CS 170 First Midterm 15 Feb 2001

NAME (1 pt): ______________________________

TA (1 pt): ______________________________

Name of Neighbor to your left (1 pt): ______________________________

Name of Neighbor to your right (1 pt): ______________________________

**Instructions**: This is a closed book, closed calculator, closed computer, closed network, open brain exam, but you are permited a 1 page, double-sided set of notes, large enough to read without a magnifying glass.

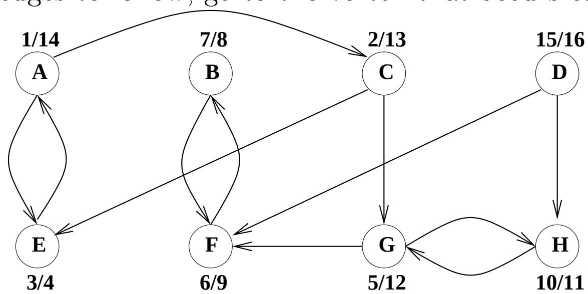
All directed graphs in this exam are assumed not to have self-loops.

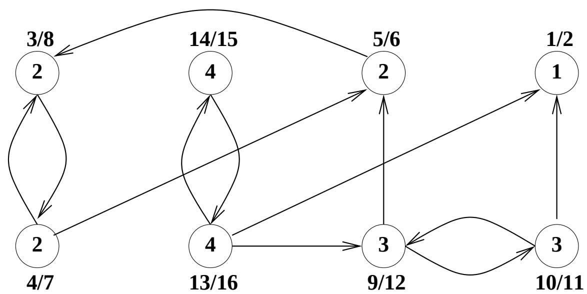
You get one point each for filling in the 4 lines at the top of this page. Each other question is marked by the number of points it is worth.

Write all your answers on this exam. If you need scratch paper, ask for it, write your name on each sheet, and attach it when you turn it in (we have a stapler).

| | |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| Total | |

**1a) (20 points)** Run the Strongly Connected Components (SCC) algorithm on the following graph. Show the pre and post numbers for both passes of DFS. Where there are several choices of vertex to start at or edges to follow, go to the vertex that occurs earliest alphabetically.



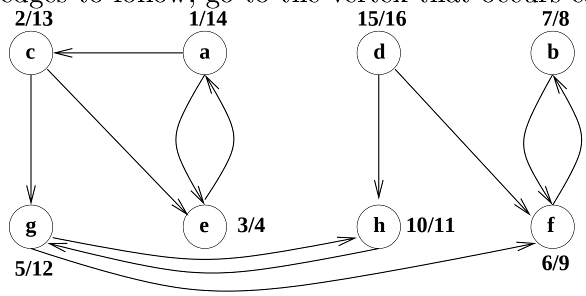



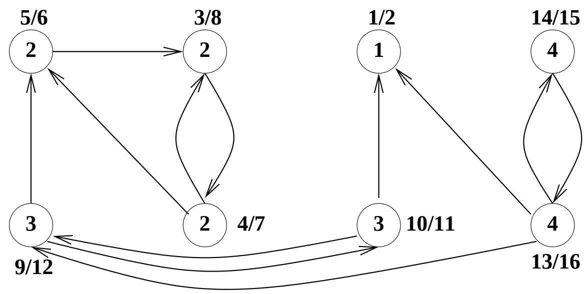

**1b)** How can we topologically sort the SCCs *without* running another pass of DFS? Assume you still have all the information generated by the SCC algorithm available, including pre and post numbers generated during each pass of DFS, and the SCC number assigned to each vertex.

**ANSWER.** During the second pass of DFS in the SCC algorithm, each connected component of $G$ was visited in topological order, so we can simply output the components in the order they were visited.

Another solution is to sort the components in reverse order by the maximum `post` number in each component from the first pass of DFS. Or, sort the components in forward order by the minimum/maximum `pre`/`post` number in each component from the second pass of DFS.

**1a) (20 points)** Run the Strongly Connected Components (SCC) algorithm on the following graph. Show the pre and post numbers for both passes of DFS. Where there are several choices of vertex to start at or edges to follow, go to the vertex that occurs earliest alphabetically.



