Choose a description for participants:

✔ **System Crawler** 2018-05-31

B - Product on the segment by modulo

CodeChef - SEGPROD

Read problems statements in Mandarin chinese, Russian and Vietnamese as well.

Given an array **A** of **N** integers and an integer **P** (not necessarily prime). You are required to answer **Q** queries, **i**-th query is described by two numbers $\mathbf{L_i}$, $\mathbf{R_i}$, the answer to the query is the product of all numbers in array A from index $\mathbf{L_i}$ to index R_i modulo **P**. You have to answer the queries in online mode, that is you can't know a query before you answer all queries that are before it.

Input

the first line contains one integer **T** , the number of test-cases.

the first line of each test-case contains **N, P, Q** .

the third line contains **N** space-separated integers, the of elements array **A** .

the fourth line contains array **B** which consists of floor(Q/64) + 2 integers . Let's number the queries starting from 0. If **i** is divisible by 64 then:
$\mathbf{L_i = (B_{i\ /\ 64} + x) \bmod N}$, $\mathbf{R_i = (B_{i\ /\ 64\ +\ 1} + x) \bmod n}$.
Otherwise: $\mathbf{L_i = (L_{i-1} + x) \bmod N}$, $\mathbf{R_i = (R_{i-1} + x) \bmod N}$. If $\mathbf{L_i > R_i}$ you should swap them. here x is equal to 0 if it's the first query, otherwise it's equal to the answer of the previous query plus 1 modulo **P**.

Output

After each test-case you have to print one number - the value of x after the the last query modulo P

Constraints

- $\mathbf{1 \le T \le 100}$
- $\mathbf{1 \le N \le 10^6}$
- $\mathbf{1 \le Q \le 2 * 10^7}$
- $\mathbf{2 \le P \le 10^9}$

- $0 \leq A_i \leq 10^9$
- $0 \leq B_i \leq n - 1$
- The sum of **N** over all testcases does not exceed 10^6 and sum of **Q** over all testcases does not exceed $2 * 10^7$

Subtasks

- **Subtask 1** (20 points) : **P** is prime and all numbers of A in range [1; **P** - 1]
- **Subtask 1** (20 points) : $P = x^k$ where x is prime and k is integer and all numbers of A in range [1; **P** - 1]
- **Subtask 3** (60 points) : Original constraints

Example

Input:

```
2
6 113 3
1 2 3 4 5 6
1 4
6 113 70
1 2 3 4 5 6
1 4 3
```

Output:

```
8
```

All Copyright Reserved ©2018 Xu Han

Server Time: 2018-06-02 17:57:14 BST

PROBLEM:

Given a non-changing array, answer queries of the form "**? l r**: what is $\prod_{i=l}^{r} a_i$ modulo P (P is not necessarily prime)?" (subarray product query). The queries are online, and the constraints are so tight that an $O(1)$ per query solution is required.

QUICK EXPLANATION:

The intended solution does not utilize the fact that the query operation is product modulo P. The proposed data structure is capable of handling any associative operation query on subarray in $O(1)$ (online, provided that the array is not changing).

The idea of the intended data structured is somewhat similar to the idea of a sparse table: we will precompute the operation results on some subarrays of the array so that every input query subarray can be represented as a union of constant (at most 2) number of subarrays we already know the answer for. However, the difference with the classic sparse table (used for static RMQ problem) is that the given operation $\oplus$ is not idemptotent: $a \oplus a \neq a$ for the most of a's, so the union we will use must be disjoint. The operation is also not invertible, so we can't use the prefix-sums approach.

For each $k = 1, \dots, \lceil \log_2 N \rceil$ consider all indices of the array that are divisible by 2^k as "pivots". Precompute the product on subarrays whose left or right bound is a "pivot", and that do not contain any other pivots.

Claim: for each $[l, r]$ query we can choose pivot element $2^k \cdot x$ in such way that we already precomputed the operation for some subarray with indices $[l, 2^k \cdot x)$, $[2^k \cdot x, r]$.

EXPLANATION:

Build the following data structure: for each $k = 1, \dots, \lceil \log_2 N \rceil$, for each $i = 0, 1, \dots, N - 1$ (we use 0-indexation of the array) compute (we omit modulo P for simplicity; assume that all integers are members of the ring Z_P, with multiplication defined accordingly):

$$A_{k,i} = \prod_j a_j \text{ for } \lfloor \frac{i}{2^k} \rfloor \cdot 2^k \leq j \leq i$$

$$B_{k,i} = \prod_j a_j \text{ for } i \leq j \leq \lceil \frac{i+1}{2^k} \rceil \cdot 2^k - 1$$

Here $\sim$ denotes bitwise negation, $\&$ denotes bitwise AND operation.

The meaning of the expressions is the following:

1. $\lfloor \frac{i}{2^k} \rfloor \cdot 2^k$ is the largest number not exceeding i that is divisible by 2^k. Implementation-wise, this number I is characterized by the property $I \ \& \ 2^k = 0$ ($\&$ denotes bitwise AND).
2. $\lceil \frac{i+1}{2^k} \rceil \cdot 2^k - 1$ is by one less than the smallest number more than i that is divisible by 2^k. Implementation-wise, this number J is characterized by the property $J \& 2^k = 2^k - 1$ ($\&$ denotes bitwise AND).

Let's notice the following: suppose we have a query $[L, R], L \neq R$, and for selected k the range of indices $[L + 1, R]$ contains unique index I divisible by 2^k. Then

$$B_{k,L} \cdot A_{k,R} = \prod_{i=L}^{R} a_i$$

Indeed, if that index of the form $I = 2^k \cdot x \in [L + 1, R]$ exists and is unique, then, by the meaning of the corresponding expressions above:

$$I = \lfloor \frac{R}{2^k} \rfloor \cdot 2^k$$

$$I = \lceil \frac{L+1}{2^k} \rceil \cdot 2^k$$

And by definition of $B_{k,L}$ and $A_{k,R}$:

$$B_{k,L} \cdot A_{k,R} = \prod_{j=L}^{I} a_j \cdot \prod_{j=I-1}^{R} a_j = \prod_{j=L}^{R} a_j$$

Can we find the k for which I is unique? Surely, such k exists because if for some k there are $X > 1$ indices divisible by 2^k in the range $[L+1, R]$, for $k+1$ there are $\frac{X}{2}$ such indices if X is even and either $\lfloor \frac{X}{2} \rfloor$ or $\lceil \frac{X}{2} \rceil$ if X is odd. Incrementing k sufficient number of times we arrive to $X = 1$.

It remains to notice that k can be chosen as $k = \max_l : 2^l \leq L \oplus R$ ($\oplus$ denotes bitwise XOR).

AUTHOR'S AND TESTER'S SOLUTIONS:

Author's solution can be found here.

Tester's solution can be found here.

RELATED PROBLEMS:

nov17 medium-hard editorial segprod

This question is marked "community wiki".

asked **13 Nov '17, 14:09**

2★ melfice
71 ●7●24
accept rate: 0%

edited **14 Nov '17, 19:29**

admin ♦♦
18.5k ●348●495●529

---

11 Answers:

oldest answers newest answers **popular answers**

7

If you think of the array as a tree (chain), this is similar to centroid decomposition. Dividing the array(tree) into O(NlogN) subarrays(paths), in a way that any subarray can be represented as a concatenation of two of those O(NlogN) subarrays. If we append 1's to the array until its size becomes 2^k-1 and then perform the decomposition, the nodes at height 'h' will have 2^(k-1-h), (assuming root is at height 0) as the largest power of 2 that divides them. The LCA of two nodes l,r will be some number l<=i<=r of the form x*2^k as mentioned in the editorial. We need to decrease 'r' until it becomes a multiple of some power of 2 having power as large as possible. This is same as picking a '1' in 'r's binary representation and setting all bits to the right of it to '0'. But, the '1' we pick cannot be to the left of the first point of difference in the binary representations of l,r, because then 'r' would become less than 'l'. That is exactly what the largest set bit in (l XOR r) represents.

For example- Consider array of size 15, then 8 is the root with children 4 and 12 and so on. Consider the query from 5 to 7 and we get the required number by which splitting is done as 6. (their lca). Similar is the case with query 7 and 11 and lca as 8.

link

edited **15 Nov '17, 23:04**

6★ likecs
3.4k ●10●51

answered **15 Nov '17, 00:23**

6★ hemanth_1
1.3k ●9
accept rate: 26%

1 nice explanation!

