CS1510 Dynamic Programming Problems 22 & 27, Reduction Problems 3 & 4

Rebecca Negley, Sean Myers

October 17, 2011

22. The input to this problem is n points $x_1, \ldots, x_n$ on a line. A good path P has the property that one endpoint of P is the origin and every x_i is covered by P. Note that P need not be simple. That is, it can backtrack over territory that it has already covered. Assume a vehicle moves along this path from the origin at unit speed. The response time r_i for each x_i is the time until the vehicle first reaches x_i. The problem is to find the good path that minimizes $\sum_{i=1}^{n} r_i/n$, the average response time. Give a polynomial time algorithm for this problem.

    Solution: Suppose at time t, a path P is at point x_p. The next unvisited point that P visits must be the unvisited point closest to x_p on the left or the unvisited point closest to x_p on the right. If P instead goes to some point farther away, it must pass through one of those points on the way, so one of those two points will in fact be the next one visited. Create two sequences $x_{1_1}, \cdots, x_{1_k}$ and $x_{2_1}, \cdots, x_{2_m}$ $(k+m=n)$ such that $x_{1_i} \geq 0$ for all $i \in [1,k]$ and $x_{2_j} < 0$ for all $i \in [1,m]$. Additionally, $x_{1_i} \leq x_{1_{i+1}}$ for all $i \in [1,k-1]$ and $x_{2_j} \geq x_{2_{j+1}}$ for all $j \in [1,m-1]$. Then, since the path starts at the origin (between the two sequences), the closest unvisited point to the left of the current point is $\min_j(x_{2_j} | x_{2_j}$ unvisited). Similarly, the closest unvisited point to the right of the current point is $\min_i(x_{1_i} | x_{1_i}$ unvisited).

    Now we can construct our tree. At level lvl, each node corresponds to a sequence of lvl points. Each node has two children, one corresponding to the the smallest unvisited x_{1_i} (corresponding to min i) being added to the sequence and one corresponding to the greatest unvisited x_{2_j} (corresponding to min j) being added to the sequence. Then, we have a binary tree with n levels, so we have 2^n possible solutions at the leaves. To reduce complexity, we will prune the tree as follows:

    (a) If there are two nodes on the same level that have sequences ending at the same point and have the same closest unvisited points on the left and the same closest unvisted points on the right, prune the one with the greater average response time.

