HW 1 Solution: CS 473: Fundamental Algorithms, Spring 2013

1. (30 PTS.) The closet mayhem problem.

A new broom closet was built for the Magical Science department of the Unseen University. The department has n students, and every student has a broom. The closet has $m \geq n$ slots where one can place a broom. The ith broom b_i, can be placed only into two possible slots, denoted by x_i and x'_i, where $x_i, x'_i \in \{1, \ldots, m\}$, for $i = 1, \ldots, n$. Given these locations, design an algorithm that decides in polynomial time if there is a legal way to place all of them in the closet, such that no two brooms share a slot. To this end, answer the following.

(A) (5 PTS.) Consider the graph G with $2n$ nodes, where for every broom there are two nodes $[i : x_i]$ or $[i : x'_i]$. For $\alpha \in \{x_i, x'_i\}$ and $\beta \in \{x_j, x'_j\}$, place an edge from $[i : \alpha]$ to $[j : \beta]$, if placing the ith broom at α implies that the jth broom must be placed in the slot β because the other placement of the jth broom is α. How many edges can this graph has in the worst case? What is the running time of your algorithm to compute this graph?

Solution:

We scan the input, and we register a broom, with each slot that it want to be placed in. Now, for every slot, if it has k brooms registered with it, we need to generate $k(k-1)$ edges in the corresponding graph. Indeed, if L_i is the list of brooms registered with the ith slot, then for for every ordered pair of brooms in this list, we need to introduce an edge in the graph.

Since in the worst case every pair of brooms might share a slot, in the worst case this graph might have $n(n-1)$ edges.

(B) (5 PTS.) If there is a path in G from $[i : \alpha]$ to $[j : \beta]$, then we say that $b_i = \alpha$ ***forces*** $b_j = \beta$. Prove that if $b_i = x_i$ forces $b_j = x_j$ then the "reverse" must hold; that is, $b_j = x'_j$ forces $b_i = x'_i$.

Solution:

Proof: Formally, the proof of this claim is by induction on the length of the path. Say there is a path from $[i : x_i]$ to $[j : x_j]$, and it is of length one, then there is an edge between these two vertices. This specifies that if we place the ith at slot x_i then the jth broom must be placed in the slot x_j; namely, the other slot of the jth broom (i.e., x'_j) must be equal to x_i. In particular, if we place the jth broom at slot $x'_j = x_i$, then we must place the ith broom in x'_i. That is, there is an edge in the graph between $[j : x'_j]$ $[i : x'_i]$.

So assume the claim holds for paths of length k, and consider a path π in the graph of length $k+1$, from say, $[i_1 : x_{i_1}]$ to $[i_{k+1} : x_{i_{k+1}}]$, where the tth edge in this path is from $[i_t : x_{i_t}]$ to $[i_{t+1} : x_{i_{t+1}}]$. By induction, for $t = 1, \ldots, k$, since the edge $[i_t : x_{i_t}]$ to $[i_{t+1} : x_{i_{t+1}}]$ is in the graph, we have that the edge $[i_{t+1} : x'_{i_{t+1}}]$ to $[i_t : x'_{i_t}]$ is also in the graph. This implies that there is a path this graph from $[i_{k+1} : x'_{i_{k+1}}]$ to $[i_1 : x'_{i_k}]$, which is what we had to prove. ■

(C) (5 PTS.) Prove that if $[i : x_i]$ and $[i : x'_i]$ are in the same strong connected component of G, then there is no legal way to place the brooms in the closet.

Solution:

We are assuming here that $x_i \neq x'_i$.
This is of course obvious, if we try to place the ith broom in the i slot. Now, the implication edge of the graph tell us where to place other brooms. Indeed, if there is an edge $[i : x_i]$ to $[j : x_j]$ then we must place the jth broom in the slot x_j, whether we like it or not (we don't). Thus, a path from $[i : x_i]$ to $[i : x'_i]$ implies that in such a situation we must place the ith broom in the slot x'_i, which is of course, a contradiction. Same argument works for the other direction.

