Solutions for 18th of July

Binary Search on Arrays 2

The problem is: Given a sorted array $a[1], a[2], \cdots, a[n]$ and some integers $x_1, x_2, \cdots, x_m$ to search, find the position of x_i in the array a (if exists)

Because the array a is sorted and strictly increasing (it's mentioned in the 'input format'), we can use a binary search for each x_i. You may use the code from the lecture slides:

```
int l = 0, r = n-1; // If x is in the array, it must be inside a[l..r]
while(l <= r) {
   int m = (l + r) / 2;
   if(x < a[m]) {
      r = m - 1;
   } else if(x > a[m]) {
      l = m + 1;
   }else { // we found it!
      printf("we found %d in position %d\n", x, m);
      break;
   }
}
```

One of the hard part was to read the input. The input was in a weird format, as it was in the problem 'Binary search on arrays 1'. The reasons we decided to give the input in that way are: (1) there were some problems on other regionals that required to read inputs in that way (2) we wanted to avoid solutions that read all inputs first and solve the problem.

Let's see how to calculate the sequence $x_1, x_2, \cdots, x_m$ from the input sequence $y_1, y_2, \cdots, y_m$. The definition was like:

- $x_1 = y_1$.
- For all $j \geq 2$, $x_j = (x_{j-1} + y_j) \bmod 10^9$ if x_{j-1} exists in array a, and $x_j = (x_{j-1} - y_j) \bmod 10^9$ otherwise.

So to know x_j we only need to know the result of the $(j-1)$-th query (in other words, the query that the program processed just before) and the value of x_{j-1}. So, we are going to store the of the last query result (if exists, store 1, otherwise, store 0)in a variable *p*, and store x_{j-1} in a variable *q*.

In the definition, x_1 is considered specially. (as there is no 0-th query) However, if we just assume *p* = 1 and *q* = 0, we can just use the second line of the definition without considering x_1 specially.

So, the code will be something like this:

```
int p = 1, q = 0;
for(int j = 1; j <= m; j++) {
  int yj; scanf("%d", &yj);
  int xj = (p == 1) ? (q + yj) % 1000000000 : (q - yj + 1000000000) %
1000000000;
  int result = my_binary_search(.., yj);
  p = (result != 0) ? 1 : 0;
  q = xj;
}
```

Cutting Sticks

This problem was taken from the lecture slides of today. You can refer to the solution in the lecture slides. (It's in the "Some problems can be solved by binary search on functions" part)

Wrong approach

Some students used approach like this:

1. Calculate the sum of the lengths of each sticks ($\sum a_i$)
2. Print $\left\lfloor \frac{\sum a_i}{k} \right\rfloor$.

This approach **doesn't work** for this problem. (although it works for the examples, sorry for that) Why?

In this problem, we are not allowed to *reuse* the sticks that doesn't have the same length. For example, if we fixed the length of the new stick as 5, and there are sticks with length 1, 2 and 2 left, we cannot make a additional stick with length 5 = 1 + 2 + 2. (It is mentioned in the statement)

However, if we just calculate the sum and divide it into k, we are actually 'reusing' the remaining sticks. (Just think that we connected all given sticks into one, and then cut it into k pieces)

EKO

Define $f(x)$ as the total number of meters of cutted wood if we set the 'height parameter' as x. From the definition of the problem, we can calculate $f(x)$ easily by the formula below:

$$f(x) = \sum_{i=1}^{n} \{a_i - \min(a_i, x)\}$$

And from the formula (and our intuition), we can find that $f(x)$ is a *monotonically decreasing function*, as we move the height parameter to the top, we must have smaller cut.

In the problem, we are asked to find the *maximum integer x such that* $f(x) \geq M$ ('maximum integer height of the sawblade that still allows him to cut off at least M metres of wood'). We can do this by just using a binary search on x.

This is the excerpt of the binary search part:

```
while (low <= high) {
    mid = (low + high) / 2;
    if (f(mid) >= M) {
        ans = mid;
        low = mid + 1;
    } else {
        high = mid - 1;
    }
}
```

Finding Roots 1

The problem was: Given four non-negative integers A, B, C, D such that $A^2 > B^2 + C^2$ and $D \geq C$, find the root of the equation $Ax + B \sin x + C \cos x = D$. It should be accurate to three points under the decimal point. $0 \leq A, B, C \leq 100$ and $D \leq 10{,}000$.

The first thing we should know is that the function is *strictly increasing*. This holds because:

- We can write $f(x) = Ax + B \sin x + C \cos x = Ax + \sqrt{B^2 + C^2} \sin(x + \alpha)$ where $\cos \alpha = \frac{B}{\sqrt{B^2+C^2}}$, $\sin \alpha = \frac{C}{\sqrt{B^2+C^2}}$.
- Differentiate f by x: $\frac{d}{dx} f(x) = A + \sqrt{B^2 + C^2} \cos(x + \alpha) \geq A - \sqrt{B^2 + C^2} > 0$

So we can write a simple binary search (on real numbers) to find the root of the equation, like this. I set the beginning interval as $[0,10050]$ because:

- $f(0) = C \leq D$. So the root must be non-positive.
- If $B = C = 0$, we have to solve the equation $Ax = D$. If $A = 1$ (smallest possible integer) and $D = 10000$ (largest possible integer, from the constraints), $x = 10000$. So the upper bound should be 10,000 or more.
- If $B^2 + C^2 \neq 0$, $\sqrt{B^2 + C^2} \sin(x + \alpha)$ may be negative and we need to increase x a little bit more than 10,000. However, as $A > \sqrt{B^2 + C^2}$, we don't need to increase x *that* much (actually at most 1, we guess) So we just set the limit as 10,050.

```
double low = 0, high = 10050.0;
for(int rep = 0; rep < 100; rep++) {
   double mid = (low + high) / 2.0;
   if(f(mid) <= D) {
       low = mid;
   }else {
       high = mid;
   }
}

double answer = (int)(low * 1000) / 1000.0;
printf("%.3f\n", answer);
```

This was our intention, but it turned out the problem was super hard than we expected! If you run this code on the judge server, you will get WRONG-ANSWER. Even though we repeated many times, it seems hard to achieve the desired precision.

So we need to use another way. As we only need three digits below the decimal point, we can just multiply 1,000 to that number and think of it as an integer. Then, we can run a similar binary search (on integers) like this:

```
int low = 0, high = 10050*1000, ans = -1;
while(low <= high) {
  int mid = (low + high) / 2;
  if(f(mid / 1000.0) <= D) {
    low = mid+1;
    ans = mid;
  }else {
    high = mid-1;
  }
}
printf("%d.%03d\n", ans / 1000, ans % 1000);
```

I used the condition “f(mid / 1000.0) **<=** D” because the value “mid / 1000.0” is obviously smaller than or equal to the actual root.