# Homework 6

Mitchell Gant[1]

[1]Georgia Institute of Technology

April 26, 2019

## Problem 1

(a)

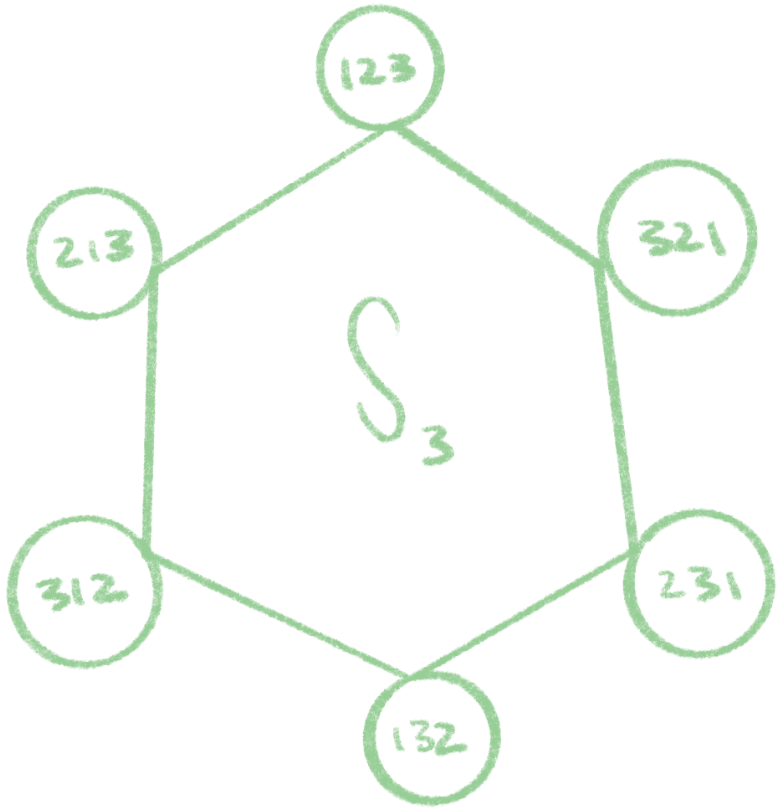


Figure 1: $S_3$

(b) number of processors in $S_n$ = number of distint permutations of first $n$ positive integers = $n!$

(c) Algorithm for finding path from one processor in $n^{th}$ order star network to another:

**Step 1:** Compare the $i^{th}$ integer in the source rank with the $i^{th}$ integer in the destination rank ,where $i \in \mathbb{Z}, [2, n]$. Append the source rank to a list of processors through which you would have to 'hop' to get to the destination.

**Step 2:** During the comparison, if the $i^{th}$ integer in the destination and the source processor do not match we switch the integer in the $1^{st}$ position with that of the $i^{th}$ position in the source processor. If the source and destination processors' $i^{th}$ element was the same then we check to see if the source and destination processor have the same rank and if so we stop running the algorithm. If not, we set $i$ equal to $((i+1)mod(n))$ and if $i = 1$ we set $i$ to 2 to keep it within its range.

**Step 3:** We repeat steps 1 and 2 for a maximum of $n$ iterations.

(d) Using the algorithm above after making one iteration or one hop, it would take $(2n-1)$ subsequent iterations or hops to come back to the source processor. Given this information we can conclude that the diameter of $S_n = O(\frac{2n-1+1}{2}) = O(n)$.

## Problem 2

If $P_i$ and $P_j$ are processors in a $d$ dimensional hypercube then both processors' ranks' are composed of $d$ bits. We can create unique parallel paths from $P_i$ to $P_j$ by flipping a unique bit in $P_i$ at every iteration until $P_i$ is the binary equivalent of $P_j$. The result is d unique parallel paths with d as the upperbound given that d is the maximum number of separate connections to one processor in the d-dimensional hypercube. We can also think about this in terms of $P_i$ and $P_j$ which differ in h bits. If we instantiate $h$ different paths each starting with $P_i$ and flip a different bit in which $P_i$ and $P_j$ differ for each path to get the next step in the path for h iterations, we will notice that no two nodes are hopped to amongs the h different paths until the final hop in which all paths converge to $P_j$. This leads to $h$ unique parallel paths each with a length of $h$. If you think about the remaining $d-h$ bits which do not differ in the ranks of $P_i$ and $P_j$, these could signify paths that must be first routed away by flipping one of the $d-h$ bits in which $P_i$ and $P_j$ are the same to a different value and then routing the path back by flipping the bit back. Flipping the bit at the beginning and end of the path ensures that the path's remain unique and flipping the bit twice leads to an extra 2 steps in the path. This makes the path $h+2$ hops long. The result is a total of $d$ parrallel paths.

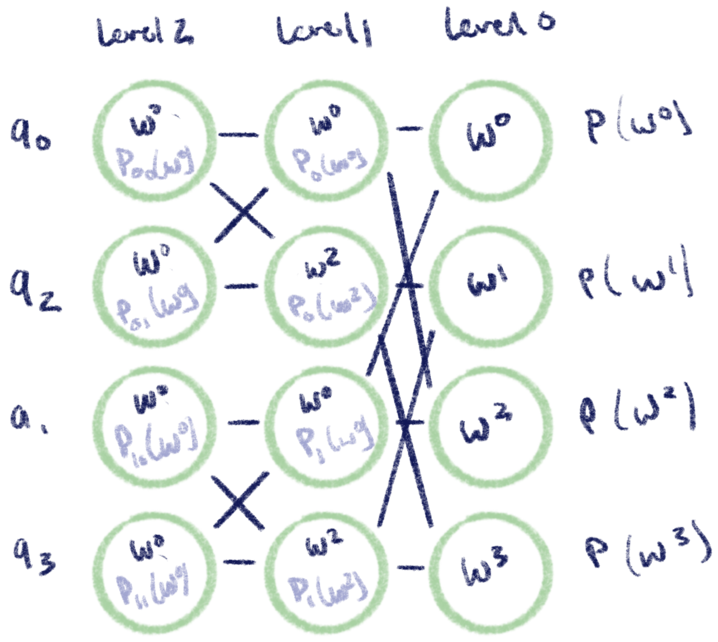
## Problem 3

## Problem 4

This algorithm is incorrect as it will not work when Q is a polynomial with degree greater than the degree of P and the algorithm also breaks down when any of the coefficients in Q are zero. The result of the algorithm will also only be the quotient in polynomial division but will not include the remainder.

## Problem 5

If we look at the result of the matrix vector multiplication of a vector and a Toeplitz matrix we will see that the result is very similar to that of a polynomial multiplication where one polynomial has the values in the vector as coefficients and the other has the unique values of the Toeplitz matrix as coefficients in the polynomial. The Toeplitz matrix can be mapped to a polynomial of the form $T(x) = \sum_{i=0}^{m} t_i x^i$; where $m = 2n-2$ and the vector can be mapped to a polynomial of the form $A(x) = \sum_{i=0}^{n-1} a_i x^i$. When looking



at the resultant vector $R(x)$, each element with index $j$ in the vector can be mapped to the polynomial $\sum_{j=0}^{n-1} a_j t_{j-i+n-1}$, which would be the same as the computing the coefficients of $x^{j+n-1}$ after calculating $T(x) * A(x)$ using FFT.