Editorial พีทซิลล่าอยากดัง (PZ_Camera)

ที่มา: ข้อสอบท้ายค่ายสองศูนย์ ม.บูรพา รุ่น 17

Problem Statement

พีทซิลล่าเป็นเจ้าของร้านดอกไม้ที่ได้วางดอกไม้เรียงไว้หน้าร้านของเขาเป็นเส้นตรงไว้จำนวน N ต้นโดยแต่ละต้นนั้นอาจจะมีสีแดง สีฟ้า สีเหลือง หรืออาจเป็นสีดำ เราจะกำหนดว่าเป็นสี C_1, C_2, C_3, ..., C_n ตามลำดับ

ในแต่ละวันพีทซิลล่าต้องการโปรโมทร้านของเขาโดยการจ้างช่างภาพมาถ่ายดอกไม้ที่จัดเรียงไว้จำนวน Q วัน วันละหนึ่งคน แต่ปัญหามันอยู่ที่ช่างภาพจะชอบถ่ายรูปดอกไม้ที่ตำแหน่งต่างกัน โดยช่างภาพคนที่ j จะชอบถ่ายดอกไม้เป็นช่วงที่ติดกันซึ่งอยู่ภายในช่วงดอกไม้ L_j ถึง R_j เท่านั้น

ตัวพีทซิลล่าเองก็ชอบให้ช่างภาพถ่ายดอกไม้หลายๆ สีเพื่อความหลากหลายในภาพ โดยเขาจะพอใจกับภาพดอกไม้ที่ถ่ายได้ก็ต่อเมื่อมีสับเซตของสีของดอกไม้ขนาดอย่างน้อย K ที่แต่ละคู่ของสีมีหมายเลขที่คูณกันแล้วไม่เป็นกำลังสองสมบูรณ์ทุกคู่

ตัวอย่างเช่นในกรณี K = 3 ถ้ารูปของช่างภาพมีดอกไม้สี 1, 3, 4, 2, 9, 6 จะสามารถทำให้พีทซิลล่าพอใจได้เพราะว่าในสีดังกล่าวมีสับเซต 1, 3, 2 และ 6 ที่มีขนาดเท่ากับ 4 ที่ทุกๆ คู่ของสีนั้นคูณกันแล้วไม่เป็นกำลังสองสมบูรณ์ทุกคู่ แต่ถ้าช่างภาพมีดอกไม้สี 1, 4, 18, 9, 2, 8 นั้นไม่สามารถหาสับเซตที่มีขนาดอย่างน้อย 3 ที่ทำให้ทุกๆ คู่ของสีนั้นคูณกันแล้วไม่เป็นกำลังสองสมบูรณ์ทุกคู่ได้ แต่สับเซตที่มีขนาดใหญ่ที่สุดก็คือ 1, 2 ที่มีขนาด 2 เท่านั้น

งานของคุณ

จงเขียนโปรแกรมเพื่อช่วยช่างภาพหาว่าสามารถถ่ายดอกไม้ให้พีทซิลล่าพอใจได้กี่แบบ

ข้อมูลนำเข้า

บรรทัดแรก รับจำนวนเต็มบวก 3 จำนวน คือ N, Q และ K โดยที่ 1 <= N, Q, K <= 100,000

บรรทัดที่สอง รับจำนวนเต็มบวก N จำนวน คือ C_1, C_2, C_3, ..., C_n

โดยที่ $1 <= C_1, C_2, C_3, ..., C_n <= 10^6$

อีก Q บรรทัดต่อมา รับจำนวนเต็มบวก 2 จำนวน คือ L_j และ R_j โดยที่ $1 <= L_j < R_j <= N$

ข้อมูลส่งออก

มี Q บรรทัด โดยแต่ละบรรทัดให้แสดงจำนวนวิธีในการถ่ายดอกไม้ให้พีทซิลล่าพอใจในแต่ละวัน

ตัวอย่าง

ข้อมูลนำเข้า	ข้อมูลส่งออก
10 3 3 1 1 2 3 2 4 1 4 1 3 2 4 3 7 2 2	1 4 0

คำอธิบายตัวอย่าง

มี 3 วันในการถ่ายรูปดอกไม้

วันแรก มีช่วงที่ช่างภาพทำให้พีทซิลล่าพอใจได้เพียงช่วงเดียวคือถ่ายดอกไม้ดอกที่ 2, 3, 4

วันที่สอง มีช่วงที่ช่างภาพทำให้พีทซิลล่าพอใจได้ 4 ช่วง ได้แก่ (4, 5, 6), (4, 5, 6, 7), (3, 4, 5, 6) และ (3, 4, 5, 6, 7)

