Strongly Connected Components (SCC's)

(CLRS 22.5)

Basics

- $\boxed{\text{A strongly connected component (SCC) of a directed graph } G = (V, E) \text{ is a maximal set of vertices such that any two vertices in the set are mutually reachable.}}$
- Example: All vertices along a directed cycle are in the same SCC.
- Intuitively, we think of a SCC as a cycle.
- Given a directed graph, some questions that one may be interested in are:

  (1) Given a vertex v, compute it's strongly connected component (SCC).

  (2) Is G strongly connected (SC)?

  (3) Compute the SCCs of G.

Practice:

- Draw a graph with 3 vertices and (a) 1SCC; (b) 2 SCCs; (c) 3 SCCs.
- What happens to SCC's as you add an edge to the graph?
- Show an example when adding an edge does not change the SCC's, and show one where it does.

1 Computing the SCC of a given vertex v

All vertices in the same SCC as v must be able to reach v and v must be able to reach them. This suggests the following approach, which runs in linear time $O(V + E)$:

- Perform DFS(v) (or BFS(v)) to find all vertices reachable from v.
- Perform DFS(v) using the *incoming* edges to find the vertices that reach v (a different way to say this is that we perform DFS in the transpose graph G^T).
- Find the vertices that are reached by both DFS's. These are the vertices that are in the same SCC. Why? Let u, u' be two vertices that are both reachable from v and that reach v. Then it follows that $u \rightsquigarrow v$ and $v \rightsquigarrow u'$ and thus $u \rightsquigarrow u'$. Similarly, $u' \rightsquigarrow v$ and $v \rightsquigarrow u$ and thus $u' \rightsquigarrow u$.

2 Finding whether a graph G is strongly connected

For a graph to be SC, every vertex needs to be able to reach every other vertex. Take an arbitrary vertex v and run DFS(v). If this does not reach all vertices, G is not SC. If DFS(v) reaches all vertices, we continue with DFS(v) using the *incoming* edges (in other words, DFS(v) on G^T): if this does not reach all vertices, then clearly G is not SC. Otherwise, we know that v reaches all vertices, and all vertices reach v. Does that mean that G is strongly connected? Yes. We have the following approach which runs in linear time $O(V + E)$:

1. Pick an arbitrary vertex v and perform DFS(v).
2. Perform DFS(v) in the transpose graph G^T.
3. If both searches reach *all* vertices, G is SC. Otherwise it's not.

Proof: Let u, u' arbitrary vertices in G. We know that $u \rightsquigarrow v$ (because of (2)) and $v \rightsquigarrow u'$ (because of (1)) and thus $u \rightsquigarrow u'$. Similarly $u' \rightsquigarrow u$. Thus any u, u' are mutually reachable.

3 Find the SCCs in G

- **Naive:** Ok, so we want to compute the SCC's in G. We always start a problem by coming up with the most straightfirward solution we can possibly think of. For e.g. we can start with an arbitrary vertex v and compute its strongly connected component; then we delete v and all vertices in its SCC from the graph, and repeat. This finds the components one at a time; it can be shown this runs in the worst case $|V|$ searches, for a total of $O(V(V + E)$. Can you come up with a graph where this approach will take $\Theta(V + E)$?
- We ask the usual question: can we do better?
- Yes! There is a very clever algorithm by Kosaraju-Sharir that finds the SCCs in linear time! It's very short, but it's quite hard to understand why it works.
- It's key ingredient, and a concept that's useful when thinking of SCC is the *component graph*: this is a graph with one vertex per strongly connected components, and with an edge $C \to C'$ if there exists an edge (u, v) in G with u in C and v in C'.
- Exercise: There's one important property that the component graph has: What is it?

  Answer: The component graph cannot have cycles.

  Proof: Consider two SCC's, C and C', and assume there is a cycle between them: this means than an arbitrary vertex in C can reach an arbitrary vertex in C', and the otehr way around, via this cycle. This means that C, C' would be a single strongly component, which violates our assumption that they are separate, maximal SCC's.
- Exercise: If we perform DFS on a graph with several SCC's, is it possible that one SCC be broken across two difefrent DFS trees?

