Quiz 2 Review

Scope

- Quiz 1 material fair game but explicitly **not emphasized**
- 6 lectures on graphs, L09-L14, 2 Problem Sets, PS5-PS6

Graph Problems

- Graph reachability by BFS or DFS in $O(|E|)$ time
- Graph exploration/connected components via Full-BFS or Full-DFS
- Topological sort / Cycle detection via DFS
- Negative-weight cycle detection via Bellman-Ford
- Single Source Shortest Paths (SSSP)

Restrictions		SSSP Algorithm	
Graph	Weights	Name	Running Time $O(\cdot)$
DAG	Any	DAG Relaxation	$\|V\| + \|E\|$
General	Unweighted	BFS	$\|V\| + \|E\|$
General	Non-negative	Dijkstra	$\|V\| \log \|V\| + \|E\|$
General	Any	Bellman-Ford	$\|V\| \cdot \|E\|$

- All Pairs Shortest Paths (APSP)
  - Run a SSSP algorithm $|V|$ times
  - Johnson's solves APSP with negative weights in $O(|V|^2 \log |V| + |V||E|)$

Graph Problem Strategies

- Be sure to **explicitly describe a graph** in terms of problem parameters
- Convert problem into finding a shortest path, cycle, topo. sort, conn. comps., etc.
- May help to duplicate graph vertices to encode additional information
- May help to add auxiliary vertices/edges to graph
- May help to pre-process the graph (e.g., to remove part of the graph)

Graph Problem Common Mistakes

- Define your graphs! Specify vertices, edges, and weights clearly (and count them!)
  - (e.g., construct graph $G = (V, E)$ with a vertex for each...
    and a directed edge (u, v) with weight w for each...)
- State the problem you are solving, not just the algorithm you use to solve it
  - (e.g., solve SSSP from s by running DAG Relaxation...)
- Connect the graph problem you solve back to the original problem
  - (e.g., the weight of a path from s to t in G corresponds to the sum of tolls paid along a driving route, so a path of minimum weight corresponds to a route minimizing tolls)

Problem 1. Counting Blobs (S18 Quiz 2)

An **image** is a 2D grid of black and white square pixels where each white pixel is contained in a **blob**. Two white pixels are in the same blob if they share an edge of the grid. Black pixels are not contained in blobs. Given an $n \times m$ array representing an image, describe an $O(nm)$-time algorithm to count the number of blobs in the image.

Solution: Construct a graph G with a vertex per white pixel, with an undirected edge between two vertices if the pixels associated with them are both white and share an edge of the grid. This graph has size at most $O(nm)$ vertices and at most $O(nm)$ edges (as pixels share edges with at most four other pixels), so can be constructed in $O(nm)$ time. Each connected component of this graph corresponds to a blob, so run Full-BFS or Full-DFS to count the number of connected components in G in $O(nm)$ time.

Problem 2. Unicycles (S18 Quiz 2)

Given a **connected** undirected graph $G = (V, E)$ with strictly positive weights $w : E \to \mathbb{Z}^+$ where $|E| = |V|$, describe an $O(|V|)$-time algorithm to determine a path from vertex s to vertex t of minimum weight.

Solution: Given two vertices in a weighted tree containing only positive weight edges, there is a unique simple path between them which is also the minimum weight path. A depth-first search from any source vertex s in the tree results in a directed DFS tree in $O(|V|)$ time (since $|E| = |V| - 1$). Then relaxing edges in topological sort order of the directed DFS tree computes minimum weight paths from s in $O(|V|)$ time. Since G has one cycle, our strategy will be to break the cycle by removing an edge, and then compute the minimum weight path from s to t in the resultant tree.

First, we find the vertex v closest to s on the cycle by running depth-first search from s in $O(|V|)$ time (since $|E| = |V|$). One edge e_1 of the cycle will not be in the tree returned by DFS (a back edge to v), with the other edge of the cycle incident to v being a single outgoing DFS tree edge e_2. If s is on the cycle, $v = s$; otherwise the unique path from s to v does not contain e_1 or e_2.

