25 (100 PTS.) Rainbow walk

We are given a directed graph with n vertices and m edges ($m \geq n$), where each edge e has a color $c(e)$ from $\{1, \ldots, k\}$.

25.A. (20 PTS.) Describe an algorithm, as fast as possible, to decide whether there exists a closed walk that uses all k colors. (In a walk, vertices and edges may be repeated. In a closed walk, we start and end at the same vertex.)

Solution:

The algorithm is simple:

1. First compute the strongly connected components.
2. Return true iff some component contains all k colors.

Runtime. We can compute the strongly connected components in $O(m)$ time by the algorithm from class. For each component, one way to check whether it uses all k colors is to use a mark bit (initialized to false) for each color: examine each edge and turn on the mark bit for its color; at the end, we check that all colors are marked, and then make another pass over the edges to unmark all colors encountered, before proceeding to the next component. The running time of this step is proportional to the number of edges, $O(m)$.

Correctness. If a component contains edges $(u_1, v_1), \ldots, (u_k, v_k)$ using all k colors, we can connect these edges together to form a closed walk (since by definitiion of strongly connected components, there are paths from v_1 to u_2, v_2 to u_3, etc.).

Conversely, if there is a closed walk using all k colors, all vertices in the walk belong to the same strongly connected component, so this component uses all k colors.

25.B. (80 PTS.) Now, assume that there are only 3 colors, i.e., $k = 3$. Describe an algorithm, as fast as possible, to decide whether there exists a walk that uses all 3 colors. (The start and end vertex may be different.)

Solution:

Let $G = (V, E)$ be the given directed graph. Construct a new directed graph G':

- For each $v \in V$ and subset $S \subseteq \{1, 2, 3\}$, create a new vertex (v, S) in G'.
- For each edge $(u, v) \in E$ with color $c(u, v)$ and subset $S \subseteq \{1, 2, 3\}$, create an edge from (u, S) to $(v, S \cup \{c(u, v)\})$ in G'.
- Furthermore, create a new start vertex s' in G', and add edges from s' to $(v, \emptyset)$ in G' for every $v \in V$.

We then run BFS or DFS in G' from the start vertex s', and return true iff some vertex of the form $(v, \{1, 2, 3\})$ is reachable from s'.

Runtime. G' has at most $8n + 1 = O(n)$ vertices and $8m = O(m)$ edges. The BFS/DFS takes $O(m)$ time.

Justification. A vertex (v, S) is reachable from s' in G' iff there is a walk in G that ends at v and uses precisely the colors in S. (This claim can be verified by straightforward induction.) Thus, the algorithm returns true iff there is a walk using all colors in $\{1, 2, 3\}$.

26 (100 PTS.) Stay safe

We are given an *undirected* graph with n vertices and m edges ($m \geq n$), where each edge e has a positive real weight $w(e)$, and each vertex is marked as either "safe" or "dangerous".

26.A. (35 PTS.) Given safe vertices s and t, describe an $O(m)$-time algorithm to find a path from s to t that passes through the smallest number of dangerous vertices.

Solution:

First compute the connected components of the subgraph induced by the safe vertices, by BFS or DFS in $O(m)$ time. Let G be the original graph. Construct a new undirected graph G':

- For each component C of safe vertices, add C as a vertex in G'.
- For each dangerous vertex v in G, add v as a vertex in G'.
- For each edge uv in G where u is in a component C_u of safe vertices and v is a dangerous vertex, add $C_u v$ as an edge in G'.
- For each edge uv in G where u and v are dangerous, create a new "dummy" vertex that is joined to both u and v.

The new graph G' has $O(m)$ vertices and edges. Finally, run BFS to compute the unweighted shortest path in G' between the components C_s and C_t containing s and t; the BFS takes $O(m)$ time.

There is a path from s to t in G through ℓ dangerous vertices iff there is a path from C_s to C_t in G' with 2ℓ edges (the path alternates between components/dummy vertices and dangerous vertices).

