****

**Demux Summer Hackathon -4**

Solution

**1.The Power Sum**

The obvious brute-force solution for this problem is simply to enumerate all possible ways to express **X**  as a sum of distinct th powers.

A good approach here would be to use recursive backtracking, a good general-purpose technique for enumerating things.

By the way, one must be careful not to double count equivalent expressions. For instance, this problem considers **103+93** and **93** +**103**  to be the same sum. (The explanation section clarifies this.) We should make sure we generate each expression only once.

How can we check if two expressions are the same? Well, one way is simply to sort their summands and compare term by term. This works because for every expression, there is a unique equivalent expression whose terms are in sorted order. This suggests that we should only look at sorted expressions. Thus, it's sufficient to just ensure that the list of numbers we're generating is always in sorted order, i.e., increasing.

**Code:**

#include <iostream>

#include <vector>

using namespace std;

int ipow(int b, int e) {

if (e == 0)

return 1;

return e == 0 ? 1 : b \* ipow(b, e - 1);

}

int count\_expressions(int x, int n, vector<int>& vals) {

int s = 0;

for (int v : vals) {

s += ipow(v, n);

}

if (s == x) {

return 1;

} else {

int answer = 0;

int v = vals.empty() ? 1 : vals.back() + 1;

while (s + ipow(v, n) <= x) {

vals.push\_back(v);

answer += count\_expressions(x, n, vals);

vals.pop\_back();

v++;

}

return answer;

}

}

int main() {

int x, n;

cin >> x >> n;

vector<int> vals;

cout << count\_expressions(x, n, vals) << endl;

}

**JAVA:**

import java.util.Scanner;

public class PowerSum {

int count=0;

int sum;

int pow;

public static void main(String[] args) {

Scanner in=new Scanner(System.in);

int x=in.nextInt();

int n=in.nextInt();

PowerSum p=new PowerSum();

p.sum=x;

p.pow=n;

int N=(int)Math.pow(x, (1.0/n));

p.getcount(p.sum,N,true);

p.getcount(p.sum,N,false);

System.out.println(p.count);

in.close();

}

void getcount(int sum1,int n,boolean lenyani){

if(lenyani==true){

sum1=sum1-(int)Math.pow(n, pow);

}

if(sum1<0) return;

if(sum1==0){

count++;

return;

}

if(n==1) return;

getcount(sum1,n-1,true);

getcount(sum1,n-1,false);

}

}

# 2. Crossword Puzzle

Recursively, try every possible valid position to place a word.

Suppose we have **k** words, then start from first word and try to place it at valid position in the grid, either horizontally or vertically, and then move ahead with the updated grid for the next word. If for any word there is no valid position found, then backtrack and restore the previous grid(i.e, grid before when last word is not placed) and so on.

If we place all the words in the grid for any configuration, then print the grid as an answer and terminate the resursion.

**Code:**

#include <bits/stdc++.h>

using namespace std;

vector<string> grid(10);

vector<string> words;

bool f;

void call(int ind)

{

    if(!f) {

        return;

    }

    if(ind == words.size()) {

        if(f) {

            for(auto word: grid) {

                cout<<word<<endl;

            }

            f=false;

        }

        return;

    }

    int i,j,p,q,k;

    for(i=0;i<10;++i) {

        for(j=0;j<10;++j) {

            p=i,q=j;

            for(k=0;k<words[ind].size() && p+k<10;++k) {

                if(grid[p+k][q] != '-' && grid[p+k][q] != words[ind][k]) {

                    break;

                }

            }

            if(k==words[ind].size()) {

                vector<string> temp = grid;

                for(k=0;k<words[ind].size();++k) {

                    grid[p+k][q] = words[ind][k];

                }

                call(ind+1);

                grid = temp;

            }

            for(k=0;k<words[ind].size() && q+k<10;++k) {

                if(grid[p][q+k] != '-' && grid[p][q+k] != words[ind][k]) {

                    break;

                }

            }

            if(k==words[ind].size()) {

                vector<string> temp = grid;

                for(k=0;k<words[ind].size();++k) {

                    grid[p][q+k] = words[ind][k];

                }

                call(ind+1);

                grid = temp;

            }

        }

    }

}

int main()

{

    f=true;

    int i,j;

    for(i=0;i<10;++i) {

        cin>>grid[i];

    }

    string s,w;

    cin>>w;

    for(auto x: w) {

        if(x==';') {

            words.push\_back(s);

            s="";

        } else

            s+=x;

    }

    words.push\_back(s);

    call(0);

    return 0;

