1. (10 points). A dag is a directed acyclic graph (acyclic means no cycles). The topological sorting problem is: Given a dag G, sort the vertices of G in such a way that if G has an edge (u,v) then u occurs earlier than v in the sorted order.

One topological sorting algorithm given in our textbook (page 615) uses DFS as its main component. However there are simpler/more direct ways to do this.

i. Write an algorithm to give a topological sort of a dag G = (V,E) which does not use a depth first search.

ii. Carry out your algorithm on the graph in figure 22.8 on page 615 of our book.

iii. Show that your algorithm can be implemented in O(n+m) steps where n = |V| and m = |E|.

Answer:

We can assume we are given the dag as a list of vertices and edges. The idea now is to keep track, for each vertex v, of its in-degree, where in-d(v) = the number of of vertices u where u → v is an edge in G. Additionally we need to keep track of which vertices have been output by the algorithm and the list of edges leaving v, for each vertex v. We return to how to efficiently do this at the end of this answer.

The algorithm now is: Go through all of the vertices outputting those v that have in-degree 0 (in-d(v) = 0). Once a vertex v is outputted in this was we reduce by one the in-degree of all u's for which v → u in G. We keep repeating these two steps until all vertices have been outputted.

In the example on page 615 this results in the graph being processed in the following steps:

output m,n,p. reduce indegrees of x,q,r and q,o and o,s,z by 1 each time it occurs.
output q,o and reduce t and r,v,s
output s and reduce r output r and reduce u,y
output u,y and reduce t and v
output w,x,t and reduce z
output z and halt

Finally, returning to the issue of doing all this efficiently (that is in O(n+m) time), we need to:

(i) For each vertex u, construct and keep a list of all of w’s where u → w is in G
(ii) Keep a list of all vertices u which have their in-d(w) values reduced to 0. Every time we output a vertex we find such a w we put also put it on the list of 0 in-degree vertices and we output those vertices at the next iteration of the algorithm.

In essence we guaranty that every edge in all of the lists of edges we set up in the beginning phase where we “keep track of the list of edges leaving v, for each vertex v” is gone through exactly once during the running of the algorithm. This takes place when vertex v is outputted.

2. (12 Points) Answer the 4 T (true) or F (false) questions below. Justify your answers here by saying why the statement is true or by giving a small counter example.

i. Let (S,T) be a minimum s-t cut in the network flow graph G, and let (u,v) be an edge that crosses the cut in the forward direction, i.e., vertices s,u are in S and v,t in T.

If you increase the capacity of the edge (u,v) in G then you always also increase the maximum flow of G.

Answer: False. Consider the following graph G.

(S, T) = ({s, a}, {b, t}) is a min cut of G and edge (a, b) crosses the cut. Increasing the capacity of edge (a, b) doesn’t increase the maximum flow of G though since the flow is still constrained by the capacity of edges (s, a) and (b, t).

ii. If all capacities of a flow graph G are multiplied by some integer multiple of 5, then the values of the max flow of G is also a multiple of 5.

Answer: True. If all the capacities of G are multiplied by 5 then the FF algorithm will proceed just as before, every augmenting path found during Ford-Fulkerson will have its path value multiplied by 5. Hence also the sum of the augmenting paths during FF and the total max flow will also be multiplied by 5.

iii. There is no known polynomial time algorithm to solve the max-flow problem.

Answer: False. The FF method is not polynomial time but the Karp-Edmonds version is in P.

iv. Let G be a bipartite graph with n vertices and m edges. There is an algorithm, to test if a matching for G is maximum which runs in time O(n+m) steps.

Answer: True. A matching in G corresponds to a flow in the flow graph G' as discussed in class. A max matching corresponds to a max flow f in the flow graph. As with any max flow, the resulting residual graph G'f then has no s-t path. Setting up the flow network and the residual graph takes O(n+m) time.
(check this), and BFS can be used to check that there is no augmenting path in O(n+m) time as well.

3. (10 points) Use the Ford Fulkerson algorithm to find the maximum flow in the following flow graph.
   Find the min s-t cut of the graph as well.
   Answer:
   The flow graph corresponding to the maximum flow is given below. The value of the maximum flow is 17.

   The minimum cut is (\{S, A, B, C\}, \{D, T\}).

4. (5 points) Give an example of a bipartite graph P = (L,R) whose corresponding flow graph G (as described at the beginning of Section 26.3 of the textbook) has a single unique max-flow and also the property that for every s-t min-cut (S, T) of G both S and T contain at least 2 vertices.
   For the example you should draw the picture of P (showing R and L) and the corresponding picture of G, and then write down all of the s-t min-cuts in G and as well as the one max flow of G.
   Answer:
   Let P be the bipartite graph given below.

