

COT 5405 - Spring 2009

Homework #5 Solutions

March 23, 2009

Grading Policy:

- Please contact TA Chunchun Zhao by email (cot5405sp09@cise.ufl.edu) or in his office hours for any grading issues.
- Maximum score is 100 points.
- Each completed question worth 10 points.
- Partial credit is given if you don't answer a question completely.
- Problem 4 is graded for 60 points.
- Please notify the TA if you find anything wrong in this solution.

Problem 1 (Problem 25.1-5 in page 628)

Solution:

The all-pairs shortest-paths algorithm in Section 25.1 computes

$$L^{(n-1)} = W^{n-1} = L^{(0)} \cdot W^{n-1}$$

where W is the weight matrix and $L^{(0)}$ is the identity matrix. That is, the entry in the i^{th} row and j^{th} column of the matrix "product" is the shortest-path distance from vertex i to vertex j , and row i of the product is the solution to the single-source shortest-paths problem for vertex i .

Notice that in a matrix "product" $C = A \cdot B$, the i^{th} row of C is the i^{th} row of A "multiplied" by B . Since all we want is the i^{th} row of C , we never need more than the i^{th} row of A .

Thus the solution to the single-source shortest-paths from vertex i is $L_i^{(0)} \cdot W^{n-1}$, where $L_i^{(0)}$ is the i^{th} row of $L^{(0)}$ – a vector whose i^{th} entry is 0 and whose other entries are ∞ .

Doing the above “multiplications” starting from the left is essentially the same as the BELLMAN-FORD algorithm. The vector corresponds to the d values in BELLMAN-FORD – the shortest-path estimates from the source to each vertex.

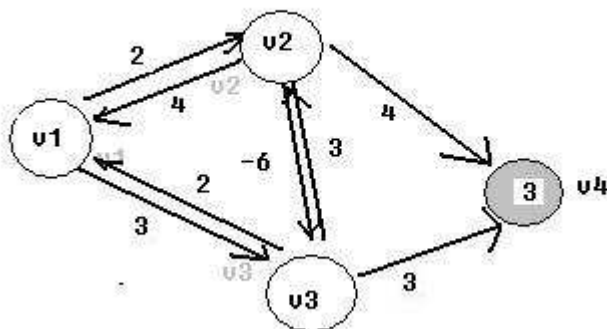
- The vector is initially 0 for the source and ∞ for all other vertices, the same as the values set up for d by INITIALIZE-SINGLE-SOURCE.
- Each “multiplication” of the current vector by W relaxes all edges just as BELLMAN-FORD does. That is, a distance estimate in the row, say the distance to v , is updated to the smaller estimate, if any, formed by adding some $w(u, v)$ to the current estimate of the distance to u .
- The relaxation/multiplication is done $n - 1$ times.

Problem 2 (Exercise 25.3-6) page 640

Solution:

The problem arises when we choose a vertex s , from which no negative-weight cycles are reachable. However, note that this is possible in directed graphs which are not strongly connected.

For example, consider the following graph which has four vertices, v_1, \dots, v_4 with a negative-weight cycle v_2, v_3 . Also, there are two more edges $(v_2, v_4), (v_3, v_4)$. Now, if $s = v_4$, clearly Bellman Ford won't be able to detect the negative-weight cycle. Obviously, this problem is alleviated when the graph is strongly connected.



Problem 3 (Problem 26-4 in page 694)

1) Solution:

a) First, observe that if the residual capacity of (u,v) (after executing Ford-Fulkerson) is still non-zero, increasing its capacity further does not change the flow.

b) Now suppose the residual flow were zero.

In the residual graph, change residual flow of (u,v) from 0 to 1

c) check to see if there are any augmenting paths, if so augment the flow and output.

Using BFS, this takes $O(V+E)$. Augmenting flow takes $O(E)$.

2) Solution:

a) if the residual capacity of (u,v) were non-zero, decreasing it does not change the flow.

b) If not, then decreasing the capacity with the current flow violates the capacity constraints ((u,v) is now taking in more flow than its capacity).

i) First, find a path from s to t via (u,v) in the original flow graph.

ii) Now pump back one unit of flow from t to s. This increases the residual capacity of all edges occurring on this path by 1 (and decreases net flow by 1).

iii) Now, it is possible that augmenting paths exist in the new residual graph (this would amount to rerouting the flow, around the zero residual edge). So, check for augmenting paths in this residual graph and augment the flow, if this is the case.

This algorithm uses BFS atmost thrice and is therefore $O(V+E)$. Augmenting flow takes $O(E)$ and is done atmost twice.

Problem 4 Problem 33-1 on page 962

a) solution:

1) using Jarvis's march to find first layer, time $O(n * h)$, h is the number of points on the convex hull

2) Remove points on the first layer

3) repeating the procedure by using Jarvis's march find ith convex hull.

Time complexity : $O(\sum n_i * h_i) < O(\sum n * h_i) = O(n * \sum h_i) = O(n^2)$

b) solution:

we can prove it by contradiction

Suppose there exists an algorithm which can find convex layers in $o(n \log n)$ time. Then, this means we can find all convex layers in $o(n \log n)$ time, including the convex hull of the given input points.

Now, consider a set of non-negative numbers ($A_1, A_2, A_3, \dots, A_n$) to be sorted.

Since we already assumed that there exists an algorithm which can find all the convex layers of a given point set in $o(n \log n)$ time, then we can use this algorithm to find all the convex layers of the following set of points generated from the given set of numbers: $\{ (A_1, A_1^2), (A_2, A_2^2), (A_3, A_3^2), \dots, (A_n, A_n^2) \}$.

Note that, we only need the outmost convex layer which is the convex hull of the points, since all the points are located on the $y = x^2$ curve when $x > 0$. Every three physical (in x-increasing order) successive points are forming a convex angle in clockwise order.

If we traverse the convex hull starting from the left most point, we can get the sorted list of the given set of numbers in $o(n \log n)$ time.

Finding the convex layer takes $o(n \log n)$ time, finding the minimum x value(left most point) requires $O(n)$ time and traversing the convex hull would take $O(n)$ time making the total running time $o(n \log n)$.

However, we already know that lower bound for sorting is $n \log n$, so this is a contradiction. Therefore, such an algorithm for convex layer problem cannot exist.