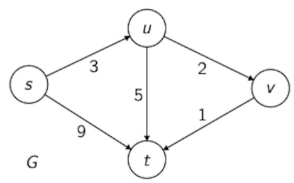
Lab 13: Shortest Paths

Module: Graphs

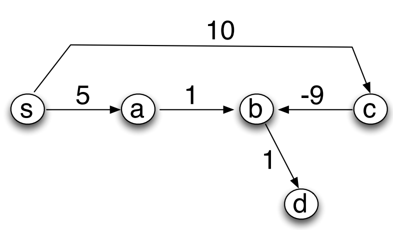
COLLABORATION LEVEL 0 (NO RESTRICTIONS). OPEN NOTES.

1. Step through `Dijkstra(G, s, t)` on the graph shown below. Complete the table below to show what the arrays d[] and p[] are at each step of the algorithm, and indicate what path is returned and what its cost is. Here D represents the set of vertices that have been removed from the PQ and their shortest paths found (in the notes we denoted it by S). .



	d[s]	d[u]	d[v]	d[t]	p[s]	p[u]	p[v]	p[t]
When entering the first while loop for the first time, the state is:	0	∞	∞	∞	None	None	None	None
Immediately after the first vertex is explored	0	3	∞	9	None	s	None	s
Immediately after the second vertex is explored								
Immediately after the third vertex is explored								
Immediately after the fourth vertex is explored								

2. Consider the directed graph below and assume you want to compute SSSP(s).



(a) Run Dijkstra's algorithm on the graph above step by step. Are there any vertices for which d[x] is correct? Are there any vertices for which d[x] is incorrect? Why?

(b) Now run Bellman-Ford algorithm, and assume the edges are relaxed in the following order: $\{bd, cb, ab, sc, sa\}$. For each round of relaxation, show the distances d[x] at the end of that round.

(c) How many rounds of relaxation are necessary for this graph, if the edges are relaxed in this specific order?

(d) In general, what is the worst-case number of rounds in Bellman-Ford algorithm for a graph of $|V|$ vertices?

(e) The bound aabove is not tight. Make a connection between the longest shortest path and the number of rounds that are necessary (in the worst case) for Bellman-Ford to converge.

(f) Give an order of relaxing edges for the graph above which correctly computes shortest paths for all vertices after just one round.

3. Give example of a graph G=(V,E) with an arbitrary number of vertices for which one round of relaxation in Bellman-Ford algorithm is always sufficient, no matter the order in which the edges are relaxed.

4. Give example of a graph G=(V,E) with an arbitrary number of vertices for which $|V| - 1$ rounds of relaxation in Bellman-Ford algorithm are always necessary in the worst case.

5. Consider Bellman-Ford algorithm and remember that by one round of relaxation we mean that *all* edges in the graph are relaxed (in arbitrary order). Fill in the sentences below so that they are true:

   (a) After one round of edge relaxation, it is guaranteed that $d[x] = \delta(s, x)$ for all vertices x whose shortest paths from s consist of ....

   (b) After i rounds of edge relaxation, it is guaranteed that $d[x] = \delta(s, x)$ for all vertices x whose shortest paths from s consist of ....

6. You are given an image as a two-dimensional array of size $m \times n$. Each cell of the array represents a pixel in the image, and contains a number that represents the color of that pixel (for e.g. using the RGB model).

   A segment in the image is a set of pixels that have the same color and are **connected**: each pixel in the segment an be reached from any other pixel in the segment by a sequence of moves up, down, left or right.

   Design an efficient algorithm to find the size of the largest segment in the image.

Additional problems: Optional

1. **Longest simple paths don't have optimal substructure:** Consider a directed graph G, and assume that instead of shortest paths we want to compute *longest paths.* Longest paths are defined in the natural way, i.e. the longest path from u to v is the path of maximum weight among all possible paths from u to v. Note that if the graph contains a positive cycle, then longest paths are not well defined (for the same reason that shortest paths are not well defined when the graph has a negative cycle). So what we mean is the *longest simple path*, (a path is called *simple* if it contains no vertex more than once).

   Show that the the *longest simple path* problem does not have optimal substructure by coming up with a small graph that provides a counterexample.

   Note: Finding longest (simple) paths is a classical *hard* problem, and it is known to be NP-complete.

2. **Arbitrage:** Suppose the various economies of the world use a set of currencies $C_1, C_2, ..., C_n$ –think of these as dollars pounds, bitcoins, etc. Your bank allows you to trade each currency C_i for any other currency C_j and finds some way to charge you for this service (in a manner to be elaborated in the subparts below). We will devise algorithms to trade currencies to maximize the amount we end up with.

   (a) Suppose that for each ordered pair of currencies (C_i, C_j), the bank charges a flat fee of $f_{ij} > O$ dollars to exchange C_i for C_j (regardless of the quantity of currency being exchanged). Devise an efficient algorithm which, given a starting currency C_s, a target currency C_t, and a list of fees f_{ij} for all $i, j \in \{1, ..., n\}$ computes the cheapest way (that is, incurring the least in fees) to exchange all of our currency in C_s into currency C_t. Justity the correctness of your algorithm and its runtime.

   (b) Consider the more realistic setting where the bank does not charge flat tees, but instead uses exchange rates. In particular, for each ordered pair (C_i, C_j), the bank lets you trade one unit of C_i to r_{ij} units of C_j. Devise an efficient algorithm which, given starting currency C_s, target currency C_t, and a list of rates r_{ij}, computes a sequence of exchanges that results in the greatest amount of C_t. Justify the correctness of vour algorithm and its runtime.

   Hint: How can you turn a product of terms into a sum?

   (c) Due to fluctuations in the markets, it is occasionally possible to find a sequence of exchanges that lets you start with currency A, change into currencies B, C, D, etc.. and then end up changing back to A with more money than you started (this is called *arbitrage*). Come up with an algorithm that, given a set of currencies and the exchange rates r_{ij} between them, determines if arbitrage is possible.