Exercise 20.1-5

20.1-5

The ***square*** of a directed graph $G = (V, E)$ is the graph $G^2 = (V, E^2)$ such that $(u, v) \in E^2$ if and only if G contains a path with at most two edges between u and v. Describe efficient algorithms for computing G^2 from G for both the adjacency-list and adjacency-matrix representations of G. Analyze the running times of your algorithms.

Sol)

With the adjacency-matrix representation, computing the square of a graph is akin to squaring a matrix, but keeping all entries as 0 or 1. We also have to add self-loops for all vertices because each vertex contains a path with 0 edges from itself to itself.

```
SQUARE-ADJACENCY-MATRIX(G)
  allocate n × n matrix G² = (g²_ij), with all entries initially 0
  for i = 1 to n
      g²_ii = 1                         // path of length 0: i ⇝ i
      for k = 1 to n
          if g_ik == 1
              g²_ik = 1                 // path of length 1: i ⇝ k
          for j = 1 to n
              if g_ik == 1 and g_kj == 1
                  g²_ij = 1             // path of length 2: i ⇝ k ⇝ j
  return G²
```

This procedure takes $O(V^3)$ time.

With the adjacency-list representation, a procedure can go through each edge (i, j) and then find all the edges (j, k) in j's adjacency list, adding edge (i, k) to G^2. There is one hitch, however: making sure not to add an edge multiple times. If there are edges (i, j), (j, k), (i, h), and (h, k), then there are two paths of length 2 from i to k ($i \rightsquigarrow j \rightsquigarrow k$ and $i \rightsquigarrow h \rightsquigarrow k$), but edge (i, k) should appear in i's adjacency list in G^2 just once. We'll build an adjacency matrix for G^2, just like the output of SQUARE-ADJACENCY-MATRIX, but by going through the adjacency lists of G. As we'll see, that takes $O(V^2 + VE)$ time, which beats $O(V^3)$ for SQUARE-ADJACENCY-MATRIX if $|E| = o(V^2)$. Once the adjacency matrix of G^2 is built, converting it to adjacency lists takes $O(V^2)$ time, for a total of $O(V^2 + VE)$ time.

```
SQUARE-ADJACENCY-LISTS(G)
  // Since G² is the output adjacency list representation, use a different name
  // for the adjacency matrix.
  allocate n × n matrix M = (m_ij), with all entries initially 0
  for each vertex i in G.Adj
      m_ii = 1
      for each vertex j in G.Adj[i]
          m_ij = 1
          for each vertex k in G.Adj[j]
              m_ik = 1
  // M is the complete adjacency matrix for G². Convert to adjacency lists.
  allocate G².Adj with |G.V| entries, each an empty linked list
  for each vertex i in G.Adj
      for each vertex j in G.Adj
          if m_ij == 1
              add edge (i, j) to the linked list in G².Adj[i]
  return G²
```

Allocating M and converting M to adjacency lists takes $O(V^2)$ time. The first set of triply-nested **for** loops takes $O(VE)$ time because for each of the $|E|$ edges (i, j) in the middle loop, at most $|V|$ edges appear in j's adjacency list. Thus, the entire procedure takes time $O(V^2 + VE)$.

Exercise 20.2-8

20.2-8

The ***diameter*** of a tree $T = (V, E)$ is defined as $\max\{\delta(u, v) : u, v \in V\}$, that is, the largest of all shortest-path distances in the tree. Give an efficient algorithm to compute the diameter of a tree, and analyze the running time of your algorithm.

Sol)

To find the diameter of tree T, select any vertex x in T, and run BFS from x. Let y be the last vertex discovered in the BFS from x. Now run BFS from y, and let

z be the last vertex discovered in this second BFS. We claim that the diameter of T is the value of $z.d$ from the second BFS. In other words, since there is a unique simple path between each pair of vertices in T, we claim that the diameter equals $\delta(y, z)$.

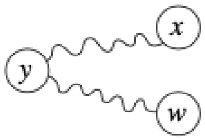
To prove that the diameter of T is the distance between y and z, let u and v be any two vertices in T such that $\delta(u, v)$ equals the diameter of T.

