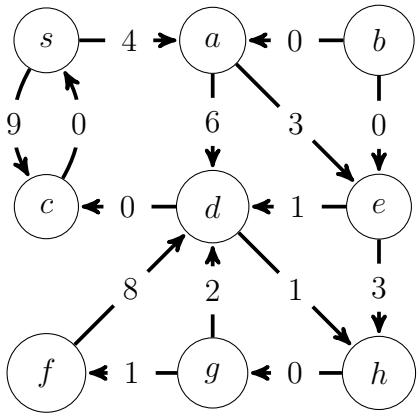
Problem Set 7

All parts are due on April 12, 2019 at 6PM. Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. Solutions should be submitted on the course website, and any code should be submitted for automated checking on `alg.mit.edu`.

Problem 7-1. [20 points] **Dijkstra Practice**

(a) [16 points] Run Dijkstra on the following graph from vertex s to every vertex in $V = \{a, b, c, d, e, f, g, h, s\}$. Write down (1) the minimum-weight path weight $\delta(s, v)$ for each vertex $v \in V$, and (2) the order that vertices are removed from Dijkstra's queue.



Solution:

Vertex v	a	b	c	d	e	f	g	h	s
$\delta(s, v)$	4	∞	8	8	7	10	9	9	0
Removal order	2	9	5	4	3	8	7	6	1

Rubric:

- 8 points for minimum-weight path weights
- -1 point per incorrect minimum-weight path weight, minimum zero points
- 8 points for vertex order
- -1 point per vertex order inversion, minimum zero points

(b) [4 points] Change the weight of edge (f, d) to -3. Identify a vertex v for which running Dijkstra from s as in part (a) would result in an incorrect output for $\delta(s, v)$.

Solution: If Dijkstra were to be run on the graph with the weight of edge (f, d) changed to -3, nodes would still be processed in the same order, and all reported distances would be the same, since f is the last vertex to be processed. However, the cycle (d, h, g, f, d) has negative weight, so any vertex v reachable from this cycle should have $\delta(s, v) = -\infty$. So any vertex accept for vertex b may be identified.

Rubric:

- 4 points for correctly identifying any vertex that is not b

Problem 7-2. [20 points] **Color Limit**

A graph is **two-colored** if each vertex is assigned a color either red or blue. Given a two-colored weighted directed graph $G = (V, E)$ having only positive weights and positive integer k, describe an efficient algorithm to return a path of minimum weight from a blue vertex s to a blue vertex t which passes through red vertices at most k times, or return that no such path exists.

Solution: Construct a new graph G': for each vertex $v \in V$, G' contains $k+1$ vertices $\{v_0, \ldots, v_k\}$; and for each directed edge $(u, v) \in E$, G' contains either edges (u_i, v_i) for $i \in \{0, \ldots, k\}$ if u is blue, or edges (u_i, v_{i+1}) for $i \in \{0, \ldots, k-1\}$ if u is red. Graph G' has $O(k|V|)$ vertices and $O(k|E|)$ edges and can be constructed in $O(k(|V| + |E|))$ time. G' is constructed so that a path from s_0 to t_i in G' corresponds directly to a path in G from s to t that passes through red vertices exactly i times. So to find a minimum weight path from s to t passing through red vertices at most k times, run Dijkstra from s_0 to compute minimum distances to each t_i for $i \in \{0, \ldots, k\}$, and then return a minimum weight path to any t_i for which $\delta(s_0, t_i)$ is minimum. Dijkstra in this graph takes $O(k|E| + k|V| \log(k|V|)) = O(k(|E| + |V| \log |V|))$ time.

