# Homework 5

MPCS 51100

due: Wed Dec. 12, 2018

In problems 1-3, consider the shortest path problem for a directed, weighted graph $G(E, V)$. We learned that Dijkstra's algorithm finds the distance between any source node S and all other nodes in the graph in $O(|V|^2)$ operations. A similar problem, the all-source shortest path problem, finds the minimum distance between EACH vertex v in a graph and all other vertices.

1. Write a function `dijkstra_all_src()` that solves the all source shortest path problem by applying Dijkstra $n = |V|$ times, i.e. once for each vertex in $G$. Note that this algorithm has complexity $n^3$. Include a driver that tests the code on the two test graphs in the homework dropbox. Report the timings. Name this source file p1.c

2. Write a shared memory parallel version of `dijkstra_all_src()` using OpenMP. Your strategy will be straightforward – simply assign groups of different source vertices to each core. Assume $n > p$, where p is the number of processors (i.e. cores) on a multicore machine. Be careful to properly address any load balance issues. What is the complexity of the parallel algorithm in terms of $n$ and $p$? Compare the performance of the serial and parallel approach using the provided sample graph. Comment briefly on the observed timings. This source file will be p2.c.

3. Finally, consider how you might use a shared memory parallel processor to accelerate Dijskstra *within* a single source computation? There are several possible approaches, and developing a reasonable approach yourself is part of the exercise. In this case, to guide your thinking you may assume that $n$ is much larger than $p$ – i.e there are many more vertices than cores. Consider ways to partition the graph among processors to minimize contention and load balance – ie to keep all cores busy but require minimum synchronization. Describe your approach and comment briefly on observed timings. Name your function `dijkstra()`, include a driver, and name the file p3.c.

4. Write a function `graph_period` that efficiently calculates the *period* of a directed graph $G$. Then write a driver that takes the Markov matrix in the hw directory as input and calculates its period (as a reminder *aperiodic* is equivalent to a period of 1). Name this source file p4.c.

5. Write a function `graph_is_irreducible` and an associated driver that takes a Markov matrix as input and calculates whether or not it is *irreducible* (as a reminder *irreducible* means strongly connected for a directed graph). Apply this function the the Markov graph provided in the homework directory. What is the result? Name this source file p5.c.

6. Write a function `markov_chain` that takes a Markov matrix as input and uses randomization to execute realizations to estimate the stationary probabilities. Write a driver that calls this function on the Markov graph provided in the homework directory. What are the resulting stationary probabilities? What do the answer to the previous two questions tell you about the values that you calculated? Name this source file p6.c.

7. Write a function called `GE_solve()` that takes an $n \times n$ input matrix $A$ and RHS vector $b$ and returns the vector $x$ that satisfies $Ax = b$. Solve the system using Gaussian elimination with back-substitution. You may assume that the linear system has a unique solution. Include a simple test driver that initializes a small test matrix and RHS vector, calls `GE_solve()`, and prints the results. Name this source file p7.c.

8. Devise an $O(n)$ algorithm to solve $Ax = b$ using elimination and backsubstitution *assuming that $A$ is tridiagonal.* Implement this in a function called `tri_solve()`. Use a data structure for A that does not require storage of the zeros that are not part of the diagonal, upper diagonal, or lower diagonal. Include a simple test driver that initializes a small tridiagonal test matrix and RHS vector, calls `tri_solve()`, and prints the results. Name this source file p8.c

9. Write a shared memory parallel version of `GE_solve()` using either OpenMP or PThreads. Include a simple test driver that allows the user to specify the number of threads, initializes a small test matrix and RHS vector, calls `GE_solve()`, and prints the results. Name this source file p9.c.

10. Carry out a performance analysis of the parallel and serial versions of `GE_solve()`,exploring a range of matrix sizes and thread counts.