(D) (5 PTS.) Assume that there is a legal solution, and consider a strong connected component X of G involving brooms, say, $b_1, \ldots, b_t$ in G; that is, X is a set of vertices of the form $[1 : x_1], \ldots, [t : x_t]$. Then, prove that $[1 : x'_1], \ldots, [t : x'_t]$ form their own connected component in G. Let this component be the ***mirror*** of X.

Solution:

Duh. Since X is a strong connected component of G then for any $[i : x_i]$ and $[j : x_j]$ in X there is a directed path from one to the other. Now, by (B), this implies that as $[i : x_i]$ forces $[j : x_j]$, then $[j : x'_j]$ forces $[i : x'_i]$; namely, there is a path from $[j : x'_j]$ forces $[i : x'_i]$. This implies that the set of vertices $X_M = \left\{ [i : x'_i] \;\middle|\; [i : x_i] \in X \right\}$ is strongly connected in G, since there is a directed path between any pair of them.

(E) (5 PTS.) Prove that if X is a strong connected component of G that is a sink in the meta graph $\mathsf{G}^{\mathrm{SCC}}$, then the mirror of X is a source in the meta graph $\mathsf{G}^{\mathrm{SCC}}$.

Solution:

By the above the mirror set X_M is strongly connected. Now, assume for the sake of contradiction, that there is an edge in the graph from $[1 : x'_1]$ to $[2 : x'_2]$, such that $[1 : x'_1]$ is not in X_M, but $[2 : x'_2]$ is in X_M (i.e., X_M is not a source vertex - it has incoming edges). Then, by (B), we have that $[2 : x_2] \in X$, and $[1 : x_1] \notin X$, and the edge from $[2 : x_2]$ to $[1 : x_1]$ is in the graph. But then, X can not be a sink in the strong connected component graph, since there is an edge leaving one of its vertices. A contradiction.

(F) (5 PTS.) Consider the algorithm that takes the sink X of the meta-graph $\mathsf{G}^{\mathrm{SCC}}$, use the associated slots as specified by the nodes in X, remove the vertices of X from G and the mirror of X from G, and repeating this process on the remaining graph. Prove that this algorithm generates a legal placement of the brooms in the closet (or otherwise outputs that no such placement exists). Also, describe how to implement this algorithm efficiently. What is the running time of your algorithm in the worst case as a function of n and m.

Solution:

Algorithm: We compute the graph G. That takes $O(n^2)$ time in the worst case. Next, we compute its strong connected components. If there is a broom that both vertices that correspond to it are in the same strong connected component, the algorithm stops and

output that there is no solution.
Next, we scan the meta graph, and create for every node in the meta-graph a list of all its incoming edges. For every node, we have a counter of how many edges are incoming into it, and how many edge leave it. This preprocessing takes linear time in the size of the meta-graph, which is $O(n^2)$ in the worst case. We also create a FIFO queue with all the strong connected components that have no outgoing edges (i.e., the sinks).
Next, as described, the algorithm takes the strong connected component that is a sink of the meta-graph, write down the assignments of brooms to slots for all the brooms in this component, remove X and X_M from the meta graph, and continues to the next iteration. When removing X and X_M, we scan all the relevant edges that are being deleted, and we decrease the counter of number of outgoing edges. For example, if there is an edge (S, X) in the meta-graph, then when we remove X, we decrease the counter of the number of edges outgoing from S. If the counter of S becomes zero, we add it to the queue of sinks. The algorithm continues in this fashion, repeatedly removing a sink, by extracting it from the queue. Clearly this algorithm has linear running time in the size of meta-graph, and number of vertices in the original graph. Which is $O(n^2)$.
Correctness: We are left with the correctness proof. Observe that when the algorithm removes X, it removes all the complementary assignments in X_M. Furthermore, since X is a sink and X_M is a source in the meta-graph, there is no path from a vertex of X to a vertex of X_M (in the original graph!). As such, there can never be a path in the graph for an assigned slot to its complement assignment. Clearly, by the end of this process all the brooms were assigned a slot (because we consumed the whole graph).
We need to prove that it is not possible that the algorithm assigned the same slot to two different brooms. In particular, assume for the sake of contradiction, and it did happen for brooms b_1 and b_2, for the slot $x_1 = x_2$. Assume that either broom b_1 was assigned the slot before b_2, or they were assigned the slot in the same time. But then $1x_1$ forces $2x'_2$, and if the two brooms were assigned in the same time then $2x_2$ and $2x'_2$ are both in X when it was removed, which is impossible. Otherwise, since X is a sink, it must be that we assigned b_2 the slot x'_2 (because it is forced), and then $[2 : x_2]$ was removed from the graph since it is in X_M, and as such this assignment did not happen. In either case, this is not possible. Namely, the generated assignment is valid.

