Binary Search & its applications (Programming Club 4)

Tomasz Kosciuszko

12.02.2018

1 Introduction

Binary search, you may have heard about it, because it is one of the most basic algorithmic concepts. If you have not, do not worry, it is very easy to understand. Binary search is more less what you do when you are looking for a word in a dictionary. You choose a page at random and then you decide whether your word is further than that page and you repeat the action, until you have found what you were looking for.
This strategy is possible because words in a dictionary are sorted lexicographically. Unsurprisingly, we can apply Binary search on any monotonic structure, like a sorted list, increasing function or even the least amount of resources needed to complete a certain task.

2 Problem example - "Mafia"

Let's have a look at this problem's statement:
codeforces.com/problemset/problem/348/A

So we have a game of "Mafia". In every round we need one supervisor who does not take part in the game. For each person we know how many rounds they wish to participate in. We have to say what is the smallest number of rounds in which we can satisfy everyone. The problem looks easy, but at least for me it is not obvious how to dive into coding. We have an intuition that persons with smaller a_i can be supervisors more often, but it does not give us a straightforward solution.
Let's think in the following way: what if we knew the number of rounds to be played and wanted to determine whether it is enough. That is much easier to do, because we know how many supervisors we can afford. So let's try this strategy: we guess a number of rounds x. Then we check whether it is sufficient. If it is we know that the result must not be greater that x. Otherwise the result must be bigger then x. Sounds like Binary search? Here is part of the code that reads the input:

```
int n, a[100000];
int main() {
        cin >> n;
        for (int i = 0; i < n; i++) {
                cin >> a[i];
        }
```

Not very complicated, I am declaring $a[]$ to have 10^5 elements, because that is the biggest input possible.
Now, let's write the code that will deal with suggesting the number of rounds. I start here just with the knowledge that my answer is somewhere in the interval $[1, 2 \cdot 10^9]$. With each step I will cut the possible interval by half:

```
long long left = 1, right = 2000000000, middle;
while (left < right) {
        middle = (left + right) / 2;
        if (enough(middle)) {
                right = middle;
        } else {
                left = middle + 1;
        }
}
cout << left << "\n";
```

Ok, the only piece missing is the *enough*() function. For each player a_i, they will be free for $rounds - a_i$ rounds to supervise the game. Moreover if for any player $rounds < a_i$ we can return $false$. Otherwise the number of rounds is sufficient if we have at least as many supervisors as rounds:

```
bool enough(long long rounds) {
        long long supervisors = 0;
        for (int i = 0; i < n; i++) {
                if (a[i] > rounds) {
                        return false;
                }
                supervisors += rounds - a[i];
        }
        if (supervisors >= rounds) {
                return true;
        } else {
                return false;
        }
}
```

And that is all! The complexity of our solution is $O(n \log n)$, just enough for the given constraints.

3 Important remarks

- Whenever you notice a situation where something behaves like a monotonic function it may be a good opportunity to apply Binary search. But make sure that it really makes sense to use it!
- Binary search is a good way to cut your complexity from $O(n)$ to $O(n \log n)$ or from $O(n^2)$ to $O(n \log n)$, which can be a dramatic improvement!
- Sometimes you have to preprocess your data to run some Binary search on it, you may for example wish to sort a list before you do any searches.
- If you are not sure how to deal with **standard input and output** in your favourite language, have a look at some accepted solutions by other people: codeforces.com/problemset/status.

4 Practice problems

- Queries (easy): codeforces.com/problemset/problem/600/B
- Cowbells (easy): codeforces.com/problemset/problem/604/B
- Maximum Value (medium): codeforces.com/problemset/problem/484/B
- Exams (medium): codeforces.com/problemset/problem/732/D
- Levko and Array (hard): codeforces.com/problemset/problem/360/B

5 Hints

- Queries: Try to sort the array before you start answering the queries.
- Cowbells: Which cowbells are you going to transport just one per box?
- Maximum Value: Do you have time to iterate through all numbers divisible by a_i?
- Exams: If the number of days was fixed, would you know how to tell whether you have enough time?
- Levko and Array: Try Binary search + DP!