Rubric:

- 9 points for description of a correct algorithm
- 3 points for a correct argument of correctness
- 3 points for a correct analysis of running time
- 5 points if correct algorithm is $O(k(|E| + |V| \log |V|))$

Problem 7-3. [20 points] **Duper Dario**

Duper Dario is a video game in which plumber Dario explores time and space by traversing g game levels. Game levels are connected via w **warp pipes**. Each warp pipe is one-way, where a warp pipe from level a to level b allows Dario to travel from level a to level b, and every level is reachable from every other level by traveling a sequence of warp pipes. Each warp pipe (a, b) is labeled with an integer t_{ab}: if Dario travels in warp pipe (a, b), the game clock will change by time t_{ab}, meaning Dario will go forward in time if t_{ab} is positive, and back in time if t_{ab} is negative (the game clock does not change while Dario is in a level). The warp pipes in the game are designed so that it is impossible for Dario to go back in time to the same location, i.e., if Dario warps away from any level a when the game clock shows time t, the game clock will always be at a higher time than t if he ever returns to level a. Describe an efficient algorithm to determine for **every** game level, the set of levels that Dario can reach from that level while going **strictly** back in time.

Solution: Construct a graph G on the g game levels, with a directed edge from level a to level b with weight t_{ab} if there is a warp pipe from level a to level b with label t_{ab}. Then traveling from one level to another will go back in time if the path between them has negative weight, and G does not contain negative weight cycles, because it is impossible for Dario to go back in time to the same location. Run Johnson's algorithm on G to compute all pairs shortest paths, and for each level a, return all levels reachable from a with shortest path length strictly less than zero. Graph G has g vertices and w edges, so Johnson's algorithm runs in $O(gw + g^2 \log g)$ time.

Rubric:

- 9 points for description of a correct algorithm
- 3 points for a correct argument of correctness
- 3 points for a correct analysis of running time
- 5 points if correct algorithm is $O(gw + g^2 \log g)$

Problem 7-4. [20 points] **Egg Rage**

Wester Rabbit is upset about last Halloween: apparently many neighbors thought dressing as a human was not a good Halloween costume and refused to give Wester candy. Now, whenever Wester passes one of these neighbors' houses, Wester descends into an uncontrollable fit of rage and throws an egg at the house. If Wester passes such a house without an egg, Wester will turn violent, which is most distressing. Wester plans to attend an egg festival across town and wants to avoid violence. Wester has a map showing the m two-way roads connecting all n intersections in town, including the intersection where the festival will be held. Wester marks each road with the number of houses on the road that refused candy. Wester also marks some intersections as having grocery stores where Wester can buy eggs, including the intersection where Wester starts. Wester's backpack has limited space, so Wester can only carry at most k eggs at any given time. Assuming that Wester always has enough money to buy eggs when needed, describe an efficient algorithm to determine the minimum number of eggs Wester must throw and still reach the festival carrying at least $12 < k$ eggs, without turning violent along the way (or return that violence is inevitable).

Solution: For this problem, it's unclear what an 'efficient' algorithm should actually be, as it depends on the relationship between n and k. We present two approaches here: one that is $O(n(m+n\log n))$ and the other that is $O(k(m+n\log n))$. An efficient algorithm would choose the smaller of the two, leading to an $O(\min(n,k)(m+n\log n))$ solution. Here, we will accept either solution for full points.

$O(n(m+n\log n))$ approach: We can use an approach similar to problem 6-5. Construct undirected graph G on the n road intersections, with edges corresponding to the m roads in the town, where each edge is weighted by the number of neighbors on that road who did not give out candy. Wester never wants to run out of eggs, so construct a new graph G' on the intersections containing grocery stores and the festival intersection t, with a directed edge from a grocery store u to a grocery store v if v is reachable from u by throwing at most k eggs, and a directed edge from u to t if it is reachable from u by throwing at most $k-12$ eggs. To compute edges from grocery store u, run Dijkstra from u in G to find all grocery stores reachable from u by throwing at most k eggs (or the festival intersection reachable from u by throwing at most $k-12$ eggs), letting the shortest path weight found be the corresponding edge weight $w \le k$ in G'. Then, any path from the starting grocery store s to the festival intersection t in G' corresponds to a path in G from s to t that avoids violence arriving at t with 12 eggs, so run Dijkstra in G' from s to t and return the weight of the shortest path found. G has n vertices, m edges, and at most n grocery stores, so running Dijkstra from each grocery store takes a total of $O(n(m+n\log n))$ time. Additionally, G' has at most $O(n)$ vertices and at most $O(n^2)$ edges, so running Dijkstra once on G' takes at most $O(n^2+n\log n)$ time, leading to a total running time of $O(n(m+n\log n))$.

$O(k(m+n\log n))$ approach: If $k = o(n)$, then we can do better than the above approach by keeping track of the amount of eggs in Wester's backpack along the way. Construct directed graph G'': for each intersection v, G'' contains $k+1$ vertices $\{v_0,\ldots,v_k\}$ (where v_i corresponds to Wester being at intersection v having i eggs); and if there is a road connecting intersections u and v in G with weight w, add a directed edge with weight w from u_i to v_{i-w} for every $i \in \{w,\ldots,k\}$ if v does not contain a grocery store, or a directed edge from u_i to v_k for every $i \in \{0,\ldots,k\}$ if v contains a grocery store. Then any path in G'' from s_k to any t_i for $i \in \{12,\ldots,k\}$ corresponds to a path in G from s to t that avoids violence arriving at t with 12 eggs, so run Dijkstra from s_k and return the weight of the shortest path found to any t_i for $i \in \{12,\ldots,k\}$. G'' has $O(kn)$ vertices and at most $O(km)$ edges, so running Dijkstra from s_k in G'' takes $O(km+kn\log(kn))$, i.e., $O(k(m+n\log n))$ time.

Rubric:

- 9 points for description of a correct algorithm
- 3 points for a correct argument of correctness
- 3 points for a correct analysis of running time
- 5 points if correct algorithm is $O(\max(k,n)(m+n\log n))$
- (so $O(k(m+n\log n))$ and $O(n(m+n\log n)$ are both fine)

Problem 7-5. [20 points] **Hiking Hibernation**

Steryl Chrayed is planning a winter hike through Canyonmaze National Park, a large network of deep canyons. Steryl has a map of the park depicting all the canyons and clearings in Canyonmaze, where a single canyon is a trail with high-walls that directly connects two clearings, where each clearing connects at most six canyons. Each canyon on the map is marked with its length. Steryl leaves her car at one clearing and wants to hike across the park to her favorite clearing, but she does not want to disturb the wildlife along the way. In particular, some clearings are home to groundhogs that hibernate in the winter. Hikers will disturb groundhogs if they hike too close. Steryl knows which clearings are home to groundhogs, and has marked each groundhog-occupied clearing c_i on the map with a (possibly different) positive **safety distance** d_i, meaning if Steryl is ever within hiking distance d_i of clearing c_i, she would disturb the groundhogs that live at c_i. Given the lengths of the n canyons in Canyonmaze, along with the g groundhog-occupied clearings and their associated safety distances, describe an efficient algorithm to determine whether Steryl can hike from her car to her favorite clearing without disturbing any groundhogs.

Solution: Construct a graph G with the vertices being the $O(n)$ clearings connected to some canyon, and edges being the n canyons connecting clearings weighted by their length (if neither the parking clearing or favorite clearing are connected to a canyon, return that there is no such hike). We would like to remove vertices from G that are within the safety distance of any groundhog-occupied clearing. We could remove all such vertices by running Dijkstra once from each groundhog-occupied clearing, but we can actually find all such vertices by running Dijkstra only once. Add a new vertex x to G to form graph G' with a directed edge from x to each groundhog-occupied clearing c_i with weight $D - d_i$ where $D = \max_i d_i$. Then, a vertex in G is within the safety distance of a groundhog-occupied clearing if and only if its shortest distance from x is less than or equal to D. Run Dijkstra from x in G' to find all vertices within distance D and remove them from G to form graph G''. Then run DFS or BFS in G'' to determine whether the parking clearing and the favorite clearing are in the same connected component in G''. Both graphs G' and G'' have $O(n)$ vertices and $O(n)$ edges, so performing Dijkstra once one each takes $O(n \log n)$ time in total.

Rubric:

- 9 points for description of a correct algorithm
- 3 points for a correct argument of correctness
- 3 points for a correct analysis of running time
- 5 points if correct algorithm is $O(n \log n)$