Remark. For an alternative solution without computing components first, we can simply reduce the problem to single-source shortest paths in a graph where all edge weights are 0 or 1 (namely, assign an edge (u, v) a weight of 1 if u is dangerous and 0 if u is safe). If we use Dijkstra's algorithm, this would give $O(n \log n + m)$ time, but with some thought, it is possible to implement Dijkstra's algorithm in $O(m)$ time in the 0-or-1 special case (or modify BFS directly to handle 0-weight edges)...

Solution:

Alternative solution: **DFS** and **BFS** combined (the 0/1 Dijkstra hinted to above).

Do a **DFS** from s, where the **DFS** does not handle the outgoing edges out of dangerous vertices (the **DFS** treat them has not having any outgoing edges). Let D_0 be all the set of dangerous vertices that this **DFS** reached. All the vertices visited during the **DFS** are marked as (wait for it) visited. Let U_0 be this set of visited vertices.

Now, do **DFS** from each of the vertices of D_0 (ignoring their marking as already visited), where the algorithm ignores vertices that were already visited (as in the usual **DFS**). Again the **DFS** does not use any outgoing edges for dangerous vertices that are visited for the first time. Let D_1 be the newly visited dangerous vertices. Again, all the vertices visited are marked as such (let U_1 be the set of vertices visited during this stage).

We repeat this process till the graph is exhausted. In the ith iteration doing **DFS** from the vertices of D_{i-1} to compute the newly visited set of vertices U_i, and the newly discovered dangerous vertices D_i.

It is easy to verify using boring induction, that all the vertices in $U_i \setminus D_i$ are reachable by paths using exactly i dangerous vertices, and furthermore, there are no less dangerous paths to these vertices from s. In particular, if $t \in U_k$, then the least dangerous path to t uses only k dangerous vertices.

As for the running time analysis, it is easy to verify that a vertex can participate in at most two stages. Each stage might visit all the outgoing edges for such a vertex. It follows that the overall running time is proportional to the number of edges, that is $O(m)$.

Remark. Another (essentially equivalent) alternative is to modify BFS directly, with a deque (double-ended queue). When exploring the neighbors v of a vertex u, if v is safe, we insert v to the front of the deque; if v is dangerous, we insert v at the back... (Yet another alternative is to use two queues...)

26.B. (65 PTS.) Given safe vertices s and t and a value W, describe an algorithm, as fast as possible, to find a path from s to t that passes through the smallest number of dangerous vertices, subject to the constraint that the total weight of the path is at most W.

Solution:

Let $G = (V, E)$ be the original graph. Construct a new directed graph G':

- For each $v \in V$ and number $k \in \{0, \ldots, n\}$, create a vertex (v, k) in G'.
- For each $uv \in E$ of weight w and number k, if v is dangerous, create an edge from (u, k) to $(v, k+1)$ of weight w (if $k < n$); if v is safe, create an edge from (u, k) to (v, k) of weight w. (Do the same with u and v swapped.)

The new graph G' has $O(n^2)$ vertices and $O(mn)$ edges.

Run Dijkstra's single-source shortest path algorithm on G' from $(s, 0)$. Dijkstra's algorithm takes time $O(|V'| \log |V'| + |E'|) = O(n^2 \log n + mn)$.

Observe that a path from $(s, 0)$ to (v, k) in G' corresponds to a path from s to v in G that uses k dangerous vertices (possibly including v). Thus, the problem can be solved by identifying the smallest k^* for which the shortest path distance from $(s, 0)$ to (t, k^*) is at most W. This takes at most $O(n)$ additional time (e.g., naively by a linear search).

The total running time is $O(n^2 \log n + mn)$.

(*Remark.* If k^* is small, it is possible to improve the running time to $O(nk^* \log n + mk^*)$.)

