Binary Search

Daniel Wisdom

November 10, 2017

1 Basic Binary Search

Let's say we have an array in increasing order and want to find the **smallest element that is greater than or equal to** K. I will call this element arr[answer]. We could search the entire array in O(N) time, but we can do better. If we find that arr[10]$<K$, then we know that answer$>$10. Likewise, if arr[10]$\geq K$ then answer$\geq$10. Using this we can eliminate half the possibilities every step.

Algorithm 1 Basic Binary Search

```
function BINARYSEARCH(arr, K)
    lowIndex ← 0                          ▷ lowIndex is the lowest possible value of answer, inclusive
    highIndex ← N − 1                     ▷ highIndex is the highest possible value of answer, inclusive
    while lowIndex ≠ highIndex do
        midIndex ← (highIndex+lowIndex+1)/2          ▷ Truncated to a whole number
        if arr[midIndex]< K then
            lowIndex ← midIndex + 1
        else
            highIndex ← midIndex
    return arr[lowIndex]
```

This will run in O(log N) time, much better than O(N). For other search requirements, such as the largest element strictly less than K you will need to tweak when to add one and the comparisons.

The example allows us to define a general binary search algorithm.

1. *Answer Range*: We need to know a maximum range that *answer* falls inside. These bounds will be the initial values of $lowIndex$ and $highIndex$. In the example we can assume the answer falls within the size of the array, [0,$N-1$].

2. *Test*: This is a test on a guess G that tells whether $answer < G$ or $answer > G$, or if G might be the answer. In the example we simply compare arr[$midIndex$] to K as our test.

This algorithm works in O(O(Test) $*$ log(Range)) for any problem where we can eliminate half the data at every step. This means that, as long as the test is reasonably fast, we can search through massive ranges.

2 Examples

Unfortunately, it is sometimes hard to tell when a binary search could be useful. Often in USACO Bronze or Silver it will be fairly straightforward, but binary search problems in Gold and Platinum often involve binary searching on the input to another algorithm in a counter intuitive way.

2.1 Singing Cows

Farmer John has a large group of N cows, each with a singing talent in the range [0, 100,000,000]. After all the auditions, Farmer John has a lot of questions, each in the form of: If I only accept cows with more than K talent, how many cows will I have in my choir? Given a list of the cow's talent levels, answer all Farmer John's questions quickly.

Hint: Sort first, ask questions later.

2.2 Singing Cows Solution

Because the cows are given in unsorted order, is will be useful to sort them by talent. To make math slightly simpler, I would sort the cows in decreasing order of talent.

Now, given that Farmer John will only take cows with more than K talent, let's use a binary search to find the most talented (earliest in the ordering) cow that will be rejected from the choir. This means that all cows in front of that cow will be accepted. If that unfortunate cow occurs at index i (zero indexed) in the sorted order, there are i cows in front of him, and therefore Farmer John will have i cows in his choir.

The binary search will take O(log N) time, giving an overall run time of O(log N * questions + N log N).

2.3 Milk Production

In Farmer John's herd of cows, all cows produce milk with different fat percentages. Every cow produces a certain percentage of milk ($0.00 \leq p \leq 100.00$) and a certain number of gallons per week. You job is to quickly find, for a given percentage P_{max}, how many gallons of milk with a lower fat percentage per week Farmer John can produce. There will be a large number of queries, so O(N) time for each query will not work.

2.4 Milk Production Solution

Let's sort the cows by increasing percentage milk fat and see how that helps us. We can now binary search this array to find the last cow, at index i, whose percentage is still less than P_{max} quickly. We then know that all cows from 0 to i inclusive contribute to the milk production. If we loop over this range and sum up their milk productions, it will be correct but slow.

Now the problem becomes: Given a list of milk productions, quickly find the sum from the start of that array to index i. Prefix sums do exactly this. After sorting the cows we can make a list of the sum from 0 to k every index k. This takes O(N), but allows us O(1) lookup.

The final solution is:

1. Sort all the cows by percentage
2. Find the sum of milk productions up to each cow, store this array for later.

3. For each query binary search on P_{max} to find the index of the last included cow. Now look up the sum of milk productions up to and including that cow from the prefix table.

Including the initial sort and prefix table calculation, the complexity is O($N \log N$ + queries * $\log N$).

2.5 Ground Leveling

Farmer John has an uneven plot of land he wants to level. Every square has an elevation h ($0 \leq h \leq 1,000,000$). After all work is done, every square of the plot must have the same exact height. It will cost FJ \$10 per foot to buy concrete to raise a square. FJ can also hire workmen to dig out a square, lowering it as much as he wants. If FJ lowers a space by x feet, it will cost him x^2 dollars.

However, due to strange discounts in his area, FJ will only have to pay for either workmen or concrete, whichever is more expensive. The cheaper one will be free. This means if FJ needs \$ 20 of concrete and \$36 of workmen total for all squares, it will only cost him \$36.

Given a list of initial heights of the squares, find the minimum amount FJ will pay to level the plot of land.

Hint: look at the graphs of concrete cost vs final height and worker cost vs final height.

3 Problems

Counting Haybales, Silver December 2016
Farmer John has just arranged his N haybales ($1 \leq N \leq 100,000$) at various points along the one-dimensional road running across his farm. To make sure they are spaced out appropriately, please help him answer Q queries ($1 \leq Q \leq 100,000$), each asking for the number of haybales within a specific interval along the road.

Cow Dance Show
After several months of rehearsal, the cows are just about ready to put on their annual dance performance; this year they are performing the famous bovine ballet "Cowpelia".

The only aspect of the show that remains to be determined is the size of the stage. A stage of size K can support K cows dancing simultaneously. The N cows in the herd ($1 \leq N \leq 10,000$) are conveniently numbered 1...N in the order in which they must appear in the dance. Each cow i plans to dance for a specific duration of time d(i). Initially, cows 1...K appear on stage and start dancing. When the first of these cows completes her part, she leaves the stage and cow K+1 immediately starts dancing, and so on, so there are always K cows dancing (until the end of the show, when we start to run out of cows). The show ends when the last cow completes her dancing part, at time T.

Clearly, the larger the value of K, the smaller the value of T. Since the show cannot last too long, you are given as input an upper bound T_{max} specifying the largest possible value of T. Subject to this constraint, please determine the smallest possible value of K.