# EC330 HW8 Solution

Emre Ateş

December 1, 2016

1. (a) We can use a modified Dijksta's algorithm to calculate the shortest path to our destination.

```
Function Main(G = (E,V), source, dest):
    max_heap Q
    for v ∈ V do
        if v = source then
            BW[v] = ∞
            prev[v] = v
        else
            BW[v] = 0
            prev[v] = NIL
        end
        Q.add_with_priority(v, BW[v])
    end
    while Q is not empty do
        v = Q.extract_max()  /* v is the vertex with the largest bandwidth */
        for u | (v,u) ∈ E do
            newBW = min{BW[v], bandwidth(v,u)}
            if newBW > BW[u] then
                BW[u] = newBW
                prev[u] = v
            end
        end
    end
    return BW[dest]
```

   (b) We could make the algorithm stop quicker by checking if there is a change in the distance matrix at the end of every loop. If there is no change, this means that we found all of the shortest paths, and we don't need to run the algorithm to completion.

2. (a) We can calculate the minimum number of coins required for up to $K$ cents. For our algorithm, we will store each value inside an array $A$. Our base case is zero, for which we need zero coins ($A[0] = 0$). After that, for each value $k$,

the minimum number of coins required to add up to $k$ is:

$$\min(A[k-c_i])+1,\ \forall i \in [1,N]$$

When programming this solution, we must consider the case where $k-c_i<0$, which would result in an invalid array access. The algorithm is as follows:

```
Function Main(K, c_1...c_N):
    int A[K+1] = ∞
    A[0] = 0
    for k ∈ 1..K do
        min_coins = ∞
        for c_i ∈ c_1..c_N do
            if c_i > k then
                break   /* Assuming that the coins are in increasing order */
            end
            if A[k − c_i] + 1 < min_coins then
                min_coins = A[k − c_i] + 1
            end
        end
        A[k] = min_coins
    end
    return A[K]
```

(b) In order to find the longest subsequence of strings $A[M]$ and $B[N]$, we can construct an array of longest subsequences, which is $X[M][N]$. This array represents the length of the longest subsequence between the substrings $A[1..m]$ and $B[1..n]$, $m \leq M$, $n \leq N$. The base case is when both substrings have zero length, which means the longest subsequence is also has zero length ($X[0][0]=0$). After that, we can say that if $A[m]=B[n]$, we can increase the longest subsequence length up to $m-1$, $n-1$ by 1:

$$X[m][n] = \begin{cases} \max\{X[m-1][n-1]+1, X[m-1][n], X[n-1][m]\} & \text{if } A[m]=B[n] \\ \max\{X[m-1][n], X[m][n-1]\} & \text{otherwise} \end{cases}$$

```
Function Main(A, B):
    int X[M + 1][N + 1]
    X[0][*] = 0
    X[*][0] = 0
    for m ∈ 1..M do
        for n ∈ 1..N do
            if A[m] == B[n] then
                X[m][n] = max{X[m − 1][n − 1] + 1, X[m − 1][n], X[n − 1][m]}
            else
                X[m][n] = max{X[m − 1][n], X[n − 1][m]}
            end
        end
    end
    return X[M][N]
```

3. (a) We can use a modified version of the Floyd-Warshal algorithm to detect negative weight cycles.

```
Function Main(G = (E,V)):
    for v ∈ V do
        dist[v][v] = 0
    end
    for (u, v) ∈ E do
        dist[u][v] = w(u, v)
    end
    for k = 1 → |V| do
        for i = 1 → |V| do
            for j = 1 → |V| do
                if dist[i][j] > dist[i][k] + dist[k][j] then
                    dist[i][j] = dist[i][k] + dist[k][j]
                end
            end
        end
        for v ∈ V do
            if dist[v][v] < 0 then
                return k        /* k is # edges in the negative weight cycle */
            end
        end
    end
    return 0                    /* We didn't find any negative weight cycles */
```