Lecture 18: DFS and MST

We continue with the strongly connected component problem. I claim that the node with the largest finish time will be in the source component in the component graph. To see this, consider two components \((C_1, C_2)\) (i.e. there is an edge from \(C_1\) to \(C_2\) in component graph). Note we can not go from a node in \(C_2\) and reach nodes in \(C_1\) (component graph \(G_x\) has no cycles). In class, we verified that either \(C_1\) or \(C_2\) is visited first, we all get contradictions to the assumption that a node of \(C_1\) has the largest finish time. You should verify this yourself and this helps you to gain more understanding of the directed graph.

But this is not what we want: we want a node in the sink not the source component. The trick is to create the complement of \(G\) (called \(G'\)) by reversing the edges of \(G\). \(G'\) has same components as \(G\), but the component graph is also reversed. Now this gives the algorithm: (1) first do DFS; (2) Construct \(G'\) (the complement graph); (3) do DFS on \(G'\) by visiting the nodes according to decreasing order of finish time of the first DFS; (4) Output nodes found inside one DFS-visit (i.e. the DFS trees) as strongly connected components. The key to make this work is that in the complemented graph, the second DFS from the node with highest (original) DFS finish time will not visit other components; this also holds when we visit the other nodes in the decreasingly order because the components that are reachable from a node has already been visited by the second DFS and we will not revisit them. Read the textbook again to ensure you understand what I am writing here.

Now minimum spanning tree (ch. 23). For a connected (undirected) graph \(G\), find a tree that connects all the nodes such that the total edge costs are the smallest. The well-known Kruskal’s algorithm is a greedy algorithm: first sort the edges by edge weights; then for each edge, add edge if not cycles is created by this edge. We discuss in class to claim this algorithm will always lead to a spanning tree (see your notes/textbook).

Question: is this optimal? Yes, and we ow show each added edge is safe to add: safe means we are not making mistakes at that step (the edges we picked so far belong to some MST) at each step.

We first assume all edge costs are distinct. A fundamental property of graph cut: let \(S, V - S\) be partitions the nodes, and edge \(e = (v, w)\) is the cheapest edge across \(S\) and \(V - S\) (say \(v \in S\)). Claim: every MST contains \(e\). Proof: consider spanning tree \(T\) does not contain \(e\). Will prove \(T\) is not MST, by the exchange argument (used several times before fro greedy algorithm): will find an edge \(e'\) of \(T\) that is most costly than \(e\) and we will replace \(e'\) with \(e\) to get a new cheaper \(T'\). Now, since \(T\) does not have \(e = (v, w)\). There is a path in \(T\) from \(v\) to \(w\). Since \(v\) and \(w\) on different partition, we follow this path from \(v\) to \(w\) and we will leave \(S\) at node \(v'\) and enter \(V - S\) at \(w'\) (for the first time). And \(e' = (v', w')\) belongs to \(T\). Now, we remove \(e'\) and include \(e\) instead. Claim: what we get is a spanning tree. First it is tree: still connected: each pair of node can still reachable (now going through \(e\) instead of \(e'\)). It is also not hard to show what we have is acyclic. The changed tree \(T\) is less costly.

Now we use this property to show Kruskal’s algorithm is correct (meaning optimal). Consider one edge \(e = (v, w)\) added by the algorithm. We let \(S\) be the nodes reachable from \(v\) by the selected edges by the algorithm so far. In class, we show this \(S\) works: first \(v\) in \(V\) but not \(w\) (why?). Also, \(e\) must be the first edge between \(S\) and \(V - S\) (why?). Thus, \(e\) is the cheapest between \(S\) and \(V - S\). So \(e\) belongs to every MST according to the “fundamental property of cut”. Moreover, although I did not explicitly show that the selected edges by the Kruskal’s algorithm must form a spanning tree, this is in fact easy to see. Suppose the selected edges do not form a spanning tree, then suppose these edges will connect nodes in two (the case of having three or more components is similar) disconnected components \(C_1\) and \(C_2\). Let \(S\) be the nodes in \(C_1\) and the rest nodes are in \(C_2\). Then there must be some edge crossing the cut of \((C_1, C_2)\) (why?). In this case, Kruskal’s algorithm will select some edges crossing this cut since such edge will not lead to cycles. This is a contradiction to that \(C_1\) and \(C_2\) are disconnected. 

Finally, I will show that we can make Kruskal’s algorithm work even when the edge weights are not distinct. In this case, as I explain in class, we can disturb the edges in \(G\) by very small amount s.t. the resulting graph \(G'\) has all distinct edge weight. Then we can find an MST \(T\) from \(G'\) using Kruskal’s algorithm. Now I claim \(T\) is also an MST for the original ( unperturbed) graph \(G\), if the perturbation of edge weights is small enough. I will explain why this is the case in the next lecture.