Competitive Programming and Mathematics Society

Programming Workshop #3

Binary Search

Jonathan Lam

Today's Workshop

CPMSOC

Quick Refresher on Binary Search

CPMSOC

- Simple algorithm to find an item in a sorted array
- The algorithm:
  - The search maintains a search space in the array, which initially contains all the elements
  - At each step, check the middle element of the active region.
  - If this is the target element, we are done.
  - Otherwise, search recursively on the left or right half of the middle element (depending on whether it is above or below the target element).
- Time complexity: $O(\log n)$
  - Each step reduces the size of the search space by a half.
- The implementation is notorious for being error prone and off-by-one errors.
- There are built in functions: `C` has `bsearch` and `C++ <algorithm>` has `binary_search`, `lower_bound` and `upper_bound`.

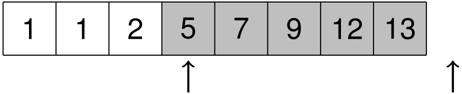
Example on an array

Searching for 7

1	1	2	5	7	9	12	13

Example on an array

Searching for 7



Example on an array

Searching for 7

1	1	2	5	7	9	12	13
			↑		↑		

Example on an array

CPMSOC

Searching for 7

1	1	2	5	7	9	12	13

Implementation

CPMSOC

```
#include <cstdio>                                    // The same as stdio.h, gives printf/scanf

int main() {
    int search_val = 7;
    int arr[] = {1, 1, 2, 5, 7, 9, 12, 13};
    int l = -1;
    int r = 1000000;
    while (l < r-1) {
        int mid = (l+r)/2;
        int mid_val = arr[mid];
        if (mid_val >= search_val) r = m;
        else                       l = m;
    }
    printf("%d\n", r);
}
```

There are many other ways of implementing binary search, although this is the one I was taught and the one I prefer. Also it's probably better to use a loop rather than recursion to avoid the overhead.

Implementation

CPMSOC

```
int l = 0, r = n-1;
while (l <= r) {
    int mid = (a+b)/2;
    if (array[mid] == search_val) {
        // search_val found at index mid
    }
    if (array[mid] > x)  r = mid-1;
    else                 l = mid+1;
}
```

Binary Search on Functions

CPMSOC

We have seen so far binary search applied to static sorted arrays. However, binary search can also be applied in non-obvious ways.

- Let `bool canDo(int x)` be a boolean monotone function.
  For example,

x	0	1	2	3	4	$\cdots$	$k-1$	k	$k+1$	$k+2$	$\cdots$
canDo(x)	0	0	0	0	0	$\cdots$	0	1	1	1	$\cdots$

- So `canDo(x)` is false for $x < k$ and true for $x \geq k$.
- We can then binary search for the value of k.
- Of course, the function can also go from true to false instead of false to true.

Implementation - Binary Search on Functions

CPMSOC

```
#include <cstdio>                        // The same as stdio.h, gives printf/scanf

bool canDo(int A) {
    return A >= 12;
}

int main() {
    int l = -1;
    int r = 1000000;
    while (l < r-1) {
        int mid = (l+r)/2;
        if (canDo(mid)) r = m;
        else            l = m;
    }
    printf("%d\n", r);
}
```

Example - Giants

CPMSOC

Problem Statement:
Your army consists of a line of N giants, each with a certain height. You must designate precisely $L \leq N$ of them to be leaders. Leaders must be spaced out across the line such that every pair of leaders must have at least $K \geq 0$ giants standing between them.

Find the maximum height of the shortest leader among all valid choices of L leaders.

Input:
First line 3 integers, N, L and K. The second line will contain N integers, H_i, the height of the ith leader.

Output:
A single integer with the maximum height of the shortest leader among all valid choices of L leaders.

Example - Giants

CPMSOC

Suppose $N = 10$, $L = 3$, $K = 2$, $H = [1, 10, 4, 2, 3, 7, 12, 8, 7, 2]$.

We want to choose 3 leaders out of 10 soldiers, such that there are at least 2 giants in between each leader.

Optimal choice: Pick the leaders with heights 10, 7 and 7.

Example - Giants

CPMSOC

- It's worth trying to attempt this with a brute-force/greedy/DP technique to appreciate why this problem is tricky.
- The question did not give constraints on N, L or K, although suppose they are sufficiently large enough so that a brute force solution is not practical.
- We need another plan. . .
- Binary search is very useful for problems that ask you to find the 'max of the min' or the 'min of the max' of something.
- We will need to split the problem into two variants: the *decision* problem and the *optimisation* problem.
- Typically the decision problem is *a lot* easier than the optimisation problem.

Example - Giants

CPMSOC

- Define the decision problem:
  `canDo(T)`: Does there exist some valid choice of leaders satisfying the constraints whose shortest leader has height *at least* T?
- The decision problem is a boolean function (either there exists a possible configuration, or not)
- It can be solved in $O(N)$ time by a straight-forward greedy algorithm.
  1. Let $L_{\text{count}} = 0$. This variable will store the amount of giants that we have selected to be leaders so far.
  2. Start off with the first element in H where $H_i \geq T$.
  3. Move to H_{i+K+1} and check if its value $\geq T$.
     - If it is, choose this giant as our soldier, and increment L_{count}. Move to the next giant K places down and repeat the above check.
     - Else, move to the next soldier 1 place down and repeat the above check
  4. Check if $L_{\text{count}} \geq L$.

Example - Giants

CPMSOC

- Note: We define it to be *at least* T (rather than having the shortest giant have height exactly T). This makes it monotone.
  - That is, the function looks like

    `1 1 1 1 ...  1 1 1 0 0 0 0 ...`
  - This makes intuitive sense as lower values of T allow more giants to be eligible to become leaders, so the required positions for leaders could be filled by more available 'qualified' giants.
  - And as we increase T, the number of eligible giants decreases, decreasing the available pool of possible leaders.
  - Eventually, we will reach a point where we will run out of leaders to meet the criteria.
- Then the optimisation problem is asking for the largest T such that `canDo(T)` is true.
- We can then just perform a binary search on T.
  - T are the heights of the soldiers, so we'll need a sorted list of the heights.
  - Create a sorted copy of H, and call this H'. This will take $O(N \log N)$ time.
  - Binary search on the values in H'. This will take $O(\log N)$ time.
- This takes $O(N \log N)$ time.

Example - Giants

CPMSOC

- For example, with

$$H = [1, 10, 4, 2, 3, 7, 12, 8, 7, 2]$$

  we have

$$H' = [1, 2, 2, 3, 4, 7, 7, 8, 10, 12]$$

- Is it possible for 4 to be the lower bound (but not necessarily the minimum height) of the giants? We can select 10, 7, 8. So yes.
- Is it possible for 8 to be the lower bound? Select the giants with height 10, 12, and ... there's no more giants. So it is not possible.
- Is it possible for 7 to be the lower bound? We can select 10, 7, 7. So yes.
- Thus, binary search gives us the answer of $T = 7$.
- Note binary search saved us from having to check all the values in H'.

Summary

CPMSOC

- Binary search can be surprisingly powerful when searching on non-obvious functions.
- Some questions can be solved by binary searching the answer and running a simulation for each of the possible answers to see if some activity is possible (the decision problem)
- Think about using binary search when you are asked to find (along the lines of) the largest/smallest x such that $f(x)$ is less than/greater than/equal to/... y.
- Other applications include finding zeroes of a function (interval bisection method).

Problems