  Answer: No. Once DFS reaches a vertex of a SCC, it will reach the entire component, eventually. That is to say, a DFS tree will visit the whole SCC, or not visit it at all.

- Exercise: Is it possible that a DFS tree contains more than one SCC?

  Answer: Yes.

- Kosaraju-Sharir idea is to call DFS on the vertices in G in such an order so that a DFS tree identifies precisely *one* SCC.

- Exercise: Can you think of a vertex v to call DFS(v) so that all verties reached are in *only one* SCC?

  Answer: v can be any of the vertices in the "last" SCC of G in topological order.

- So we could do this:

  1. If we somehow knew the component graph, and if we could find reverse topological order of vertices in the component graph, then
  2. For each vertex v in reverse topologial order, call DFS(v). The claim is that everything reached by this DFS is the SCC of v, nothing more, nothing less.

  This looks good, the problem is that to know the component graph, we would need to know the SCC's, and that's precisely what we are trying to compute.

- As it turns out, there is no good way to identify the components in reverse topological order. But it is possible to identify them in topological order.

- Remember that tpological order on a DAG is given by reverse order of finish time.

- Let's say we perform $DFS(G)$, and let v be the vertex with the largest finish time. G is not a DAG, so we think of its component graph.

- By now we have the intuition. Traversing the vertices in reverse finish time gives us topological order in the component graph. We formalize with the following claim:

- Notation: For a SCC C, denote $f(C)$ to be the largest finish time of a vertex in C. Denote $d(C)$ to be the smallest start/discover time of a vertex in C.

- Claim: Let C, C' be two SCC's in G. If there is an edge (u, v) from a vertex $u \in C$ to a vertex $v \in C'$, then C must be finished *after* C': $f(C) > f(C')$.

  Proof:

  - Case 1: Component C is discovered before C' by DFS(G), $d(C) < d(C')$: Let x be the first vertex in C discovered by DFS. From x, DFS will reach all vertices in C, and, through edge (u, v), it will reach component C'. It's not possible to have an edge from C' back to C (it would cause a cycle), and so the entire C' is finished before vertex u, and thus before x. Thus we have $f(x) > f(C')$ and thus $f(C) > f(C')$.
  - Case 2: Component C' is discovered first: $d(C') < d(C)$: Let y be the first vertex discovered in C'. From y DFS will reach all vertices in C'. There cannot be an edge from a vertex in C' to a vertex in C (because it would cause a cycle), hence no vertex in C is reachable from y. This means that when DFS finishes, all vertices in C are still white. This means C is discovered and finished after $f(C')$.

- Ok, so we perform $DFS(G)$ and now let v be teh vertex with the largest finish time. What can you say about it? What can v reach, and what can reach v?
- We are going to perform DFS on v but considering the *incoming* edges, not the outgoing edges.
- This gives us the following algorithm:

  SCC Algorithm (Kosaraju-Sharir)

  1. Perform DFS(G) to compute finish times for all vertices.
  2. For each vertex u in G in reverse order of finish time: run $DFS(u)$ on *incoming edges* and label all vertices in its DFS tree as a separate SCC.

- Instead of performing the DFS on teh incoming edges, we could use the concept of *transposed graph*: Let G^T be the reversed, or transpose graph, where the direction of each edge is reversed. We can think of G^T as storing all teh edges that are incoming to vertices.
- Exercise: How would you build G^T given G and how fast?
- Exercise: What is reached by $DFS(v)$ in G^T? What is reached by $DFS(v)$ in G^T?

  Answer: All vertices reachable from v. All vertices that reach v.
- Exercise: What can you say about the SCC's of G^T?

  Answer: They are the same as in G.
- This gives us the following algorithm:

  SCC Algorithm (Kosaraju-Sharir)

  1. Perform DFS(G) to compute finish times for all vertices.
  2. Compute G^T.
  3. For each vertex u in reverse order of finish time: run $DFS(u)$ in G^T and label all vertices in its DFS tree as a separate SCC.