Josephus Problem

Table of Contents

Problem situation - Given the natural numbers n and k. All natural numbers from 1 to n are written in a circle. First count the k-th number starting from the first one and delete it. Then k numbers are counted starting from the next one and the k-th one is removed again, and so on. The process stops when one number remains. It is required to find the last number.

This task was set by **Flavius Josephus** in the 1st century (though in a somewhat narrower formulation: for $k = 2$).

This problem can be solved by modeling the procedure. Brute force modeling will work $O(n^2)$. Using a segment tree we can improve it to $O(n \log n)$. We want something better though.

Modeling a $O(n)$ solution

We will try to find a pattern expressing the answer for the problem $J_{n,k}$ through the solution of the previous problems.

Using brute force modeling we can construct a table of values, for example, the following:

$n \setminus k$	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	2	1	2	1	2	1	2	1	2	1
3	3	3	2	2	1	1	3	3	2	2
4	4	1	1	2	2	3	2	3	3	4
5	5	3	4	1	2	4	4	1	2	4
6	6	5	1	5	1	4	5	3	5	2
7	7	7	4	2	6	3	5	4	7	5
8	8	1	7	6	3	1	4	4	8	7
9	9	3	1	1	8	7	2	3	8	8
10	10	5	4	5	3	3	9	1	7	8

And here we can clearly see the following **pattern**:

$$J_{n,k} = (J_{(n-1),k} + k - 1) \mod n + 1$$

$$J_{1,k} = 1$$

Here, 1-indexing somewhat spoils the elegance of the formula; if you number the positions from 0, you get a very clear formula:

$$J_{n,k} = (J_{(n-1),k} + k) \mod n$$

So, we found a solution to the problem of Joseph, working in $O(n)$ operations.

Simple **recursive implementation** (in 1-indexing)

```
int josephus(int n, int k) {
    return n > 1 ? (joseph(n-1, k) + k - 1) %
}
```

Non-recursive form :

```
int josephus(int n, int k) {
    int res = 0;
    for (int i = 1; i <= n; ++i)
      res = (res + k) % i;
    return res + 1;
}
```

This formula can also be found analytically. Again here we assume 0-indexed. After we killed the first person, we have $n - 1$ people left. And when we repeat the procedure then we will start with the person that had originally the index $k \bmod m$. $J_{(n-1),k}$ would be the answer for the remaining circle, if we start counting at 0, but because we actually start with k we have $J_{n,k} = (J_{(n-1),k} + k) \mod n$.

Modeling a $O(k \log n)$ solution

For relatively small k we can come up with a better solution than the above recursive solution in $O(n)$. If k is a lot smaller than n, then we can kill multiple people ($\lfloor \frac{n}{k} \rfloor$) in one run without looping over. Afterwards we have $n - \lfloor \frac{n}{k} \rfloor$ people left, and we start with the $(\lfloor \frac{n}{k} \rfloor \cdot n)$-th person. So we have to shift by that many. We can notice that $\lfloor \frac{n}{k} \rfloor \cdot n$ is simply $n \bmod k$. And since we removed every k-th person, we have to add the number of people that we removed before the result index.

Also, we need to handle the case when n becomes less than k - in this case, the above optimization would degenerate into an infinite loop.

Implementation (for convenience in 0-indexing):

```
int josephus(int n, int k) {
    if (n == 1)
        return 0;
    if (k == 1)
        return n-1;
    if (k > n)
        return (joseph(n-1, k) + k) % n;
    int cnt = n / k;
    int res = joseph(n - cnt, k);
    res -= n % k;
    if (res < 0)
        res += n;
    else
        res += res / (k - 1);
    return res;
}
```

Let us estimate the **complexity** of this algorithm. Immediately note that the case $n < k$ is analyzed by the old solution, which will work in this case for $O(k)$. Now consider the algorithm itself. In fact, on each iteration of it, instead of n numbers, we get about $n\left(1-\frac{1}{k}\right)$ numbers, so the total number of x iterations of the algorithm can be found roughly from the following equation:

$$n\left(1-\frac{1}{k}\right)^x = 1,$$

on taking logarithm, we obtain:

$$\ln n + x \ln\left(1-\frac{1}{k}\right) = 0,$$

$$x = -\frac{\ln n}{\ln\left(1-\frac{1}{k}\right)},$$

using the decomposition of the logarithm into Taylor series, we obtain an approximate estimate:

$$x \approx k \ln n$$

Thus, the complexity of the algorithm is actually $O(k \log n)$.

Analytical solution for $k = 2$

In this particular case (in which this task was set by Josephus Flavius) the problem is solved much easier.

In the case of even n we get that all even numbers will be crossed out, and then there will be a problem remaining for $\frac{n}{2}$, then the answer for n will be obtained from the answer for $\frac{n}{2}$ by multiplying by two and subtracting one (by shifting positions):

$$J_{2n,2} = 2J_{n,2} - 1$$

Similarly, in the case of an odd n, all even numbers will be crossed out, then the first number, and the problem for $\frac{n-1}{2}$ will remain, and taking into account the shift of positions, we obtain the second formula:

$$J_{2n+1,2} = 2J_{n,2} + 1$$

We can use this recurrent dependency directly in our implementation. This pattern can be translated into another form: $J_{n,2}$ represents a sequence of all odd numbers, "restarting" from one whenever n turns out to be a power of two. This can be written as a single formula:

$$J_{n,2} = 1 + 2\left(n - 2^{\lfloor \log_2 n \rfloor}\right)$$

Analytical solution for $k > 2$

Despite the simple form of the problem and a large number of articles on this and related problems, a simple analytical representation of the solution of Joseph's problem has not yet been found. For small k, some formulas are derived, but apparently they are all difficult to apply in practice (for example, see Halbeisen, Hungerbuhler "the Josephus Problem" and Odlyzko, Wilf "Functional iteration and the Josephus problem").