    With this pruning rule, each level can have at most $(k+1)*(m+1) < n^2$ possible pairs of closest unvisited right point and closest unvisited left point. We can create our tree in an $n \times n \times n \times 2 \times 2$ matrix MGP, the first dimensions corresponding to the level of the tree, the second to the index of the closest unvisited point to the right, third to the index of the closest unvisited point to the left, the fourth to whether the path is at the right or left input (0 for right, 1 for left), and the fifth to whether we are storing the minimum average response time or the current response time. We initialize each location MGP$[\alpha, \beta, \gamma, 0/1, 0]$ to $+\infty$ except for MGP[0,1,1,0,0], MGP[0,1,1,0,1], MGP[0,1,1,1,0], and MGP[0,1,1,1,1], which we set to 0. Then, we traverse down the levels of our matrix (or tree) by building each level from the previous one. If MGP$[\alpha, \beta, \gamma, 0, 0]$ has a value $< \infty$, then we can set MGP$[\alpha+1, \beta, \gamma+1, 0, 0]$ to the minimum of its current value and $(\alpha *$MGP$[\alpha, \beta, \gamma, 0, 0] + (MGP[\alpha, \beta, \gamma, 0, 1] + (x_{1_{\gamma+1}} - x_{1_\gamma})))/(\alpha+1)$. If we change this location from its current value, set MGP$[\alpha+1, \beta, \gamma+1, 0, 1]$ to MGP$[\alpha, \beta, \gamma, 0, 1] + (x_{1_{\gamma+1}} - x_{1_\gamma}))$, the response time at that point. Similarly, we can set the location coordinating to a left traversal. Then, when we are fiinished filling in our table, we seek return the minimum of MGP$[n, k+1, m+1, 0, 0]$ and MGP$[n, k+1, m+1, 1, 0]$, corresponding to the minimum of the path using all the points and ending at the right endpoint and the path using all the points and ending at the left endpoint. In code, our algorithm looks as follows:

```
minGoodPath($x_1, \cdots, x_n$)
    construct $x_{1_1}, \cdots, x_{1_k}$ ordered (L-R) sequence of nonnegative points
    construct $x_{2_1}, \cdots, x_{2_m}$ ordered (R-L) sequence of negative points
    construct MGP of size $n \times n \times n \times 2 \times 2$
    initialize MGP to $+\infty$
    MGP[0,1,1,0,0]=0
    MGP[0,1,1,0,1]=0
    MGP[0,1,1,1,0]=0
    MGP[0,1,1,1,1]=0
    for $lvl$ from 1 to $n$ do:
        for $i$ from 1 to $k+1$ do:
            for $j$ from 1 to $m+1$ do:
                if MGP$[lvl, i, j, 0, 0] < \infty$ do:
                    MGP$[lvl+1, i+1, j, 0, 0]$ =min{MGP$[lvl+1, i+1, j, 0, 0]$, ($lvl$*MGP$[lvl, i, j, 0, 0]$
                                        +MGP$[lvl, i, j, 0, 1] + (x_{1_i} - x_{1_{i-1}}))/(lvl+1)$}
                    if MGP$[lvl+1, i+1, j, 0, 0]$ changed:
                        MGP$[lvl+1, i+1, j, 0, 1] = lvl$*MGP$[lvl, i, j, 0, 0]$+MGP$[lvl, i, j, 0, 1] + (x_{1_i} - x_{1_{i-1}})$
                    MGP$[lvl+1, i, j+1, 1, 0]$ =min{MGP$[lvl+1, i, j+1, 1, 0]$, ($lvl$*MGP$[lvl, i, j, 0, 0]$
                                        +MGP$[lvl, i, j, 0, 1] + (x_{1_{i-1}} - x_{2_j}))/(lvl+1)$}
                    if MGP$[lvl+1, i, j+1, 1, 0]$ changed:
                        MGP$[lvl+1, i, j+1, 1, 1] = lvl$*MGP$[lvl, i, j, 0, 0]$+MGP$[lvl, i, j, 0, 1] + (x_{1_{i-1}} - x_{2_j})$
                if MGP$[lvl, i, j, 1, 0] < \infty$ do:
                    MGP$[lvl+1, i+1, j, 0, 0]$ =min{MGP$[lvl+1, i+1, j, 0, 0]$, ($lvl$*MGP$[lvl, i, j, 1, 0]$
                                        +MGP$[lvl, i, j, 1, 1] + (x_{1_i} - x_{2_{j-1}}))/(lvl+1)$}
                    if MGP$[lvl+1, i+1, j, 0, 0]$ changed:
                        MGP$[lvl+1, i+1, j, 0, 1] = lvl$*MGP$[lvl, i, j, 1, 0]$+MGP$[lvl, i, j, 1, 1] + (x_{1_i} - x_{2_{j-1}})$
                    MGP$[lvl+1, i, j+1, 1, 0]$ =min{MGP$[lvl+1, i, j+1, 1, 0]$, ($lvl$*MGP$[lvl, i, j, 1, 0]$
                                        +MGP$[lvl, i, j, 1, 1] + (x_{2_{j-1}} - x_{2_j}))/(lvl+1)$}
                    if MGP$[lvl+1, i, j+1, 1, 0]$ changed:
                        MGP$[lvl+1, i, j+1, 1, 1] = lvl$*MGP$[lvl, i, j, 1, 0]$+MGP$[lvl, i, j, 1, 1] + (x_{2_{j-1}} - x_{2_j})$
    return min(MGP$[n, k+1, m+1, 1, 0]$, MGP$[n, k+1, m+1, 0, 0]$)
```

27. Give a polynomial time algorithm for the following problem. The input consists of a two dimensional array R of non-negative integers, and an integer k. The value $R_{t,p}$ gives the number of users requesting page p at time t. At each integer time, the server can broadcast either 0 or 1 pages. If the server broadcasts page p at time t, the requests of all the users who requested page p strictly before time t are satisfied. The server can make at most k broadcasts. The goal is to pick the k times to broadcast and the pages to broadcast at those k gimes in order to minimize the total time (over all requests) that requests/users have to wait in order to have their requests satisfied.

    Solution:

3. Show that if there is an $O(n^k)$, $k \geq 1$, times algorithm for squaring a degree n polynomial, then there is an $O(n^k)$ time algorithm for multiplying two degree n polynomials. Assume that the polynomials are given by their coefficients.

Solution:

4. Consider the following variant of the minimum Steiner tree problem. The input is n points in the plane. Each point is given by its Cartesian coordinates. The problem is to build a collection of roads between these points so that you can reach any city from any other city and the total length of the roads is minimized. The collection of roads should be output as an adjacency list structure. Show by reduction that if you can solve this problem in linear time, then you can sort n numbers in linear time.

   Solution: