Tutorial 9: Single-source shortest path (SSSP)

1 No easy fix for negative edge weights

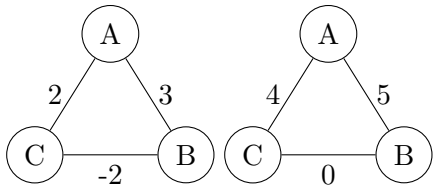
We saw a greedy algorithm (Dijkstra's algorithm) for solving the SSSP problem on graphs with non-negative weights, and a dynamic programming algorithm (Bellman-Ford algorithm) for solving the SSSP problem on weighted graphs that can have both positive and negative edge weights (but no negative cycles).

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 $W - \ell M$.

1.1 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.

Note: to get some intuition, observe that adding a constant to each edge in a path will disproportionately hurt paths with low weight but many edges, as compared to paths with high weight but few edges!

2 Coupon for a free edge

Suppose you want to find the shortest path from s to t in a weighted directed graph $G = (V, E)$ with no negative edge weights, with the added complication that I give you a **coupon for one free edge**. That is, you can pick any **one** edge and change its weight to **zero**.

(a) Find the shortest path that uses your coupon optimally. (Formally, find the path from s to t with the smallest total weight in *any* graph G' that is obtained from G by setting one edge weight to zero.) Show that this problem can be solved by reducing to Dijkstra's algorithm.

(b) HARD: Show how to modify the input graph in $O(m + n)$ time to reduce the problem in part (a) to a single Dijkstra's call. The Dijkstra's call should run with the same asymptotic time complexity as a single Dijkstra's call on the original graph G.

(c) Continuation of part (b): How could you solve a variant of this problem where you have k coupons for free edges? How much does this increase the size of the graph?

2.1 Solution

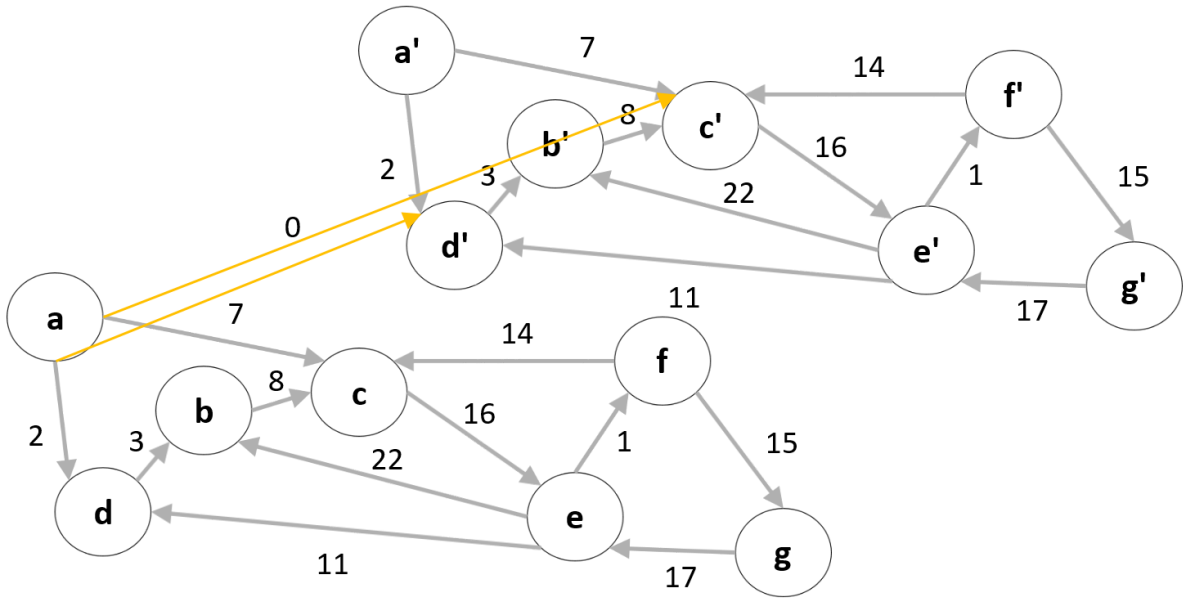
Part (a)

For each edge $e \in E$, construct a new graph G' that is the same as G, except e has weight zero, and run Dijkstra's algorithm on this graph. Return the *minimum* distance from s to t produced by any of the calls to Dijkstra's algorithm.

