Kick Start 2019 - Practice Round

Analysis: Kickstart Alarm

Kickstart Alarm: Analysis

The problem asks us to calculate the summation of power of each wakeup call: $POWER_1 + POWER_2 + \ldots + POWER_K$, where $POWER_i$ is just the summation of the i-th exponential-power of all the contiguous subarrays of the Parameter Array.

Small dataset

For Small dataset, we can iterate over every subarray of the given array and calculate the summation of $POWER_i$ for all $i \leq K$. Thus, the simplest brute solution will work for Small dataset.

Pseudocode for Small dataset:

```
result = 0
for(k in 1 to K) {
  for(L in 1 to N) {
    for(R in L to N) {
      for(j in L to R) {
        result = result + A[j] * pow(j-L+1,k)
        result %= 1000000007
      }
    }
  }
}
```

In the above pseudcode, we can precompute all the $pow(a, b)$ values for $1 \leq a \leq n$ and $1 \leq b \leq k$.
The overall time complexity is $O(N^3 \times K)$.

Large dataset

The above solution will not work for Large dataset. To solve for Large dataset, let's iterate over every position x and calculate the contribution by A_x to the result for all subarrays where this element is y-th element in the subarray.

- If $y > x$, there is no subarray such that A_x can be y-th element.
- For $y \leq x$, there is exactly one index where the subarray must start (i.e $y - 1$ places before x). Hence, all the subarrays starting at $(n - (y - 1))$ and ending on or after index x will have A_x at position y in the subarray. Therefore, the number of subarrays with element A_x at y-th position in the subarray will be $(n - x + 1)$.
  Contribution from this element as y-th element in one subarray = $A_x \times y^1 + A_x \times y^2 + \ldots + A_x \times y^K$.
  Let us denote with $S(x, y)$ as the contribution from this element as y-th element in all subarrays. Combining above observations, we can show that $S(x, y) = (n - x + 1) \times A_x \times (y^1 + y^2 + \ldots + y^K)$.
- $S(x, y) = 0$ for $y > x$.
- $S(x, y) = A_x \times K \times (n - x + 1)$ for $y = 1$.

- $S(x, y) = \frac{(n-x+1) \times A_x \times y \times (y^K - 1)}{(y-1)}$ for $y \leq x$ and $y > 1$.

Contribution by element at position x to the result (let us say $C(x)$) = $\sum S(x, y)$ for $1 \leq y \leq n$ $= (n - x + 1) \times A_x \times (K + \frac{2 \times (2^K - 1)}{(2-1)} + \frac{3 \times (3^K - 1)}{(3-1)} + \ldots \frac{x \times (x^K - 1)}{(x-1)})$.

So we can find the contribution by element at position x in $O(N \times \log(K))$. This gives us a $O(N^2 \times \log(K))$ solution to compute contribution of all the elements.

Let us define $G(x) = \frac{C(x)}{(A_x \times (n-x+1))} = K + \frac{2 \times (2^K - 1)}{(2-1)} + \frac{3 \times (3^K - 1)}{(3-1)} + \ldots \frac{x \times (x^K - 1)}{(x-1)}$.
Now if we look closely at $G(x)$ and $G(x+1)$, we can observe that
$G(x+1) = G(x) + \frac{(x+1) \times ((x+1)^K - 1)}{x}$.
Hence we can compute $G(x+1)$ from $G(x)$ in $O(\log(K))$ time. And subsequently $C(x+1)$.

Therefore the total time complexity = $O(N \times \log(K))$.

Pseudocode for Large dataset:

```
G[1] = K
C[1] = A[1] * K * n
result = C[1]
for(i in 2 to n){
  // Using the formula derived above to get G[i] from C[i-1]
  G[i+1] = G[i] + i * (i^K - 1) / (i - 1)
  C[i] = G[i] * A[i] * (n - i + 1)

  result = result + C[i]
  result %= 1000000007
}
```