Strongly Connected Components

Def. We say that vertices u and v are in the same strongly connected component of a graph if there exists a path from u to v and from v to u.

Notice that in an undirected graph, connectedness implies strongly connectedness, as every edge is bidirectional. Therefore we are going to only consider directed graphs.

Finding the strongly connected components of a graph allows us to partition the graph into strongly connected subgraphs within which one can get from each vertex to every other vertex.

Kosaraju's algorithm

Kosaraju's algorithm is what we can use to find the strongly connected components of the graph in $O(E + V)$ time. The algorithm relies on the fact that the strongly connected components of the graph are the same as the strongly connected components of the transpose graph (same graph with the direction of all the edges reversed).

Kosaraju's algorithm is essentially two DFS, one on the graph and another on the reverse graph. The first DFS on the original graph is a reverse topological sort(ish), we sort the list of vertices in such a way that u comes before v if there exists and edge from v to u. I say this is approximately a topological sort because technically one cannot topologically sort a graph if it contains a cycle, but in our case, if it contains a cycle, we ignore it, and keep going. Note that this will sometimes create forests in connected graphs as a result, but that's fine.

Now we have this list, note that the first element of the list is a node in the original graph that is either in a cycle or has no incoming edges. Now we dfs from this node on the **transpose** graph, and every element it can get to is guaranteed to be in the same strongly connected component as it, this is because the SCC of the transpose graph is the same as that of the original graph. After we dfs from that point, we move onto the next un-assigned element in the sorted list and repeat the process, thereby generating a group of strongly connected subgraphs.

Boolean Satisfiability Problems

There is a special type of problems called boolean satisfiability problems that can be solved with Kosaraju's algorithm. The problem is relatively simple, given a bunch of boolean variables $x_1, x_2, \ldots, x_n$ and a boolean expression expressed as the conjunction of several pairwise disjunctions using these variables. Our task is to determine if there exists a solution.

To solve this problem we introduce something called the **implication graph**. The implication graph is a graph where there exists an edge between two nodes if there exists a logical implication that if one node is true so is the next. We construct the implication graph as

follows: if the entire boolean expression is true, then each of the pairwise disjunctions are true (because they are all connected via conjunctions). Therefore suppose we have $(x_i \vee x_j)$ as a part of the conjunction, then we can construct two edges: $\neg x_i \to x_j$ and $\neg x_j \to x_i$.

Once we construct the implication graph, we know that the equation is **NOT** satisfiable if x_i and $\neg x_i$ are within the strongly connected component. As that would mean x_i and $\neg x_i$ must both be true.

Otherwise we can pick an arbitrary starting point, assign it `true`, its opposite `false` and floodfill those truth values to its strongly connected component.