วันที่สาม ไม่มีช่วงใดเลยที่ทำให้พีทซิลล่าพอใจได้

แนวคิด

ปัญหาของเราในข้อนี้คือต้องหา จำนวนช่วงที่ติดกันของตัวเลขภายในช่วง L_j และ R_j ที่มีเงื่อนไขหนึ่ง โดยเงื่อนไขนั้นคือ มีสับเซตที่ไม่จำเป็นต้องติดกันขนาดมากกว่าหรือเท่ากับ K ที่ไม่มีสมาชิกคู่ใด ๆ มีผลคูณเป็นกำลังสองสมบูรณ์

ขั้นตอนแรก เราจึงพิจารณาเงื่อนไขของจำนวนสองจำนวนที่มีผลคูณเป็นกำลังสองสมบูรณ์ โดยจะยกตัวอย่างเป็น $m = p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k}$ และ $n = p_1^{b_1} p_2^{b_2} \ldots p_k^{b_k}$ โดยให้ ลำดับ $\{p_i\}_{i=1}^{k}$ แทนลำดับของจำนวนเฉพาะทั้งหมดที่เป็นตัวประกอบของ m หรือ n จะได้ว่า $mn = p_1^{a_1+b_1} p_2^{a_2+b_2} \ldots p_k^{a_k+b_k}$

ซึ่งถ้า mn เป็นจำนวนกำลังสองสมบูรณ์ จะได้ว่า $a_i + b_i, \forall i \in \{1,2,\ldots,k\}$ เป็นจำนวนคู่ทั้งหมด ซึ่งหมายความว่าความเป็นคู่คี่ของเลขชี้กำลังของทุกจำนวนเฉพาะทั้งหมดที่เป็นตัวประกอบของ m หรือ n ของทั้ง m และ n นั้นเหมือนกัน กล่าวคือ $\forall i \in \{1,2,\ldots,k\}$ จะได้ว่า a_i และ b_i มีความเป็นคู่คี่เหมือนกัน

ดังนั้น ในการสังเกตว่าจำนวนสองจำนวนใด ๆ เป็นจำนวนที่คูณกันแล้วเป็นจำนวนกำลังสองสมบูรณ์หรือไม่ เราจะทำการหา $mp(M)$ แทนจำนวนเต็มบวก M ที่เปลี่ยนเลขชี้กำลังของจำนวนเฉพาะที่เป็นตัวประกอบทุกตัวเป็น 0 หรือ 1 แทนความเป็นคู่คี่ของเลขชี้กำลังนั้น ตัวอย่างเช่น

ถ้า $M = 108 = 2^2 \cdot 3^3$ จะได้ว่า $mp(M) = mp(108) = mp(2^2 \cdot 3^3) = 2^0 \cdot 3^1 = 3$

ถ้า $M = 128 = 2^7$ จะได้ว่า $mp(M) = mp(128) = mp(2^7) = 2^1 = 2$

ถ้า $mp(M)$ ของจำนวนเต็มบวกสองจำนวนเท่ากันหมายความว่าเป็นจำนวนที่คูณกันแล้วเป็นจำนวนกำลังสองสมบูรณ์ โดย Algotithm ในการหา $mp(M)$ สามารถทำได้หลายวิธี

วิธีที่หนึ่ง เราจะทำการหาร M ที่เราต้องการหา $mp(M)$ ด้วย j^2 สำหรับทุก $j \le \lfloor \sqrt{M} \rfloor$ โดยมี code ดังนี้

```
int mp(int M){
    int j;
    for(j=sqrt(M);j>=2;j--){
        if(M%(j*j)==0) M/=(j*j);
    }
    return M;
}
```

$Time\ Complexity: O(\sqrt{M})$

โดยเมื่อมีจำนวนเต็มบวกทั้งหมด N จำนวนจึงมี $Time\ Complexity: O(N\sqrt{M})$ ซึ่งอาจใช้เวลานานเกินไป

วิธีที่สอง เนื่องจากวิธีที่หนึ่ง อาจใช้เวลานานเกินไปเราจึงใช้ memoization ในการเก็บค่า $mp(M)$ ให้สามารถหาค่า $mp(M)$ ภายใน $O(1)$ สำหรับทุก ๆ จำนวนเต็มบวก M ที่เป็นไปได้ คือ 1 <= M <= 10^6

โดยมี code ดังนี้

```
void findmp(){
    int i,j;
    memset(mp,0,sizeof mp);
    for(i=1;i<=1000000;i++){
        if(mp[i]==0){
            for(j=1;i*j*j<=1000000;j++) mp[j*j*i]=i;
        }
    }
}
```

