"CS 374" Spring 2017 — Homework 8 Solutions

1. Let $G = (V, E)$ be a connected directed graph with non-negative edge weights, let s and t be vertices of G, and let $H(V_H, E_H)$ be a subgraph of G obtained by deleting some edges. Suppose we want to reinsert exactly one edge from G back into H, so that the shortest path from s to t in the resulting graph is as short as possible. Describe and analyze an algorithm that chooses the best edge to reinsert. Ideally the running time of your algorithm should be *asymptotically* the same as that of running Dijkstra's algorithm.

   Solution: Notation:

   - $|V| = n$
   - $|E| = m$
   - $E' = E \backslash E_H$

   For the sake of simplicity, assume $V = \{1, 2, ..., n\}$. The algorithm has four steps:

   (1) Compute the shortest-path distance in H from s to every other vertex using Dijkstra's algorithm, in $O(|E_H| + |V_H| \log |V_H|) = O(m + n \log n)$ time. Store these values in an array $\text{dist}_s[1..n]$

   (2) Compute the shortest path distance in H^{rev} from t to every other vertex using Dijkstra's algorithm, again in $O(m + n \log n)$ time. Store these values in an array $\text{dist}_t[1..n]$

   (3) Find the edge $u \to v \in E'$ that minimizes the function $\text{dist}_s[u]$ + l$(u \to v)$ + $\text{dist}_t[v]$, by brute force, in O(m) time.

   (4) Finally,if $\text{dist}_s[u] + \ell(u \to v) + \text{dist}_t[v] < \text{dist}_s[t]$,return an error condition `None`; otherwise, return the edge $u \to v$. The algorithm runs in $O(m + n \log n)$ time, as required.

   Proof of correctness: Since Dijkstra's is used without any modification, it correctly computes the shortest paths from s and t to all $v \in V$. We have two cases to consider:

   - No "best" edge exists: This implies that the shortest path between s and t is H is indeed the shortest path between s and t in G as well. We check for this condition in (4), in which case we correctly return that no edge needs to be reinserted.
   - Otherwise: In this case, we want to choose the edge($u \to v$) which minimizes the shortest path(P) distance between s and t. Let $P = s, x_1, x_2, .., x_i, u, v, x_{i+1} ... t$. Thus minimizing the length of P is the same as minimizing the function dist(s,u) + len(u,v) + dist(v,t), where dist(a,b) represents the length of the path from a to b. However, since distance from a to b is the same as distance from b to a, this reduces to minimizing the function dist(s,u) + len(u,v) + dist(t,u) which is essentially $\text{dist}_s[u]$ + $\ell(u \to v)$ + $\text{dist}_t[v]$ in our algorithm.

   ■

Rubric: Rubric: 10 points

- 1 for the formula $dist(s,u) + l(u \to v) + dist(v,t)$
- 2 for justification of the formula usage
- 5 for algorithm details
- 2 for time analysis

Max 5 points for the obvious $O(kmlogn)$-time(or another slower) algorithm.

2. **Solution:** The problem is equivalent to finding the maximum number of distinct nodes that can be reached in any walk from s in graph G. For a general graph G, let G^{SCC} be the SCC metagraph for G, and S_q be the SCC containing s. The algorithm follows.

   (a) Create G^{SCC} for G.

   (b) For each node S in G^{SCC} corresponding to a SCC S of G, assign a weight $w(S) = |S|$.

   (c) Find a longest node-weighted path from S_q in G^{SCC}.

   (d) Output yes iff the length of this path is at least k.

   Correctness: Consider any walk p in G starting from s. On one hand, p corresponds to the path p^{SCC} in G^{SCC} which is the sequence of SCC's of G that p enters. Also observe that p can enter each SCC at most once. On the other hand, to reach most distinct nodes, whenever p enters any SCC S, it would visit all the $w(S)$ nodes in S. Thus, the maximum number of distinct nodes that any walk in G can reach must be equal to the length of some node-weighted path in G_{SCC}, which can be maximized using dynamic programming.

   Time complexity: Let n and m be the number of nodes and edges in G. Creating G^{SCC} takes $O(n+m)$ time. Find the longest node-weighted path using dynamic programming on G^{SCC}, together with the topological ordering as a preprocessing step, takes time linear to size of G^{SCC} which is at most $O(n+m)$. Thus, the overall time complexity of the algorithm is $O(n+m)$. ■

   Rubric: 1 point for the case that G is strongly connected; 2 points for solving the case that G is a DAG; 4 points for solving the general case. 2 points for justification of correctness. 1 point for time analysis.

3. Let $G = (V, E)$ a directed graph with non-negative edge lengths. Let $R \subset E$ and $B \subset E$ be red and blue edges (the rest are not colored). Given s, t and integers h_r and h_b describe an efficient algorithm to find the length of a shortest s-t path that contains at most h_r red edges and at most h_b blue edges.

Solution: This solution explicitly uses dynamic programming.

Let $d(v, i, j, k)$ to be the minimum distance of the all paths from s to v with at most i total edges, at most j red edges, and at most k blue edges. Let G' to be the graph obtained by removing all colored edges in G. Then $d(v, i, 0, 0)$ is the shortest path distance from s to v with at most i total edges in G'. Using Bellman-Ford algorithm, we can compute $d(v, i, 0, 0)$ for all $v \in V$ and $0 \leq i \leq n-1$ in $O(mn)$ time. For $i < 0$, $d(v, i, 0, 0) = \infty$.

