# Algorithms Fall 2016 Homework 5 Solution

October 19, 2016

## PROBLEM 1

The running time of Dijkstra's algorithm depends on the implementation of EXTRACT-MIN function. Since any shortest path contains at most $|V|-1$ edges and we already know the range of the weight of each edge to be in $0,1,...,W$, for any vertex $v$ that is reachable from $s$, the weight of the path is at most $(|V|-1)W$. The basic idea of modifying the EXTRACT-MIN function is to create $(|V|-1)W+2$ cells $C[0],C[1],...,C[(|V|-1)W],C[\infty]$ and put each vertex $v \in G.V$ to the cell $C[v.d]$, each time we pick a vertex from the first non-empty cell starting from $C[0]$ to $C[\infty]$ and then renew the $d$ value for the unpicked vertices and re-assign them to corresponding cells.
To implement this data structure, we use circular double-linked list for each cell to contain the vertices. We also need a array that holds the pointers to the vertices such that we can find the vertex's position in $O(1)$ time. The pseudo-code is as follows:

```
DIJKSTRA-MODIFIED($G, w, s$)
1  Let $C[0],C[1],...,C[(|V|-1)W],C[\infty]$ be new empty doubly linked lists
2  Let $ptr[1...V]$ be an array of pointers
3  **for** each vertex $v \in G.V$
4          **if** $v == s$
5                  $C[0].insert(ptr[s])$
6          **else**
7                  $C[\infty].insert(ptr[v])$
8  **for** $i = 1 to (V-1)W$
9          **for** each vertex $u \in C[i]$
10                 **for** each vertex $v \in G.adj[u]$
11                         **if** $v.d > u.d + w(u,v)$
12                                 $v.d = u.d + w(u,v)$
13                                 $v.\pi = u$
14                                 $vnode = ptr[v]$
15                                 remove $vnode$ from the list and add it to $C[v.d]$
```

Since we have a $ptr$ array to hold the pointers to all the vertices, the time cost of removing a vertex from and adding a vertex to a cell would be $O(1)$. The outer for loop have $(V-1)W+1$ iterations, so the time cost for scanning the cells is $O((V-1)W)$. Each vertex will be visited at most once, for edge $(u,v)$, it will be visited at most once when it is relaxed. The total running time is $O((V-1)W+V+E) = O(WV+E)$.

## PROBLEM 2

**a)** Suppose we have three d-dimensional boxes $X = (x_1,x_2,...,x_d), Y = (y_1,y_2,...,y_d), Z = (z_1,z_2,...,z_d)$ and $X$ nests within $Y$ and $Y$ nests within $Z$. To prove that the nesting relation is transitive, we need to prove that $X$ nests within $Z$. By the definition, there exists two permutations $\pi_1$ and $\pi_2$ such that:

$$x_{\pi_1(1)} < y_1, x_{\pi_1(2)} < y_2, ..., x_{\pi_1(d)} < y_d$$

$$y_{\pi_1(1)} < z_1, y_{\pi_1(2)} < z_2, ..., y_{\pi_1(d)} < z_d$$

We can construct new permutation:

$$\pi_3 : \pi_3(i) = \pi_1(\pi_2(i)), \forall i = 1,2,...d$$

Then we have $x_{\pi_3(i)} = x_{\pi_1(\pi_2(i))} < y_{\pi_2(i)} < z_i$ for $i = 1,2,...,d$. That means $X$ nests within $Z$. Therefore the transitivity of the nesting relation is proved.
**b)** Suppose we have two d-dimensional boxes $X = (x_1, x_2, ..., x_d)$, $Y = (y_1, y_2, ..., y_d)$. We can determine whether $X$ nests within $Y$ by sorting the dimensions of $X$ and $Y$ in increasing order and checking if $x_i < y_i$ for all $i = 1,2...d$. The pseudo-code is as follows:

```
NEST(X, Y)
1  if $X.length \neq Y.length$
2      return FALSE
3  $X = QUICK-SORT(X, 1, X.length)$
4  $Y = QUICK-SORT(Y, 1, Y.length)$
5  for $i = 1$ to $d$
6      if $X[i] \geq Y[i]$
7          return FALSE
8  return TRUE
```

**c)** We can construct a directed acyclic graph to represent the nesting relations between boxes and convert this problem to a shortest path problem. Firstly we construct a graph $G = (V, E)$ and a weight function $w$, by following steps:

1. Initially set $V$ and $E$ to be empty sets.
2. For each box $B_i$ we add vertex $v_i$ into $V$.
3. For each pair $(B_i, B_j), i \neq j, i = 1, ..., d, j = 1, ..., d$, we add a edge $(v_i, v_j)$ to $E$ only if $B_i$ nests within $B_j$ and let $w(v_i, v_j) = 1$.
4. Add a source vertex $s$ to $V$, and add edge $(s, v_i)$ for the vertices with $in-degree = 0$ for $i = 1,2,...,d$.
5. Add a sink vertex $t$ to $V$, and add edge $(v_i, t)$ with $out-degree = 0$ for $i = 1,2,...,d$.

Since the nesting relation is transitive and anti-reflexive, the graph $G$ we build is a directed acyclic graph. The we apply the DAG-SHORTEST-PATHS function on $G$ and construct a shortest path from $s$ to $t$. The sequence of boxes on this path between $s$ and $t$ (not included) is the longest sequence we want to find.
The running time of this algorithm consists several steps as follows:

1. Sorting the dimensions of $n$ d-dimensional boxes. This step takes $O(nd \lg d)$
2. Comparing $n$ boxes to each other. Determining nesting relation between two boxes whose dimensions are already sorted takes $O(d)$ time, there this step take $O(n^2 d)$ time.
3. Constructing the graph $G$, adding vertices takes $O(n)$ and adding edges takes $O(n^2)$. This step takes $O(n^2)$.
4. Running DAG-SHORTEST-PATHS function takes $O(V + E) = O(n^2)$ time.

The total running time is $O(nd \lg d + n^2 d + n^2 + n^2) = O(n^2 d + nd \lg d)$.

## PROBLEM 3

This problem is similar to Problem 5, in the output of Floyd-Warshall algorithm $D^{(n)} = (d_{ij}^{(n)})$, $d_{ij}^{(n)}$ represents the weight of a shortest path from $i$ to $j$ consisting of at most $n-1$ edges. So if there exists negative-weight cycle in the given graph which is consisting of at most $n-1$ edges, the value $d_{ii}^{n}$ for all the vertices $i$ that is in the cycle would be negative. Therefore, we can detect if there is a negative-cycle in the graph by simply checking if there are negative values in the diagonal elements in the output of Floyd-Warshall algorithm.

## PROBLEM 4

By lemma 25.1, the total weight of any cycles will be unchanged after reweighting. This is proved in the last part of the proof. This means that in the reweighted graph, the 0-weight cycles still have the same total weight as before, which is 0. But according to the properties, each of the edge in the cycle has nonnegative weight, it has to be the case that $\hat{w}(u, v) = 0$ for every edge $(u, v)$ in $c$.