$Time\ Complexity: O(\max(M))$ และตอบค่า $mp(M)$ ภายใน $O(1)$ ในแต่ละครั้ง

เมื่อเราสามารถหา $mp(M)$ ได้แล้วเราจึงเปลี่ยนจำนวนเริ่มต้นเป็นค่า $mp(M)$ ของทุก ๆ จำนวนแล้วจะแก้ปัญหาหลักของโจทย์นั่นก็คือ หาจำนวนช่วงที่ติดกันของตัวเลขภายในช่วง L_j และ R_j ที่มีเงื่อนไขคือมีสับเซตที่ไม่จำเป็นต้องติดกันขนาดมากกว่าหรือเท่ากับ K ที่ไม่มีสมาชิกคู่ใด ๆ มี $mp(M)$ เท่ากัน เราอาจจะเปลี่ยนข้อความนี้เป็น หาจำนวนช่วงที่ติดกันของตัวเลขภายในช่วง L_j และ R_j ที่มี $mp(M)$ ที่แตกต่างกันอย่างน้อย K ค่า โดยมีการตอบคำถามนี้จำนวน Q ครั้ง

เราจะใช้ dynamic programming โดยจะนิยาม $dp[i]$ แทน index ของ array เริ่มต้น j ที่น้อยที่สุดที่ทำให้ค่า $mp(M)$ ในช่วง $i,\ i+1,\ i+2,\ ...,\ j$ มีค่าที่แตกต่างกันอย่างน้อย K ค่า โดยวิธีการหา $dp[i]$ มีวิธีการหาได้หลายวิธีดังนี้

วิธีที่หนึ่ง เราจะพิจารณาตั้งแต่ $j=i,\ i+1,\ i+2,\ ...,\ N$ โดยเริ่มต้นเราจะใช้ `std::unordered_map<int,int>` ในการตรวจสอบว่าที่เราเคยพิจารณาไปแล้วมี $mp(M)$ ที่มีค่าแตกต่างกันทั้งหมดกี่ค่า โดยใช้ ฟังก์ชัน `size()` หรือเราจะใช้ array ขนาดเท่ากับค่าที่มากที่สุดที่เป็นไปได้ของ M ซึ่งในข้อนี้คือ 10^6 ในการเก็บว่ามี M กี่จำนวนที่มีค่า $mp(M)$ เป็น i โดยในวิธีนี้จะมี $Time\ Complexity: O(N^2)$ ซึ่งใช้เวลามากเกินไป

วิธีที่สอง เราจึงใช้ความเป็นฟังก์ชันทางเดียวหรือ monotonic function ในการพิจารณาว่าถ้า $j>i$ แล้ว $dp[j]\ge dp[i]$ เนื่องจากถ้า $dp[i]$ แทน index ของ array เริ่มต้นที่น้อยที่สุดที่ทำให้ค่า $mp(M)$ ในช่วง $i,\ i+1,\ i+2,\ ...,\ dp[i]$ มีค่าที่แตกต่างกันอย่างน้อย K ค่า ซึ่งเมื่อ $j>i$ จะทำให้ค่า $mp(M)$ ที่แตกต่างกันในช่วง $j,\ j+1,\ j+2,\ ...,\ dp[i]$ นั้นมีไม่เกินค่า $mp(M)$ ที่แตกต่างกันในช่วง $i,\ i+1,\ i+2,\ ...,\ dp[i]$ เราจึงได้ว่า $dp[j]\ge dp[i]$

นั่นคือในการพิจารณา $dp[i]$ เราจะเริ่มพิจารณาตั้งแต่ $dp[i-1],\ dp[i-1]+1,\ ...,\ N$ แทน ซึ่งสามารถทำให้เราพิจารณา j ไม่เกิน N ครั้ง โดยมีตัวอย่าง code ดังนี้

```
j=0;
for(i=1;i<=n;i++){
    while(j<=n&&cnt<k){
        j++;
        mark[c[j]]++;
        if(mark[c[j]]==1) cnt++;
    }
    dp[i]=j;
    if(mark[c[i]]==1) cnt--;
    mark[c[i]]--;
}
```

โดยในวิธีนี้จะมี $Time\ Complexity: O(N)$

เมื่อเราได้ $dp[i]$ แล้วเราจึงสามารถนำไปตอบคำถามจำนวน Q คำถาม โดยที่แต่ละปัญหาย่อยจะมีวิธีทำต่างกันดังนี้

ปัญหาย่อยที่ 1 จะมี Q <= 5

เราจะทำการพิจารณาทุก ๆ index i ตั้งแต่ L, L+1, L+2, ... , R โดยพิจารณาดังนี้

กรณีที่ 1 ถ้า $dp[i]\le R$ จะได้ว่า เราสามารถเลือกช่วงที่ติดกันเริ่มที่ i ไปยัง $dp[i]$ ถึง R รวม $R-dp[i]+1$ ช่วง