Claim

$\delta(y, v) = \delta(u, v)$.

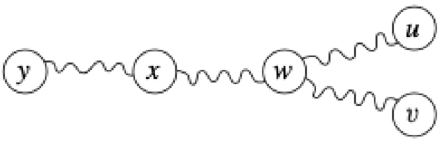
Proof of claim Let w be the first vertex on the path $u \rightsquigarrow v$ discovered during the first BFS. There are three possibilities for the path relationships among x, y, and w.

- x is not on the path $w \rightsquigarrow y$ and w is not on the path $x \rightsquigarrow y$. Then the path $x \rightsquigarrow w$ must go through y:

  Since w is farther from x than y is, we get the contradiction that y is not the last vertex discovered during the BFS from x. Therefore, this case cannot occur.

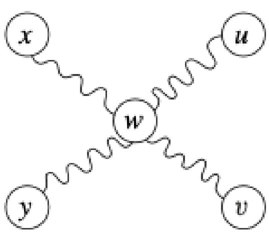
- x is on the path $w \rightsquigarrow y$. Then because w is the first vertex discovered in the BFS from x, no other vertex in the path $u \rightsquigarrow v$ can be on the path $x \rightsquigarrow w$, for it would have been discovered before w in the BFS from x:

  In this case, we have $\delta(w, y) \geq \delta(x, y)$. Since y is the last vertex discovered in the BFS from x, we have $\delta(x, y) \geq \delta(x, u)$. Since the path $x \rightsquigarrow u$ includes the subpath $w \rightsquigarrow u$, we have $\delta(x, u) \geq \delta(w, u)$. Putting these inequalities together gives $\delta(w, y) \geq \delta(x, y) \geq \delta(x, u) \geq \delta(w, u)$, so that $\delta(w, y) \geq \delta(w, u)$. Since the path $v \rightsquigarrow w$ is not a subpath of $u \rightsquigarrow y$, we can add $\delta(v, w)$ to both sides, giving $\delta(y, v) = \delta(v, y) = \delta(v, w) + \delta(w, y) \geq \delta(v, w) + \delta(w, u) = \delta(v, u) = \delta(u, v)$, so that $\delta(y, v) \geq \delta(u, v)$.

  Since the diameter equals $\delta(u, v)$, we have $\delta(y, v) \leq \delta(u, v)$, and so $\delta(y, v) = \delta(u, v)$.

- w is on the path $x \rightsquigarrow y$. As in the previous case, no other vertex in the path $u \rightsquigarrow v$ can be on the path $x \rightsquigarrow w$, for it would have been discovered before w in the BFS from x:

Since $x \rightsquigarrow w$ is a subpath of both $x \rightsquigarrow y$ and $x \rightsquigarrow u$ and y is the last vertex discovered in the BFS from x, we must have $\delta(x,w) + \delta(w,y) = \delta(x,y) \geq \delta(x,u) = \delta(x,w) + \delta(w,u)$. Subtracting $\delta(x,w)$ from both sides gives $\delta(w,y) \geq \delta(w,u)$. Because the diameter equals $\delta(v,u)$, we have $\delta(v,u) \geq \delta(v,y)$. Then, we have $\delta(v,w) + \delta(w,u) = \delta(v,u) \geq \delta(v,y) = \delta(v,w) + \delta(w,y)$, and subtracting $\delta(v,w)$ from both sides gives $\delta(w,u) \geq \delta(w,y)$. Having both $\delta(w,y) \geq \delta(w,u)$ and $\delta(w,u) \geq \delta(w,y)$ means that $\delta(w,y) = \delta(w,u)$. Now, adding back $\delta(v,w)$ to both sides gives $\delta(y,v) = \delta(v,y) = \delta(v,w) + \delta(w,y) = \delta(v,w) + \delta(w,u) = \delta(v,u) = \delta(u,v)$. ■ (claim)

Because the diameter equals $\delta(u,v)$, we have $\delta(y,z) \leq \delta(u,v)$. And because z is the last vertex discovered in the BFS from y, we have $\delta(y,z) \geq \delta(y,v)$, giving $\delta(y,v) \leq \delta(y,z) \leq \delta(u,v)$. From our claim that $\delta(y,v) = \delta(u,v)$, this inequality collapses to an equality, giving $\delta(u,v) = \delta(y,z)$, so that the diameter equals $\delta(y,z)$.