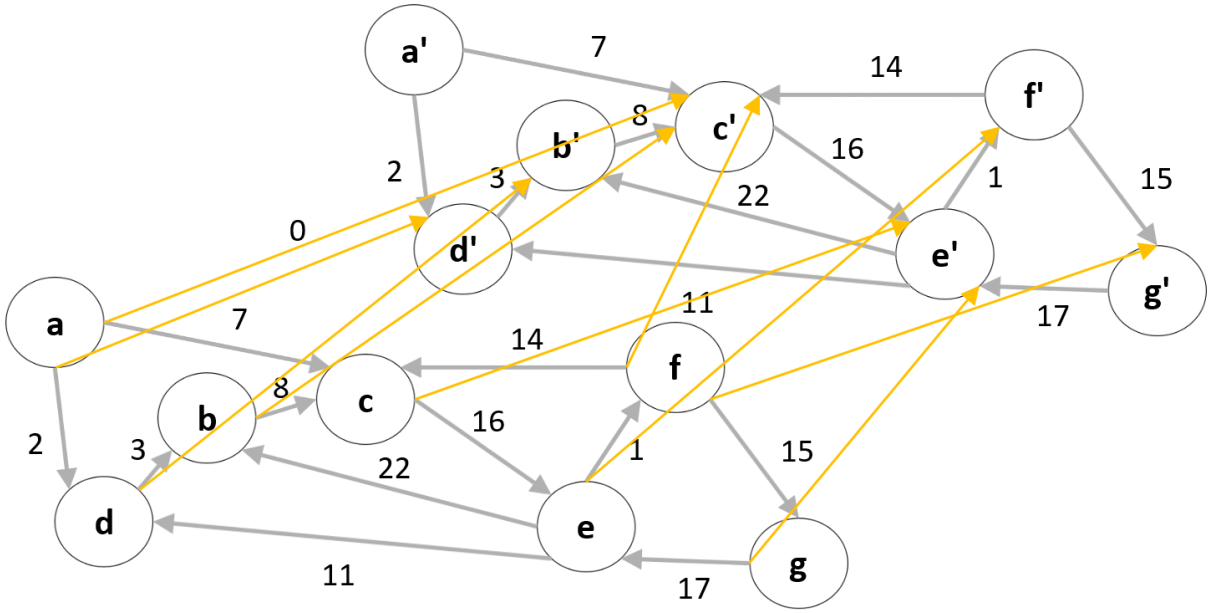
Part (b)

Create a new graph G' containing **two copies of graph** G called G_1 and G_2. (G_1 and G_2 are subgraphs of G'.) For each node u in subgraph G_1, let u' denote the *corresponding* node in G_2. For each node u in G_1, for each outgoing edge $\{u, v\}$, create an edge with weight zero in G' from u to v'. That is, a "free" edge from u in G_1 to the corresponding target of that edge in G_2. See figures on the next page for details.

The following figure shows G_1 and G_2 with free edges (in orange) from the node labeled a.



The following shows G' with **all** free edges drawn.



By taking an edge from G_1 to G_2, we can effectively take one edge in the original graph for free.

Try to convince yourself that the shortest path from s in G_1 to t' in G_2 is the shortest path in G', and is also the optimal solution to this problem.

Part (c)

Similar to the above, but instead of only two copies of G, we have $k+1$ copies representing $k+1$ *layers* of the graph, and free edges take you from one layer of the graph to the next.

This blows up the space complexity by a multiplicative factor of $(k+1)$. Runtime for, e.g., Dijkstra's algorithm also increases by the same multiplicative factor.

Interestingly, note that although the final graph G' has $(k+1)n$ nodes where n is the number of nodes in the original graph G, the number of edges in G' cannot be as large as $((k+1)n)^2$ (the naive number-of-nodes-squared upper bound we would get if we knew nothing about the structure of the graph G'). Because of the structure of G', we know it contains exactly $(k+1)m$ edges where m is the number of edges in the original graph. So, we know the number of edges in G' is $O(kn^2)$ (recall n is the number of nodes in the *original* graph).