Answer the questions below. For the questions involving both programming and analysis, please submit your code through Udacity and your analysis through T-square.

1. Let $G(V, E, c, s, t)$ be a network with integer capacities. The problem is to find an s-t cut in G of minimum capacity that has the smallest number of edges among all minimum capacity cuts of G. Show how to modify the capacities of G to create a new network $G' = (V, E, c', s, t)$ also with integer capacities such that any minimum capacity s-t cut in G' is a minimum capacity s-t cut in G with a minimum number of edges.

   Implement your algorithm here:

   https://www.udacity.com/course/viewer#!/c-ud557/l-1209378918/m-3410948546

2. Let $r = (r_1, \ldots, r_m)$ and $c = (c_1 \ldots c_n)$ be two vectors of positive integers such that for some K, $r_1 + \cdots + r_m = c_1 + \cdots + c_n = K$.
   a. Implement an $O(mnK)$ algorithm to find a matrix A, if it exists, with 0/1 entries, m rows and n columns such that it has exactly r_i ones in the ith row and exactly c_j ones in the jth column. Return None if no such A exists. Submit your answer here.

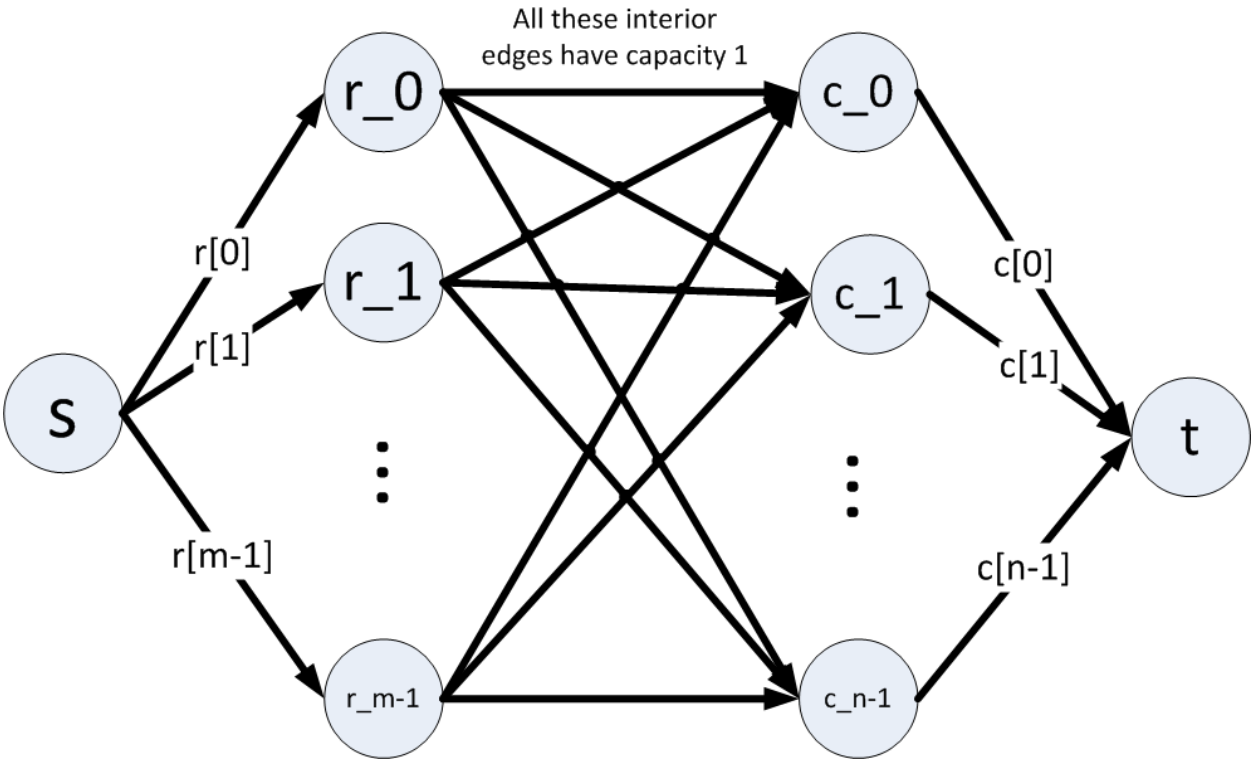
   https://www.udacity.com/course/viewer#!/c-ud557/l-1209378918/m-3373348543

   Hint: Construct an appropriate flow network and find a maximum flow in it.

   b. Prove the correctness of your algorithm.

The graph being constructed is shown below. From an intuitive standpoint, this graph encompasses everything needed to solve this problem. We have the flow from the source to the row nodes capped at the values of vector r and the flows from the column nodes to the sink are capped by the values of vector c. In the middle we have every row vertex connected to every column vertex with a maximum flow of 1 along each of these edges (i.e., a bipartite graph). In the solution, it is this flow that we are concerned about. In the output matrix, each row can contribute at most one unit of "flow" to each column. The conservation of flow both into the row vertices and out of the column vertices ensure that the matrix output will not exceed the values of the rows and columns.

However, because the maximum flow between the interior vertices can be less than that desired in the output matrix, a check is still required to see that the output matrix is correct.



c. Prove that its running time is the $O(mnK)$. Why does it run in the claimed time bound? Hint: What will the value of the augmenting flows be? How many augmenting flows will be required?

Using the graph above, since r and c both sum to K, we have K possible augmenting flows. This replaces the bound on the number of iterations in Edmonds-Karp. The other important factor, the number of edges, is dominated by the number of edges of the inner bipartite graph. There are m*n edges here.

So using the Edmonds-Karp algorithm on the unique structure of the graph results in a tighter bound of O(mnK).

3. A graph is said to be k-regular if the degree of each vertex is exactly k. Let $G = (L, R, E)$be a k-regular bipartite graph.
   a. Prove that $|L| = |R|$.

This can be proved by induction. Begin with the base case of sets L and R that have no edges between them. Add one edge from L to R, then the sum of the vertex degrees of L is equal to the sum of the vertex degrees of R, namely 1. The inductive step assumes we have vertex degree sums of n-1 for both L and R. Adding an edge from L to R adds one to each vertex degree sum, so we have that the sum of vertex degrees of both L and R are equal.

In this case with a k-regular bipartite graph, the sum of the vertex degrees of L is k*|L| and the sum of the vertex degrees of R is k*|R|. Since we know that the sum of vertex degrees of L and R are equal, we have that |L| = |R|.

b. Prove that any such G has a perfect matching. Hint: Use Frobenius-Hall.

Frobenius-Hall states that a matching of size |L| exists if and only if for every x that is a subset of L, the size of the neighborhood of x is greater than or equal to the size of x. In this case, we have that every vertex has k edges. These edges can go to the same vertex (i.e., neighbors can be shared) but there is also the possibility that the size of the neighbors of x is greater than x, so we have that the size of the neighbors of x is at least the size of x. So no matter the subset of L, the size of the neighborhood of that subset will be greater than or equal to the size of the subset (because k is greater than or equal to 1).

A perfect matching is one where L and R are the same size and the size of the matching is equal to the size of L. (In other words, every vertex in L has a match in R.) From Frobenius-Hall, we know that the size of the matching is the size of L (take the subset to be the set L and we can't have more matchings than the size of L) and from above we know that the size of L is equal to the size of R. Thus, we have a perfect matching.

I struggled to word the above, so I found the paper at this link words it much more succinctly:
http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=6&cad=rja&uact=8&sqi=2&ved=0CEEQFjAF&url=http%3A%2F%2Fwww.math.uiuc.edu%2F~kostochk%2Fmath581%2FChapter2-1.pdf&ei=rbUMVe_CMoGXNqyEgKgG&usg=AFQjCNHCQiCbak54dtjWL552X6ZrqMX2aw&sig2=Z5wdxEv6PGU0MF0Um-5zdw
Counting the edges joining subset x to N(x) yields k*|x| <= k*|N(x)| since N(x) is incident to x and perhaps others. Using the Frobenius-Hall theorem and the definition of perfect matching, we have that a k-regular bipartite graph has a perfect matching.

From my research, it looks like Frobenius proved this using the Ford-Fulkerson method we discussed in lecture. He constructed a graph with the following properties:

$$f(u,v) = \begin{cases} 1/k \; if \; (u,v) \; is \; an \; edge \\ 1 \quad if \; u = s \; or \; v = t \\ 0 \qquad otherwise \end{cases}$$

Running Ford-Fulkerson results in a positive flow along each edge between L and R, which implies there is a perfect matching.

c. Prove that the edges of a k-regular bipartite graph can be partitioned into k perfect matchings.

From above, we know that graph G has a perfect matching. Every edge of the perfect matching is incident on two vertices. Remove the edges of the matching and the resulting graph G' is a k-1 regular graph.

Applying Frobenius-Hall again shows that this graph has a perfect matching. Induction shows that this process can be repeated k times to result in k perfect matches.

4. Let $G = (V, E)$ be an undirected graph with edge costs $w: E \to Z^+$. Let P denote the set of s – t paths for two vertices s, t in G. Let C denote the set of cuts that separate s and t (each cut is a set of edges). Prove that $\underset{p \in P}{max} \underset{e \in p}{min} w(e) = \underset{c \in C}{min} \underset{e \in c}{max} w(e)$.

First I will show that a specially constructed cut can produce the left hand side of the equation above and then I will show that the remaining cuts all produce results that are removed by the min operation on the right hand side. Note: Discussions on the forum helped me formulate this answer.

Every cut must bisect every path because every path goes from s to t and every cut separates s and t. Take the edges with the minimum weight along each path and remove them. The maximum edge in this cut is the same as the maximum edge of the set of minimum path edges. So we have shown that

$$\underset{p \in P}{max} \underset{e \in p}{min} w(e) = \underset{e \in c}{max} w(e)$$

when c contains the minimum path edges.

Next, we must show that the remaining cuts all produce results that are removed during the min operation of the right hand side. Since we constructed the first cut to be the set of minimum path edges, the remaining cuts must all contain at least one edge that is not in the set of minimum path edges. This means that the maximum value of this cut will not be in the set of minimum path edges. In fact, it will be greater than any value in the set of minimum path edges. When the min operation is performed on the set C, the operation removes all of these edges and selects the value from the first cut we made, which is also the maximum of the minimum edges in the paths from s to t. Thus, we have shown that

$$\underset{p \in P}{max} \underset{e \in p}{min} w(e) = \underset{c \in C}{min} \underset{e \in c}{max} w(e).$$