HW 4 #1

1. A graph with vertices a, b, c, d, e, f, g, and edges connecting them.

1/7 chance of picking each vertex, reducing the graph to a 6 cycle.

In order to end up with 5, we cannot pick edge bc or de. There is a 5/7 chance of this happening because we can pick any of the other 5 edges.

This would give us a 6 cycle and again we cannot pick any edge connecting b and c or j and e, but this now has probability 4/6 because there are only 6 edges.

Now we have a 5 cycle and probability is 3/5. Then a 4 cycle is 2/4. And finally for the last pick, there is only a 1/3 probability that we do not reduce an edge that connects b and c or j and e.

So the probability of the algorithm returning 5 is (5/7)(4/6)(3/5)(2/4)(1/3) which equals 1/21.

There are 7 vertices and by cutting through any 2 of them you will have a min-cut. This means there are (7 choose 2) = 21 min-cuts.
The cut \( \mathcal{E}_3 \) is a max-cut of size \( 6 \).

The probability is \( \frac{1}{4} \) because there are 2 min-cuts and each one has the same minimal size of 3. So one of them will always be found by the contraction algorithm.
We can use Friedvalds algorithm to determine if they are inverses. If they multiply to the identity matrix then they are inverses of each other.

(i) We can use the following matrices:

\[ A = \begin{bmatrix} 1 & 0 \\ 0 & 15 \end{bmatrix} \]
\[ B = \begin{bmatrix} 1 & 0 \\ 0 & 6 \end{bmatrix} \]

We can use the following vector and we find we got a false positive.

\[ v = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \]

Now, applying Friedvalds Algorithm, we get

\[ \begin{bmatrix} 1 & 0 \\ 0 & 15 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \]

We can see that \( AB \neq I \) but the algorithm still comes up as true for this case. This case has a 0.5 probability of a false positive because it is completely dependent on the second entry of the test vector. If it is a 1, then it will fail. If it is a 0 then it will pass.

(ii) Since this is simply Friedvalds Algorithm, we know that the error for \( k \) iterations is less than \( \frac{1}{2^k} \). This was done in the previous homework. My work from the previous homework is as follows where \( d_{ij} \) is the element of \( D \) at row \( i \) and column \( j \).

We can say \( AB = C \) when \( AB - C = 0 \). For this algorithm we can say \( A(BV) - CV = V_0 \) where \( V_0 \) is the zero vector. Let the matrix \( D \) be \( AB - C = D \) where the entries of \( D \) are denoted \( d_{ij} \). Since we know that if \( AB = C \) that the algorithm will always output “yes”, \( d_{ij} \) for all \( i \) and \( j \) will equal 0 (all entries of \( D \) will be 0). We now need to look at the case of \( AB \neq C \). In this case \( d_{ij} \) for some \( i \) and \( j \) must be nonzero. We know \( A(BV) - CV = DV \). Let the entries of \( V \) be denoted \( v_i \). If \( d_{ij} \neq 0 \) but \( v_i = 0 \) for all values of \( i \), then it will give a false positive. Since there are only 2 options for \( v_i \), then we know that for every vector that yields a false positive, there exists a vector that answers correctly (by flipping the \( v_i \) that provide erroneous results). This means, that at most half of vectors will provide incorrect results.
Problem 3: HW 4, Problem #3

Consider the graph $G_{2n}$ formed by taking two cliques on $n$ vertices and joining them by a single edge (picking one vertex in each clique). For the first $n/2$ rounds of the proposed algorithm, each clique contains at least $n/2$ many vertices. Let $a$ and $b$ be the number of vertices from each clique in the $i$-th round for $i \leq n/2$. The probability that the only mincut survives the $i$-th round is

$$\frac{C_a^2 + C_b^2}{C_{a+b}^2} \leq \frac{2C_n^2}{C_{3n/2}^2} = \frac{n(n-1)}{3n(3n/2 - 1)} < \frac{2n(n-1)}{3n(3n-3)} < \frac{2}{9}$$

Thus, the probability that the mincut survives the first $n/2$ rounds is less than $(2/9)^{n/2}$. 
4.
If we toss twice, the distribution of probability is:

<table>
<thead>
<tr>
<th></th>
<th>Head head</th>
<th>Head tail</th>
<th>Tail head</th>
<th>Tail tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>p*p</td>
<td>p*(1-p)</td>
<td>(1-p)*p</td>
<td>(1-p)*p</td>
<td>(1-p)*(1-p)</td>
</tr>
</tbody>
</table>

Thus the probability of head and tail equals to the probability of tail and head.
So the process to generate unbiased sequence of random bits can be:
First, toss twice. If we don’t get head and tail (maybe head and head, or tail and tail), we start over. Then if we get head and tail, we regard this as 0. If we get tail and head, we regard this as 1. So the probability to generate 0 equals to the probability of generating 1.

Denote \( e \) as the expected number of times we would need to flip before we can generate one random bit.

\[
e = 2p(1 - p) + 2(1 - p)p + (2 + e)p * p + (2 + e)(1 - p)(1 - p)
\]

Then \( e = \frac{1}{p - p * p} \)
And \( p = \frac{1}{4} \), so \( e = \frac{16}{3} \) 
We need to generate 3 random bits so the final result should be \( 3e = 16 \) times 

5.
Part a
A matrix that is not rearengable but has at least one 1 on every line and every column is:

\[
\begin{array}{ccc}
0 & 1 & 0 \\
1 & 1 & 1 \\
0 & 1 & 0 \\
\end{array}
\]

Part b
Let $A$ be an array $n \times n$ of 0s and 1s. Construct the following bipartite graph $G$:

$V' = \{x_1, x_2, \ldots, x_n\} \cup \{y_1, y_2, \ldots, y_n\}$
\[ E' = \{ x_i, y_i \} | A(i, j) = 1 \]  

And make it a flow graph by adding a source and a sink from each side of the graph.
\[ V' = V \cup \{ s, t \} \]
\[ E' = E \cup \{ (s, x_i) | i \in V \} \cup \{ (y_i, t) | i \in V \} \]

Make the capacity of each link 1.

Each node on the left side corresponds to one row and each node on the right corresponds to one column. If there is an edge \((x_i, y_j)\) in the graph that means that in the array \(A(i, j) = 1\).

Find the maximum matching, (as described at section 26.3 of the book). If the maximum matching is \(n\), then the matrix is rearengable and vice versa. We are going to prove both directions.

**If the matrix is rearrangeable then the maximum matching of \(G\) is \(n\).**

Notice that every row-row switch is the same as relabeling two nodes from the right side of the graph and every column-column switch corresponds to the relabeling of two nodes of the right side of the graph, but is doesn’t affect the maximum flow of the graph, since we don’t add, remove or change edges. Since the maximum flow of the graph is the same the maximum matching also don’t change. Let \(R\) be the set of rearrangements, switches between rows and columns, that will create a matrix from \(A\), that is going to have all ones on the diagonal. We will denote as \(R(A)\) the matrix that results after applying the rearrangements. Find \(A' = R(A)\).

In the graph apply all the operations in \(R\) by relabeling the nodes, e.g. if in \(R\) there is the operation \(r_i \rightarrow r_j\), \(r_i\) denotes the row \(i\), then relabel \(x_i = x_j\) and \(x_i = x_i\), and get a new graph \(G'\), that has the same maximum matching as \(G\). This \(G'\) is the bipartite graph that corresponds to matrix \(A'\). Since in \(A'\) there are ones on the main diagonal, on \(G'\) there is an edge \((x_i, y_i) \forall i \leq n\). So the maximum matching is \(n\).

**If the maximum matching of \(G\) is \(n\), then the matrix is rearrangeable**

Let \(M\) be the set of matchings, where \(|M| = n\). For every node on the right part of the graph \(x_i\) find it’s matching in \(M\), \(y_j\). Relabel \(y_j\) as \(y_i\) and vice versa. In the matrix switch \(c_i \rightarrow c_j\), where \(c_i\) is the column \(i\). In the end the resulting graph \(G'\) will have an edge \((x_i, y_i) \forall i \leq n\). The corresponding \(A'\), that resulted by just switching columns, must have ones on the main diagonal. So \(A\) is rearrangeable.

This algorithm takes \(O(n^2)\) time to build the graph and then it takes \(O(n^5)\) to find the matching. Note that the running time of Edmond Karp is \(O(|V||E|^2)\) and we know that \(|E| \in O(|V|^2)\), so Edmond Karp runs in \(O(|V|^5)\).