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?

Answer:

(a) We give a direct proof. Consider the longest path $\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-1}, v_k\}$ between two vertices $x = v_0$ and $y = v_k$ in the tree (here the length of a path is how many edges it uses, and if there are multiple longest paths then we just pick one of them). We claim that x and y must be leaves. Suppose the contrary that x is not a leaf, so it has degree at least two. This means x is adjacent to another vertex z different from v_1. Observe that z cannot appear in the path from x to y that we are considering, for otherwise there would be a cycle in the tree. Therefore, we can add the edge $\{z, x\}$ to our path to obtain a longer path in the tree, contradicting our earlier choice of the longest path. Thus, we conclude that x is a leaf. By the same argument, we conclude y is also a leaf.

The case when a tree has only two leaves is called the *path graph*, which is the graph on $V = \{1, 2, \ldots, n\}$ with edges $E = \{\{1,2\}, \{2,3\}, \ldots, \{n-1, n\}\}$.

(b) We claim the maximum number of leaves is $n - 1$. This is achieved when there is one vertex that is connected to all other vertices (this is called the *star graph*).

We now show that a tree on $n \geq 3$ vertices cannot have n leaves. Suppose the contrary that there is a tree on $n \geq 3$ vertices such that all its n vertices are leaves. Pick an arbitrary vertex x, and let y be its unique neighbor. Since x and y both have degree 1, the vertices x, y form a connected component separate from the rest of the tree, contradicting the fact that a tree is connected.

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).

Answer: We use induction on $n \geq 1$. The base case $n = 1$ holds because in this case the graph only has two vertices $V = \{0, 1\}$, and there is 1 path connecting them. Assume the claim holds for the $(n-1)$-dimensional hypercube. Let $x = x_1 x_2 \ldots x_n$ and $y = y_1 y_2 \ldots y_n$ be distinct vertices in the n-dimensional hypercube; we want to show there are at least n edge-disjoint paths from x to y. To do that, we consider two cases:

1. Suppose $x_i = y_i$ for some index $i \in \{1, \ldots, n\}$. Without loss of generality (and for ease of explanation), we may assume $i = 1$, because the hypercube is symmetric with respect to the indices. Moreover, by interchanging the bits 0 and 1 if necessary, we may also assume $x_1 = y_1 = 0$. This means x and y both lie in the 0-subcube, where recall the 0-subcube (respectively, the 1-subcube) is the $(n-1)$-dimensional hypercube with vertices labeled $0z$ (respectively, $1z$) for $z \in \{0,1\}^{n-1}$.

   Applying the inductive hypothesis, we know there are at least $n-1$ edge-disjoint paths from x to y, and moreover, these paths all lie within the 0-subcube. Clearly these $n-1$ paths will still be edge-disjoint

in the original n-dimensional hypercube. We have an additional path from x to y that goes through the 1-subcube as follows: go from x to x', then from x' to y' following any path in the 1-subcube, and finally go from y' back to y. Here $x' = 1x_2 \dots x_n$ and $y = 1y_2 \dots y_n$ are the corresponding points of x and y in the 1-subcube. Since this last path does not use any edges in the 0-subcube, this path is edge-disjoint to the $n-1$ paths that we have found. Therefore, we conclude that there are at least n edge-disjoint paths from x to y.

2. Suppose $x_i \neq y_i$ for all $i \in \{1, \dots, n\}$. This means x and y are two opposite vertices in the hypercube, and without loss of generality, we may assume $x = 00\dots0$ and $y = 11\dots1$. We explicitly exhibit n paths $P_1, \dots, P_n$ from x to y, and we claim they are edge-disjoint.

   For $i \in \{1, \dots, n\}$, the i-th path P_i is defined as follows: start from the vertex x (which is all zeros), flip the i-th bit to a 1, then keep flipping the bits one by one moving rightward from position $i+1$ to n, then from position 1 moving rightward to $i-1$. For example, the path P_1 is given by

   $$000\dots0 \to 100\dots0 \to 110\dots0 \to 111\dots0 \to \cdots \to 111\dots1$$

   while the path P_2 is given by

   $$000\dots0 \to 010\dots0 \to 011\dots0 \to \cdots \to 011\dots1 \to 111\dots1$$

   Note that the paths $P_1, \dots, P_n$ don't share vertices other than $x = 00\dots0$ and $y = 11\dots1$, so in particular they must be edge-disjoint.

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.

Answer: Assume $n = 2m$ is even. Consider the following assignment of sources and destinations: every element in $\{0,1\}^n$ of the form $x0$ is a source with corresponding destination $0x$, where $x = x_1 \dots x_m \in \{0,1\}^m$ and $0 = 00\dots0 \in \{0,1\}^m$. There are 2^m such pairs, and we can divide the remaining vertices into sources and destinations arbitrarily (e.g., we can send every element of the form $xy \in \{0,1\}^n$ to yx, where $x, y \in \{0,1\}^n$). Observe that using the bit-fixing algorithm, every source $x0$ will be routed via the point $0 = 00\dots0 \in \{0,1\}^n$ on the way to $0x$. This means the point 0 is a bottleneck.

Note: We can also argue that at some point during the execution of the bit-fixing algorithm, there are at least $2^m/(n+1)$ packets that arrive at 0 at the same time. We argue this as follows: Each path in the bit-fixing algorithm takes at most n edges, so it traverses at most $n+1$ vertices (including the source and destination). For each path P_i, let j_i be the index at which P_i reaches the vertex 0 (i.e., 0 is the j_i-th vertex in P_i). Since there are 2^m paths but only at most $n+1$ different values of j_i, by the pigeon hole principle we conclude that there are at least $2^m/(n+1)$ paths that have the same index. That is, there is $k \in \{0,1,\dots,n\}$ such that those $2^m/(n+1)$ paths all reach 0 at the k-th index. So here our bound is weaker by a factor of $1/(n+1)$, but notice that the 2^m still grows much faster than $n+1$, so $2^m/(n+1)$ is still an exponential number of congested packets.