Tutorial 7: Graph algorithms

1 DAGs

We saw in class that we can linearize a directed acyclic graph (DAG) using DFS and sorting the vertices by the finish/post-visit times. Recall that a vertex in a directed graph is a *source vertex* if it has in-degree 0. Here's another proposed algorithm for linearizing a DAG G:

Algorithm 1: Linearizer(G)

```
1 while G is not empty do
2     Find a source vertex v in G;
3     Output v;
4     Remove v from G;
```

Does this algorithm always produce a valid linearization of DAGs? Give a proof or counter-example to justify your answer.

1.1 Solution

Theorem 1. *The Linearizer algorithm always produces a valid linearization of the DAG G.*

Proof. First, we need to argue that the algorithm is well-defined. We need to justify that there is always a source vertex in a DAG G. Since we have previously seen an algorithm to compute a linearization using DFS, we know that every DAG has a valid linearization. Furthermore, the first vertex in any valid linearization must have in-degree 0, since there are no back edges. Therefore every DAG must have at least one source vertex.

To show that this algorithm always produces a valid linearization, we will use induction on n, the number of vertices in G.

The base case is when $n = 1$. In this case, there is only one possible linearization and indeed it is valid because there are no back edges.

Suppose that for any graph G' with $n' < n$ vertices that Linearizer(G') produces a valid linearization of G'. Let v be a source vertex of G and let G' be the graph obtained from G by removing the vertex v. Observe that Linearizer(G) will output v and then perform exactly the same as a call to Linearizer(G'). The sequence of vertices produced is therefore v followed by the result of Linearizer(G'). By the inductive hypothesis, Linearizer(G') produces a valid linearization of G'. Since v is a source vertex, there are no edges from the vertices in G' to v. Therefore, v,Linearizer(G') is a valid linearization of G. □

2 Minimum Spanning Trees

Here's another proposed greedy algorithm for computing the Minimum Spanning Tree of a weighted graph:

Algorithm 2: CycleCut(G)

```
1 Sort the edges E by decreasing weight;
2 for each e ∈ E in order of decreasing weight do
3     if e is part of a cycle in (V, E) then
4         E ← E \ {e};
5 Return T = (V, E);
```

Does this algorithm always return a MST of weighted graphs? Give a proof of correctness or a counter-example to justify your answer.

2.1 Solution

Theorem 2. *CycleCut(G) always returns an MST of the graph G.*

Proof. One slick way to prove this is to observe that it is similar to Kruskal's algorithm, but in reverse. To show that this algorithm works, it suffices to show that it produces the same edge set as Kruskal's algorithm.

Of course, we can also write a direct proof, for which we need the following claim.

Claim. Let T^* be the spanning tree returned by CycleCut(G). Let C be any cycle in G and let e be the highest weighted edge in C. Then e is not in T^*.

Proof of claim: We an prove this by induction on the number of cycles in G. If there are no cycles in G, the claim is vacuously true. Assume that the claim holds for any G' with fewer cycles than G. Let e be the first edge removed from G by CycleCut. Then e is the highest weight edge in any cycle of G. Let C be a cycle in G. If C contains e, then CycleCut removes e, the highest weight edge in C. If C does not contain e, then C is a cycle in G', the graph obtained from G by removing the edge e. By the inductive hypothesis, CycleCut removes the highest weight edge of C in G', which is also the highest weight edge of C in G. By induction, the claim is proven for all graphs G.

Now that the claim is proven, we must use it to show the spanning tree T^* returned by CycleCut(G) is a minimum spanning tree.

Let T be any spanning tree which differs from T^*. Let e be an edge in T^* which is not in T. Say $e = (u, v)$. Since T is a spanning tree, it contains a path P from u to v. Together, the path P and edge e form a cycle C in G.

Case 1: e is the highest weighted edge in C. If this is true, this contradicts the above claim, because e is in T^*.

Case 2: $w(e) \leq w(e_2)$ for all edges e_2 in C. In this case, let e' be the highest weighted edge in C. Consider the graph T' formed from T by removing e' and adding e. Clearly, the sum of edge weights in T' is less than or equal to the sum of edge weights in T. Furthermore, T' is a spanning tree. Consider any pair of vertices in G, v_1 and v_2. Since T is a spanning tree, there is a path P

from v_1 to v_2 in T. If P does not contain e', then P is also a path in T'. If P does contain e', then the path P' obtained from P by replacing e' with $C \setminus \{e'\}$ is a path from v_1 to v_2 in T'. So in any case, every pair of vertices remains connected in T', so T' is a spanning tree containing the edge e with a cost at least as good as T.

□

3 Single-source shortest path

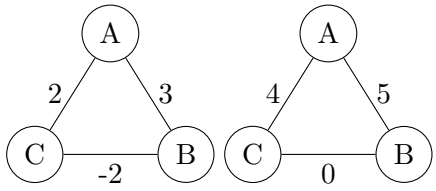
We saw a greedy algorithm (Dijkstra's algorithm) for solving the SSSP problem on graphs with non-negative weights, and a dynamic programming algorithm (Bellman-Ford algorithm) for solving the SSSP problem on weighted graphs that can have both positive and negative edge weights.

Prove that the following proposed reduction from the general SSSP problem to the SSSP problem on graphs with nonnegative weights does not correctly solve the problem:

1. Let $-M$ be the minimum edge weight in G.
2. Define G' to be the weighted graph with the same edges as G and with edge weights $w'(u, v) = w(u, v) + M$ for each $(u, v) \in E$.
3. Solve the SSSP problem for the nonnegative weighted graph G'.
4. For each vertex v with shortest path weight W in G' obtained by following a path of ℓ edges from s to v, output the distance $W - \ell M$.

3.1 Solution

One simple counterexample is the following graph G on the left and its corresponding G' on the right:



The shortest path from A to C in G is the path through vertex B which has total weight 3-2=1. The shortest path from A to C in G' is just the edge (A,C) which has weight 4. Converting this as described in the reduction gives us $4 - 1 \cdot 2 = 2$, which is not the shortest path length in G.