Illinois Institute of Technology
Department of Computer Science

Solutions to Homework Assignment 8

CS 430 Introduction to Algorithms
Spring Semester, 2017

Solution:

1. a) In a looped tree, there is $O(E) = O(2V)$. So, the running time should be $O(V \log V)$.

   b) There are two cases here to find a shortest path from node u to v.

   First, v is a descendant of u. In this case, we only need to find the path from u to v in the tree, which takes $O(V)$. Second, v is not a descendant of u. In this case, we need to find the shortest path from u to a leaf, then back around to the root and down to v again. To find the shortest path from u to a leaf, we apply modified BFS with monitoring the weight cost of the link back to u. The time cost is $O(V)$ since $O(E) = O(2V)$ in the looped tree.

2. If we find a negative edge to a vertex v that is already out of priority queue (that is vertices for which a shortest path length has already been calculated assuming there were no negative edges connecting to it), then we should calculate new shortest path through the negative edge and update the $v.d$ value and again push this new vertex to the priority queue. Therefore, we need to modify the RELAX to allow visiting a vertex more than once as shown in Algorithm 1.

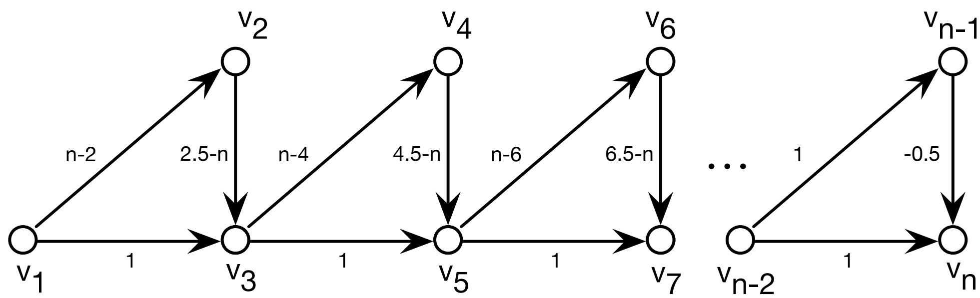
Algorithm 1: RELAX-NEGATIVE(u, v, w)

if *if* $v.d > u.d + w(u, v)$ **then**
  $v.d = u.d + w(u, v)$
  $v.\pi = u$
  if $v \notin Q$ **then**
    INSERT(Q, v)

However, the modified Dijkstra's algorithm can take exponential time in the worst case. Specifically, we can construct a weighted graph of n vertices with negative weights, such that Dijkstras algorithm calls $\Theta(2^{n/2})$ RELAX. For example, we can construct the graph with negative weights as follows. Let $T(n)$ be the number of relaxation on $v_1, \cdots v_n$. Then we can build a recurrence as

$$T(n) = 2 + T(n-2) + 1 + T(n-2) = 2T(n-2) + 3 = \Theta(2^{n/2}),$$

where the first two relaxations are for (v_1, v_2) and (v_1, v_3), $T(n-2)$ relaxations are for $v_3, \cdots, v_n$, one relaxation for (v_2, v_3) and $T(n-2)$ relaxations are for $v_3, \cdots, v_n$. Note that $v_1.d < v_3.d \cdots < v_{n-2}.d < v_{n-1}.d < v_{n-3}.d < \cdots < v_2.d$ during the execution of the algorithm.



Algorithm 2: FLOYD-WARSHALL(W)

1 $n = W.rows$
2 $D^0 = W$
3 **for** $k = 1$ *to* n **do**
4   let $D^k = d_{ij}^k$ be a new $n \times n$ matrix
5   **for** $i = 1$ *to* n **do**
6   **for** $j = 1$ *to* n **do**
7             $d_{ij}^k = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$
8   **if** $i == j$ *and* $d_{ij}^k < 0$ **then**
9                 $d_{ij}^k = -\infty$
10 **return** D^n

3. Notice that the Floyd-Warshall algorithm computes the weight of the path from a node to itself. This weight will be updated if and only if there is a negative circle. Otherwise $d_{ii} = 0$ will be the minimum for any node i. Therefore, we just need to modify the Floyd-Warshall algorithm by checking each update of d_{ii} . If any update changes d_{ii} to be smaller than 0, there exists a negative weighted cycle and we set $d_{ii} = -\infty$, and any path using that cycle will result in $-\infty$. Algorithm 2 shows the modified algorithm. Checking if $d_{ii} < 0$ takes constant time (Line 8-10) and the running time will remain to be $\Theta(n^3)$.