15-451/651 Algorithm Design & Analysis, Fall 2023

Recitation #11

Objectives

- Understand the objective of an online algorithm, how to create one, and how to bound its competitive ratio.
- Practice common techniques for geometric algorithms.

Recitation Problems

1. **(Favoritism)** You're given a rooted, unweighted tree T of n nodes (not necessarily a binary tree, or a balanced tree). For each node in the tree, one of the children of that node is designated as the "favorite child". Initially the left-most children are all favorites, but they may change over time.

   We want to perform a sequence of `Traverse`(x) operations. The `Traverse`(x) operation traverses the path from the root of the tree to the node x (assume that the algorithm knows the path it needs to take in order to get from the root to x). The cost of a traversal is the number of nodes visited (not counting the root), except that visiting a favourite child is free!

   To help improve the cost, the algorithm is allowed to change the favorite child of any node at any time. Changing the favorite child of a node costs one. Consider the following greedy online algorithm: before performing a traversal, make all of the nodes on the path the favourite child of their parent if they are not already.

   Prove that this algorithm is 2-competitive, that is, it incurs a cost at most double that of an optimal omniscient algorithm that can see the future and plan ahead.

2. **(Move to Front with Linked Lists)** Recall the *List Update* problem from lecture. We begin with a list of n items $1, 2, \ldots, n$ stored in a linked list and have the following operations:

   - `Access`(x): Accesses item x. This cost is the position of x.
   - `Swap`(k): Swaps adjacent elements at positions k and $k+1$. This costs 1.

   We now extend our linked-list functionality with a new operation `MoveForward` defined as:

   - `MoveForward`(x, k): After accessing element x, move it to any position k in front of it, i.e. $k < \texttt{pos}(x)$. This has no cost.

   Using only these operations, our goal is to devise an online-algorithm that is 2-competitive.

   (a) Consider the following possible algorithms:

   - MTF (Move to Front): After accessing the x^{th} element, move it all the way to the very front of the list
   - MoTF (Move One to Front): After accessing the x^{th} element, move it one position forward in the list
   - Frequency Count: Keep the list sorted with respect to frequency. (The highest frequency is in the front.)

   Show why *MoTF* and *Frequency Count* are <u>not</u> 2-competitive by providing a sequence of m bad accesses alongisde an offline solution.

(b) We'll choose to stick with MTF now. In lecture, we showed that MTF was 4-competitive for the original *List Update* problem. Given our new `MoveForward` operation, show that MTF is now 2-competitive for this variant of the problem (Hint: use a potential function).

3. **(Angular sorting without angles)** Recall that the first step of the *Graham scan* algorithm for convex hull that we learned in lectures is to sort the points with respect to their angle from the bottom-most point. The most straightforward way to do this is of course to simply compute the angles and then sort the points. This has some drawbacks, such as having to perform floating-point computations that are susceptible to rounding errors.

   Given a set of n points with integer-valued coordinates, describe how to sort them with respect to their angle to the bottom-most point without using any floating-point computations. (Hint: use the line-side test primitive).