CS 6901 (Applied Algorithms) – Lecture 6

Antonina Kolokolova

September 25, 2014

In this lecture, we will first finish our discussion of the Strongly Connected Components algorithm by talking about its running time and giving an outline of the correctness proof.

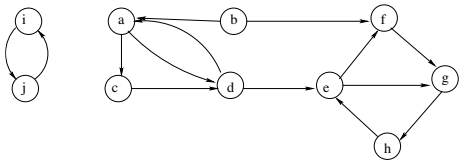
0.1 Strongly Connected Components

Recall from the last class the algorithm for computing strongly connected components and our example:

The following algorithm computes strongly connected components of a graph G.

SCC(G)
1 Run DFS on G
2 Compute G^r ▷ where G^r is G with all edge directions reversed
3 Run DFS on G^r, considering vertices in the order of decreasing finishing time computed by DFS(G)
4 **return** DFS forest from the second run; each tree is a separate strongly connected component.

Example 1 *Suppose we obtained the following start/finish times by running the above algorithm:* $a : (3, 4)$, $b : (19, 20)$, $c : (1, 14)$, $d : (2, 13)$, $e : (5, 12)$, $f : (6, 11)$, $g : (7, 10)$, $h : (8, 9)$, $i : (15, 18)$, $j : (16, 17)$ *from the first DFS run. Now, running DFS on the reverse graph* G^r*, we first start with* b*. As all edges of* b *are outgoing in* G*, all edges are incoming in* G^r*, and so there is nothing else in its connected component. Then run DFS starting with* i*, obtaining component* $\{i, j\}$*. Then proceed to* c*; here,* a*,* b *and* d *are accessible, since edge* (e, d) *is incoming to this component. As* b *has been processed already, it does not become part of this tree, leaving only* $\{c, a, d\}$ *as elements of this component. The final tree is rooted at* e*, and consists of* $\{e, h, g, f\}$.



First, let us look at the running time of this algorithm. We know that $DFS \in O(n+m)$, and G^r has the same number of vertices and edges as G, so two subsequent runs of DFS will still take time $O(n + m)$ (remember that for a constant number of blocks executed one after another we take the maximum, or, alternatively, that doubling the running time does not change the corresponding O-class). Now, to compute G^r we need to go through G and create a new adjacency list for each vertex, now containing what was formerly incoming edges to this vertex. But this can be done in time $O(n + m)$ as well, by going through all lists of G in order and inserting the corresponding entries into the lists of G^r. Therefore, the total time is $O(n + m)$.

The key fact we need to for the correctness proof is that if u and v are in different connected components C and D, and there is an edge (u, v), then there will be a vertex $w \in C$ with a large finishing time than any

vertex $v' \in D$, where the finishing time is with respect to running DFS on G. To see this, take a vertex in w with the largest finishing time of all vertices in C. Then, all vertices in C are in a DFS tree starting from w. Since there is an edge (u, v), DFS will try to follow this edge.

Now, if v is white at that time, then DFS algorithm will explore all of D before backtracking (as D is strongly connected, and there is no path in G from D to C). Then all vertices in D will be descendants of w, so for any $v' \in D$, $w.d < v'.d < v'.f < w.f$. Note that v cannot be gray at that time, as it would imply that it is already on a path from w to v, but such a path would have to leave C, go to D, and come back to C, contradicting the fact that C and D are separate strongly connected components. Thus, if v is not white it would have to be black; in that case, its finishing time for every v' in D was already computed (as they are parts of the same DFS tree), whereas $w.f$ is not assigned yet, so $w.f > v'.f$.

Finally, we need to show that $SCC(G)$ correctly computes strongly connected components. That is, all vertices in the same connected component become parts of the same $DFS(G^r)$ tree (where vertices are considered in $DFS(G^r)$ in order of decreasing finished time from $DFS(G)$), and any two vertices from different components will be in different $DFS(G^r)$ trees. To see the first fact, note that there is a path from any vertex to any other vertex within a connected component, so once DFS picks a vertex in a component it explores all of it before stopping. This was true even for a single run of $DFS(G)$, so a more interesting part is to show that in the trees of $DFS(G^r)$ each component forms a separate tree.

For the intuition, imagine running DFS on a DAG considering vertices in reverse topological order. The last vertex in this order is a sink: so DFS does not have anywhere to go, and this vertex forms a separate tree. The second to last vertex is either a sink itself (so the same reasoning applies), or has an edge to the last vertex, but since that one is already marked black, the second to last vertex becomes its own tree as well. Now, it can be shown to be true for all vertices of the DAG by induction. Assume that the last i vertices in the topological order (that is, the first i vertices DFS visits) each form a separate tree. Now consider vertex $i+1$ from the end. all of its outgoing edges go to vertices in the tree already processed, so all of them go to black vertices. Therefore, there is nothing to explore and the vertex $i+1$ forms the next DFS tree.

Now, all that is left is to generalize the argument before slightly, to talk about strongly connected components rather than vertices in a DAG. Let us argue by induction again. The vertex with the largest finishing time is in a component C_1 with no edges to other components in G^r, by the fact we discussed at the start of the proof. Thus, the first tree will contain all vertices in that strongly connected component, but no other; and all its vertices will be marked black.

Now, suppose the first i trees outputted by the algorithm form separate strongly connected components $C_1 \dots C_i$. Consider the time when $DFS(G^r)$ starts exploring component C_{i+1}. At that time, all edges from vertices $v \in C_{i+1}$ in G^r are going to vertices $u \in C_1, \dots C_i$, which are already marked black (which follows from the same key fact). Therefore, only vertices in C_{i+1} will become part of the $i+1$st $DFS(G^r)$ tree, completing the proof.