6★ likecs (15 Nov '17, 23:05)

1 What a great explanation man , thanks a ton!!

4★ gagan86nagpal (16 Nov '17, 09:56)

1

can anyone give me the easy explaination as i am not understanding the editorial..

link

answered **15 Nov '17, 18:17**

2★ soumik33
11 ●1
accept rate: 0%

https://discuss.codechef.com/questions/116992/disjoint-sparse-table

4★ adhyyan1252 (16 Nov '17, 14:23)

1

@soumik33 See here,we precompute the prefix product for all indices which are divisible by 2^k where k=1....logN So

We calulate prefix product for all subarrays from prevPivot+1 to currPrivot-1. And prefix product from currPivot to nextpivot - 1

So lets say for k=2: Indices:4,8,12,16,20,... So we calculate for following ranges: (1,3),(2,3),(3,3) (4,4),(4,5),(4,6),(4,7)

(5,7),(6,7),(7,7) (8,8),(8,9),(8,10),(8,11)

We do this for k=1..logN Now suppose a query: 1-6 (Here pivot are 2,4) So it can be (1-1)*(2-3)*(4-6)

(Here pivot is 4) Or (1-3)*(4-6) The second one takes only 1 multiplication

Consider another eg: Range 17:22 So (17:19)*(20:22)

Since we have precalculated for these results for all multiples of power of 2 it can be done in O(1).

note 20=4 * 5=(2^2)*5

So it means we have to choose such a number in the range which is multiple of some 2^k and k is maximum(as if not we have to do more than 1 multiplication)

So lets express 17:0b10001 22:10110 Exor result=111 So last set bit is 2 So pivot is multiple of 2^2

(Also note that there is only 1 multiple x for x*2^k as for x-1 it is less than L and for x+1 it is greater than R)

Proof for xor: See @hemanth_1 comment.Really nice explaination.

Hope this helps. Plz correct me if i am wrong.

link edited **15 Nov '17, 23:14** answered **15 Nov '17, 22:32**

5★ vivek_1998299
1.2k ●2●8
accept rate: 22%

Hi vivek_1998299, I want to make this clear enough. Let's say we want to find I for (14,17) 14: 0b001110 17: 0b010001 XOR: 0b011111, HEre last bit set is at 4, hence 2^4 i.e. 16. Am i correct ?

4★ gagan86nagpal (16 Nov '17, 08:09)

Yes correct

5★ vivek_1998299 (16 Nov '17, 10:10)

1

@gagan86nagpal

It is same as i discussed in my comment. They have used to arrays A,B(2d arrays)

A(k,i) denotes product of all elements from largest value of 2^k * x (less than or equal to i) to i.So that value is floor(i/(2^k)) * 2^k (consider for i=5 ,k=2 so val=4,

for i=8,k=2. Val=8)

B(k,i) denotes product of all elements from i to (smallest value of 2^k * x which is just greater than i) - 1.That value is ceil((i+1)/2^k)*2^k (Consider i=5 k=2 so val=8 - 1=7

Consider i=8,k=2. So val=12 - 1 =11)

So as i said in prev comment : For k=2 and for i=4...N //for sake of confusion not considering from 1 (as 0 doesnt comes(1 based))

A(k,i)= [(4,4), (4,5) , (4,6) , (4,7) , (8,8) , (8,9) ,.....]

B(k,i)= [(4,7) , (5,7) , (6,7) , (7,7) , (8,11) , (9,11) ,...]

Hope this helps.

Plz correct me if i am wrong.

