Strongly Connected Components & Biconnected Components

Charlie Gunn

28 September 2017

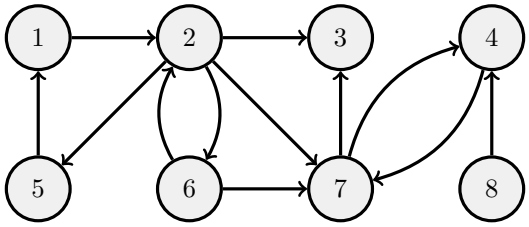

Figure 1: A directed graph. *Credit: Crash Course Coding Companion.*

1 Strongly Connected Components

A strongly connected component (SCC) is a set S of vertices such that for every pair of vertices u, $v \in S$, there exists a path from u to v. In an undirected graph the set of SCCs is equivalent to the set of connected components, and therefore can be found using a simple DFS. This works because in an undirected graph, u can reach v iff v can reach u. In directed graph, this statement is no longer true, and therefore finding the SCCs becomes harder.

2 Kernel graphs

What if we replaced each strongly connected component in a graph with a single node? This would give us what we call the *kernel graph*, which describes the edges between SCCs. Note that the kernel graph must be a directed acyclic graph (DAG), because any cycles would constitute an SCC, and therefore would be compressed into a single kernel.

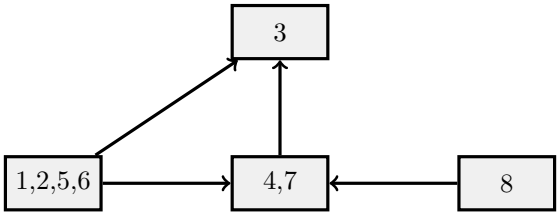

Figure 2: Kernel graph of the above directed graph.

3 Tarjan's SCCs Algorithm

In a DFS traversal of the graph, every SCC will be a subtree of the DFS forest (this subtree is not necessarily proper, i.e., it may be missing some branches which are in separate SCCs). The roots of these subtrees are called the "roots" of the SCC. Any node of an SCC could be the root, if it happens to be the first node of the SCC visited by the traversal.

If a node v can reach another node u which is an ancestor of v in the DFS, then v cannot be the root of an SCC. This is because there is a cycle between u and v, and therefore u and v must be in the same SCC. u, or some ancestor or u, is the root of the SCC. This observation leads us to the conclusion that a node is the root of an SCC iff it cannot reach any of its ancestors.

To keep track of pending nodes, we will use a stack (note, this stack is distinct from the recursive stack which stores method calls and is responsible for recursion). When a node v is visited for the first time, it is added onto the stack. After the recursive call that visits all of v's descendents, we know whether v has a path to any node earlier on the stack. If so, the call returns, leaving v on the stack. Otherwise, v must be the root of an SCC, which consists of v together all nodes above v on the stack (such nodes all have paths back to v but not to any earlier node; if they had paths to earlier nodes, then v would also have paths to earlier nodes, which we know is false). The SCC rooted at v is then popped from the stack and returned.

To aid us, we will need to do some bookkeeping: we give the nodes an ID by the order we visit them in the DFS. Every time we visit a new node we will give it the next available ID. This allows us to further define v.lowlink as the lowest node ID on the stack that is reachable from v. v can reach itself, so v.lowlink is initiated to v.ID. If, after recurring on all of v children, v.lowlink still equals v.ID then v is the root of its SCC.

Algorithm 1 Tarjan's SCC Algorithm

```
function VISIT(Vertex v)
    v.ID := nextID
    nextID++
    v.lowlink := v.ID
    push v onto the stack
    v.onStack := true
    for each edge (v,u) do
        if u has not been visited then
            Visit(u)
            v.lowlink := min(v.lowlink, u.lowlink)
        else if u.onStack then
            v.lowlink := min(v.lowlink, u.lowlink)          ▷ More on this line later
    if v.lowlink = v.ID then
        while v.onStack do
            pop u off the stack
            add u to the SCC
            u.onStack = false
```

Because a DFS starting from one node may not reach every node, we must call Visit() on each node that has not yet been visited. The overall algorithm takes $O(|V| + |E|)$ time.

Note: Tarjan's original algorithm had the noted line as v.lowlink = min(v.lowlink, u.ID). This will also work. Many online sources use this version to be true to the original. However, using u.lowlink instead of u.ID produces a nice property: v and u are in the same SCC iff v.lowlink = u.lowlink. Because this can be useful, the u.lowlink version is often preferable.

4 Biconnected Components

In an undirected graph, biconnected components are sets of nodes where each one is reachable from every other, even if any one node is removed. Note that one node can be in multiple BCCs, up to its degree. Take, for example, A–B–C: B is in a biconnected component with A and in one with C. However, every edge is in exactly one BCC. We can define BCCs by the edges they include, or by articulation points. Articulation points are nodes that divide BCCs, and belong to both (such as B in the example above). An articulation point is also any node that, when removed, divides the graph into disconnected components.

