CS 290H: Sparse matrix algorithms // Homework 1

Assigned October 3, 2004

Due by class time Thursday, October 14

1. **[15 points]** Do exercise 2.3 on page 11 of GLN. (Notice that GLN uses the same exercise numbers over and over again in different sections—be sure you're doing the right exercise 2.3!)

2. **[15 points]** Do exercise 4.3 on page 68 of GLN.

3. **[35 points]** Let A be an $n \times n$ symmetric, positive definite matrix, and let $G = G(A)$ be its graph. Fix an elimination ordering of G (that is, a labeling of the vertices of G by the integers 1 through n), and let G^+ be the filled graph for that ordering. We define the *front size* of the ordering as the maximum number of higher-numbered neighbors of any vertex in G^+,

$$s = \max_{1 \le v \le n} \{\#(w) \mid w > v \text{ and } (v, w) \text{ is an edge of } G^+\},$$

which is also equal to the maximum number of nonzeros below the diagonal in any column of the Cholesky factor L of A.

Consider the two-dimensional model problem, in which G is a square grid graph with $n = k^2$ vertices.

3(a) **[15 points]** Using the path lemma (class, Sep 28), prove that the front size of the model problem is $O(\sqrt{n}\,)$.

3(b) **[20 points]** Prove that there's no elimination order for the model problem that gives front size better than $O(\sqrt{n}\,)$—in other words, prove that for every elimination ordering on the vertices of G, the front size is $\Omega(\sqrt{n}\,)$. (Hint: First use the path lemma to prove that for every elimination ordering, the graph G^+ contains a complete subgraph with $\Omega(\sqrt{n}\,)$ vertices.)

3(c) **[Optional, extra credit]** Prove that if G is any planar graph, then there is an elimination order for G whose front size is $O(\sqrt{n}\,)$.

4. **[35 points]** Write a Matlab mexFunction `C = mult (A,B)` to multiply two real sparse matrices. The result should be a Matlab sparse matrix data structure, identical to the result of `C = A*B`. The indices of each column of C should be in sorted order, and you should remove any entries in C that become zero due to numerical cancellation. You can use your `sparseprint` routine from HW0 to help check the results.

You may assume that both A and B are square, though it's actually just as easy to write code for the general case in which A is $n \times m$ and B is $m \times p$.

There are several possible approaches to this problem. I'll talk about a few of them in class. Matlab's sparse multiplication uses a sparse accumulator, or SPA. It loops over the columns of B, computing one column of AB at a time as a dense vector in the SPA and then storing it into the

sparse data structure for the result. The running time for Matlab's algorithm is $O(f + n)$, where f is the number of nonzero multiplications (per problem 1 above) and n is the dimension of the matrices. To achieve this running time you need to be careful in your implementation of the SPA: you can take $O(n)$ time to initialize the SPA once at the beginning, but you must not take $O(n)$ time for SPA operations in each column.

Test your code by running it in Matlab on several matrices of the form `sprand (n, n, d)` for values of n ranging from 100 to as large as you can and for values of $d = 1/n$, $5/n$, and $10/n$. You should be able to get n at least into the hundreds of thousands for $d = 5/n$, and possibly bigger. Use Matlab's `etime`, or `tic` and `toc`, to get wall-clock times. Make a plot in Matlab that shows f (number of flops) on the x axis, and running time on the y axis, with a point for each matrix. Plot the running time of Matlab's `C = A*B` on the same figure (say "help hold" to Matlab). Can you beat Matlab's running time?

Turn in all your code, and also the plot with your run times and Matlab's, and also a Matlab transcript of the session that creates the plot and verifies that your output matrices agree with Matlab's. For full credit, you should submit a correct code that runs in reasonable time for n up to 100,000 and $d = 5/n$.

There are several ways to get extra credit.

[**Optional, extra credit**] Implement at least two different algorithms and compare them (along with Matlab's). What is the asymptotic running time, in $O()$ notation, of your algorithms?

[**Optional, extra credit**] Experiment with ways to do memory allocation. You don't know before you begin how much memory you'll need for the output because you don't know how many nonzeros it will have. One way (which will get you full credit but would be cheating in practice) is to allocate a fixed amount of memory that's enough for your largest test case. A second way (which Matlab uses) is to guess how much memory you will need; if you run out, you allocate a bigger block, copy your partial results into the new space, and free the old block. A third way is to do the whole matrix multiplication twice: the first time through, discard the results after each column of the output, but count the nonzeros and use the total to allocate exactly the right amount of memory to redo the multiplication.

[**Optional, extra credit**] I'll give extra credit to any code that is faster than Matlab on matrices of my choice of dimension 100,000 and more.

[**Optional, extra credit**] I'll give a little more extra credit to the fastest code submitted.