.

4. **Dijkstra with Edge Budget** Let $G = (V,E)$ be a directed graph with nonnegative edge weights, a source vertex $s \in V$, and an integer $k \leq |V| - 1$. The task is to compute the shortest path distance from s to every vertex $v \in V$ such that no path uses more than k edges. Design an algorithm. Prove that it is correct and analyse the runtime complexity.

Algorithm We define an augmented state space in which each vertex is replicated across $k+1$ layers, corresponding to the number of edges used so far. More formally, let $\text{dist}[v][j]$ denote the shortest distance from s to v using at most j edges. Initially, we set $\text{dist}[s][0] = 0$ and $\text{dist}[v][j] = \infty$ for all $v \neq s$. The algorithm proceeds using a priority queue that stores triples (d, v, j), where d is the tentative distance to vertex v with exactly j edges. From state (u, j), if $j < k$, then for every edge $(u, v) \in E$ with weight w, we perform the relaxation

$$\text{dist}[v][j+1] \leftarrow \min\{\text{dist}[v][j+1],\ \text{dist}[u][j] + w\}.$$

At the end of the computation, the shortest constrained distance to a vertex v is given by

$$\min_{0 \leq j \leq k} \text{dist}[v][j].$$

Correctness The correctness of the algorithm follows from the fact that any path from s to v using at most k edges can be decomposed into a prefix path of at most j edges and one additional edge, which directly leads to the recurrence used in the relaxation step. Since the algorithm processes states in nondecreasing order of their current distance values, no shorter path to a state is overlooked. In other words, the greedy property of Dijkstra's algorithm is preserved in the augmented state space. Hence, when the algorithm terminates, $\text{dist}[v][j]$ is the length of the shortest path from s to v with exactly j edges, and taking the minimum over $j \leq k$ yields the desired solution.

Time Complexity The augmented graph contains $O(k|V|)$ states, since each vertex is replicated across $k+1$ layers. Each edge may be relaxed up to k times, giving a total of $O(k|E|)$ relaxations. When a binary heap is used for the priority queue, each operation costs $O(\log(k|V|))$. Therefore, the total running time of the algorithm is

$$O\big((k|V| + k|E|)\log(k|V|)\big).$$

For small values of k, this remains close to the performance of the standard Dijkstra algorithm, while in the worst case ($k = |V|$) it is asymptotically slower by a factor of $|V|$.

5. **Dijkstra with Maximum Edge Weight** Let $G = (V, E)$ be a directed graph with nonnegative edge weights. Given a source vertex $s \in V$ and a threshold $W_{\max} \geq 0$, a path is called *feasible* if all its edges have weights at most $W_{\max}$. Modify Dijkstra's algorithm to compute the shortest distances from s to all vertices along *feasible* paths only. Explain why your modification is correct and analyze its time complexity.

   Solution

   Algorithm

   1. Initialize $\text{dist}[s] = 0$, and $\text{dist}[v] = \infty$ for all $v \neq s$.
   2. Use a priority queue keyed by tentative distances.
   3. When relaxing an edge (u, v) with weight w, only relax if $w \leq W_{\max}$:

   $$\text{dist}[v] \leftarrow \min(\text{dist}[v], \text{dist}[u] + w)$$

   4. Continue extraction and relaxation as in standard Dijkstra.

Correctness Ignoring edges with weight exceeding $W_{\max}$ is equivalent to running Dijkstra on the subgraph $G' = (V, E')$ with $E' = \{(u, v) \in E : w(u, v) \le W_{\max}\}$. Since all edge weights in E' are nonnegative, the greedy argument of standard Dijkstra still applies: each vertex is extracted with its minimal distance among feasible paths. Therefore, the algorithm correctly computes shortest distances restricted to feasible paths.

Time Complexity Let $|E'|$ denote the number of edges with weight at most $W_{\max}$. Using a binary heap, the running time is

$$O\big((|V| + |E'|) \log |V|\big),$$

which is asymptotically similar to standard Dijkstra, with fewer edges relaxed if many edges exceed the threshold.

6. **Dijkstra with a Pinned Node** You are given a strongly connected directed graph $G = (V, E)$ with positive edge weights along with a particular node $v_0 \in V$. Give an efficient algorithm for finding the shortest path between *all pairs of nodes*, with the one restriction that these paths must all pass through v_0. Make your algorithm as efficient as you can, perhaps as fast as Dijkstra's algorithm. Explain why your algorithm is correct and justify its running time.
Answer:
Let P be the shortest path from vertex u to v passing through v_0. Note that, between v_0 and v, P must necessarily follow the shortest path from v_0 to v. By the same reasoning, between u and v_0, P must follow the shortest path from v_0 and u in the reverse graph. Both these paths are guaranteed to exist as the graph is strongly connected. Hence, the shortest path from u to v through v_0 can be computed for all pairs u, v by performing two runs on Dijkstra's algorithm from v_0, one on the input graph G and the other on the reverse of G. The running time is dominated by looking up all the $O(|V^2|)$ pairs of distances.