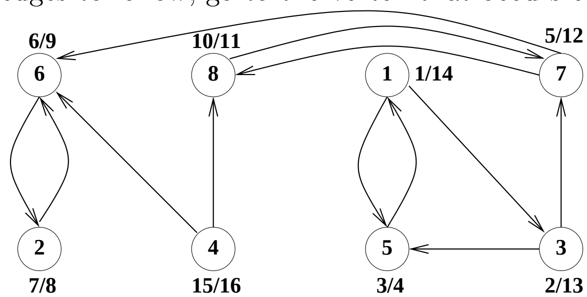



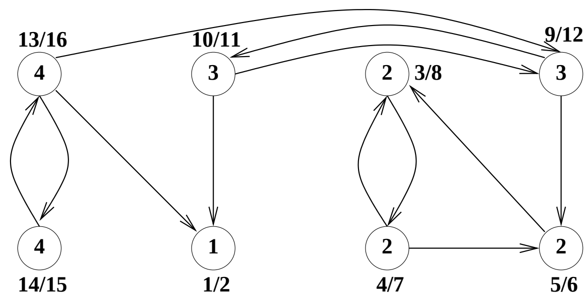

**1b)** How can we topologically sort the SCCs *without* running another pass of DFS? Assume you still have all the information generated by the SCC algorithm available, including pre and post numbers generated during each pass of DFS, and the SCC number assigned to each vertex.

**ANSWER.** During the second pass of DFS in the SCC algorithm, each connected component of $G$ was visited in topological order, so we can simply output the components in the order they were visited.

Another solution is to sort the components in reverse order by the maximum `post` number in each component from the first pass of DFS. Or, sort the components in forward order by the minimum/maximum `pre`/`post` number in each component from the second pass of DFS.

**1a) (20 points)** Run the Strongly Connected Components (SCC) algorithm on the following graph. Show the pre and post numbers for both passes of DFS. Where there are several choices of vertex to start at or edges to follow, go to the vertex that occurs earliest numerically.




**1b)** How can we topologically sort the SCCs *without* running another pass of DFS? Assume you still have all the information generated by the SCC algorithm available, including pre and post numbers generated during each pass of DFS, and the SCC number assigned to each vertex.

**ANSWER.** During the second pass of DFS in the SCC algorithm, each connected component of $G$ was visited in topological order, so we can simply output the components in the order they were visited.

Another solution is to sort the components in reverse order by the maximum `post` number in each component from the first pass of DFS. Or, sort the components in forward order by the minimum/maximum `pre`/`post` number in each component from the second pass of DFS.

**2) (24 points) True or False?** No explanation required, except for partial credit. Each correct answer is worth 2 points, but 2 points will be *subtracted* for each wrong answer, so answer only if you are reasonably certain.

1. Let $X$ be a minimum spanning tree of the weighted undirected graph $G(V,E)$. Let $G'(V',E')$ be another graph defined by augmenting $G$ with a new vertex $u$ and some weighted edges incident on $u$. In other words, $V' = V \cup \{u\}$ and $E' = E \cup F$ where $F$ is a set of edges touching $u$. Then there is always a minimum spanning tree $X'$ of $G'$ such that $X \subset X'$.

   **ANSWER.** False. If the weights of edges in $E$ are all very large, and $u$ has a very light edge connecting it to every vertex in $V$, then the unique minimum spanning tree will equal $F$.

2. $e^{c\sqrt{n}}$ is $O(e^{\sqrt{n}})$ for all $c > 0$.

   **ANSWER.** False, because $e^{c\sqrt{n}} = e^{\sqrt{n}}e^{(c-1)\sqrt{n}}$ and $e^{(c-1)\sqrt{n}}$ is not $O(1)$ for $c > 1$.

3. $n^{2+\frac{\sin(n)}{\log n}}$ is $O(n^2)$.

   **ANSWER.** True, because $n^{2+\frac{\sin(n)}{\log n}} = n^2 n^{\frac{\sin(n)}{\log n}} = n^2 2^{\sin n} \le 2n^2$.

4. Let $G$ be an undirected graph, and let $G'$ be another undirected graph. $G'$ has one vertex for each biconnected component of $G$ (and no other vertices). Two vertices in $G'$ are connected by an edge if and only if the two corresponding biconnected components in $G$ share a vertex. Then $G'$ can have a cycle.

