CS 466/666
Topic 2: Divide and Conquer/ Reductions
Lectures: Sept 28 & 23

Master Theorem review (CLRS section 4.3)

There are 3 parts to the Master Theorem, we need examples of each. Indeed for 2 cases we will give new methods.

1 Easy MThm case: "balance" giving a lg n term
Mergesort and Binary Search are examples.

2 Number of subproblems dominates:
Number multiplication in $O(n^{1.58..})$ (problem p844 ex 30-1 and Strassen Matrix Multiplication in $O(n^{2.81..})$(Section 28.2) are examples.
Do both

3 Size of "fix-up" dominates.
Here is a chance to build on the Strassen method and develop a couple of reductions.
Transitive closure of a directed graph; a simple $O(n^3)$ method is given on p633. It is a simple iteration, though one cold possibly argue for dynamic programming.
The text also discusses it as "min, plus" matrix multiplication … but note these operations don't work in "Strassen".
Go back and observe that by squaring an adjacency matrix, we get in position (i,j) the number of ways of getting from i to j in 2 steps. If we make the diagonal of the matrix all 1's, this gives the number of ways of going from i to j in "at most 2 steps".
If we keep squaring, say $\lceil \lg n \rceil$ times we will get non zero values in location i,j if there is a path from 1 to j. There is a problem that this could be a very large number; but if we replace all non zeros by 1's after each squaring (including the last) we never have values in the answers greater than n
So we can use Strassen for (Boolean) matrix multiplication and get an $O(n^{2.81..} \lg n)$ method.
Interestingly we can get rid of this lg n term.

- Get rid of the issue of cycles by finding the strongly connected components of the graph, collapsing each into a single node. (The answer to the original problem is easily recovered) Note this phase takes $O(n^2)$ time and can substantially reduce the problem size, so is probably a very good idea in practice)
- Reorder these nodes so that edges go from lower to higher numbered nodes (ie topological sort, again quadratic time)
- Let G denote the adjacency matrix of a graph and G* the adjacency matrix of the transitive closure. Then, we have manipulated the problem so that G is an upper triangular Boolean matrix.

$G = \begin{pmatrix} A & C \\ 0 & B \end{pmatrix}$ where A and B are uppertriangular with 1's on the diagonal. *A* deals with the edges in the first half of the graph and there are no "back edges", so indeed

A^* goes in the top left quarter of G^*; similarly B^* goes in in the lower right. Connections from the top to the bottom of the graph are found by "taking a few steps in the top", then moving from top to bottom according to C, then "taking a few steps in the bottom". This means

$$G^* = \begin{pmatrix} A^* & A^*CB^* \\ 0 & B^* \end{pmatrix}.$$

So, G^* can be found with 2 n/2 by n/2 transitive closures followed by 2 Boolean matrix multiplications. So

$T(n) = 2T(n/2) + O(n^{2.81...})$ (using Strassen).

From the Master Theorem, we have reduced transitive closure to Boolean matrix multiplication as long as our Boolean matrix multiplication is $\Omega(n^{2+\varepsilon})$, i.e. we have an $O(n^{2.81...})$ transitive algorithm assuming we can do arithmetic on lg n bit numbers in constant time.

Curiously the reduction works in the other direction. Suppose we want to multiply the Boolean matrices A and B. Consider the digraph whose adjacency matrix is

$$H = \begin{pmatrix} I & A & O \\ & I & B \\ & & I \end{pmatrix}$$ … What is this graph? It is tripartite, with edges going from the first third to the second and from the second to the third. Finding the transitive closure is simply a matter of filling in the appropriate values in the top right corner (where the 0 is). This is easily seen to be AB, i.e. finding the transitive close (by whatever means we like) on a 3n node graph can be used to multiply two arbitrary Boolean matrices.