Answer the questions below.

1. Let $G(V, E, c, s, t)$ be a network with integer capacities. The problem is to find a 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 $G' = (V, E, c', s, t)$ such that any minimum capacity cut in G' is a minimum capacity cut in G with the smallest number of edges.

Solution:

Here is a python implementation

```
def minedgesmincut(C,s,t):
    n = C.shape[1]

    for i in range(n):
        for j in range(n):
            if C[i][j] > 0:
                C[i][j] = n*n*C[i][j]+1

    return mincut(C,s,t)
```

Let n be the number of vertices in the graph. For any partition (A, B), let $E(A,B) = \{(a,b) \in E | a \in A, b \in B\}$. Consider two s-t cuts, (A, B) and (X, Y).
Under the definitions of c and c', we have
$c'(X,Y) - c'(A,B) = n^2(c(X,Y) - c(A,B)) + |E(X,Y)| - |E(A,B).|$

Thus, if $c(A,B) = c(X,Y)$, then $c'(X,Y) - c'(A,B) = |E(X,Y)| - |E(A,B)|$, and the cut with the few edges will have the lower capacity under c'.

On the other hand, if $c(X,Y) - c(A,B) \geq 1$, then
$c'(X,Y) - c'(A,B) \geq n^2(c(X,Y) - c(A,B)) + |E(X,Y)| - |E(A,B)| \geq n^2 - |E(A,B)| > 0.$ A cut with lower capacity under c will have a lower capacity under c' regardless of how many edges it uses.

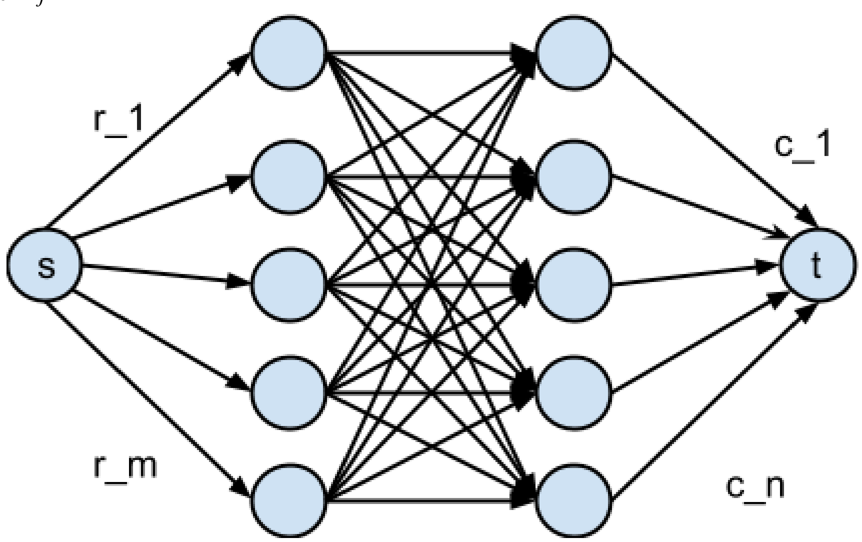
2. Let $r = (r_{1,} \dots, r_m)$ and $c = (c_1 \dots c_n)$ be two vectors of positive integers. Let $K = r_1 + \dots + r_m = c_1 + \dots + c_n.$
    a. Implement an $O(mnK)$ algorithm to find a matrix A with 0/1 entries, m rows and n columns such that it has exactly r_i ones in the ith row and exactly c_j ones in the jth column.

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

b. Prove the correctness of your algorithm in the comments.
c. Prove that its running time is the $O(mnK)$ in the comments. 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?

Solution:

a. We construct a flow network consisting of m vertices corresponding to the rows and n corresponding to the columns. Each row vertex is connected to each column vertex with an edge with capacity 1. The source is connected to every row vertex i with an edge of capacity r_i and every column vertex is connected to the sink with an edge of capacity c_j



If the maximum flow in this network has a value K , then it is possible to construct a 0/1 matrix meeting the desired constraints.

Here is a python implementation

```
def find_mtx(r,c):
    """Input: lists r and c.
       Output: A 0/1 matrix with dimensions len(r) x len(c)
               where the sum of the ith row is r[i] and the
               sum of the jth column is c[j].  If no such
               matrix exists None is returned."""
    m = len(r)
    n = len(c)

    row_sum = sum(r)
    col_sum = sum(c)
```

```
if row_sum != col_sum:
    return None
s = m+n
t = s+1

C = np.zeros((m+n+2,m+n+2))

for i in range(m):
    for j in range(n):
        C[i][m+j] = 1

#Adding edges for the sums
for i in range(m):
    C[s][i] = r[i];
for j in range(n):
    C[m+j][t] = c[j];

(flow, cut) = maxflow_mincut(C,s,t)

if flowValue(flow,s,t) != row_sum:
    return None

A = np.zeros(m,n)
for i in range(m):
    for j in range(n):
        A[i][j] = flow[i][m+j]

return A
```

b. Let A be a matrix returned by the above procedure. By the conservation of flow at all of the row vertices, the ith row of A must have r_i 1s for all $i = 1 \dots m$. By the conservation of flow at all of the column vertices, the jth column of A must have c_j 1s for all $j = 1 \dots n$. Hence, the matrix A has the desired properties.

Now let A be a 0/1 matrix satisfying the constraints. Using the notation of the given algorithm, we construct a flow over the network as follows. Let $f(s, R(i)) = r_i$ for $i \in 1 \dots m$. Let $f(C(j), t) = c_j$ for all $j \in 1 \dots n$. And let $f(R(i), C(j)) = A[i][j]$. By the constraints of the row and column sums, flow is conserved at every internal vertex, so this is a valid flow. It also saturates every edge from the source, so it must be a maximum flow and have a flow value of K. Therefore, the above algorithm will NOT return None and will instead return a matrix satisfying the constraints.

c. The flow network is constructed in $O(mn)$ time. Solving the maximum flow by Ford-Fulkerson, we have that each augmenting flow will have value 1. It follows that at most K augmenting flows will need to be found. Since each augmentation takes $O(mn)$ time via depth-first search, this gives an over time bound of $O(mnK)$ steps.

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 $|L|=|R|$.
   b. Prove that G has a perfect matching. Hint: Use Frobenius-Hall.
   c. Prove that the edges of a k-regular bipartite graph can be partitioned into k perfect matchings.

Solution:

a. Because the graph is bipartite every edge has exactly one vertex incident on L and one incident on R. Thus, $k|L| = |E| = k|R|$, and so $|L|=|R|$.

b. Let U be any subset of L, and let $N(U)$ be the neighborhood of U. Let $A = \{e \in E|\ e$ is incident on $U\}$ and let $B = \{e \in E|e$ is incident on $N(U)\}$. Note that $A \subseteq B$. Then by the fact that the graph is k-regular, $k|U| = |A| \le |B| = k|N(U)|$. Hence, by the Frobenius-Hall theorem, G has a perfect matching.

c. We argue by induction on k. In the base case, where $k=1$, the previous argument suffices. For $k>1$, there must be a perfect matching by the previous argument. Call this matching M. Subtracting the edges in M from G leaves a $(k-1)$-regular graph. Using the inductive hypothesis this graph must have a $k-1$ perfect matchings. Together with M these form a partition of the edges into k perfect matchings.

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 $\max_{p \in P} \min_{e \in p} w(e) = \min_{c \in C} \max_{e \in c} w(e)$.

   Solution:

   First, we show that $\max_{p \in P} \min_{e \in p} w(e) \le \min_{c \in C} \max_{e \in c} w(e)$. Suppose not. That is, suppose there is an s-t path p and a cut c such that $\min_{e \in p} w(e) > \max_{e \in c} w(e)$. Then $p \cap c = \emptyset$ and c does not cut the path p. This contradicts the assumption that c is a cut.

   Now, we show $\max_{p \in P} \min_{e \in p} w(e) \ge \min_{c \in C} \max_{e \in c} w(e)$. Let $w^* = \max_{p \in P} \min_{e \in p} w(e)$ and let $c^* = \{e \in E|w(e) \le w^*\}$. Note that $\max_{e \in c^*} w(e) = w^*$. To see that c^* is an s-t cut, consider an arbitrary s-t path p. By definition of w^*, we have that $w^* \geq \min_{e \in p} w(e)$, so c^* cuts the path p at it's minimum weight edge. Therefore, $\max_{p \in P} \min_{e \in p} w(e) = w^* = \max_{e \in c^*} w(e) \geq \min_{c \in C} \max_{e \in c} w(e)$.

Together the two inequalities imply the desired equality.