ECS223a Parallel Algorithms
Homework 3

You are encouraged to talk to other people about these problems, but please **write up the solutions by yourself**. Cite any conversations you had with others, as well as books, papers or Web sites you consulted.

Always explain the answer in **your own words**; do not copy text from the book, other books, Web sites, your friends' homework, etc. Explain your solution as you would to someone who does not understand it, for instance to a beginning graduate student or an advanced undergraduate. Do not give several solutions to one problem; pick the best one.

Please type your homework. If you know LaTeX, use that. If not, you may type your answers in any word processing system and write in mathematical notation by hand as necessary. Include pictures if appropriate; you can draw in pictures by hand or include them in the file.

1. Do problem 5.30. Assume the graph is given as an adjacency matrix, and that the root v of the tree is specified.

2. We looked at an algorithm for connected components for an undirected graph in the PRAM model. Let's consider the same problem in the simplified MapReduce model given in the Goodrich et al. paper, in which each node can accept or send data of size at most M where $\Omega(\lg n) = M = O(n^\epsilon)$, with $0 < \epsilon < 1$, and n is the number of nodes in the input graph.

   We will assume that the maximum degree d of any vertex is $d = O(M)$, so that an input vertex can examine all of its outgoing edges. The graph is given as a set of edges, and node i has as its initial input a list of pairs $A_0(i) = (\,(i, j_1), (i, j_2), ..., (i, j_k)\,)$ with $k \leq d$, representing the edges adjacent to i.

   Try to achieve the smallest number of rounds and the lowest communication cost that you can; certainly try to do better than simulating the PRAM algorithm on the simplified MapReduce system. Notice that one challenge with transferring the PRAM algorithm to the simplified MapReduce framework is that a super-vertex might have $> M$ adjacent edges, which would be too many for one MapReduce node to handle.

3. Do problem 9.31. The "random allocation scheme" in Step (2) is covered is in the description of perfect hashing in Cormen, Leiserson, Rivest and Stein book, *Introduction to Algorithms*, which we assume you know from cover to cover; it's Theorem 11.9 in the Third Edition, where the argument works for picking random numbers as well as for Universal Hash Functions.

4. In genomics, a *k-mer* is a sub-string of length k. Finding all the k-mers in a long string is a common operation in bioinformatics. For instance, the string:

   $$TAGGTCAGG$$

   has 3-mers TAG, AGG, GGT, GTC, TCA, CAG and AGG again. How could you use the randomized string-matching algorithm for finding k-mers, where k is an input that will be provided by the user? Can you use the same function f?