Question 10.4 from CTCI: Sorted Search, No Size

Benjamin De Brasi

September 20, 2017

Question

You are given an array-like data structure Listy which lacks a size method. It does, however, have an elementAt (i) method that returns the element at index i in 0(1) time. If i is beyond the bounds of the data structure, it returns - 1. (For this reason, the data structure only supports positive integers.) Given a Listy which contains sorted, positive integers, find the index at which an element x occurs. If x occurs multiple times, you may return any index.

Explanation and Algorithm

A trivial solution would be to use linear search. This would be O(n), but we can get a better solution.

The key is to implement binary search and figuring how to get the a range of indices in the list to search through. This can be done by starting from the first index and multiplying by 2 every time the index is less than the target number. Either one of two things will happen: You will find a number that is greater than the the target or you the list will return a -1. Either way, you will have a endpoint you can use for binary search. From that endpoint you simply divide by 2 to get a valid start point (you would have checked it's validity in the previous iteration of this doubling loop). This will be just as fast as binary search since it will take at most log(n) to reach the end of the list since $2\hat{k} = n$ and thus $k = \log n$.

Hints

1. Think about how binary search works. What will be the issue with just implementing b inary search?

2. Binary search requires comparing an element to the midpoint. Getting the midpoint requires knowing the length. We don't know the length. Can we find it?

3. We can find the length by using an exponential backoff. First check index 2, then 4, then 8, then 16, and so on. What will be the runtime of this algorithm?

Code

```
int search(Listy list, int value){

   int index = 1;

   while(list.elementAt(index) != -1 && list.elementAt(index) < value){
   index *= 2;
}
   return binarySearch(list, value, index/2, index);
}

int binarySearch(Listy list, int value, int low, int high){
   int mid;

   while (low <= high){
      mid = (low + high) / 2;
      int middle = list.elementAt(mid);

      if(middle > value || middle == -1){
         high = mid -1;
      }else if (middle < value) {
         low = mid + 1;
      }else{
         return mid;
      }
   }
   return -1;
}
```

Run Time analysis

The algorithm runs in O(log(n)). The takeaway is it takes no extra time to run binary search if you don't know the length of the list you are running it on.

Sources

Question, answer and other material taken from Cracking the Coding Interview 6th edition by Gayle Laakmann McDowell.