Sample Code:

```
#include<bits/stdc++.h>
using namespace std;

int tl[21][1000005];
int tr[21][1000005];
int a[1000005],b[1000005];

int main()
{
    ios_base::sync_with_stdio(0);

    int t; cin >> t;

    while (t--)
    {
        int n,p,q; cin >> n >> p >> q;

        for (int i = 0; i < n; ++i) cin >> a[i];

        int cnt = (q >> 6) + 2;
        for (int i = 0; i < cnt; ++i) cin >> b[i];

        /// Preprocess
        for (int k = 0; (1<<k)<=n ; k++)
        {
            int ones = (1<<k) - 1;

            long long tmp = 1;
            for (int i = 0; i < n; i++)
            {
                tmp *= a[i];
                tmp %= p;
                tl[k][i] = tmp;
                if ((i & ones) == ones) tmp = 1;
            }

            tmp = 1;
            for (int i = n - 1; i >= 0; i--)
            {
                tmp *= a[i];
                tmp %= p;
                tr[k][i] = tmp;
                if ((i & ones) == 0) tmp = 1;
            }
        }
```

```
        int prev=0, l=0, r=0;

        for (int i = 0; i < q; ++i)
        {
            if ((i%64) == 0){
                l = b[(i/64)] + prev;
                r = b[(i/64) + 1] + prev;
            }
            else{
                l += prev;
                r += prev;
            }

            l %= n; r %= n;
            if (l > r) swap(l, r);

            long long ans;
            if (l == r){
                ans = a[l];
            }
            else{
                int num = 31 - __builtin_clz(l ^ r);
                ans = tl[num][r];
                ans *= tr[num][l];
            }

            ans = (ans + 1) % p;
            prev = ans;
        }

        cout << prev << "\n";
    }
    return 0;
}
```

Codechef SEGPROD:

Given a non-changing array **a** of size n($1<=n<=10^6$), answer Q($1<=Q<=2*10^7$) queries of the form "? l r: what is:

$$\prod_{i=l}^{r} a_i \text{ modulo } P \ (P \text{ is not necessarily prime})?"$$

(subarray product query). The queries are online. and the constraints are so tight that an O(1) per query solution is required.

Problem Link: https://www.codechef.com/problems/SEGPROD

Ideas:

O(1) per query solution(Divide & Conquer optimization):

There is a way of building sparse table in so that you won't have to consider element twice.

You have to do something like Divide & Conquer optimization.

Consider that you now need to precalculate something for the interval [L,R] You should take and calculate answers for queries [M,M], [M,M + 1], ..., [M,R] Calculate same thing for queries [M - 1,M - 1], [M - 2,M - 1], ..., [L,M - 1]

Now you can answer any query [X,Y], such that $X \leq M \leq Y$ if you can merge answers for [X,M - 1] and [M,Y] in O(1).

Now we can deal with any query that consists M and we have to deal with intervals [L,M - 1] and [M + 1,R].

We have wasted O(R - L) calculation time, so whole recursion will be *O(NlogN).*

It is obvious that M that we are looking for must be such that A <= M <= B. Thus, it is obvious that leading bits these are equal in A and B must remain same in M. It it also obvious that if we were starting our recursion with L=0 and $R=2^k—1$, then M such that it has whole equal prefix of A and B with a single 1-bit right afterwards in the position where L and R were split in the first place.

The basic idea of this DS: Define M to be the midpoint of the array. 1/2 of all possible queries would be such that its Left endpoint is before M and its right endpoint is after M. So it would be efficient to keep prefix products from M to all other indices (even the ones before M). Any query which has a Left endpoint on the left of M and right endpoint on the right of M can be easily answered in O(1) using this prefix. Now the idea is to recursively do the same thing for the left and right half separately. That's why memory and pre-processing takes O(NlogN) time/mem.

This DS can subsitute anything a segment tree can do but in O(1) time. However, it takes more memory and doesn't support updates.

I think finding the index/level is the hardest part in this technique and it took me the longest time to figure out how to do it in O(1). I rounded the size up to the

nearest power of 2, because then all the mid points are like this: Lev 0- {1000}, Lev 1- {0100, 1100}, Lev 2- {0010, 0110, 1010, 1110} etc. Now by intuition I came up with a solution which can find the level in O(1). Define A = L XOR R. The most significant bit of A will tell you the level. C++ has a function which hopefully runs in O(1) to find the MSB, which is __builtin_clz().

