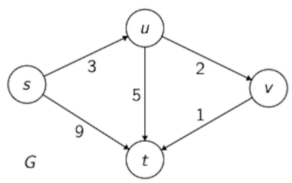
Lab 13: Shortest Paths

Module: Graphs

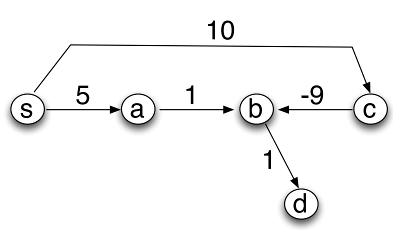
Collaboration level 0 (no restrictions). Open notes.

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 element of D is added, the state is:	0	3	∞	9	None	s	None	s
Immediately after the second element of D is added, the state is:								
Immediately after the third element of D is added, the state is:								
Immediately after the fourth element of D is added, the state is:								

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? In general, what is the worst-case number of rounds in Bellman-Ford algorithm for a graph of $|V|$ vertices?(in this case, $|V| = 5$) Why the difference? Can you connect the number of rounds necessary with something in the graph?

(d) 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. Prove that the following claim is false by showing a counterexample:

   Claim: Let $G = (V, E)$ be a directed graph with negative-weight edges, but no negative-weight cycles. Let $w, w < 0$, be the smallest weight in G. Then one can compute SSSP in the following way: transform G into a graph with all positive weights by adding $-w$ to all edges, run Dijkstra, and subtract from each shortest path the corresponding number of edges times $-w$. Thus, SSSP can be solved by Dijkstra's algorithm even on graph with negative weights.

7. 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

All-Pair-Shortest-Paths via matrix multiplication: In the APSP problem the goal is to compute the shortest path between all pairs of vertices $u, v \in V$. Note that the output is of size $\Theta(|V|^2)$ which means any algorithm for APSP runs in $\Omega(|V|^2)$.

1. We can solve the problem simply by running Dijkstra's algorithm $|V|$ times. What is the running time of this approach? What does the running time become for sparse graphs ($E = \theta(V)$) and for dense graphs ($E = \theta(V^2)$)?

   We can obtain another APSP algorithm by working on adjacency matrix of the graph, which we denote by A: for weighted graphs, a_{ij} is equal to the weight w_{ij} of the edge (v_i, v_j); w_{ij} is assumed to be ∞ is the edge does not exist.

   Let A, B be two matrices, and let $C = A \cdot B$. Remember that

   $$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

   We redefine the $\sum$ and $\cdot$ operators in matrix multiplication to mean *minimum* and $+$, respectively. That is,

   $$c_{ij} = min_{k=1..n}\{a_{ik} + b_{kj}\}$$

2. What does $A \cdot A$ represent in terms of paths in graph G?

3. What about $\min\{A, A \cdot A\}$?

4. How fast can you compute $B = \min\{A, A \cdot A\}$?

5. Sketch an algorithm for computing APSP using this approach and estimate its running time. Hint: express it as computing some power B^k. What is a sufficient value of k ?

6. Improve your algorithm by being smart about how you compute powers.

   Hint: aim to compute a^n in $O(\lg n)$ rather than in $O(n)$ time.

7. Describe how this corresponds to dynamic programming.

   Hint: consider the following subproblem: $d_k(u, v)$ is the shortest path from u to v that consists of at most k edges. What does $d_1(u, v)$ correspond to? How do you define $d_2(u, v)$ in terms of $d_1(u, v)$?