CS1510 Dynamic Programming Problems 24,& 25, Reduction Problem 2

Rebecca Negley, Sean Myers

October 14, 2011

24. Give a polynomial time algorithm for the following problem. The input consists of a sequence $R = R_0, ..., R_n$ of non negative integers, and an integer k. The number R_i represents the number of users requesting some particular piece of information at time i (say from a www.server. If the server broadcasts this information at some time t, the requests of all the users who requested the information strictly before time t are satisfied. The server can broadcast this information at most k times. The goal is to pick the k times to broadcast in order to minimize the total time (over all requests) that requests/users have to wait in order to have their requests satisfied

    Solution:

25. Assume that you are given a collection $B_1, ..., B_n$ of boxes. You are told that the weight in kilograms of each box is an integer between 1 and some constant L, inclusive. However you do not know the specific weight of any box, and you do not know the specific value of L. You are also given a pan balance. A pan balance functions in the following manner. You can give the pan balance any two disjoint sub-collections, say S_1 and S_2 of the boxes. Let $|S_1|$ and $|S_2|$ be the cumulative weight of the boxes in S_1 and S_2, respectively. The pan balance then determines whether $|S_1| < |S_2|$, $|S_1| = |S_2|$, or $|S_1| > |S_2|$. You have nothing else at your disposal other than these n boxes and the pan balance. The problem is to determine if one can partition the boxes into two disjoint sub-collections of equal weight. Give an algorithm for this problem that makes at most $O(n^2L)$ uses of the pan balance. For partial credit, find an algorithm where the number of uses is polynomial in n and L.

    Solution:

26. Show that if there is an $O(n^k)$, $k \geq 2$, time algorithm for inverting a nonsingular n by n matrix C then there is an $O(n^k)$ time algorithm for multiply two arbitrary n by n matrices A and B.
    For a square matrix A, A inverse, denoted A^{-1}, is the unique matrix such that $AA^{-1} = I$, where I is the identity matrix with 1's on the main diagonal and 0's every place else. Not that not every square matrix has an inverse, e.g. the all zero matrix.

    Solution: If we can reduce multiplying two arbitrary n x n matrices to inverting a matrix, and the most inefficient part of the algorithm is the inversion, then the algorithm will run in at least $O(n^k)$.

    If we set up a matrix C, where the matrix of size 3n x 3n, where the matrix looks like: $\begin{vmatrix} \mathrm{I} & \mathrm{A} & 0 \\ 0 & \mathrm{I} & \mathrm{B} \\ 0 & 0 & \mathrm{I} \end{vmatrix}$

    (I being identity matrix). The setup of this matrix would take $9 * n^2$, since it is trivial to set up an identity matrix, copy a matrix or 0-pad a matrix like so.

    Then the inverse(derivation not shown) would look something like: $\begin{vmatrix} \mathrm{I} & \mathrm{-A} & \mathrm{A*B} \\ 0 & \mathrm{I} & \mathrm{-B} \\ 0 & 0 & \mathrm{I} \end{vmatrix}$

    Then all we would need to do is extrapolate the top right node (which takes n^2 amount of time), and we have our matrix multiplication.

The time to convert to the input matrix C is $O(n^2)$, the inversion takes $O(n^k)$ where k must be greater than or equal to 2 and then once we have the output, the time to translate to the desired output is $O(n^2)$. Hence, the algorithm's slowest possible run time is that of the inverse multiplication $O(n^k)$.