2. (30 PTS.) Heavy time.
Consider a DAG G with n vertices and m edges.

(A) (5 PTS.) Assume that s is a sink in G. Describe how to compute in linear time a set of new edges such that s is the only sink in the resulting graph G (G has to be a DAG). How many edges does your algorithm add (the fewer, the better)?

Solution:

Scan the edges, and count for every vertex the number of edges incoming into it. The vertices with zero incoming edges are the sinks. Put an edge from each sink to s. Clearly, if there are k sinks, then this introduces $k - 1$ edges, and the running time is $O(n + m)$. The new graph is still a DAG and only s has no outgoing edges; that is, it is the only sink. Clearly, any sink except s needs an edge coming out of it to "kill" its sinkiness. Thus,

$k-1$ is the minimum number of edges required to do so.

(B) (10 PTS.) Assume G has a sink vertex s. Some of the vertices of G are marked as being ***significant***. Show an algorithm that in linear time computes all the vertices that can reach s via a path that goes through at least t significant vertices, where t is a prespecified parameter. (Hint: Solve the problem first for $t = 1$ and then generalize.)

Solution:

Add edges to G such that s is the only sink, using (A). This takes $O(n+m)$ time.
Next, topologically sort the vertices of G. Let $v_1, \ldots, v_n$ be the resulting ordering. Observe that $v_n = s$, and that all the edges in G are of the form (v_i, v_j) where $i < j$.
Assume that we had computed for all the nodes v_{i+1} to v_n the paths with the largest number of significant nodes on them. Formally, for each such node v_τ, we will have a count $C(v_\tau)$ of the number of significant nodes in the best path starting with v_τ, and the edge to the next vertex in this best path starting at v_τ. Computing $C(v_n)$ is obvious – the next pointer is null, and $C(v_n)$ is one if v_n is significant, and zero otherwise. To compute this quantity for v_i, we compute

$$x_i = \max_{(v_i, v_j) \in E} C(v_j),$$

where E is the set of edges of G. we set $C(v_i)$ to x_i if v_i is not significant, and to $x_i + 1$ if it significant. In either case, we set the next edge point to the edge realizing the max. We decrease i, and repeat this process till we are down with all vertices in the graph.
Since every edge get considered only once in this process, the overall running time of this algorithm is $O(n+m)$.
It is now straightforward to prove by induction that $C(v_j)$, for all j, is the maximum number of significant vertices that might be visited by a path starting at v_j and ending at s. We can now scan the quantities $C(v_j)$, and output all the vertices for which $C(v_j)$ is $\geq t$.

(C) (10 PTS.) Assume the edges of G have weights assigned to them. Show an algorithm, as fast as possible, that computes for all the vertices v in G the weight of the ***heaviest*** path from v to s.

Solution:

We use the same algorithm as above. Except that now every vertex v_τ would have weight $W(v_\tau)$ which is the maximum weight path from v_τ to s. The formula for computing v_i, after we computed $v_{i+1}, \ldots, v_n$ is

$$W(v_i) = \max_{(v_i, v_j) \in E} \left(w(v_i, v_j) + W(v_j) \right),$$

where $w(v_i, v_j)$ is the weight of the edge (v_i, v_j). Again, an easy induction proves that this algorithm is correct.
The running time is $O(n+m)$, as usual.

(D) (5 PTS.) Using the above, describe how to compute, in linear time, a path that visits all the vertices of G if such a path exists.

