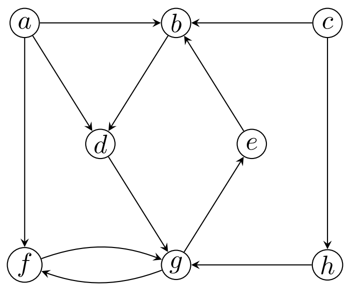
Problem 1 (20 points). You are given the following directed graph $G = (V, E)$.

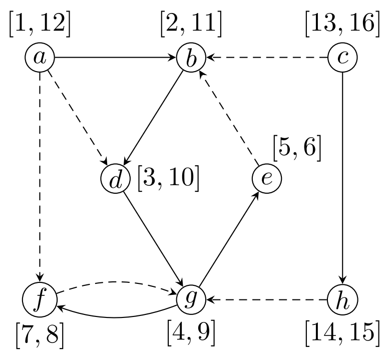


1. Use adjacency list to represent G.

   Solution.
   $a \to b, d, f$
   $b \to d$
   $c \to b, h$
   $d \to g$
   $e \to b$
   $f \to g$
   $g \to e, f$
   $h \to g$

2. Run the DFS (with-time) algorithm using your above adjacency list as input. Give the pre/post interval for each vertex in V. Classify each edge in E as a tree edge, forward edge, back edge, or cross edge.

   Solution. Tree edges are presented as solid edges, non-tree edges are represented as dashed edges. Among these non-tree edges, (a,d), (a,f) are forward edges. (f,g), (e,b) are back edges, (c,b), (h,g) are cross edges.



Problem 2 (20 points). For a given graph G, we know that different adjacency list representation of G leads to different DFS runs, and consequently may result in different search forests. For each of the following questions, either give such an instance, or prove that there does not exist such instance.

1. Can you design an instance of a directed graph, such that one DFS run reveals forward-edge(s), but another DFS run does not?

   Solution. Yes. Consider an instance $G = (V, E)$ with $V = \{a, b, c\}$ and $E = \{(a,b), (a,c), (b,c)\}$. Let a be the first vertex that is explored. Within explore (G, a), if vertex b is explored before c, then edges (a,b) and (b,c) will be tree edges and edge (a,c) will be a forward edge; otherwise, if vertex c is explored before b, then edges (a,c) and (a,b) will be tree edges and edge (b,c) will be a cross edge (and there will be no forward edge).

2. Can you design an instance of a directed graph, such that one DFS run reveals cross-edge(s), but another DFS run does not?

   Solution. Yes. The same instance as above applies.

3. Can you design an instance of a directed graph, such that one DFS run reveals back-edge(s), but another DFS run does not?

   Solution. No. This is because DFS reveals back edge(s) if and only if the graph contains cycle(s), which is a property of the graph and therefore independent of the choices of the adjacency list.

4. Can you design an instance of a directed graph, such that two DFS runs reveals different number of tree-edges?

   Solution. Yes. Consider an instance $G = (V,E)$ with $V = \{a,b\}$ and $E = \{(a,b)\}$. If vertex a is explored before b, then edge (a,b) will be a tree edge. Otherwise, if vertex b is explored before a, then edge (a,b) will be a cross edge (and there will be no tree edge).

Problem 3 (20 points).

1. Given an undirected graph $G = (V,E)$ and an edge $e = (u,v) \in E$, design a linear time algorithm to determine whether there exists a cycle in G that contains e.

   Solution. Let G' be the graph after removing edge $e = (u,v)$ from G. Clearly, we have that there exists a cycle in G that contains e if and only if there exists a path connecting u and v in G'. Therefore, we can design the following algorithm.
   Algorithm. Remove (u,v) in the adjacency list of G; let the new graph be G'. Run explore (G', u) using this updated adjacency list as input. If in the resulting visited array we have visited[v] = 1, (i.e., v can be reached from u in G'), then return true, i.e., there exists a cycle in G that contains e; otherwise return false, i.e., there does not exist a cycle in G that contains e.
   Running Time. Revising the adjacency list to remove $e = (u,v)$ takes $O(|E|)$ time. Run explore (G', u) also takes $O(|E|)$ time. Therefore, the entire algorithm runs in linear time.

2. Given a directed graph $G = (V,E)$ and an edge $e = (u,v) \in E$, design a linear time algorithm to determine whether there exists a cycle in G that contains e.

   Solution. Let G' be the graph after removing edge $e = (u,v)$ from G. Clearly, we have that there exists a cycle in G that contains e if and only if there exists a path from v and u in G'. Therefore, we can design the following algorithm.
   Algorithm. Remove (u,v) in the adjacency list of G; let the new graph be G'. Run explore (G', v) using this updated adjacency list as input. If in the resulting visited array we have visited[u] = 1, (i.e., u can be reached from v in G'), then return true, i.e., there exists a cycle in G that contains e; otherwise return false, i.e., there does not exist a cycle in G that contains e.
   Running Time. Revising the adjacency list to remove $e = (u,v)$ takes $O(|E|)$ time. Run explore (G', v) also takes $O(|E|)$ time. Therefore, the entire algorithm runs in linear time.

Problem 4 (20 points). In an undirected graph, the *degree* of vertex v, denoted as $d(v)$, is defined as the number of neighbors v has, or equivalently, the number of edges adjacent to v.

1. Given an undirected graph $G = (V,E)$ represented as an adjacent list, design an $O(|E|)$ time algorithm to compute the degree for all vertices.

   Solution. The algorithm scans the adjacency list once: for each vertex counts and stores the number of vertices in its corresponding list (which is the degree of this vertex). The algorithm therefore runs in $O(|E|)$ time.

2. For vertex $v \in V$, define $d^2(v)$ as the sum of the degree of all neighbors of v, i.e., $d^2(v) = \sum_{u:(v,u)\in E} d(u)$. Design an $O(|E|)$ time algorithm to compute $d^2(\cdot)$ for all vertices.

   Solution. The algorithm first call the above precedure to compute the degree of all vertices and store them in array $d[1\cdots n]$. The algorithm then allocates another array $d^2[1\cdots n]$, all of which are initialized as 0, to store the $d^2(\cdot)$ for all vertices. The algorithm then scans the adjacency list: for each vertex v_i, traverses the corresponding list of v_i, and for each edge $(v_i, v_j) \in E$, adds $d[j]$ to $d^2[i]$. The entire algorithm traverses the adjacency list twice and therefore it runs in $O(|E|)$ time.

Problem 5 (20 points). We have three containers with sizes of 10 pints, 7 pints, and 4 pints, respectively. The 7-pint and 4-pint containers start with full of water, but the 10-pint container is initially empty. We are allowed the following operation: pouring the contents of one container into another, stopping only when the source container is empty, or the destination container is full. We want to determine, is there a sequence of such operations that leaves exactly 2 pints in the 7-pint or 4-pint container.

1. Model this task as a graph problem: give a precise definition of the graph (i.e., what are the vertices and what are the edges), and then state the question about this graph that needs to be answered.

   Solution. Let $G = (V,E)$ be the corresponding directed graph. Each vertex in V represents a tuple (x,y,z) satisfying $x+y+z = 11$, indicating that there are x pints, y pints, and z pints of water, in the 10-pint container, 7-pint container, and 4-pint container respectively. Formally, $V = \{(x,y,z) \mid x+y+z = 11, \text{ and } x,y,z \text{ are non-negative integers}\}$. We add an edge (v_1, v_2) to E if we can perform one operation to transform $v_1 = (x_1,y_1,z_1)$ into $v_2 = (x_2,y_2,z_2)$. There are at most 6 out-edges for each vertex, corresponding to the 6 possible operations. For instance, following are the out-edges for vertex $(4,5,2)$:
   $(2,7,2)$: pouring water from 10-pints container to the 7-pints container until the later is full;
   $(2,5,4)$: pouring water from 10-pints container to the 4-pints container until the later is full;
   $(9,0,2)$: pouring water from 7-pints container to the 10-pints container until the former is empty;
   $(4,3,4)$: pouring water from 7-pints container to the 4-pints container until the later is full;
   $(6,5,0)$: pouring water from 4-pints container to the 10-pints container until the former is empty;
   $(4,7,0)$: pouring water from 4-pints container to the 7-pints container until the former is empty.
   Clearly, there is a sequence of such operations that leaves exactly 2 pints of water in the 7-pint or 4-pint container, if and only if in the above graph $G = (V,E)$ there exists a path from vertex $(0,7,4)$ to any vertex (x,y,z) satisfying $y = 2$ or $z = 2$. If we have $y = 2$ (resp. $z = 2$), then we can perform another operation to pour all water in the 4-pint (resp. 7-pint) container to the 10-pint container to arrive at vertex $(9,2,0)$ (resp. $(9,0,2)$). And we can also perform one operation to transfer $(9,2,0)$ into $(9,0,2)$, and back. Therefore, there is a sequence of operations that leaves exactly 2 pints of water in the 7-pint or 4-pint container, if and only if in the above graph $G = (V,E)$ there exists a path from vertex $(0,7,4)$ to vertex $(9,0,2)$.

2. What algorithm should be applied to solve the task?

   Solution. Run explore $(G, v = (0,7,4))$ and verify if visited$[(9,0,2)]$ equals to 1.