A shortest path from s to t cannot traverse both edges e_1 and e_2, or else the path would visit v at least twice, traversing a cycle of positive weight. Removing either e_1 or e_2 results in a tree, at least one of which contains the minimum weight path from s to t. Thus, find the minimum weight path from s to t in each tree using the algorithm described above, returning the minimum of the two in $O(|V|)$ time.

Problem 3. Doh!-nut (S18 Quiz 2)

Momer has just finished work at the FingSprield power plant at location p, and needs to drive to his home at location h. But along the way, if his driving route ever comes within driving distance k of a doughnut shop, he will stop and eat doughnuts, and his wife, Harge, will be angry. Momer knows the layout of FingSprield, which can be modeled as a set of n locations, with two-way roads of known driving distance connecting some pairs of locations (you may assume that no location is incident to more than five roads), as well as the locations of the d doughnut shops in the city. Describe an $O(n \log n)$-time algorithm to find the shortest driving route from the power plant back home that avoids driving within driving distance k of a doughnut shop (or determine no such route exists).

Solution: Construct a graph G with a vertex for each of the n city locations, and an undirected edge between two locations if there is a road connecting them, with each edge weighted by the positive length of its corresponding road. The degree of each vertex is bounded by a constant (i.e., 5), so the number of edges in G is $O(n)$. First, we identify vertices that are within driving distance k of a doughnut shop location: create an auxiliary vertex x with a 0-weight outgoing edge from x to every doughnut shop location, and run Dijkstra from x. Remove every vertex from the graph whose shortest path from x is less than or equal to k, resulting in graph $G' \subset G$. If either p or h are not in G', then no route exists. Otherwise, run Dijkstra from p in G'. If no path exists to h, then no valid route exists. Otherwise, Dijkstra finds a shortest path from p to h, so return it (via parent pointers). This algorithm runs Dijkstra twice. Since the size of either graph is $O(|V|)$, Dijkstra runs in $O(|V| \log |V|) = O(n \log n)$ time (e.g. using a binary heap to implement a priority queue).

Problem 4. Long Shortest Paths

Given directed graph $G = (V, E)$ having arbitrary edge weights $w : E \to \mathbb{Z}$ and two vertices $s, t \in V$, describe an $O(|V|^3)$-time algorithm to find the minimum weight of any path from s to t containing **at least** $|V|$ edges.

Solution: Our strategy will compute intermediate values for each vertex $v \in V$:

1. the minimum weight $w_1(v)$ of any path from s to v using **exactly** $|V|$ edges, and then
2. the minimum weight $w_2(v)$ of any path from v to t using any number of edges.

First, to compute (1), we make a duplicated graph similar to Bellman-Ford, but without edges corresponding to remaining at a vertex. Specifically, construct a graph G_1 with

- $|V| + 1$ vertices for each vertex $v \in V$: vertex v_k for $k \in \{0, \ldots, |V|\}$ representing reaching v from s along a path containing k edges; and
- $|V|$ edges for each edge $(u, v) \in E$: edge (u_{k-1}, v_k) of the same weight for $k \in \{1, \ldots, |V|\}$.

Now a path in G_1 from s_0 to $v_{|V|}$ for any $v \in V$ corresponds to a path from s to v in G through exactly $|V|$ edges. So solve SSSPs in G_1 from s_0 to compute the minimum weight of paths to each vertex traversing exactly $|V|$ edges. This graph is acyclic, and has size $O(|V|(|V| + |E|)) = O(|V|^3)$, so we can solve SSSP on G_1 via DAG relaxation in $O(|V|^3)$ time.

Second, to compute (2), we make a new graph G_2 from G where every edge is reversed. Then every path to t in G corresponds to a path in G_2 from t, so compute SSSPs from t in G_2 to find the minimum weight of any path from v to t in G using any number of edges, which can be done in $O(|V||E|) = O(|V|^3)$ time using Bellman-Ford.

Once computed, finding the minimum sum of $w_1(v) + w_2(v)$ over all vertices $v \in V$ will provide the minimum weight of any path from s to t containing at least $|V|$ edges, since every such path can be decomposed into its first $|V|$ edges and then the remainder. This loop takes $O(|V|)$ time, so the algorithm runs in $O(|V|^3)$ time in total.