Solution:

Mark all the vertices of the graph as significant, and use (B) with $t = n$. If there is a vertex that has a path visiting n significant vertices, that it induces the desired path, and we can extract the path by following the next pointers starting from this vertex.
The running time is $O(n + m)$, as usual.

3. (40 PTS.) Wishful graph.
Let $\mathsf{G} = (V, E)$ be a directed graph. Define a relation R on the nodes V as follows: uRv iff u can reach v or v can reach u.

(A) (10 PTS.) Is R an equivalence relation? If yes, give a proof, otherwise give an example to show it is false.

Solution:

No. It is not an equivalence relationship. Consider the graph over $\{1, 2, 3\}$ that has the edges $(1, 2), (3, 2)$. Clearly, we have that $1R2$ and $3R2$, but $1R3$ does not hold.

(B) (30 PTS.) Call G ***uselessly-connected*** if for every pair of nodes $u, v \in V$, we have that there is either a path from u to v in G, or a path from v to u in G. Give a linear time algorithm to determine if G is uselessly-connected, here linear time is $O(m + n)$, where $m = |E|$ and $n = |V|$.

Solution:

Algorithm: Compute the meta-graph M of strong connected components of G. Compute the topological sorting of the DAG M. Let $S_1, \ldots, S_k$ be the resulting topological ordering of the connected components. We verify that for all i there is an edge in M from S_i to S_{i+1}. If this is not true, we output that G is not uselessly connected, otherwise, it is uselessly connected, and we output this fact.

Running time: Computing the strong components takes $O(m + n)$ time. Topological sorting of M takes $O(|V(M)| + |E(M)|)$ time, which is $O(n+m)$. Verifying that ever two consecutive vertices in the topological ordering of the connected components is connected can be done in linear time by scanning the edges, from the last vertex to the first vertex in this ordering (see previous problem). So overall this is $O(n + m)$ time.
Correctness:

Lemma 0.1. *A DAG M has a unique topological ordering, if and only if, in any topological ordering of M, any two consecutive vertices are connected by an edge (oriented in the right direction, of course).*

Proof: One direction is easy – if there is an ordering where every two consecutive vertices are connected, then the ordering is uniquely defined, and there is only one topological ordering.

As for the other direction, consider an ordering where there is an edge missing between v_i and v_{i+1}, where $v_1, \ldots, v_n$ is the topological ordering. But then, clearly, both topological orderings are valid:
(A) $v_1, \ldots, v_{i-1}, v_i, v_{i+1}, v_{i+2}, \ldots, v_n$.
(B) $v_1, \ldots, v_{i-1}, v_{i+1}, v_i, v_{i+2}, \ldots, v_n$.
Namely, the topological ordering of M is not unique. ■

Lemma 0.2. *In a DAG M, if there are two vertices X and Y such that there is no path between X and Y, and no path between Y and X then the topological ordering of M is not unique.*

Proof: Consider the two DAGs $M_{(X,Y)} = M \cup \{(X,Y)\}$ and $M_{(Y,X)} = M \cup \{(Y,X)\}$. Clearly, they are both DAGs, but they define two different topological orders on their vertices. In one of them X must appear before Y, and in the other one Y must appear before X. In any case, any valid topological ordering for $M_{(X,Y)}$ and $M_{(Y,X)}$ is a valid topological ordering for M. We conclude that M has at least two different topological orderings. ■

Lemma 0.3. *The graph G is uselessly-connected if and only if in the topological ordering of the SCC, there is an edge between any two consecutive components.*

Proof: If G is not uselessly connected, then there are two vertices x and y in G such that there is no path from x to y, and from y to x. Namely, x and y are in two different strong connected components of G, and these two components are not connected in M. That is, the strong connected components X and Y, containing x and y, respectively, are incomparable. Namely, by Lemma 0.2 there are at least two different topological ordering of M, and by Lemma 0.1 there is a missing consecutive edge in the topological ordering of M.
If G is uselessly connected, then there is an edge between any two strong connected components of M, and as such the topological ordering of M is unique, and there is an edge also between any two consecutive connected components in the topological ordering of M. ■
Clearly, Lemma 0.3 implies the correctness of the algorithm.