Monotonic Queues and USACO Review

Daniel Wisdom

January 8 2019

1 Monotonic Queues

1.1 Sample Problem

We are given a list of N numbers. We want to find all the minimums of every range of size K. If N=7,K=3 this means the minimum of [0,2],[1,3],[2,4],[3,5], and [4,6]. We could easily do this in O(NK) by looping through every window. For examples, the list A=2,4,3,8,1,0,4. The window minimums are 2,3,1,0,0.

1.2 Solution

This looks like a problem using queues. We can sweep through the array and keep a queue of the numbers in the current window. We will push the first K elements of A into the queue. We can push the next element of A and pop the last element of the queue to shift our window down one. We always keep K elements in the queue. Before every shift we need to know what the smallest element in the queue is.

The key is realizing that some elements in the queue don't matter. If my queue is 3,4,2, where 2 is the next to be removed, then 4 will never be the minimum in the queue. Because 3 will be in the queue longer than 4, and 3 is smaller than 4, 4 can never be the minimum. I can write that queue as $3^1, 2$. Where the superscript indicates an anonymous, never-minimum number. Likewise 4,3,5,8,1,3,7,3 can be written as $4, 3^2, 1^3$. Popping once will leave the queue as $4, 3^2, 1^2$. Popping twice more will leave $4, 3^2$. When I add an element to the queue, if the element in front of it is larger, I can superscript the preceding number. I will repeat this until the preceding number is smaller, or I have superscripted ever other number in the queue. Using this method, the minimum element will always be the first element of the queue. Finding the minimum is now O(1).

Every element will store its value, along with a count of superscripted elements. You can superscript a larger element by $myCount = myCount + hisCount + 1$ and deleting the larger element. Popping from the queue decrements the first element's count, or removes it if the count is 0. Finding the minimum and popping are both O(1). Consider adding 0 to 9,8,7,6,5,4,3,2,1. That individual operation will be O(n) and result in a queue of 0^9. Because every element is only superscripted at most once, adding is still amortized O(1).

2 Redistricting

2.1 Problem

USACO Platinum January 2019

The greater metropolitan area of Bovinopolis consists of a line of N pastures ($1 \leq N \leq 3 * 10^5$), each containing a single cow, which is either a Holstein or a Guernsey.

The government of Bovinopolis wants to divide the greater metropolitan area into some number of contiguous districts, so that each district contains at most K pastures ($1 \leq K \leq N$), and every pasture is contained in exactly one district. Since the government is currently controlled by Holsteins, they want to find a way to redistrict which minimizes the number of Guernsey-majority or tied districts (a district is tied if the number of Guernseys equals the number of Holsteins). Find the minimum number of districts which are either Guernsey-majority or tied.

2.2 Solution

Note: This solution is different from the one posted on usaco.org.

I will consider Holsteins to be a +1 and Guernseys to be a -1. Conveniently this makes a district Holstein majority if the range sum is greater than 0. I can computer the prefix sums so that a district is good (Holstein majority) if $pSum[end] - pSum[start] > 0$. Note that I refer to districts using [start,end) and pSums[x] is the sum up to x-1, but not including x.

I want to greedily go as far as possible without making a bad district. I will find, for each x, whether it is possible to make only good districts, with the last district ending at x-1. For example in **HHGG**H**G**GG bold letters are valid ending+1 places for good districts. x is a valid ending+1 if a previous position y is valid and $pSums[x] - pSums[y] > 0$ and $x - y \leq K$. This suggests a monotonic queue because we are interested in the minimum pSum[y] of the last K elements. We can sweep the monotonic queue across the array, and check the minimum pSum[y] to know whether a position is a valid ending+1 before putting it in the monotonic queue.

Of course, our solution needs to account for situations where we need a bad district. If the last K positions are all invalid, we need one bad district. We can check this by looking at the minimum pSums[y]: if it is invalid then we need a bad district. Increment the bad district counter. Now, I could end my bad district anywhere in the last K elements. They are all valid starting positions. Consider K=5, GGGHHGGH. We need to set the two middle H's to valid, otherwise we would need 2 bad districts. Likewise the G's are valid, but it turns out we would always include them in the bad district. So when I need a bad district, the K past positions become valid. Note that I only need to validate the entire queue once every K elements, so this operation only adds O(N) work.

It was convenient for me to denote invalid positions by putting pSums[y]+1000000 into the monotonic queue. When I needed to make all the positions valid, I could just subtract 1000000 from every element. This doesn't change the ordering, so the monotonic queue doesn't need to be restructured.

The whole algorithm runs in O(N), and a fast O(N) too. In fact my solution finished each test in under 31ms.

3 Shortcut

3.1 Problem

USACO Gold January 2019

Every evening, Farmer John rings a giant bell that summons his cows to the barn for dinner. Eager to get to the barn as quickly as possible, they all follow the shortest possible route to get there. The farm is described by a set of N fields ($1 \leq N \leq 10,000$), conveniently numbered 1...N, with the barn residing in field 1. The fields are connected by a set of M bidirectional trails ($N - 1 \leq M \leq 50,000$). Each trail has a travel time associated with it, and there is a path from every field to the barn using some set of trails.

Field i contains c_i cows. Upon hearing the dinner bell, these cows all walk to the barn along a route that takes the minimum amount of time. If there are several routes tied for the minimum time, the cows take whichever of these is "lexicographically" smallest (i.e., they break ties between two routes by favoring the one using the lower-indexed field at the first place where the routes differ, so for example a path that visits fields 7, 3, 6, 1 would be preferable to one that visits 7, 5, 1, assuming both had the same travel time).

Farmer John is worried about the barn being far away from some fields. He adds up the travel time experienced by each cow, summed over all the cows, calling this number the total travel time. He would like to reduce this number as much as possible by adding one extra "shortcut" trail which has a travel time of T ($1 \leq T \leq 10,000$), from the barn (field 1) to some other field of his choosing. If a cow stumbles upon the shortcut trail while traveling along her usual path to the barn, she will take it if it gets her to the barn faster. Otherwise, a cow will follow her usual route, even if it might have been possible to use the shortcut to improve her travel time.

Please help Farmer John determine the greatest possible amount of decrease in total travel time he can achieve by adding his shortcut trail.

3.2 Solution

I can easily find the maximum time saved if I know two things about every field: its distance from the barn and the number of cows which start at or pass through that field. The time saving if I put the shortcut to a certain field is $cows * (dist[field] - T)$. I can easily find the distance to the barn using a Dijkstra out from the barn.

Now I want to simulate the cows walking to the barn, and count how many cows pass through each node. I could trace out the path of cows from each field, but that would be $O(N^2)$. The key is that, once two groups of cows from different fields meet, they will follow the exact same path to the barn. I want to process the nodes in an order so that one large, merged, group leaves each node after all cows have arrived at that node. I can sort the fields in decreasing order of distance from the barn to guarantee that all cows arrive at a field before I process that field.

I will start with the field furthest from the barn. I will check the adjacent nodes to see which, accounting for edge weight and lexicographic tiebreaks, is closest to the barn. The cows travel down that edge and I add them to the cow counter of that field.

When this finishes all the cows have moved to the barn. I can evaluate $cows * (dist[field] - T)$ for each field and output the maximum. Note that T could be so large the shortcut is useless. The above formula will be negative for fields where the shortcut is useless. Make sure to output 0 if this happens.

The slowest part of this algorithm is the Dijkstra, which runs in $O(M \log M)$

4 Cow Poetry

4.1 Problem

USACO Gold January 2019

Unbeknownst to Farmer John, Bessie is quite the patron of the arts! Most recently, she has begun studying many of the great poets, and now, she wants to try writing some poetry of her own. Bessie knows N $(1 \leq N \leq 5000)$ words, and she wants to arrange them into poems. Bessie has determined the length, in syllables, of each of her words, and she has also assigned them into "rhyme classes". Every word rhymes only with other words in the same rhyme class.

Bessie's poems each include M lines $(1 \leq M \leq 10^5)$, and each line must consist of K $(1 \leq K \leq 5000)$ syllables. Moreover, Bessie's poetry must adhere to a specific rhyme scheme. That is, specific sets of lines must all end in the same rhyme class. However, you may reuse a rhyme class for multiple sets. For example "moo, moo, moo, moo" satisfies ABAB. There are at most 26 different letters in the rhyme scheme.

Bessie would like to know how many different poems she can write that satisfy the given constraints.

4.2 Solution

The first thing to note is that, since all lines must be K syllables long, I can precompute the number of possible lines that end in each rhyme class. I will first compute the number of x-syllable lines, ignoring rhymes. This is a basis $O(NK)$ DP. If K=100, and I have a 3-syllable word, moo-OO-oo, there are DP[97] lines that end in moo-OO-oo. I will add DP[97] to the number of possible lines ending in moo-OO-oo's rhyme class.

The order of the rhyme scheme does not matter. The number of ways to satisfy ABCAB is the same as the number of ways to satisfy AABBC or BBAAC. The other observation is that the different letters in the rhyme scheme are totally independent. In ABABCDA, I can find the number of sets oflines that satisfy A, times the number for B, times C, times D. This gives the total ways to satisfy ABABCDA.

If I need m lines that all rhyme, how many possibilities are there? I have already calculated the number of possible lines ending in each rhyme class, P. P^m gives me the number of ways to make m rhyming lines for that rhyme class. I sum $P[rhyme]^m$ over all the rhyme classes to get the number of possible m rhyming lines.

Now, in the rhyme scheme I count the number of As,Bs,Cs... and find the number of rhyming lines for each. Multiply all these together to get the total possible poems satisfying the rhyme scheme.

5 Hopscotch

You start at position 0 on a line of N $(1 \leq N \leq 3 * 10^5)$ squares. You can jump up to K squares at once. There is a cost c_i for landing on each tile. Find the minimum cost to reach square N.