Algorithms - HW 5

Kiran Shettar

10/17/16

Problem 1 24.3-8 (pg 664)

Dijkstra (G, w, s)

1. Initialize Single Source (G, s)
2. S ← { }
3. Q ← V[G]
4. while Q != { }
   a. do u ← EXTRA-MIN(Q)
   b. S ← S ∪ {u}
   c. for each vertex v ∈ Adj [v]
      ∴ Do RELAX (u, v, w)

The running time of Dijkstra's algorithm depends on the implementation of min-priority queue. In this algorithm, we process those vertices close to the source vertex first. Because each edge has atmost weight 'w', we know the maximum possible value of the longest path in the graph is (v-1) w.

We can prioritize the vertices based on their d [] values.

The queue consists of $(v-1)w$ buckets. Vertex 'v' can be found in bucket $d[v]$. Since all other than source have $d[v]$ value between 1 & $(v-1)w$, so they can be found in buckets. $1 \ldots (v-1)w$.

If 's' is source vertex then $d[s]=0$. So, 's' can be found in bucket 0. Line 1 of algo ensures that for all vertices 'v' other than root, $d[v]$ is initialized to ∞.

After initializing all of the vertices, we have seen the buckets from 0 to $(v-1)w$. When a non-empty bucket is encountered, the first vertex is removed, and all the adjacent vertices are relaxed. This step is repeated until we have reached the end of the queue in $O(vw)$ time. Since we relax a total of E edges, the total running time for this algorithm is $O(vw+E)$

Problem 3 25.2-6 (pg 700)

Firstly we have to check the main diagonal entries of the result matrix for a negative value.

If $d_{ii}^{(n)} < 0$ where i = vertex then we can say that there's a negative weight cycle. So we can say that there exists a cycle with negative weight.

A negative weight cycle will always contain vertex 'n' or it'll not when it's containing vertex i. Then the value $d_{nn}^{(n-1)}$ will be negative, since the cycle is starting & ending in vertex 'n'. And it'll not include vertex 'n' as an intermidiate vertex.

OR

We can run the FLOYD-WARSHALL algorithm so that it'll run for one extra iterations. ~~So~~ And we can check if the value of 'd' changes. If the shortest path costs are cheaper, then there'll be negative cycle. And, if the 'd' value doesn't change, then we can say that there's no negative cycle. And the algorithm gives correct short paths.

24-2 (pg 678)

(a) Let us consider boxes with dimension
$x = (x_1, \dots x_d)$ & $y = (y_1, \dots\dots y_d)$ &
$z = (z_1, z_2 \dots z_d)$

Consider $x_{\pi(i)} < y_i$ for $i = 1, \dots d$ &
$y_{\pi'(i)} < y_i$ for $i = 1, \dots d$ so that

x will lie in y & y will lie in z.

Then, we should do $\pi''(i) = \pi'(\pi(i))$
Then, for $i = 1, \dots n$ we have

$$x_{\pi''(i)} = x_{\pi'(\pi(i))} \leq y_{\pi'(i)} < z_i$$

Hence, we can say that
'x' nests inside z.

(b) Now, we have to sort the boxes according to their dimension from longest to shortest. A box 'X' with sorted dimensions $(x_1, x_2 \dots x_d)$ nests inside a box 'y' with sorted dimensions $(y_1, \dots y_d)$ if & only if $x_i < y_i$ for $i = 1, 2 \dots d$

Hence, we can get to know that the sorting can be done in $O(d \lg d)$ time, and the test for nesting can be done in $O(d)$ time, and so the algorithm runs in $O(d \lg d)$ time.

This algorithm will work because a d-dimesional box can be oriented so that every permutation of its dimensions is possible.

(c) Now, consider $G = (V, E)$ which is intialized, where each vertex v_i corresponds to box B_i & $(v_i, v_j) \in E$ if & only if box B_i nests inside box B_j. ~~Graph G is~~ The time to construct this is $O(dn^2 + dn \lg d)$, for comparing each of the $\binom{n}{2}$ pairs of boxes after ~~boxes~~ sorting the dimensions of each.

Add a supersource vertex 's' & a supersink vertex t to 'G', and add edges (s, v_i) for all the vertices v_i with in degree 0 & (v_j, t) for all vertices v_j without degree 0. Call the resulting G', which takes $O(n)$ time.

find longest path from s to t in G'. This will help in nesting boxes. Hence the time to find the longest path is $O(n^2)$, since G' has n+2 vertices & $O(n^2)$ edges.

Over all this algorithm has a running time $O(dn^2 + dn \lg d)$ //

Problem 4 25.3-5 : Consider a '0' weight cycle a-b-c in a directed graph 'G': $w(a,b)+w(b,c)+w(c,a)=0$. Now add a vertex 's' to the graph 'G'. such that there's an edge be/n s to every vertex in the graph 'G'.

$\hat{w}(a,b) = w(a,b) + (-w(a,b))$
$\hat{w}(a,b) = w(a,b) - w(a,b)$
$\hat{w}(a,b) = 0$

Similarly,
$\hat{w}(b,c) = 0$
$\hat{w}(c,a) = 0$

$\therefore$ If any graph 'G' has a 0-weighted cycle, the new weights of every edge in the cycle is '0'.