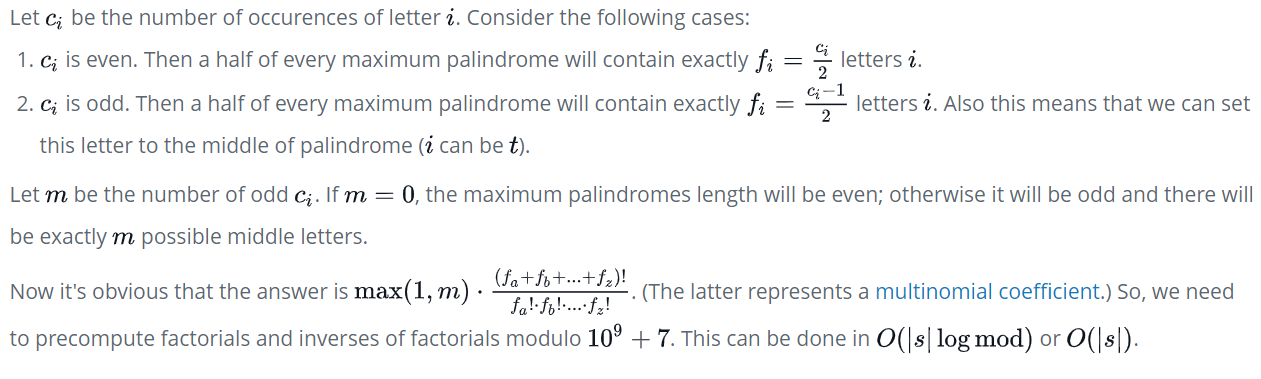
}

## 3. Maximum Palindromes

## Suppose we want to answer a query [l,r]. For each letter, we need to know the number of its occurrences. We can calculate it in  O(1) time using prefix sums.

We can bisect every palindrome. If its length is odd, we have "remainder" — the middle letter.

We want to know the number of pairs of strings(w,t) , where w is half of some maximum palindrome, and  is the palindrome remainder. Note that **t**  is an empty string if the maximum palindromes length is even. The number of these pairs will be the answer because we can build palindrome  **w+t +reversed(w)**and, vice versa, we can build such pair by the maximum palindrome.



#include <iostream>

#include <memory.h>

using namespace std;

typedef long long ll;

const int N = 100001;

const int A = 'z' - 'a' + 1;

const ll MOD = (ll) 1e9 + 7;

ll power(ll x, ll y) {

if (y == 0) {

return 1;

}

if (y & 1) {

return power(x, y - 1) \* x % MOD;

} else {

ll tmp = power(x, y / 2);

return tmp \* tmp % MOD;

}

}

ll fact[N], rfact[N];

int n, q;

char s[N];

int cnt[N][A];

ll calc(int l, int r) {

int sum = 0;

int odd = 0;

ll res = 1;

for (int i = 0; i < A; i++) {

int cur = cnt[r][i] - cnt[l - 1][i];

sum += cur / 2;

res = res \* rfact[cur / 2] % MOD;

if (cur % 2 == 1) {

odd++;

}

}

res = res \* max(1, odd) % MOD;

res = res \* fact[sum] % MOD;

return res;

}

int main() {

fact[0] = 1;

rfact[0] = 1;

for (int i = 1; i < N; i++) {

fact[i] = fact[i - 1] \* i % MOD;

rfact[i] = power(fact[i], MOD - 2);

}

memset(cnt, 0, sizeof cnt);

scanf("%s %d", s, &q);

n = strlen(s);

for (int i = 0; i < n; i++) {

cnt[i + 1][s[i] - 'a']++;

}

for (int i = 0; i <= n; i++) {

for (int j = 0; j < A; j++) {

cnt[i][j] += cnt[i - 1][j];

}

}

for (int i = 0; i < q; i++) {

int l, r;

scanf("%d%d", &l, &r);

printf("%d\n", (int) calc(l, r));

}

}

## **4.Goodland Electricity**

Let's think about what an optimal answer looks like. If the index of the leftmost tower is not **<K** , then one or more of the leftmost cities will never be able to have electricity. Bearing this in mind, we greedily choose the rightmost city with a tower having an index  **<K** which helps us minimize the number of towers that need to be switched on.

We need to iterate through the towers from **0  to n-1**, switching on towers. Let x  be last tower that was switched on thus far during the traversal. The next tower's index must **be <x+k+k**, and (for the same reasons as in the first paragraph) we want to choose the rightmost tower satisfying that inequality. We find this using a *two pointers* approach. Finally, if **x+k>n**  then we don't need to take another tower.