   **ANSWER.** True. Let $G$ consist of three triangles (a triangle is a graph with 3 vertices and 3 edges) sharing a single common vertex. Then each triangle is a biconnected component, and $G'$ is also a triangle.

5. Let $G$ be a directed graph. The Bellman-Ford algorithm will find the *longest* path (if it is finite, or report if it is not) from the start vertex $s$ to any other vertex $v$ in a graph if we negate all the edge weights $W(e)$ before running the algorithm.

   **ANSWER.** True, because minimizing $\sum_{e \in path} -W(e)$ over paths is the same as maximizing $\sum_{e \in path} W(e)$.

6. Let $G$ be a directed graph with positive edge weights $W(e)$. Let $W_{max} = \max_e W(e)$ be the maximum edge weight. Dijkstra's algorithm will find the *longest* path from the start vertex $s$ to any other vertex $v$ in a graph if we replace all the edge weights $W(e)$ by the new positive weights $1 + W_{max} - W(e)$ before running the algorithm.

   **ANSWER.** False, because minimizing $\sum_{e \in path}(1 + W_{max} - W(e)) = Length(path) \cdot (1 + W_{max}) - \sum_{e \in path} W(e)$ depends on the length of the path, and so is not necessarily the same as maximizing $\sum_{e \in path} W(e)$.

7. After we run DFS on a directed graph $G$, it is possible for a single vertex $v \in G$ to be simultaneously incident on at least one back edge, forward edge, cross edge and tree edge.

   **ANSWER.** True

8. It is always cheaper to use a binary heap instead of a linked list to implement Dijkstra's algorithm.

   **ANSWER.** False. A binary heap can cost $O(n^2 \log n)$ on a dense graph whereas a linked list costs $O(n^2)$.

9. Let $T(0) = 1$ and $T(n) = 2^{T(n-1)}$ for $n \geq 1$. Then the cost of computing $T(n)$ and printing it out as a binary integer is $\Theta(T(n-1))$.

   **ANSWER.** True, because $T(n)$ is $T(n-1)$ bits long, which is how long it takes just to write down the answer.

10. If $T(n) = 49T(n/7) + n^2 \cos \sqrt{n}$, then $T(n) = O(n^2 \log n)$.

    **ANSWER.** True, because $T(n)$ is bounded above by $S(n)$ where $S(n) = 49S(n/7)+n^2$, and from the Master Theorem $S(n) = \Theta(n^2 \log n)$.

11. Let $G(V, E)$ be an undirected graph. Then $G$ has a cycle if and only if it is possible to put directions on all the edges in $E$ so that every vertex in $V$ has an edge pointing to it.

    **ANSWER.** False. It would be true if $G$ were connected, by the following argument: If there is a cycle, direct all the edges in the cycle "circularly." Then traverse the rest of the graph by DFS starting from vertices in the cycle, and make all the tree edges point away from the cycle. Back edges can be directed arbitrarily. If there is no cycle, then $G$ is a tree, with $|E| = |V| - 1$, so there aren't enough edges to point to every vertex. But if $G$ is disconnected with one acyclic component and one component with a cycle, then the statement is not true.

12. The paths computed by Bellman-Ford after the 2nd iteration might be the shortest.

    **ANSWER.** True. In fact Bellman-Ford might be done after one iteration if it is applied to a chain in topological order.

**2) (24 points) True or False?** No explanation required, except for partial credit. Each correct answer is worth 2 points, but 2 points will be *subtracted* for each wrong answer, so answer only if you are reasonably certain.

1. It is always cheaper to use a binary heap instead of a linked list to implement Dijkstra's algorithm.

   **ANSWER.** False. A binary heap can cost $O(n^2 \log n)$ on a dense graph whereas a linked list costs $O(n^2)$.

2. Let $G$ be an undirected graph, and let $G'$ be another undirected graph. $G'$ has one vertex for each biconnected component of $G$ (and no other vertices). Two vertices in $G'$ are connected by an edge if and only if the two corresponding biconnected components in $G$ share a vertex. Then $G'$ can have a cycle.