We will use a similar algorithm with lowlinks to find the articulation points, where instead of assigning unique node IDs, we keep track of node depth. In this version we need to use Tarjan's original version of the algorithm, where v.lowlink = min(v.lowlink, u.depth) at the line noted. A node is an articulation point if it has a subtree which cannot reach up the DFS tree past the node. More formally, a node v is an articulation point if v has a child u with u.lowlink $\geq v$.depth. The only special case we have to deal with is the root of the DFS tree: if the root has more than one child, then it is an articulation point. This is because the root will only have multiple subtrees if those subtrees are disconnected, making the root a cut vertex.

5 Block-cut Trees

Similarly to how SCCs compress to form a kernel graph, when we compress each BCC to a node, and add additional nodes for shared articulation points, we get the block-cut tree of the graph. This is an undirected graph (which cannot have cycles because any cycle forms a BCC).

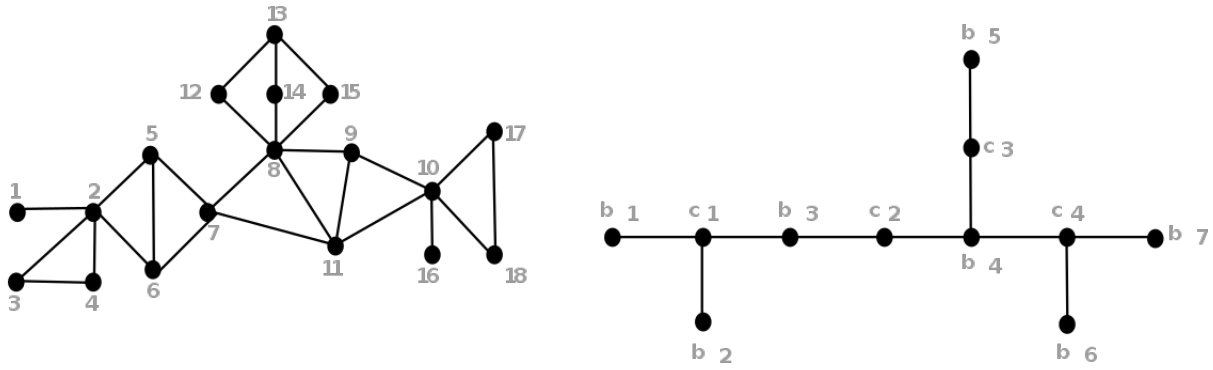

Figure 3: Undirected graph & corresponding block-cut tree

6 Problems

Mining Your Own Business (ICPC World Finals, 2011)

John Digger is the owner of a large illudium phosdex mine. The mine is made up of a series of tunnels that meet at various large junctions. Unlike some owners, Digger actually cares about the welfare of his workers and has a concern about the layout of the mine. Specifically, he worries that there may a junction which, in case of collapse, will cut off workers in one section of the mine from other workers (illudium phosdex, as you know, is highly unstable). To counter this, he wants to install special escape shafts from the junctions to the surface. He could install one escape shaft at each junction, but Digger doesn't care about his workers that much. Instead, he wants to install the minimum number of escape shafts so that if any of the junctions collapses, all the workers who survive the junction collapse will have a path to the surface. Write a program to calculate the minimum number of escape shafts needed.

Mowing the Field (USACO January 2016, Platinum)

In an effort to better manage the grazing patterns of his cows, Farmer John has installed one-way cow paths all over his farm. The farm consists of N fields ($1 \leq N \leq 100000$), conveniently numbered $1..N$, with each one-way cow path connecting a pair of fields. For example, if a path connects from field X to field Y, then cows are allowed to travel from X to Y but not from Y to X.

Bessie wonders how much grass she will be able to eat if she breaks the rules and follows up to one path in the wrong direction. Please compute the maximum number of distinct fields she can visit along a route starting and ending at field 1, where she can follow up to one path along the route in the wrong direction. Bessie can only travel backwards at most once in her journey. In particular, she cannot even take the same path backwards twice.

Push A Box (USACO December 2017, Platinum)

The barn can be modeled as an NxM rectangular grid. Some of the grid cells have hay in them. Bessie occupies one cell in this grid, and a large wooden box occupies another cell. Bessie and the box are not able to fit in the same cell at the same time, and neither can fit into a cell containing hay.

Bessie can move in the 4 orthogonal directions (north, east, south, west) as long as she does not walk into hay. If she attempts to walk to the space with the box, then the box will be pushed one space in that direction, as long as there is an empty cell on the other side. If there is no empty cell, then Bessie will not be able to make that move.

A certain grid cell is designated as the goal. Bessie's goal is to get the box into that location.

Given the layout of the barn, including the starting positions of the box and the cow, and the target position of the box, determine if it possible to win the game.