Sliding Window Maximum

Nick Haliday

2012-10-05

1 Intro

Sliding window maximum is a fairly useful and general algorithm often used in dynamic programming problems. I say fairly useful because it generally only provides a factor of $O(\log n)$ speedup over more naïve algorithms.

Nonetheless, here, as with many other algorithms, the underlying principles can be useful in more general contexts, even if the specific realization differs. In these more general contexts the ideas used in sliding window maximum can provide much more substantial speedups. Essentially, it's nice to know sliding window minimum, but the real goal of this lecture is that you grasp the principles underlying it, which are often needed to solve more difficult DP problems.

2 The Problem

Given a sequence of n integers a and a length $k \leq n$, for each index i such that $0 \leq i \leq n-k$, compute the maximum of $a[i, i+k) = (a_i, a_{i+1}, \ldots, a_{i+k-1})$. You can imagine this as a window of fixed length k sliding over the sequence a, where we want to keep track of the maximum at each shift.

The trivial solution is $O(nk)$. Using a heap, you can make a just slightly less naïve algorithm with runtime $O(n \log n)$ ($\log n$ is usually less than k by a good margin, so this is generally better). In this lecture, we present a solution that runs in $O(n)$.

3 Solving It

We will compute the maximums progressively, using old maximums to save time computing new maximums. We will store these old maximums in a deque (double-ended queue). Each of these maximums will be stored as a pair (i, a_i), the index where the maximum occurs and its value, respectively.

The basic idea underlying sliding window maximum is that we throw away values that don't matter anymore, that can never be optimal. This allows us to reduce the update times to amortized $O(1)$.

Suppose we have two maxima (i, a_i) and (j, a_j) stored in our deque, and we have $i < j$. When can that happen? If $a_i < a_j$, then we can always choose (j, a_j) over (i, a_i). Any new index $k > i, j$ will be closer to j then i, so there's no reason not to choose the greater value a_j.

We can have $a_i > a_j$, however, while still keeping both values. Eventually i could go out of range of the window, and then we'd need j around to find the new, lower maximum.

What this shows is that the deque of maxima, when sorted by their indices (which will happen naturally if we always push to the back in order), will also have strictly decreasing values a_i.

To shift the window forward by one, we repeatedly check the front of the deque, and pop it off if the index is not within the window. We then need to process the value at our current index i. To do that, we repeatedly look at the back of the deque, and pop it (j, a_j) off if $a_j \leq a_i$. Finally, we just push (i, a_i) onto the back. To query the current maximum, we can just inspect the front of the deque.

Each maximum (i, a_i), is pushed/popped at most once, and each of those operations is constant time, so the total runtime is $O(n)$.

4 The Underlying Principles

Now we get into those underlying principles I referred too. There aren't a whole lot of subalgorithms that work with a lot of DP problems besides some things like binary indexed trees or range trees. Each problem varies a little bit in what the state and transitions will be. What's important is that you have a good feel for how a DP solution can be constructed, and the principles/intuition that might lead you to the solution. Incidentally, the best way to obtain that intuition is through solving problems. In keeping with that idea, while some of the problems at the end of this lecture require straightforward implementations of sliding window maximum, several require slightly different versions or just build on one or more of the ideas from this list:

1. Throw away useless values. Often you will have to search to find where to insert a new value, and irrelevant values slow down that search.
2. Order the values in some way. This often relates to which values are useless, and where you should insert new values. Keep your collection of values sorted by this order.
3. In any particular scheme based around these ideas, individual updates may take a long time, but each value can be inserted/thrown away at most once, so the amortized time will be aymptotically faster.

5 Problems (Yay!)

1. Billboards [Interviewstreet]

   ADZEN is a very popular advertising firm in your city. In every road you can see their advertising billboards. Recently they are facing a serious challenge, MG Road the most used and beautiful road in your city has been almost filled by the billboards and this is having a negative effect on the natural view.

   On people's demand ADZEN has decided to remove some of the billboards in such a way that there are no more than K billboards standing together in any part of the road.