   **ANSWER.** True. Let $G$ consist of three triangles (a triangle is a graph with 3 vertices and 3 edges) sharing a single common vertex. Then each triangle is a biconnected component, and $G'$ is also a triangle.

3. Let $G$ be a directed graph. The Bellman-Ford algorithm will find the *longest* path (if it is finite, or report if it is not) from the start vertex $s$ to any other vertex $v$ in a graph if we negate all the edge weights $W(e)$ before running the algorithm.

   **ANSWER.** True, because minimizing $\sum_{e \in path} -W(e)$ over paths is the same as maximizing $\sum_{e \in path} W(e)$.

4. Let $G$ be a directed graph with positive edge weights $W(e)$. Let $W_{max} = \max_e W(e)$ be the maximum edge weight. Dijkstra's algorithm will find the *longest* path from the start vertex $s$ to any other vertex $v$ in a graph if we replace all the edge weights $W(e)$ by the new positive weights $1 + W_{max} - W(e)$ before running the algorithm.

   **ANSWER.** False, because minimizing $\sum_{e \in path}(1 + W_{max} - W(e)) = Length(path) \cdot (1 + W_{max}) - \sum_{e \in path} W(e)$ depends on the length of the path, and so is not necessarily the same as maximizing $\sum_{e \in path} W(e)$.

5. After we run DFS on a directed graph $G$, it is possible for a single vertex $v \in G$ to be simultaneously incident on at least one back edge, forward edge, cross edge and tree edge.

   **ANSWER.** True

6. Let $X$ be a minimum spanning tree of the weighted undirected graph $G(V, E)$. Let $G'(V', E')$ be another graph defined by augmenting $G$ with a new vertex $u$ and some weighted edges incident on $u$. In other words, $V' = V \cup \{u\}$ and $E' = E \cup F$ where $F$ is a set of edges touching $u$. Then there is always a minimum spanning tree $X'$ of $G'$ such that $X \subset X'$.

   **ANSWER.** False. If the weights of edges in $E$ are all very large, and $u$ has a very light edge connecting it to every vertex in $V$, then the unique minimum spanning tree will equal $F$.

7. If $T(n) = 49T(n/7) + n^2 \cos\sqrt{n}$, then $T(n) = O(n^2 \log n)$.

   **ANSWER.** True, because $T(n)$ is bounded above by $S(n)$ where $S(n) = 49S(n/7) + n^2$, and from the Master Theorem $S(n) = \Theta(n^2 \log n)$.

8. The paths computed by Bellman-Ford after the 2nd iteration might be the shortest.

   **ANSWER.** True. In fact Bellman-Ford might be done after one iteration if it is applied to a chain in topological order.

9. Let $G(V, E)$ be an undirected graph. Then $G$ has a cycle if and only if it is possible to put directions on all the edges in $E$ so that every vertex in $V$ has an edge pointing to it.

   **ANSWER.** False. It would be true if $G$ were connected, by the following argument: If there is a cycle, direct all the edges in the cycle "circularly." Then traverse the rest of the graph by DFS starting from vertices in the cycle, and make all the tree edges point away from the cycle. Back edges can be directed arbitrarily. If there is no cycle, then $G$ is a tree, with $|E| = |V| - 1$, so there aren't enough edges to point to every vertex. But if $G$ is disconnected with one acyclic component and one component with a cycle, then the statement is not true.

10. Let $T(0) = 1$ and $T(n) = 2^{T(n-1)}$ for $n \geq 1$. Then the cost of computing $T(n)$ and printing it out as a binary integer is $\Theta(T(n-1))$.

    **ANSWER.** True, because $T(n)$ is $T(n-1)$ bits long, which is how long it takes just to write down the answer.

11. $e^{c\sqrt{n}}$ is $O(e^{\sqrt{n}})$ for all $c > 0$.

    **ANSWER.** False, because $e^{c\sqrt{n}} = e^{\sqrt{n}} e^{(c-1)\sqrt{n}}$ and $e^{(c-1)\sqrt{n}}$ is not $O(1)$ for $c > 1$.

12. $n^{2+\frac{\sin(n)}{\log n}}$ is $O(n^2)$.

    **ANSWER.** True, because $n^{2+\frac{\sin(n)}{\log n}} = n^2 n^{\frac{\sin(n)}{\log n}} = n^2 2^{\sin n} \leq 2n^2$.

**2) (24 points) True or False?** No explanation required, except for partial credit. Each correct answer is worth 2 points, but 2 points will be *subtracted* for each wrong answer, so answer only if you are reasonably certain.

1. Let $G$ be a directed graph. The Bellman-Ford algorithm will find the *longest* path (if it is finite, or report if it is not) from the start vertex $s$ to any other vertex $v$ in a graph if we negate all the edge weights $W(e)$ before running the algorithm.

   **ANSWER.** True, because minimizing $\sum_{e\in path} -W(e)$ over paths is the same as maximizing $\sum_{e\in path} W(e)$.

2. Let $G$ be a directed graph with positive edge weights $W(e)$. Let $W_{max} = \max_e W(e)$ be the maximum edge weight. Dijkstra's algorithm will find the *longest* path from the start vertex $s$ to any other vertex $v$ in a graph if we replace all the edge weights $W(e)$ by the new positive weights $1 + W_{max} - W(e)$ before running the algorithm.

   **ANSWER.** False, because minimizing $\sum_{e\in path}(1 + W_{max} - W(e)) = Length(path) \cdot (1 + W_{max}) - \sum_{e\in path} W(e)$ depends on the length of the path, and so is not necessarily the same as maximizing $\sum_{e\in path} W(e)$.

3. $e^{c\sqrt{n}}$ is $O(e^{\sqrt{n}})$ for all $c > 0$.

   **ANSWER.** False, because $e^{c\sqrt{n}} = e^{\sqrt{n}}e^{(c-1)\sqrt{n}}$ and $e^{(c-1)\sqrt{n}}$ is not $O(1)$ for $c > 1$.

4. $n^{2+\frac{\sin(n)}{\log n}}$ is $O(n^2)$.

   **ANSWER.** True, because $n^{2+\frac{\sin(n)}{\log n}} = n^2 n^{\frac{\sin(n)}{\log n}} = n^2 2^{\sin n} \le 2n^2$.

5. After we run DFS on a directed graph $G$, it is possible for a single vertex $v \in G$ to be simultaneously incident on at least one back edge, forward edge, cross edge and tree edge.

   **ANSWER.** True

6. It is always cheaper to use a binary heap instead of a linked list to implement Dijkstra's algorithm.

   **ANSWER.** False. A binary heap can cost $O(n^2 \log n)$ on a dense graph whereas a linked list costs $O(n^2)$.

7. Let $X$ be a minimum spanning tree of the weighted undirected graph $G(V, E)$. Let $G'(V', E')$ be another graph defined by augmenting $G$ with a new vertex $u$ and some weighted edges incident on $u$. In other words, $V' = V \cup \{u\}$ and $E' = E \cup F$ where $F$ is a set of edges touching $u$. Then there is always a minimum spanning tree $X'$ of $G'$ such that $X \subset X'$.

   **ANSWER.** False. If the weights of edges in $E$ are all very large, and $u$ has a very light edge connecting it to every vertex in $V$, then the unique minimum spanning tree will equal $F$.

8. Let $G(V, E)$ be an undirected graph. Then $G$ has a cycle if and only if it is possible to put directions on all the edges in $E$ so that every vertex in $V$ has an edge pointing to it.

   **ANSWER.** False. It would be true if $G$ were connected, by the following argument: If there is a cycle, direct all the edges in the cycle "circularly." Then traverse the rest of the graph by DFS starting from vertices in the cycle, and make all the tree edges point away from the cycle. Back edges can be directed arbitrarily. If there is no cycle, then $G$

is a tree, with $|E| = |V| - 1$, so there aren't enough edges to point to every vertex. But if $G$ is disconnected with one acyclic component and one component with a cycle, then the statement is not true.

9. The paths computed by Bellman-Ford after the 2nd iteration might be the shortest.

   **ANSWER.** True. In fact Bellman-Ford might be done after one iteration if it is applied to a chain in topological order.

10. Let $T(0) = 1$ and $T(n) = 2^{T(n-1)}$ for $n \geq 1$. Then the cost of computing $T(n)$ and printing it out as a binary integer is $\Theta(T(n-1))$.

