Mahesh Yarasi
12/4/16

Algorithms Homework 8

1. a) To find the maximum bandwidth of a path b/w a & b, we can use a modified version of Dijkstra's algorithm. Dijkstra finds the shortest path but this modified version will find the longest path from a source. Initialy the cost b/w vertices is the min but now it will be the max bandwidth. So, we traverse the graph the same way but now we initialize the source to ∞ and all the other nodes to -1. Then we find the minimum of the source and the weight of the path => $\min(u, w(u,v))$. Then we find the max $(\min(u, w(u,v)), v)$. - this process is when we relax the edges. This process is basically extract_max instead of extract_min. The runtime is the same as if we used min queue => Initialization is $\Theta(V)$, extracting (V) times is $\Theta(V \log V)$. Relaxing on all the edges is $\Theta(E \log V)$ times. This results in an overall runtime of $\Theta(E \log V)$.

b) We can modify Bellman-Ford by remembering if v (destination) is relaxed or not. If v is relaxed, then we see if v is updated. If v was not updated we terminate. Since the greatest number of edges on any shortest path from the source is m, then after m iterations, every vertex has recieved its shortest-path weight $w(u,v)$. After m iterations, no weights will change. So, no weights will change in the $(m+1)^{st}$ iteration. Since We cannot know in advance, we can't make the algorithm iterate exactly m times and terminate. But if we make the algorithm stop when nothing changes any more, It will stop after m+1 iterations.

2. a) First we create an array that holds all the denominations of the coins, coins[].

Then create an empty array to hold the min # of coins to make an amount of change. This will be initialized to be size (k+1), so that the k^th element in the array is the min # of coins to make k cents Each element in this array is initialized to ∞. Fill out the table by looping through the list of coins, in decreasing order. The first coin denomination less than k, refer to the number of coins to make k subtracted by the denomination. then add 1 for that coin. ~~There~~ For example, if we had a 7 cent coin and a 1 cent coin, we would need 6 one cent coins for k = 6, when k = 7, refer to element k - 7, or 0 ¢, add 1 for the 7 cent coin price, vs. 7 1 cent coins. For k = 14, refer to k = 7 and add 1, for 2.

2b) First we initialize an array common[ ][ ] size $\overbrace{(M+1)*(N+1)}^{\substack{\text{M is the length A} \\ \text{N is the length B}}}$. This 2-D array will hold the common elements of A[0. . . . M] and B[0 . . . N]. The solution will be at common[M][N]. In order to have a runtime O(MN) we use a nested for loop, one that goes 0 -> M for A, and the other which goes 0 -> N for B. If either M or N is zero, they have 0 elements in common. ~~If~~ If A[M-1] == B[N-1], then common[M][N] is common[M-1][N-1]+1 since the m^{th} letter & n^{th} letter match. If not, then take either common[m-1][n] or common[m][n-1], whichever is greater since that is the # of common elements w/o the m^{th} or n^{th} element of A or B.

3. Similar to dynamic programming, run the Bellman-Ford Algoroithm on each vertex of the graph as a source. Using a Distance Matrix, store the shortest path from a source to another vertex. Bellman-Ford will detect a negative weight cycle for the source, S. If we are able to relax edges, a neg weight cycle has been found. Use the distance matrix to find the cycle and store the length of it. By running Bellman Ford on every vertex we can discover all the negative weight cycle from a vertex. Then we compare to get the min.

4. a) My algorithm calls the transform function, which loops through the nodes in tree 2. It finds the level # a particular nod is in V1 and if V1 & V2 are on the same level, we set and traverse down the array. ~~to de~~ If not we find the parent or grandparent of V1 so that ~~it~~ the node value matches V2 and ZIG or ZAG accordingly.