My code: https://www.codechef.com/viewsolution/16141888

Sample Code:

```
#include<iostream>
using namespace std;

const int N = 1058576, LOGN = 22;
long long v[N][LOGN];
long long a[N], p, n, q;

void build(int i, int b, int e)
{
	if(b == e) return;
	int m = (b + e)/2;
	v[m][i] = a[m]%p;
	for(int j = m-1; j >= b; j--){
		v[j][i] = (v[j+1][i]*a[j])%p;
	}
	if(m + 1 <= e){
		v[m+1][i] = a[m+1]%p;
		for(int j = m+2; j <= e; j++){
			v[j][i] = (v[j-1][i]*a[j])%p;
		}
	}
	build(i+1, b, m);
	build(i+1, m+1, e);
}

int main()
{
	ios::sync_with_stdio(false); cin.tie(0);
	int t; cin>>t;
	while(t--)
	{
		cin>>n>>p>>q;

		for(int i = 0; i < n; i++) cin>>a[i];

		int LEV = __builtin_clz(n);
		int SZ = 1<<(31-LEV);
		if(n != SZ)LEV--, SZ *= 2;
```

```
            build(0, 0, SZ-1);

            int bl = q/64 + 2;
            long long b[bl];

            for(int i = 0; i < bl; i++) cin>>b[i];

            long long prev = -1;
            long long l, r;

            for(int i = 0; i < q; i++)
            {
                 if(i%64 == 0){
                      l = (b[i/64] + (prev+1)%p)%n;
                      r = (b[i/64 + 1] + (prev+1)%p)%n;
                 }
                 else{
                      l = (l + (prev+1)%p)%n;
                      r = (r + (prev+1)%p)%n;
                 }
                 if(l > r) swap(l, r);
                 if(l == r){
                      prev = a[l]%p;
                      continue;
                 }
                 unsigned int temp = __builtin_clz(l^r);

                 unsigned int lev = temp-LEV-1;
                 prev = (v[r][lev]*v[l][lev])%p;
             }

            cout<<(prev+1)%p<<endl;

            for(int i = 0; i < SZ; i++)a[i] = 0;
       }

       return 0;
  }
```

Here's the solution in O(logN) per query:

Note that you can do it like on a segment tree now, when you are in some node [L, R] and want to process interval [A,B] you need to either answer the query instantly if A <= (L+R)/2 <= B or whole interval [A,B] is in [L, (L+R)/2] or [((L+R)/2)+1, R] so you can move here to answer query thus having O(logN) time complexity.

We can not optimize segment tree enough to solve this problem. Here I used a different data structure which is a bit like segment tree and a bit like sparse table.

Sample Code:

```
#define FAST_IO ios_base::sync_with_stdio(false);cin.tie(0)
#define MAXN 1000000
using namespace std;

struct data{
     vector<int> left;         //  left[i]    -> [M-1-i, M-1]
     vector<int> right;        // right[i] -> [M, M+i]
};
int n, P, Q, A[MAXN],B[312502];
data ST[4*MAXN];

void build(int node=1, int l=0, int r=n-1)
{
     if(l == r){
          ST[node].right.assign(1, A[l]);
     }
     else
     {
          int m = (l+r)/2;
          ST[node].right.resize(r-m+1);
          ST[node].left.resize(m-l);

          ST[node].right[0] = A[m];
          for(int i=m+1,j=1;i<=r;i++,j++)
               ST[node].right[j] = ((long long)ST[node].right[j-1] * A[i])%P;

          if(m > l){
               ST[node].left[0] = A[m-1];
               for(int i=m-2,j=1;i>=l;i--,j++)
               ST[node].left[j] = ((long long)ST[node].left[j-1] * A[i])%P;
          }

          build(2*node, l, m);
          build(2*node+1, m+1, r);
     }
}
```

```
int query(int x, int y, int node=1, int l=0, int r=n-1)
{
     int m = (l+r)/2;
     if(x <= m && m <= y){
          long long ans = ST[node].right[y-m];
          if(x < m){
               ans *= ST[node].left[m - 1 - x];
               ans %= P;
          }
          return ans;
     }
     else if(l <= x && y <= m)
          return query(x, y, 2*node, l, m);
     else
          return query(x, y, 2*node+1, m+1, r);
}
int main()
{
     FAST_IO;
     int t; cin>>t;
     while(t--)
     {
          cin>>n>>P>>Q;
          for(int i=0;i<n;i++) cin>>A[i];
          for(int i=0, x = (Q>>6) + 2;i<x;i++) cin>>B[i];

          build();

          int x = 0, l, r;
          for(int i=0;i<Q;i++)
          {
               if(i%64 == 0){
                    l = (B[i>>6] + x) % n;
                    r = (B[(i>>6) + 1] + x) % n;
               }
               else{
                    l = (l + x) % n;
                    r = (r + x) % n;
               }
               if(l > r) swap(l, r);

               x = (query(l, r) + 1) % P;
          }
          cout<<x<<'\n';
     }
     return 0;
}
```