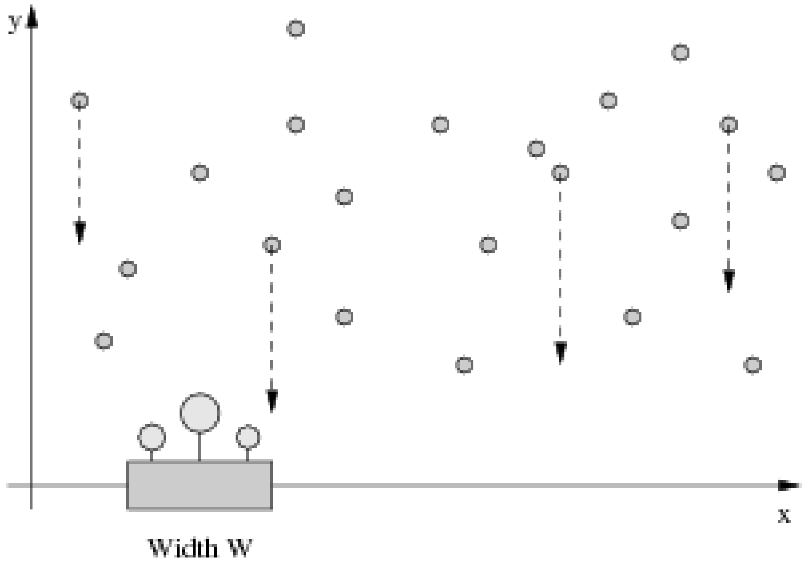
   You may assume the MG Road to be a straight line with N billboards. Initially there is no gap between any two adjecent billboards.

   ADZEN's primary income comes from these billboards so the billboard removing process has to be done in such a way that the billboards remaining at end should give maximum possible profit among all possible final configurations.Total profit of a configuration is the sum of the profit values of all billboards present in that configuration.

   Given N,K and the profit value of each of the N billboards, output the maximum profit that can be obtained from the remaining billboards under the conditions given.

2. Flowerpot [Brian Dean, 2012]

   Farmer John has been having trouble making his plants grow, and needs your help to water them properly. You are given the locations of N raindrops ($1 \leq N \leq 100,000$) in the 2D plane, where y represents vertical height of the drop, and x represents its location over a 1D number line:

   Each drop falls downward (towards the x axis) at a rate of 1 unit per second. You would like to place Farmer John's flowerpot of width W somewhere along the x axis so that the difference in time between the first raindrop to hit the flowerpot and the last raindrop to hit the flowerpot is at least some amount D (so that the flowers in the pot receive plenty of water). A drop of water that lands just on the edge of the flowerpot counts as hitting the flowerpot.

Given the value of D and the locations of the N raindrops, please compute the minimum possible value of W.

3. Cow Hopscotch [John Pardon, 2010]

   The cows have reverted to their childhood and are playing a game similar to human hopscotch. Their hopscotch game features a line of N ($3 \leq N \leq 250,000$) squares conveniently labeled $1..N$ that are chalked onto the grass.

   Like any good game, this version of hopscotch has prizes! Square i is labeled with some integer monetary value V_i ($-2,000,000,000 \leq V_i \leq 2,000,000,000$). The cows play the game to see who can earn the most money.

   The rules are fairly simple:

   - A cow starts at square "0" (located just before square 1; it has no monetary value).
   - She then executes a potentially empty sequence of jumps toward square N. Each square she lands on can be a maximum of K ($2 \leq K \leq N$) squares from its predecessor square (i.e., from square 1, she can jump outbound to squares 2 or 3 if $K = 2$).
   - Whenever she wishes, the cow turns around and jumps back towards square 0, stopping when she arrives there. In addition to the restrictions above (including the K limit), two additional restrictions apply:
   - She is not allowed to land on any square she touched on her outbound trip (except square 0, of course).
   - Except for square 0, the squares she lands on during the return trip must directly precede squares she landed on during the outbound trip (though she might make some larger leaps that skip potential return squares altogether).

   She earns an amount of money equal to the sum of the monetary values of all the squares she jumped on. Find the largest amount of cash a cow can earn.

4. Poklon [COCI 2008]

   Mirko got a set of intervals (size of set bounded by 10^5) for his birthday. There are many games he can play with them. In one of them, Mirko must find the longest sequence of distinct intervals such that each interval in the sequence is in the set and that each interval contains the one that follows in the sequence. Write a program which finds one such longest sequence.