Week 2: Competitive Coding QSTP

Binary Search

SUMMER 2021

"If all the names in the world are written down together in order and you want to search for the position of a specific name, binary search will accomplish this in a maximum of 35 iterations."

Theory

https://www.topcoder.com/community/competitive-programming/tutorials/binary-search

Things to keep in mind : The article provided above is quite comprehensive and flawless for theory of binary search, at the same time it might be overwhelming for those who aren't already familiar with binary search, to them I'd like to advise to not read the article at once. Try to understand the gist of the article prior to the heading "Taking it further: the main theorem", then proceed further.

How to identify if a problem can be solved using binary search?

To check if a problem can be solved using binary search, you will have to identify the element of monotonicity in the problem. Does your output increase with increase in input, does it decrease with increase in input and vice-versa. Despite the existence of such a straight forward thumb rule, it is difficult to identify if a problem uses binary search because it sometimes gets tough to identify the element of monotonicity. If we talk in a less mathematical way, try to break the problem in a yes or no question. If the problem follows the structure below, then binary search can be used.

STRUCTURE:

If you answered yes for some potential solution, x means that you'd also get a yes answer for any element after x. Similarly, if you got no, you'd get a no answer for any element before x. As a consequence, if you were to ask the question for each element in the search space (in order), you would get a series of no answers followed by a series of yes answers.

It can be easily observed that Yes and No can swap places in the description above, which means we will have a series of yeses followed by nos. In this case, yes for any x will mean that we will get yes for all elements before x and no for any x will mean that we will get no for all elements after x.

So to conclude with, if your question can be boiled to a function whose output varies with increasing (or decreasing) input as follows:

0,0,0,0,0. . .1,1,1,1,1,1,1,1,1,1,1. . .

OR

1,1,1,1,1. . .0,0,0,0,0,0,0,0,0,0,0,0. . .

And you are supposed to find the point of transition, i.e. the point from where onwards only 1s occur (as in the first case), if such are the characteristics of your problem, you can use binary search.

A simple visualization

if you want to have an idea why with binary search you can solve a problem, you can imagine that you have a big stick and you have to find something in that stick, so if you break the stick in a half, and in some way you know that you can drop one half and just search in the other half, then you have a smaller space where you have to search.

Applying the same principle many times you are gonna have a very small stick, where you can say if whatever you are searching is there or no.

Lower_bound and upper_bound in STL

http://www.cplusplus.com/reference/algorithm/lower_bound/

http://www.cplusplus.com/reference/algorithm/upper_bound/

Suggested Problems

- https://www.spoj.com/problems/AGGRCOW/
- https://codeforces.com/problemset/problem/1352/C
- https://codeforces.com/problemset/problem/1279/B
- https://codeforces.com/problemset/problem/1223/C
- https://www.spoj.com/problems/ABCDEF/
- https://codeforces.com/problemset/problem/822/C