Finding the diameter, therefore, takes two executions of BFS, each of which takes $\Theta(V + E)$ time. Since T is a tree, $|E| = |V| - 1$, giving a total time of just $\Theta(V)$.

Exercise 20.4-2

20.4-2

Give a linear-time algorithm that, given a directed acyclic graph $G = (V, E)$ and two vertices $a, b \in V$, returns the number of simple paths from a to b in G. For example, the directed acyclic graph of Figure 20.8 contains exactly four simple paths from vertex p to vertex v: $\langle p, o, v \rangle$, $\langle p, o, r, y, v \rangle$, $\langle p, o, s, r, y, v \rangle$, and $\langle p, s, r, y, v \rangle$. Your algorithm needs only to count the simple paths, not list them.

Sol)

To count the number of simple paths from s to t in dag G, add an attribute *count* to each vertex and then execute the following procedure.

COUNT-PATHS(G, s, t)

for each vertex $v \in G.V$
    $v.count = 0$
$t.count = 1$
topologically sort G, and let the topologically sorted order be
    $\langle v_1, v_2, \ldots, v_n \rangle$, where $n = |G.V|$
let $s = v_i$ and $t = v_j$, where $i \leq j$
for $k = j$ **downto** i // process vertices in reverse topo order from t to s
    for each vertex u in $G.Adj[v_k]$
        $v_k.count = v_k.count + u.count$
return $v_i.count$ // $v_i.count$ is same as $s.count$

We show that $s.count$ is correct by the following loop invariant:

Loop invariant: After an iteration of the second **for** loop (the loop with the header "**for** $k = j$ **downto** i"), each value $v_k.count, v_{k+1}.count, \ldots, v_n.count$ contains the number of simple paths from v_k to t.

Initialization: Because vertices $v_{j+1}, v_{j+2}, \ldots, v_n$ all follow $v_j = t$ in the topologically sorted order, their *count* values never change from their initial values of 0, which is correct because there are no paths to t from vertices following t in the topologically sorted order. The first iteration of the **for** loop sets $t.count$ to 1, which is correct since there is just one simple path from t to itself: a path with no edges.

Maintenance: An iteration for vertex v_k sets $v_k.count$ to the sum of the *count* values for all vertices adjacent to v_k. By the loop invariant, these *count* values are correct for these vertices. For each vertex u that is adjacent to v_k, there are simple paths from v_k to t going $v_k \to u \rightsquigarrow t$, so that for a fixed vertex u, the number of such paths is given by $u.count$. Summing the *count* values of the v_k's neighbors into v_k gives the total number of simple paths from v_k to t.

Termination: The loop terminates because it visits vertices from t to s in reverse topologically sorted order. By the loop invariant, after the iteration for $v_k = v_i = s$, the value $v_i.count = s.count$ contains the number of simple paths from s to t.

Exercise 21.1-3

21.1-3

Show that if an edge (u, v) is contained in some minimum spanning tree, then it is a light edge crossing some cut of the graph.

Sol)

Let T be a minimum spanning tree containing edge (u, v). Let $T' = T - \{(u, v)\}$ be T with edge (u, v) removed, and define the cut $(S, V - S)$ such that

$$S = \{x \in V : \text{there is a path } u \leadsto x \text{ in } T'\} ,$$

$$V - S = \{x \in V : \text{there is a path } v \leadsto x \text{ in } T'\} .$$

Let (y, z) be a light edge crossing this cut, so that $w(y, z) \leq w(u, v)$, and define the spanning tree $T'' = T' \cup \{(y, z)\}$. Because $w(y, z) \leq w(u, v)$, we have that $w(T'') \leq w(T)$. Since T is a minimum spanning tree, we also have $w(T) \leq w(T'')$, which implies that $w(T'') = w(T)$ and hence $w(y, z) = w(u, v)$. Therefore, (u, v) is a light edge for the cut $(S, V - S)$.