   P's corresponding flow graph G is below.
$G$ has a unique max flow, which sends one unit of flow along the path $S \rightarrow A \rightarrow D \rightarrow T$. The S-T cuts and their sizes are listed in the table below.

<table>
<thead>
<tr>
<th>S</th>
<th>T</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>{S}</td>
<td>{A, B, C, D, T}</td>
<td>2</td>
</tr>
<tr>
<td>{S, A}</td>
<td>{B, C, D, T}</td>
<td>2</td>
</tr>
<tr>
<td>{S, B}</td>
<td>{A, C, D, T}</td>
<td>1</td>
</tr>
<tr>
<td>{S, C}</td>
<td>{A, B, D, T}</td>
<td>3</td>
</tr>
<tr>
<td>{S, D}</td>
<td>{A, B, C, D}</td>
<td>3</td>
</tr>
<tr>
<td>{S, A, B}</td>
<td>{C, D, T}</td>
<td>1</td>
</tr>
<tr>
<td>{S, A, C}</td>
<td>{B, D, T}</td>
<td>3</td>
</tr>
<tr>
<td>{S, A, D}</td>
<td>{B, C, T}</td>
<td>2</td>
</tr>
<tr>
<td>{S, B, C}</td>
<td>{A, D, T}</td>
<td>2</td>
</tr>
<tr>
<td>{S, B, D}</td>
<td>{A, C, T}</td>
<td>2</td>
</tr>
<tr>
<td>{S, C, D}</td>
<td>{A, B, T}</td>
<td>4</td>
</tr>
<tr>
<td>{S, A, B, C}</td>
<td>{D, T}</td>
<td>2</td>
</tr>
<tr>
<td>{S, A, B, D}</td>
<td>{C, T}</td>
<td>1</td>
</tr>
<tr>
<td>{S, A, C, D}</td>
<td>{B, T}</td>
<td>3</td>
</tr>
<tr>
<td>{S, B, C, D}</td>
<td>{A, T}</td>
<td>3</td>
</tr>
<tr>
<td>{S, A, B, C, D}</td>
<td>{T}</td>
<td>2</td>
</tr>
</tbody>
</table>

One min cut is ($\{S, B\}, \{A, C, D, T\}$), which is smaller than any cut containing only one vertex.

5. (10 points) (i). Recall the min cut problem for undirected graphs. Give an example of a graph $G$ whose is min cut is not any single vertex of the graph. That is, for any vertex $v$ in $G$ the cut consisting of $v$ on one side and all other vertices on the other is not a min cut.

(ii). Assume $G$ is a 10 node graph and that $G$ has a min cut of cost 5. Explain why $G$ must have at least 25 edges.

(iii). Assume $H$ is a 10 node graph which is a simple cycle. That is $H$ is connected and every node in $H$ has exactly 2 neighbors. Clearly the size of $H$’s min cut is 2. How many different min cuts does $H$ have? Explain your reasoning.

Answer:

i.

Every vertex in this graph has at least degree two, so any cut with a single vertex on one side and the rest of the graph on the other has size at least 2. The min cut, however, has size 1 and is ($\{1, 2, 3\}, \{4, 5, 6\}$)

ii. Suppose there were fewer than 25 edges. Since each edge $\langle u, v \rangle$ contributes 1 to $u$’s degree and 1 to $v$’s degree, there must be a vertex $a$ of degree less than 5, since if every vertex had degree 5 there would have to be $\frac{1}{2}(10 \times 5) = 25$
edges. Consider the cut $\{(a), V - \{a\}\}$. The cost of this cut is $a$’s degree, which is less than 5. This is a contradiction since the min cut was assumed to have cost 5.

iii. Assuming the cut $(A, B)$ is equal to the cut $(B, A)$, $H$ has $\frac{n(n-1)}{2} = \frac{10(9)}{2} = 45$ min cuts. Choose from $n$ nodes where to start set $A$, and choose from the leftover $n - 1$ nodes where to end it ($n - 1$ choices because you can’t have $A = V$) and then divide by 2 since $(A, B) = (B, A)$. These are the min cuts since if you start at node $a$ and end at node $c$ and $b$ is between $a$ and $c$ in the cycle but is not included in $A$, you increase the size of the cut since the edges from $b$ will cross the cut in addition to the edges from $a$ and from $c$. Assuming $(A, B)$ is different than $(B, A)$, you get just $n(n - 1) = 10(9) = 90$ min cuts.