CS 70 Discrete Mathematics and Probability Theory
Spring 2015 Vazirani Discussion 4W

1. Leaves in a tree

A *leaf* in a tree is a vertex with degree 1.

(a) Prove that every tree on $n \geq 2$ vertices has at least two leaves.

(b) What is the maximum number of leaves in a tree with $n \geq 3$ vertices?

2. Edge-disjoint paths in hypercube

Prove that between any two distinct vertices x, y in the n-dimensional hypercube graph, there are at least n edge-disjoint paths from x to y (i.e., no two paths share an edge, though they may share vertices).

3. Congestion in hypercube routing

In HW4, we described the "bit-fixing" algorithm to send a packet from vertex x to vertex y in a hypercube:

> In each step, the current processor compares its address to the destination address of the packet. Let's say that the two addresses match up to the first k positions. The processor then forwards the packet and the destination address on to its neighboring processor whose address matches the destination address in at least the first $k+1$ positions. This process continues until the packet arrives at its destination.

In this problem we explore how well the bit-fixing algorithm works for parallel routing. Recall that in the n-dimensional hypercube architecture for a parallel computer, the 2^n vertices represent processors and the edges represent wires. Computation is divided into *compute phases*, where all processors compute separately, and *communicate phases*, where the processors use the wires to send messages to each other. In a communicate phase the processors are paired up (according to the communication needs of the algorithm being executed), where in each pair one processor is designated the source and the other the destination. Each source processor sends a message to its destination processor during the communication phase.

As you see in HW4, the bit-fixing algorithm ensures that each such message in isolation travels quickly. Of course, the problem is that during the communication phase the messages don't travel in isolation, but instead 2^{n-1} messages must simultaneously travel through the network, and this might lead to congestion. If particular, if there is a vertex or edge through which a large number of these messages must pass, then that creates a bottleneck, and the total time for the communication phase will be proportional to the maximum number of messages that must pass through any bottleneck. In this question you will show that the bit-fixing algorithm can suffer from a terrible bottleneck. In particular, show that you can assign the sources and destinations such that there is a vertex through which at least $2^{n/2}$ packets will be routed by the bit-fixing algorithm (you can assume $n = 2m$ is even).

Note: One way to avoid congestion is to *randomize* the protocol as follows. For every pair of source x and destination y, instead of sending the packet from x to y directly, first choose a random vertex z and send the packet from x to z, then from z to y (both using the bit-fixing algorithm). Then we can show that we (almost) never have congestion, no matter what configuration of sources and destinations we start with. This is an example of a *probabilistic method*, which we shall explore further in the second half of the course.