27 (100 PTS.) Stay stable

We are given a directed graph with n vertices and m edges ($m \geq n$), where each edge e has a weight $w(e)$ (you can assume that no two edges have the same weight). For a cycle C with edge sequence $e_1 e_2 \cdots e_\ell e_1$, define the *fluctuation* of C to be

$$f(C) = |w(e_1) - w(e_2)| + |w(e_2) - w(e_3)| + \cdots + |w(e_\ell) - w(e_1)|.$$

27.A. (10 PTS.) Show that the cycle with the minimum fluctuation cannot have repeated vertices or edges, i.e., it must be a simple cycle.

Solution:

Let C be a cycle with the minimum $f(C)$. If there is more than one such optimal cycle, pick the one with the smallest number of edges. Let $e_1, e_2, \ldots, e_\ell, e_1$ be its edge sequence. Suppose that a vertex or edge is repeated. Then the end vertex of e_i is equal to the start vertex of e_j for some $j > i + 2$. By the triangle inequality,

$$|w(e_i) - w(e_{i+1})| + \cdots + |w(e_{j-1}) - w(e_j)| \;\geq\; |w(e_i) - w(e_j)|.$$

Thus

$$\begin{aligned} f(C) &= |w(e_1) - w(e_2)| + \cdots + |w(e_{i-1}) - w(e_i)| + \\ &\quad |w(e_i) - w(e_{i+1})| + \cdots + |w(e_{j-1}) - w(e_j)| + \\ &\quad |w(e_j) - w(e_{j+1})| + \cdots + |w(e_\ell) - w(e_1)| \\ &\geq |w(e_1) - w(e_2)| + \cdots + |w(e_{i-1}) - w(e_i)| + \\ &\quad |w(e_i) - w(e_j)| + \\ &\quad |w(e_j) - w(e_{j+1})| + \cdots + |w(e_\ell) - w(e_1)|. \end{aligned}$$

So, the cycle $e_1, \ldots, e_i, e_j, e_{j+1}, \ldots, e_\ell, e_1$ has smaller, or same, fluctuation and is thus also optimal but has fewer edges: a contradiction.

27.B. (90 PTS.) Describe a polynomial-time algorithm, as fast as possible, to find the cycle with the minimum fluctuation.

Solution:

Let $G = (V, E)$ be the original directed graph. Construct a new directed graph G':

- For each edge $e \in E$, make e a vertex in G'.
- For each pair of edges (u, v) and (v, y) in E, create an edge from (u, v) to (v, y) with weight $|w(u, v) - w(v, y)|$.

The new graph $G' = (V', E')$ has m vertices, and at most $O(mn)$ edges (since for each edge $(u, v) \in E$, there are at most n choices for y).

A cycle with minimum fluctuaton in G corresponds to a cycle with minimum weight in G'.

We can find a shortest-weight cycle in G' by running an all-pairs shortest paths algorithm: a shortest cycle corresponds to a shortest path from u to v, followed by an edge (v, u), so it suffices to minimize the shortest path distance from u to v plus the weight of

(v, u). The computation takes $O(|V'|^2 \log |V'| + |V'||E'|)$ time by repeatedly running Dijkstra's algorithm $|V'|$ times from every start vertex. Since $|V'| = O(m)$ and $|E'| = O(mn)$, the algorithm takes $O(m^2 n)$ time.

Remark on improvement. It is possible to find a sparser graph G'' that preserves distances as G', with $O(m)$ vertices and only $O(m \log n)$ edges. (Hint: given two sets of numbers $A = \{a_1, \ldots, a_d\}$ and $B = \{b_1, \ldots, b_d\}$, there is a clever construction of a directed graph with vertices $A \cup B$ plus $O(d)$ extra vertices, and $O(d \log d)$ edges, such that there is a path from a_i to b_j iff $a_i \geq b_j$. (Hint within the hint: use divide-and-conquer.) Furthermore, we can assign weights to edges so that this path has weight $a_i - b_j$ if $a_i \geq b_j$. By a symmetric construction, we get a path of weight $b_j - a_i$ if $a_i \leq b_j$. Now apply this construction to the neighborhood of each vertex in $V \ldots$)

With this improvement, the running time reduces to $O(|V'|^2 \log |V'| + |V'||E'|) = O(m^2 \log n)$.