    **ANSWER.** True, because $T(n)$ is $T(n-1)$ bits long, which is how long it takes just to write down the answer.

11. If $T(n) = 49T(n/7) + n^2 \cos \sqrt{n}$, then $T(n) = O(n^2 \log n)$.

    **ANSWER.** True, because $T(n)$ is bounded above by $S(n)$ where $S(n) = 49S(n/7) + n^2$, and from the Master Theorem $S(n) = \Theta(n^2 \log n)$.

12. Let $G$ be an undirected graph, and let $G'$ be another undirected graph. $G'$ has one vertex for each biconnected component of $G$ (and no other vertices). Two vertices in $G'$ are connected by an edge if and only if the two corresponding biconnected components in $G$ share a vertex. Then $G'$ can have a cycle.

    **ANSWER.** True. Let $G$ consist of three triangles (a triangle is a graph with 3 vertices and 3 edges) sharing a single common vertex. Then each triangle is a biconnected component, and $G'$ is also a triangle.

**3) (18 points)** Let $G(V, E)$ be a directed graph, with a weight $W(v)$ associated with each vertex $v \in V$. Let $p =< v_0, v_1, v_2, ..., v_n >$ be a path in $G$, with no repeated vertices. Then the weight of the path $p$ is defined as the sum of the vertex weights on the path: $W(p) = \sum_{i=0}^{n} W(v_i)$.

**3a)** Suppose the vertex weights are positive. Let $s \in V$ be a vertex. Give an efficient algorithm for finding the paths of minimum weight from $s$ to all other vertices in $G$. You should give high-level pseudo code for the algorithm, justify its correctness, and analyze its complexity in a $O()$ sense. You may use any algorithm presented and analyzed in class so far without repeating its correctness proof or complexity analysis.

**ANSWER.** This part was worth 9 points, 3 each for a correct algorithm, a proof of correctness, and a complexity analysis. The reason the answer is not just "Dijkstra" is that the weights of $G$ are on the vertices, not the edges, where Dijkstra needs them. There are a number of solutions, all with the same pattern:

1. Define a new graph $G'(V', E')$, closely related to $G$ but with *edge* weights instead of *vertex* weights, that has the same set of shortest (edge weight) paths between any pair of vertices.

2. Run Dijkstra (or Bellman-Ford, for part 3b)) on $G'$.

Full credit required identifying the graph $G'$, showing that the shortest paths were the same, and applying Dijkstra to $G'$; saying the complexity was therefore the same as Dijkstra was enough.

Here are some ways to build $G'(V', E')$. At least one student found each solution.

1. Let $V' = V$ and $E' = E$, i.e. $G'$ and $G$ has the same vertices and edges. Then let the weight $W_e(u, v)$ of edge $(u, v)$ be the weight of vertex $v$: $W_e(u, v) = W(v)$. (The subscript on $W_e$ reminds us that it gives edge weights on $G'$.) Now consider any path $p =< s, v_1, v_2, ..., v_n, d >$ from $s$ to $d$ in $G$ or $G'$. Its weight in $G$ is $W(p) = W(s) + W(d) + \sum_{i=1}^{n} W(v_n)$, and its weight in $G'$ is

$$W_e(p) = W_e(s, v_1) + \sum_{i=1}^{n-1} W_e(v_i, v_{i+1}) + W_e(v_n, d) = \sum_{i=1}^{n} W(v_i) + W(d)$$

   that is $W_e(p) = W(p) - W(s)$. So all paths from $s$ to $d$ in $G'$ have the same weights as in $G$, minus $W(s)$. In particular, the shortest paths are all the same, so Dijkstra's method finds them correctly. To get the correct $W(p)$ as well, one needs to add $W(s)$ to the paths found by Dijkstra, but this was not explicitly required by the question.

2. Let $V' = V$ and $E' = E$ as above, but now let $W_e(u, v) = W(u)$ instead of $W(v)$. The same algebra as above shows $W_e(p) = W(p) - W(d)$ instead of $W(p) - W(s)$, but again the shortest paths from $s$ to $d$ in $G$ and $G'$ are the same.

