Problem 1 (Fast SSSP, 35 pts)

In class we observed that BFS solves single source shortest path on *unweighted* graphs in time $O(|V| + |E|)$. In this problem we will show how to solve SSSP in the same time complexity when G is a weighted graph with integer weights in the range $[1, c]$, where c is a constant. In SSSP for weighted graphs, we want to find the path that has the shortest path weight/distance, which is the sum of the weights of all the edges in the path. We can view SSSP for unweighted graphs as a special case where each edge has the same weight of 1.

Specifically, let c be a constant. Consider weighted graphs $G = (V, E, w)$ such that the weight $w(u, v)$ is an integer between 1 and c for each edge $(u, v) \in E$.

Design an algorithm to solve SSSP in time $O(|V| + |E|)$ on such graphs. Your algorithm should output `parent` encoding the shortest paths, and `dist` encoding the shortest path weight/distance of each node, as we saw in the BFS example in class. Your algorithm should use BFS as a subroutine. Justify the correctness and runtime of your proposed algorithm.

Problem 2 (Shortest Separable Path, 35 pts)

Let P be a program which, given as input a directed graph $G = (V, E)$ as well as two vertices $s, t \in V$, outputs the shortest path between them. Let $T(P)$ be the run-time of this program. Imagine that P is already highly optimized (say for running on a GPU) such that $T(P)$ runs significantly faster than your own shortest path algorithm. As such, we will use P to solve the following problem.

We are given a directed graph G where each of its edges is colored either red or blue. We want to find the shortest path from some vertex s to some other vertex t, with the requirement that our path must be *separable*. In order to be separable, the path we output must consist of first some number (possibly 0) of red edges followed by some number (possibly 0) of blue edges. In other words, once you use a blue edge, all following edges must be blue.

Design an algorithm to find the shortest separable path from s to t. Your algorithm should invoke P *only once* on a carefully constructed graph, and it should run in time $O(|V| + |E|) + T(P)$. You should *not* explore the graph yourself (i.e., do not implement BFS); use the optimized program P. Justify the correctness and runtime of your proposed algorithm.

(Hint: Consider the subgraph formed by restricting the edges to only the red ones. Then, consider the subgraph with only the blue edges. Notice that you are to spend some number of steps in the first subgraph and then move to the second subgraph and stay there. Can you combine the two graphs somehow?)

Problem 3 (DFS Olympics, 30 pts)

Consider running depth first search on a graph G of n nodes. As in lecture, for each vertex v we will keep track of $\texttt{disc}[v]$ and $\texttt{finish}[v]$, the times when v is first discovered and when v finishes being processed respectively.

For some graph $G = (V, E)$, we can consider $\max_{v \in V} \texttt{disc}[v]$ and $\max_{v \in V} \texttt{finish}[v]$, the largest values of $\texttt{disc}[v]$ and $\texttt{finish}[v]$ in the graph.

(a) Over all graphs $G = (V, E)$ of size $|V| = n$, what is the maximum value of $\max_{v \in V} \texttt{disc}[v]$? What is the minimum value? Give two graphs on 6 vertices satisfying these two values respectively.

(b) Over all graphs $G = (V, E)$ of size $|V| = n$, what is the maximum value of $\max_{v \in V} \texttt{finish}[v]$? What is the minimum value? Give two graphs on 6 vertices satisfying these two values respectively.

(c) Over all graphs $G = (V, E)$ of size $|V| = n$, what is the maximum value of $\max_{v \in V}(\texttt{finish}[v] - \texttt{disc}[v])$? What is the minimum value? Give two graphs on 6 vertices satisfying these two values respectively.