For $j, k \geq 0$ and at least of them is not equal to 0.

$$d(v,i,j,k) = \min \begin{cases} d(v, i-1, j, k) \\ d(v, i, j-1, k) \\ d(v, i, j, k-1) \\ \min_{(u,v)\in R} d(u, i-1, j-1, k) + l(u, v) \\ \min_{(u,v)\in B} d(u, i-1, j, k-1) + l(u, v) \\ \min_{(u,v)\in E\setminus(R\cup B)} d(u, i-1, j, k) + l(u, v) \end{cases}$$

where $l(u, v)$ is the length of edge (u, v). And we wish to compute for $d(v, n-1, h_r, h_b)$ for all $v \in V$. There are at most $O(n^2 h_r h_b)$ subproblems. To compute $d(v, i, j, k)$ based on the recursive definition, we need time that is proportional to the in-degree of v for each $v \in V$. The total sum of all in-degree of $v \in V$ is at most $O(m)$. Thus, the overall runtime is $O(mnh_r h_b) + O(mn) = O(mnh_r h_b)$.

All length values can be stored in a 4-D array. Space for this array is $O(n^2 h_r h_b)$. $d(t, n-1, h_r, h_b)$ will return the result to the problem. Evaluation order involves four layers nested loops. First layer loop goes through vertex from t to its adjacent ancestors. Second layer loop goes through i from $n-1$ to 0. Third layer loop goes through j from h_r to 0 and fourth layer loop goes through k from h_b to 0.

■

Solution: This solution uses the generalized technique from HW7 Q1.

New graph: Construct a new directed graph $G' = (V', E')$. Induce a subgraph H with vertices and noncolored edges from G. Duplicate H for $h_r \times h_b$ times. Each copy H can be indexed by $H_{i,j} = (V_{i,j}, E_{i,j})$ which indicates at most i red edges and j blue edges can be used if the shortest path ends in this $H_{i,j}$. Adjacent graphs H such as $H_{i,j}$ and $H_{i-1,j}$ are linked with red edges. And $H_{i,j}$ and $H_{i,j-1}$ are linked by blue edges. Then, use super source s' and super sink t' to connect all s's and t's.

$$\begin{aligned} V' &= \{v_{i,j} \mid \forall v \in V, i = 0, \cdots, h_r, j = 0, \cdots, h_b\} \cup \{s', t'\} \\ E' &= \{(u_{i,j}, v_{i,j}) \mid (u, v) \in E\setminus(R \cup B), i = 0, \cdots, h_r, j = 0, \cdots, h_b\} \\ &\quad \cup \{(u_{i,j}, v_{i+1,j}) \mid (u, v) \in R, i = 0, \cdots, h_r - 1, j = 0, \cdots, h_b\} \\ &\quad \cup \{(u_{i,j}, v_{i,j+1}) \mid (u, v) \in B, i = 0, \cdots, h_r, j = 0, \cdots, h_b - 1\} \\ &\quad \cup \{(s', s_{i,j}) \mid i = 0, \cdots, h_r, j = 0, \cdots, h_b\} \\ &\quad \cup \{(t_{i,j}, t') \mid i = 0, \cdots, h_r, j = 0, \cdots, h_b\} \end{aligned}$$

Reduced graph problem: The original problem is equivalent to find the shortest length path from s' to t'.

Algorithm: Apply Dijkstra's algorithm directly since non-negative edge lengths.

Original problem solution: Given an output in the form $s' \to s_{i_0,j_0} \to \cdots \to t_{i_k,j_k} \to t'$, the solution for the original problem is path $v_0, \cdots, v_k$.

Runtime: Graph construction takes $O(h_r h_b(|V|+|E|)) = O(h_r h_b(m+n)) = O(m^2(m+n))$ time. Dijkstra's algorithm takes $O(|E|+|V|\log|V|) = O(h_r h_b(m+n\log h_r h_b n)) = O(m^2(m+n\log m^2 n))$. Generate original path takes at most $O(|E|) = O(m)$ time. Therefore, total runtime is $O(m^2(m+n\log m^2 n)) = O(m^3 + m^2 n \log mn)$.

Proof of correctness:

Claim: A shortest path p' from s' to t' in G' is equivalent to a shortest path p from s to t that contains at most h_r red edges and h_b blue edges.

Proof:

($\Rightarrow$) Assume that p is not a shortest path that contains at most h_r red edges and h_b blue edges. Then, there must exist another path q from s to t and satisfies the requirement. Then, we can construct a new shortest path q' in G' which is shorter than p'. Thus, it is a contradiction.

($\Leftarrow$) Assume that p' is not the shortest path in G', there exists q' which is the shortest path in G'. Then, we can construct q which is shorter than p. Thus, it is a contradiction.

■

Rubric: 10 points

Solution 1

- Standard DP rubric

Solution 2

- 3 points for correct directed graph construction (vertices and edges).
  - -1 point if directed is not mentioned
- 1 point for stating the correct graph problem.
- 2 points for applying the correct algorithm.
- 2 points for justification of the correctness
- 2 points for runtime analysis in terms of input parameters.
  - -1 point if graph build time is not included