# Quantum Information Science HW P2

## Harris A. Ransom

## 11/11/24

In [2]:
```python
# Imports
import numpy as np
```

## Question 1

Each Tanner graph is considered a **regular bipartite** graph, since it can be divided into two sets of nodes that do not have any intraconnecting edges within their respective sets. For a random (3;6) biregular Tanner graph, we can compute the **degree distribution generating polynomials** as:

- $\lambda(x) = \sum_{d=1}^{3} \Lambda_d x^{d-1} = x^2$
- $\rho(x) = \sum_{d=1}^{6} P_d x^{d-1} = x^5$

## Question 2

### Question 2a

Successful decoding condition: $\epsilon\lambda\left(1-\rho(1-x)\right) \leq x \text{ for } 0 < x < \epsilon$

We can express $(1-\rho(1-x))$ using the above polynomial as $\alpha = (1-(1-x)^5)$

From this, we get $\lambda(\alpha)$ = $\left(1-(1-x)^5\right)^2$. Note that, from close observation, $\lambda(\alpha)$ is a CDF function, potentially of the degree distribution.

Plugging this back into the decoding condition, we get: $\epsilon\left(1-(1-x)^5\right)^2 \leq x$ for $0 < x < \epsilon$

$\frac{1}{x} \cdot \left(1-(1-x)^5\right)^2 \leq \frac{1}{\epsilon}$ for $0 < x < \epsilon$

$x^9 - 10x^8 + 45x^7 - 120x^6 + 210x^5 - 250x^4 + 200x^3 - 100x^2 + 25x \leq \frac{1}{\epsilon}$ for $0 < x < \epsilon$

Since $\epsilon$ represents the maximum channel erasure rate, we want it to fall between 0 and 1. Therefore, we can adjust our condition as: $x^9 - 10x^8 + 45x^7 - 120x^6 + 210x^5 - 250x^4 + 200x^3 - 100x^2 + 25x \leq \frac{1}{\epsilon}$ for $0 < x < \epsilon \leq 1$

We can rewrite the condition into a better form:
$\epsilon \cdot (x^9 - 10x^8 + 45x^7 - 120x^6 + 210x^5 - 250x^4 + 200x^3 - 100x^2 + 25x) \leq 1$ for $0 < x < \epsilon \leq 1$

From this condition, for values of $\epsilon$ between 0 and 1, we find that the theoretical maximum erasure rate $\epsilon$ that satisfies the condition is $\epsilon \approx 0.4295$

## Question 2b

In [3]:

```
# Gets the degree of a check node
def getCheckNodeDegree(A, node):
        degree = np.sum(A[:, node])
        return degree

# Set up adjacency matrix and message
print("Setting up Tanner graph parameters...")
VARIABLE_NODES = 2000
CHECK_NODES = 1000
A = np.zeros((VARIABLE_NODES, CHECK_NODES)) # Adjacency matrix
m = np.zeros(VARIABLE_NODES) # Message (all zeros)

# Distribute connections between variable and check nodes
# Rows: 3 (degree 3)
# Cols: 6 (check 6)
# TODO: Figure out why generation hangs on node 1999
print("Generating (3;6) Tanner graph...")
colCounts = np.zeros(CHECK_NODES) # Tracks how many columns are full
for i in range(0, VARIABLE_NODES):
```