Assignment 3

Sample Solutions

CSCI 3110 — Summer 2018

1 The Algorithm

We represent every person P_i by two vertices b_i and d_i representing their times of birth and death, respectively. Since it is better for P_i to be born before he or she dies, we also require that $b_i < d_i$; we represent this requirement by a directed edge from b_i to d_i.

The two types of constraints in the question can be represented using additional edges as follows:

- If P_i died before P_j was born, we must have $d_i < b_j$; we represent this by adding an edge from d_i to b_j to the graph.
- If the life spans of P_i and P_j overlapped, it must be true that P_i was born before P_j died and vice versa. We represent this by adding two directed edges to G, one from b_i to d_j and one from b_j to d_i.

The question we are now asking is whether there is some way to arrange the events of birth and death represented by the vertices in the graph in some order so that all the constraints represented by the edges are satisfied. But this is simply the problem of topological ordering, which we can solve in time linear in the size of the graph.

If the algorithm succeeds in finding a topological ordering, we can now inspect the nodes in topological order and assign increasing dates to them, so all constraints in the input are satisfied by these dates.

If the algorithm fails to find a topological ordering, we need to find a directed cycle that shows that the facts are not internally consistent. The DFS-based topological sorting algorithm is easily augmented to report such a cycle: The algorithm computes a postorder numbering of the vertices and arranges them in reverse postorder. If there exists an edge (u, v) such that $u < v$ in postorder (that is, u comes after v in the ordering the algorithm outputs), then observe that (u, v) cannot be a backward cross edge, a tree edge or a forward edge because all these edges satisfy $u > v$, as discussed in class. Since a DFS-forest does not contain any forward cross edges, (u, v) is thus a back edge, that is, v is an ancestor of u in the DFS forest and the path from v to u in the forest plus the edge (u, v) form a cycle. We find this cycle by following the path from u to the root of its tree and collecting the visited vertices until we reach v.

2 Correctness

The correctness of the algorithm is fairly obvious. If we successfully compute a topological ordering of the vertices of the graph and then assign increasing dates to the different vertices in topologically sorted order, then

- Every person P_i is born before he or she dies (because of the edge $b_i d_i$),
- If one of our facts said that P_i died before P_j was born, then our dates confirm this because of the edge $d_i b_j$, and
- If one of our facts said that the life spans of P_i and P_j overlapped, then $b_i < d_j$ and $b_j < d_i$ because of the edges $b_i d_j$ and $b_j d_i$, which implies that the dates again confirm this fact.

Thus, the output of our algorithm is a set of dates that confirms that all the constraints imposed by the facts can be satisfied simultaneously, that is, the given set of facts is internally consistent.

Now assume that there exists an assignment of dates that satisfies all the facts we have learned. Then we claim that sorting the vertices by their associated dates, breaking ties arbitrarily, produces a topological ordering of the graph. This is true because:

- For every person P_i, $b_i < d_i$ and, hence, b_i precedes d_i in the vertex ordering,
- For every person P_i that is known to have died before P_j was born, we have $d_i < b_j$, that is, the constraint of the edge $d_i b_j$ is satisfied, and
- For every two persons P_i and P_j whose life spans have overlapped, we have $b_i < d_j$ and $b_j < d_i$, that is, the constraints of the edges $b_i d_j$ and $b_j d_i$ are satisfied.

Since these are all the edges in the graph, the ordering is topological ordering. Since the graph dose not have a topological ordering if our algorithm reports a directed cycle in the graph, this shows that the set of facts we have learned are not internally consistent in this case.

3 Running Time

Once we have constructed the graph, the running time of the algorithm is easily seen to be $O(n+m)$. Indeed, if n' and m' denote the number of vertices and edges of the graph, then topological sorting and testing for cycles takes $O(n'+m')$ time. Since $n' = 2n$ and $m' \leq n + 2m$, the claim follows.

For the construction of the graph, the running time is a little harder to determine because it partially depends on the representation of the input. We assume here that every person is represented by a unique ID (a name) and that every fact is represented by a value that tells us the type of the fact and the names of the two people this fact concerns. Then we need a way to translate the facts into graph

edges and, more importantly, recognize when two facts talk about the same person and, thus, should be translated into edges incident to the same vertices.

The easiest way to do this translation is to scan the set of facts and maintain a dictionary of people we already know from previous facts. For every person P_i, the dictionary entry also stores pointers to the corresponding graph nodes b_i and d_i that have been added to G.

Now, when we encounter a fact involving persons P_i and P_j, we first perform a lookup on the dictionary to see whether we have already created vertices b_i, d_i, b_j, and d_j and retrieve these vertices if we have. If we do not find P_i (or P_j) in the dictionary, we add two vertices b_i and d_i (or b_j and d_j) to the graph and insert the record (P_i, b_i, d_i) into the dictionary. Once we have vertices b_i, d_i, b_j, and d_j in our hands, adding the correct edges to G is a constant-time operation per edge.

The number of dictionary operations we perform is $O(m)$, $O(1)$ per fact. The cost of this depends on the dictionary. If we use a deterministic comparison-based dictionary such as a red-black tree, the cost per operation is $O(\log n)$ (because the dictionary never stores more than n entries, one per person). Thus, the total cost of constructing the graph is $O(m \log n)$. If we use a hash table to represent the dictionary, every operation has expected constant cost, and the graph construction has expected cost $O(m)$ (but $O(mn)$ cost in the worst case).