Exercise 21.1-5

21.1-5

Let e be a maximum-weight edge on some cycle of connected graph $G = (V, E)$. Prove that there is a minimum spanning tree of $G' = (V, E - \{e\})$ that is also a minimum spanning tree of G. That is, there is a minimum spanning tree of G that does not include e.

Sol)

Let T be a minimum spanning tree for G. If T does not contain e, then we are done.

So now suppose that T contains e. We will construct another minimum spanning tree that does not contain e. Let $e = (u, v)$ and let $T' = T - \{(u, v)\}$ be T with edge (u, v) removed. Define the cut $(S, V - S)$ such that

$$S = \{x \in V : \text{there is a path } u \leadsto x \text{ in } T'\} ,$$

$$V - S = \{x \in V : \text{there is a path } v \leadsto x \text{ in } T'\} .$$

Because e is on a cycle, some other edge e' in the cycle crosses the cut $(S, V - S)$, and by the definition of e, we have $w(e') \leq w(e)$. Construct the tree $T'' = T' \cup \{e'\}$. Tree T'' is a spanning tree for G with weight $w(T'') = w(T') + w(e') = w(T) - w(e) + w(e') \leq w(T)$. Since we assume that T is a minimum spanning tree for G, T'' must be one as well, and it does not include e.

Exercise 21.1-9

21.1-9

Let T be a minimum spanning tree of a graph $G = (V, E)$, and let V' be a subset of V. Let T' be the subgraph of T induced by V', and let G' be the subgraph of G induced by V'. Show that if T' is connected, then T' is a minimum spanning tree of G'.

Sol)

Because T' is a subgraph of T and T is a tree, T' must be a forest. Moreover, because T' is connected and is the subgraph induced by V', it must be a spanning tree of G'.

To see that T' is minimum spanning tree of G', suppose that G' has a spanning tree S such that $w(S) < w(T')$. Let $\widehat{T} = T - T'$ be the edges in T that are not in T', so that $T = \widehat{T} \cup T'$, and let $T'' = \widehat{T} \cup S$. We claim that T'' is a spanning tree of G with $w(T'') < w(T)$, which contradicts the assumption that T is a minimum spanning tree of G.

We first show that $w(T'') < w(T)$. Just as the edges in T' are disjoint from the edges in $\widehat{T}$, so are the edges in S (or any spanning tree of G'). Thus, we have

$$\begin{aligned} w(T'') &= w(\widehat{T} \cup S) \\ &= w(\widehat{T}) + w(S) \\ &< w(\widehat{T}) + w(T') \\ &= w(\widehat{T} \cup T') \\ &= w(T) \,. \end{aligned}$$

To see that T'' is a spanning tree of G, we show that T'' is acyclic and that $|T''| = |T|$. The latter property follows easily, since both T' and S are spanning trees of G', so that $|T'| = |S|$, and

$$\begin{aligned} |T''| &= \left|\widehat{T} \cup S\right| \\ &= \left|\widehat{T}\right| + |S| \\ &= \left|\widehat{T}\right| + |T'| \\ &= \left|\widehat{T} \cup T'\right| \\ &= |T| \,. \end{aligned}$$

To see that T'' is acyclic, suppose that it has a cycle. That cycle must include edges from both $\widehat{T}$ and S, since each of these sets are, on their own, acyclic. Since the cycle includes at least one edge from S, it must include two vertices $u, v \in V'$. Because S is a tree connecting u and v, there is a unique simple path $u \rightsquigarrow v$ in S. Similarly, there is a unique simple path $u \rightsquigarrow v$ in T'. Adding the edges in $\widehat{T}$ to the edges in T' creates a cycle in the resulting set T of edges, contradicting the assumption that T is a tree. Thus, T'' is acyclic and thus a spanning tree of G with weight less than $w(T)$, contradicting the assumption that T is a minimum spanning tree.