3. Let $V' = V$, $E' = E$ and $W_e(u, v) = \alpha W(u) + \beta W(v)$ where $\alpha \geq 0$, $\beta \geq 0$ and $\alpha + \beta > 0$. This includes the above two solutions as special cases ($\alpha = 0$ and $\beta = 1$ for the first algorithm; $\alpha = 1$ and $\beta = 0$ for the second algorithm), another solution turned in ($\alpha = \beta = .5$) and many more besides. The same algebra as before shows that

$$W_e(p) = \alpha W(s) + (\alpha + \beta) \sum_{i=1}^{n} W(v_i) + \beta W(d)$$

   This means that if $p_1$ and $p_2$ are any two paths from $s$ to $d$, then $W(p_1) \geq W(p_2)$ if and only if $W_e(p_1) \geq W_e(p_2)$, i.e. the shortest paths in $G$ and $G'$ are the same.

4. One last solution uses a different $G'(V', E')$. For each $v \in V$, create two vertices $v_{in} \in V'$ and $v_{out} \in V'$. Let there be an edge $(v_{in}, v_{out}) \in E'$ for every $v \in V$, and let $W_e(v_{in}, v_{out}) = W(v)$. To get the remaining edges in $E'$, take every $(u, v) \in E$ and convert it to $(u_{out}, v_{in}) \in E'$. (Draw a picture to see how this works.) Again consider any path $p =< s, v_1, v_2, ..., v_n, d >$ from $s$ to $d$ in $G$. There is a corresponding path

$$p' =< s_{in}, s_{out}, v_{1,in}, v_{1,out}, v_{2,in}, v_{2,out}, ..., v_{n,in}, v_{n,out}, d_{in}, d_{out} >$$

in $G'$. It is easy to see that this defines a one-to-one correspondence between paths from $s$ to $d$ in $G$, and paths from $s_{in}$ to $d_{out}$ in $G'$, since any path must have every other edge be of the form $(v_{in}, v_{out})$. Furthermore, it is obvious that $W_e(p') = W(p)$. Therefore, shortest paths from $s$ to $d$ in $G$ are the same as shortest paths from $s_{in}$ to $d_{out}$ in $G'$, and Dijkstra works again. The cost for a binary heap implementation is $O(|E'| \log |V'|) = O((|E|+|V|) \log 2|V|)$, which is essentially the same as Dijkstra applied to $G$ itself.

**3b)** Now Suppose the vertex weights may take any values, not just positive ones, and answer the same question as in part 3a.

**ANSWER.** The approach is the same as 3a: create graph $G'$ and run Bellman-Ford, for a total cost of $O(|V'| \cdot |E'|)$,

**4) (18 points)** Let $T$ be a MST in the weighted graph $G(V, E)$. Let $G'(V', E')$ be a graph obtained by removing one vertex $u$ from $G$ (so that $V' = V - \{u\}$), and by removing all the edges incident on $u$.

Let $e \in E$ be any edge of $T$ not incident on $u$. Prove that there is an MST of $G'$ that contains $e$.

Hint: The *cut property* says:

> Let $X \subseteq T$ where $T$ is a Minimum Spanning Tree (MST) in the weighted graph $G(V, E)$. Let $S \subset V$ such that no edge in $X$ crosses between $S$ and $V - S$; i.e. no edge in $X$ has one endpoint in $S$ and one endpoint in $V - S$. Among edges crossing between $S$ and $V - S$, let $e$ be an edge of minimum weight. Then $X \cup \{e\} \subseteq T'$ where $T'$ is a MST in $G(V, E)$.

**ANSWER.** If we were to remove the edge $e$, $T$ would be partitioned into two subtrees $T_1$ and $T_2$. We claim $e$ is a minimum edge crossing from $T_1$ to $T_2$. Why? If some other edge $e'$ crossing from $T_1$ to $T_2$ has weight $w(e') < w(e)$, then $T_1 \cup T_2 \cup \{e'\}$ is a smaller spanning tree in $G$ than $T$. But $T$ is a MST in $G$, so $w(e')$ cannot be less than $w(e)$.

Let $S$ be the set of vertices spanned by $T_1$, excepting $u$. Because we leave out $u$, $S \subset V'$. By setting $X = \{\}$ in the cut theorem, we find that some MST of $G'$ contains $e$.