**Note:** Don't forget to be mindful of **-1**  cases!

The time complexity for this is **O(n).**

### C++

#include <bits/stdc++.h>

using namespace std;

const int N = 3e5 + 5;

int n, k, p, prv[N];

int main() {

    scanf("%d %d", &n, &k);

    int last = -1, ans = 0;

    for(int i = 0; i < n; i++) {

        scanf("%d", &p);

        if(p == 1) last = i;

        prv[i] = last;

    }

    for(int i = 0; i < n; ) {

        int take = prv[min(i + k - 1, n - 1)];

        if(take == -1 || take + k <= i) { printf("-1\n"); return 0; }

        i = take + k; ans++;

    }

    printf("%d\n", ans);

    return 0;

}

# 5. Gaming Array

## **Observations**

* All the numbers are distinct, so at any stage, we know exactly which index the current player must choose.
* We can determine the winner by finding the number of moves needed to finish the game. Let's call this number **k**. If **k**  is even, ANDY will win the game, otherwise BOB will win the game.

## **Calculating the Number of Moves**

Some index **i** is chosen as the maximum element at some stage of the game if there is no index **j** such that **j<i  and aj>ai**.

## **Solution**

We can solve this by scanning the array from left to right and maintaining a variable that simulates the number of moves (we'll call it **k**). Whenever you get an integer that's larger than the current running maximum, increment the value of k  by 1 . Once we've iterated through the entire array, we simply need to check whether k is odd or even. If there were an odd number of moves, then Bob wins; if there were an even number of moves, then Andy wins.

The complexity of this is O(n).

**C++:**  
  
#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int t;

cin >> t;

while (t--) {

int n;

cin >> n;

int k = 0, x = 0, y;

for(int i=0;i<n;i++){

cin >> y;

if (x < y) {

k++;

x = y;

}

}

if (k%2 == 1)

cout << "BOB\n";

else

cout << "ANDY\n";

}

return 0;

}

**Java:**  
import java.util.\*;

public class Solution {

/\*\* @return The total number of moves in the game. \*\*/

public static int solve(int[] a) {

int currentMax = 0;

int count = 0;

for (int i = 0; i < a.length; i++) {

if (a[i] > currentMax) {

currentMax = a[i];

count++;

}

}

return count;

}

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int g = in.nextInt();

for(int a0 = 0; a0 < g; a0++){

int n = in.nextInt();

int[] a = new int[n];

for(int i = 0; i < n; i++) {

a[i] = in.nextInt();

}

int turns = solve(a);

System.out.println( ((turns & 1) == 0) ? "ANDY" : "BOB" );

}

in.close();

}

}

## 6. Array Manipulation

You are given a list of size n, initialized with zeroes. You have to perform m queries on the list and output the maximum of final values of all the n elements in the list. For every query, you are given three integers a, b and k and you have to add value k to all the elements ranging from index a to b(both inclusive).

**Sub-Optimal Brute Force:**

Given each **update(a,b,k)** , for each index in the range from **[a,b ],** add the value **k** to each number in the range.

The final step is to go through the whole array and find the maximum value and print that maximum value.

The complexity of this solution is **O(n.m)** which is too high to pass in time.

**Optimal:**

* Given a range[a,b ] and a value  **k**we need to add  to all the numbers whose indices are in the range from [a,b ].
* We can do an O(1) update by adding **k** to index  **a** and add  -**k** to index (b+1).
* Doing this kind of update, the **ith** number in the array will be prefix sum of array from index 1 to i because we are adding **k** to the value at index **a** and subtracting **k** from the value at index **b+1** and taking prefix sum will give us the actual value for each index after **m** operations .
* So, we can do all **m** updates in O(m) time. Now we have to check the largest number in the original array. i.e. the index i such that prefix sum attains the maximum value.
* We can calculate all prefix sums as well as maximum prefix sum in O(n) time which will execute in time.

**Optimal:**

* This can be further optimized to run in O(m logm) time because we have to check the value of prefix sum at only **2\*m**  indices. i.e. a and b values of all the updates.
* We have, in total **m** queries and each query has a range [a,b ] which needs to be updated. So, in total we have **2\*m** indices.
* For each query, we can insert both a,k  and b+1, -k in an array and sort the array.
* Now, we have to just take the prefix sum of the array and find the maximum element which will be our answer.

**Note:** If you thought of solving it using segment tree with lazy propogation, it won't pass here as **n**  can be as high as **107**.

Code:

#include <bits/stdc++.h>

using namespace std;

long a[400009];

int main() {

int n;

int m;

cin >> n >> m;

vector < pair < int, int > > v;

for(int a0 = 0; a0 < m; a0++) {

int a;

int b;

int k;

cin >> a >> b >> k;

//storing the query

//this will add k in the prefix sum for index >= a

v.push\_back(make\_pair(a, k));

//adding -1\*k will remove k from the prefix sum for index > b

v.push\_back(make\_pair(b+1, -1 \* k));

}

long mx = 0, sum = 0;

sort(v.begin(), v.end());

for(int i=0 ; i<2\*m; i++) {

sum += v[i].second;

mx = max(mx, sum);

}

cout<<mx<<endl;

return 0;

}

# 7. