Please read the instructions carefully:

1. There are two stages to each assignment:
   
   (a) For the first stage, you need to submit your homework solution as a single pdf file to NYU Classes. It can be either a scanned copy of your written solution or photos joined into a pdf. You are also allowed to typeset the solutions in MS Word/LaTeX. In case of scanned copies it is your responsibility that the solution is readable. There are no late submissions accepted, as sample solution outlines (to be used for the second stage of the homework) will be given out on the day of the submission of the first stage of the homework. You should leave around 5 to 6 cm of space between each part of the answers for the second stage of the homework.

   (b) For the second stage of the assignment, you are required to critique your own solutions and analyze the mistakes that you made. Based on the outlines provided, you need to write the self criticism in the empty space that you left in your homework. Do not rewrite your solution, you need to understand and point out the mistakes that you made when writing the original solutions.

   A good analysis of your solution should summarize the key steps, ideas or techniques that you previously missed. It should prompt you to reorganize your thoughts and think about what you can do to improve the solutions. Even in the case where you completely missed the solution it is insightful to point out why the approach adopted by you does not work and identify the step where it fails. This will allow you to get full credit on the self evaluation part of the homeworks and of course help you in understanding the question.

   Mark correct solutions in green, incorrect solutions in red and if you are unsure about any solutions write it in blue.

2. For each question where you are required to design an algorithm, you need to clearly explain the algorithm and prove the running time complexity as asked in the question.

3. You can assume $O(1)$ complexity of arithmetic of numbers (e.g. multiplication/division of numbers and calculations of $n!$) unless otherwise stated in the question.
HW 6

Question 1. (15 points) Every minimum cut
For any cut, \( C \), in the graph, Karger's contraction algorithm finds it with probability \( \frac{1}{\binom{n}{2}} \).
Let there be \( l \) minimum cuts, \( C_1, C_2, \ldots, C_l \), in the graph. Let \( 1_i \) be the indicator variable for Karger’s algorithm returning the cut \( C_i \). We showed that \( \Pr[1_i = 1] \geq \frac{1}{\binom{n}{2}} \). We know that \( \Pr[(1_1 = 1) \lor (1_2 = 1) \lor \ldots \lor (1_l = 1)] \leq 1 \)

\[
\Pr[(1_1 = 1) \lor (1_2 = 1) \lor \ldots \lor (1_l = 1)] = \sum_{i=1}^{l} \Pr[1_i = 1] \geq \sum_{i=1}^{l} \frac{1}{\binom{n}{2}} = \frac{l}{\binom{n}{2}}
\]

So we have that \( \frac{l}{\binom{n}{2}} \leq 1 \implies l \leq \binom{n}{2} \).

Note 1.1. We achieve this bound when the graph is just a cycle on \( n \) nodes. In this case choosing any two edges gives us a minimum cut.

Question 2. (15 points) The unbearable isolation of being

\[
E[I] = \sum_{i=1}^{n} E[1_i] = \sum_{i=1}^{n} \Pr[1_i = 1]
\]

The probability that a vertex is isolated is \((1 - p)^{n-1}\), as each of its \( n - 1 \) neighbours are not connected to it. Which gives us that

\[
E[I] = n(1 - p)^{n-1}
\]

To calculate the \( \text{Var}[I] \), we use the formula

\[
\text{Var}[I] = E[I^2] - E[I]^2
\]

\[
E[I^2] = E[(\sum_{i=1}^{n} 1_i)^2]
\]

\[
= E[\sum_{i=1}^{n} 1_i^2 + \sum_{i \neq j} 1_i 1_j]
\]

\[
= \sum_{i=1}^{n} E[1_i^2] + \sum_{i \neq j} E[1_i 1_j]
\]

\[
= n(1 - p)^{n-1} + n(n - 1) E[1_1 1_2] \quad \text{from fig. 1}
\]

\[
E[1_1 1_2] = (1 - p)^{2n-3}
\]

\[
E[I^2] = n(1 - p)^{n-1} + n(n - 1)(1 - p)^{2n-3}
\]

\[
\text{Var}[I] = n(1 - p)^{n-1} + n(n - 1)(1 - p)^{2n-3} - n^2(1 - p)^{2n-2}
\]
For both 1, 2 to be isolated there are $2n - 3$ edges that should not be present in the graph.

**Note 2.1.** This finishes one part of the proof for the phase transition in graph connectivity.

**Question 3.** (15 points) Double hashing

$$
\Pr[\phi_k(h_k(a)) = \phi_k(h_k(b))] = \Pr[\phi_k(h_k(a)) = \phi_k(h_k(b)) | h_k(a) = h_k(b)] \cdot \Pr[h_k(a) = h_k(b)]
$$

$$
+ \Pr[\phi_k(h_k(a)) = \phi_k(h_k(b)) | h_k(a) \neq h_k(b)] \cdot \Pr[h_k(a) \neq h_k(b)]
$$

$$
\leq \epsilon + \delta \cdot (1 - \epsilon)
$$

**Question 4.** (20 points) Longest palindromic substring

Let the palindromic substring be of length $2l$ (assume even for now, we can easily see how to solve it for odd length palindromes after we solve this case).

We will assume a rolling hash function $h_k$, in $\mathbb{Z}_p$, which hashes $l$ length inputs. The basic idea is to calculate the rolling hash of $l$ length substrings as we move along the string and to calculate the reverse rolling hash of previous $l$ characters and then compare them.

$$\ldots, \underbrace{A_{i-l}, A_{i-l+1}, \ldots, A_{i-1}}_{\text{hash these } l \text{ characters to get } h_k(i')}, \underbrace{A_i, A_{i+1}, \ldots, A_{i+l-1}}_{\text{hash these } l \text{ characters to get } h_k(i)} \ldots$$

Now we can calculate $h_k(i)$ from $h_k(i-1)$ in $O(1)$ and using a similar technique we can also calculate $h'_k(i)$ from $h_k(i - 1)$.

Here we have $h_k(i) = \sum_{j=i}^{i+l-1} a_j k^{j-i}$, $h'_k(i) = \sum_{j=i-1}^{i-1} a_j k^{j-i}$, note that the sum in the second hash is in a decreasing order of $j$. 

Figure 1: Two vertices being isolated
Hence we can see if a string is a palindrome in $O(1)$ time, if we know $l$.

To modify this technique to work for odd length palindromes, we can easily leave a character in the middle, when calculating the two hashes and now be able to find all odd length palindromes.

We are doing a total of $n \log n$ hashes and comparing them, where each has a probability of collision to be equal to $\frac{l-1}{p} \leq \frac{n}{p}$, giving an expected number of collisions to be $\frac{n^2 \log n}{p}$. For each collision we have to do $O(n)$ work, giving an extra time complexity of $\frac{n^2 \log n}{p}$. So if we take $p \approx n^3 \log n$, we will only have a expected $O(1)$ extra time complexity, giving us a total running time of $O(n \log n)$.

**Question 5. (20 points) Fraud detection**

First let us calculate the number of pairs $(i,j)$ such that $i$ is not friends with $j$ and they have the same set of neighbours.

Look at the adjacency matrix of the graph.

$$
M = \begin{bmatrix}
\vdots & \cdots \\
\vdots & \cdots \\
i & 0 & 1 & \ldots & 1 & 0 \\
\vdots & \cdots \\
j & 0 & 1 & \ldots & 1 & 0 \\
\vdots & \cdots \\
\end{bmatrix}
$$

If $i$ and $j$ have the same neighbours then their rows are the same. Hence we can hash each row of the matrix using the rolling hash function and create $n$ hash values to get $h = [h_1, h_2, \ldots, h_n]$. Now sort $h$ and we can find how many hashes are the same, as they will be in one contiguous block in the array. I.e. if there is a block of size $k$ of the same hashes then it means that $\binom{k}{2}$ pairs of vertices share the same set of neighbours.

Now notice that we can calculate the rolling hash from the adjacency matrix as we don’t need the actual adjacency matrix to calculate the hashes. This means we can create the array $h$ in $O(n + m)$, after which we can finish in $O(n \log n)$.

If we want to find the pairs of vertices such that $i$ and $j$ are friends but their set of neighbours are the same, the above calculation won’t work as they will have different hashes. Instead we can modify the hash of each of these pairs by removing the component $k^i$ from the hash of row $j$ and $k^j$ from the hash of row $i$ and then compare the remaining hashes. There won’t be more than $O(m)$ of these kinds of pairs, as they need to be friends, which gives us an $O(m)$ algorithm, giving a total running time of $O(n \log n + (n + m))$.

**Question 6. (30 points - Extra credit) Encrypting parts**

Look at the code attached with the solutions.