Question 1 (Priority Queue Tradeoffs, 30 points). Suppose that Dijkstra’s algorithm is being run on a graph $G = (V, E)$ with the priority queue implemented using a $d$-ary heap. By adjusting $d$, we can try to achieve better runtimes for the algorithm.

(a) Show that if $|E| > |V|^{11/10}$, that $d$ can be selected so that Dijkstra’s algorithm runs in linear time. [10 points]

(b) What is the runtime if $d$ is selected to be $|E|/|V|$ (assuming that this is an integer at least 2)? Show that no other choice of $d$ improves on this runtime by more than a constant multiple. [10 points]

(c) For a fixed value of $|V|$, what is the biggest possible ratio between this runtime and the runtime of Dijkstra using a Fibonacci heap? [10 points]

Solution:

(a) Given: $|E| > |V|^{11/10}$

Run time of Dijkstra’s algorithm is $O((d|V| + |E|)\log_d|V|)$.

Let’s take $d = |V|^{1/10}$

\[ \text{Runtime} = O((|V|^{11/10} + |E|)\log_{|V|^{1/10}}|V|) \]

\[ = O(|E| \frac{\log |V|}{\log |V|^{1/10}}) \]

\[ = O(|E| * 10^{\frac{\log |V|}{\log |V|}}) \]

\[ = O(10 * |E|) = O(|E|) \]

Thus, the Dijkstra’s algorithm runs in linear time in this case.

(b) Runtime of Dijkstra’s algorithm is $O((d|V| + |E|)\log_d|V|)$.

Given: $d = |E|/|V|$ ($\geq 2$)

We know, $|E| = |V|^{1+\delta}$ where $0 \leq \delta < 1$

\[ \therefore \text{Runtime} (T) = O(2|E|\log_d|V|) = O(|E|\log_d|V|) \]

Let’s take a different value of $d$. Let $d’ = c * |E|/|V|$.

\[ \therefore \text{Runtime} (T_{new}) = O((c + 1)|E|\log_{d’}|V|) \]

Break this into two cases:

1. $c < 1$ i.e. $d’ < |E|/|V|$
   
   Runtime $(T_{new}) = O(|E|\log_{d’}|V|)$ (term $|E|\log_{d’}|V|$ dominates as the decrease_key operation becomes worse). Considering, $2 \leq d’ < |V|$, the runtime cannot be improved more than a constant multiple.

2. $c > 1$ i.e. $d’ > |E|/|V|$
   
   Runtime $(T_{new}) = O(d’|V| \log_{d’}(|V|))$ (term $d’|V|\log_{d’}|V|$ dominates as the delete_min operation becomes worse). Increasing $c$ will only increase the runtime.

In both the cases, the runtime is at least $O(|E|\log_{|E|/|V|}|V|)$. Hence, changing $d$ doesn’t improve the runtime more than a constant multiple.
(c) Let’s break it into two cases: (1) \(|E| \geq |V| \log |V|\) and (2) \(|E| < |V| \log |V|\)

(1) Runtime of Fibonacci Heap is then \(O(|E|)\). The ratio reduces to \(\frac{\log |V|}{\log(|E|/|V|)}\). In order to maximize the term, we should take minimum value of \(|E|\) which is \(|V| \log |V|\). Taking \(|E| = |V| \log |V|\) and simplifying it we get ratio to be \(\frac{\log |V|}{\log |V|}\).

(2) Runtime of Fibonacci Heap is then \(O(|V| \log |V|)\). The ratio reduces to \(\frac{|E|}{|V| \log(|E|/|V|)}\). As \(|E|\) grows faster than \(\log |E|\), we should take the largest possible value of \(|E|\). Taking \(|E| = |V| \log |V|\) and simplifying it we get ratio to be \(\frac{\log |V|}{\log |V|}\). Thus, the biggest ratio possible is \(\frac{\log |V|}{\log |V|}\).

**Question 2** (Vertex labelling, 30 points). Suppose that you are given a directed graph \(G = (V,E)\) and a function \(f : V \rightarrow \mathbb{R}\). We would like to compute another function \(g : V \rightarrow \mathbb{R}\) so that for every vertex \(v \in V\) we have that

\[ g(v) + \sum_{(v,w) \in E} g(w) = f(v). \]

(a) Show that if \(G\) is a DAG that there is a linear time algorithm for computing such a \(g\). Hint: consider determining the values \(g(v)\) one at a time in some particular order. [20 points]

(b) Given an example to show that this is not always even possible if \(G\) is not a DAG. [10 points]

**Solution:**

(a) Clearly, the calculation of the function \(g(v)\) for a vertex \(v\) is dependent on the \(g(w)\) for all vertices \(w\) for which \((v,w)\) is an edge. This problem can be solved by finding a topological ordering of graph \(G\) since a topological sort gives a linear ordering of vertices such that for every directed edge \((v,w)\), vertex \(v\) comes before \(w\) in the ordering. Then, compute the function \(g\) for the vertices in that order.

Input: adjlist(G), f
Output: g
Algorithm:

Find topological ordering (T) of the graph $G$. for all v in T (in order from sink to source):

\[ \text{sum} = f(v) \]

for all w in adjlist(v):

\[ \text{sum} = \text{sum} - g(w) \]

assign \(g(v) = \text{sum}\)

Run time Complexity: DFS is a linear time algorithm. Thus, finding topological order takes \(O(|V| + |E|)\). Computing function \(g\) for all vertices in the same order also takes \(O(|V| + |E|)\).

(b) If \(G\) is not a DAG, that means it has cycles. Topological sort is only valid for DAGs. Circular dependency will result in the infinite loop in the algorithm above and will not result in the correct computation of the function \(g\).

Example: Consider the following graph with 2 vertices, \(A\) and \(B\) with a cycle. Let \(f(A) = 0\) and \(f(B) = 1\) for some function \(f\).

Then, for vertex \(A\) : \(g(A) + g(B) = 0\) and for vertex \(B\) : \(g(B) + g(A) = 1\), which is not possible at the same time.
**Question 3** (Shopping Trip, 40 points). Jade lives in a city whose roads are described by a directed graph $G$. She wants to go on a shopping trip where she visits every vertex in some subset $S$ of $V$. She wants to find an efficient algorithm to determine whether or not it is possible to do this.

(a) If $G$ is a DAG, use a topological sort to figure out what order the vertices of $S$ must be visited in. [10 points]

(b) Use the above to find a linear time algorithm to solve Jade’s problem when $G$ is a DAG. Hint: you might need to remove edges from $G$ that skip stores that she needs to visit. [15 points]

(c) Find a linear time algorithm to solve Jade’s problem in an arbitrary directed graph, $G$. Hint: first find the strongly connected components and reduce to the case of a DAG. [15 points]

**Solution:**

(a) Topological sorting of the graph $G$ gives us the ordering of vertices in which edge is only in the forward direction. The vertices in set $S$ must be visited in the order which they appear in the topological ordering of graph $G$.

(b) We will remove edges $(v, w) \in E$ if there is a vertex $u \in S$ that comes between $v, w$ in the topological ordering. Jade should not use such edges during the shopping trip because such edges would skip vertices like $u \in S$ that should be visited by Jade. Such edge removal can be done in linear time by scanning through the vertex set $V$ in topological order and for each outgoing edge $(v, w)$ of $v \in V$ checking if the $w$ comes after the next vertex in $S$ (in topological order). Finally, we would run DFS from the start vertex and check if all the vertices in $S$ are visited.

(c) Create a metagraph $G' = (V', E')$ from $G$. In $v \in V'$, we will store all possible vertex in $S$ that can be visited. Now, the problem reduces to part (b).