กรณีที่ 2 ถ้า $dp[i]>R$ จะได้ว่า เราไม่สามารถเลือกช่วงที่ติดกันได้ รวม 0 ช่วง

ตัวอย่าง code เป็นดังนี้

```
answer=0;
for(i=l;i<=r;i++){
    if(dp[i]<=r) answer+=(r-dp[i]+1);
}
printf("%lld\n",answer);
```

$Time\ Complexity: O(N)$ ต่อคำถาม

ปัญหาย่อยที่ 2 จะมี Q <= 100,000

เนื่องจากความเป็น monotonic function ของ $dp[i]$ จะทำให้เราสามารถหา index i ที่น้อยที่สุดที่มี $dp[i] > R$ ได้

ถ้า $i \le L$ จะได้ว่า สำหรับทุก ๆ i ตั้งแต่ L, L+1, L+2, ... , R ไม่มี $dp[i] \le R$ หมายความว่าไม่มีช่วงที่ติดกันที่ต้องการ

ถ้า $i > L$ จะได้ว่า ตั้งแต่ $L,\ L+1,\ ...,\ i\text{-}1$ มี $dp[i] \le R$ ซึ่งทำให้มีช่วงที่ต้องการทั้งหมด $\sum_{j=L}^{i-1}\left(R - dp[j] + 1\right)$ ช่วง

ซึ่งถ้าเราพิจารณาตั้งแต่ $L,\ L+1,\ ...,\ i\text{-}1$ เช่นเดียวกับปัญหาย่อยที่ 1 จะทำให้มี $Time\ Complexity:\ O(N)$ ซึ่งช้าเกินไป

เราจึงเก็บค่า $qs[i]$ ซึ่งนิยามเป็นจำนวนช่วงที่ต้องการทั้งหมดเมื่อเริ่มตั้งแต่ index $i,\ i+1,\ ...,\ N$

อาจเขียนเป็นสมการคือ $qs[i] = \sum_{j=i}^{N}\left(N - dp[i] + 1\right)$

ทำให้เราได้ความสัมพันธ์ในการหา $qs[i]$ คือ $qs[i] = qs[i+1] + N - dp[i] + 1$

นั่นคือเราสามารถลดขั้นตอนการคำนวณจำนวนช่วงที่ต้องการทั้งหมดให้อยู่ใน $Time\ Complexity:\ O(1)$ ได้โดย

$$\sum_{j=L}^{i-1}\left(R - dp[j] + 1\right) = \sum_{j=L}^{i-1}\left(N - dp[j] + 1 - N + R\right) = \sum_{j=L}^{i-1}\left(N - dp[j] + 1\right) - \sum_{j=L}^{i-1}\left(N - R\right) = qs[L] - qs[i] - (N - R)\cdot(i - L)$$

ตัวอย่าง code เป็นดังนี้

```
i=upper_bound(dp+1,dp+n+1,r)-dp;
if(i<=l) answer=0;
else answer=qs[l]-qs[i]-(n-r)*(i-l);
printf("%lld\n",answer);
```

$Time\ Complexity:\ O(\log N)$ ต่อคำถาม

Full Source Code

```
#include<bits/stdc++.h>
using namespace std;
#define N 100010
#define MAXM 1000010
long long c[N],dp[N],mark[MAXM],qs[N],mp[MAXM];
int main(){
    long long n,q,i,j,k,cnt=0,l,r,answer=0;
    scanf("%lld %lld %lld",&n,&q,&k);
    for(i=1;i<=1000000;i++){
        if(mp[i]==0){
            for(j=1;i*j*j<=1000000;j++) mp[j*j*i]=i;
        }
    }
    for(i=1;i<=n;i++){
        scanf("%lld",&c[i]);
        c[i]=mp[c[i]];
    }
    j=0;
    for(i=1;i<=n;i++){
        while(j<=n&&cnt<k){
            j++;
            mark[c[j]]++;
            if(mark[c[j]]==1) cnt++;
        }
        dp[i]=j;
        if(mark[c[i]]==1) cnt--;
        mark[c[i]]--;
    }
```

```
    for(i=n;i>=1;i--){
        if(dp[i]<=n) qs[i]=qs[i+1]+n-dp[i]+1;
    }
    while(q--){
        scanf("%lld %lld",&l,&r);
        i=upper_bound(dp+1,dp+n+1,r)-dp;
        if(i<=l) answer=0;
        else answer=qs[l]-qs[i]-(n-r)*(i-l);
        printf("%lld\n",answer);
    }
return 0;
}
```

Time Complexity : $O(\max(M)+N+Q\log N)$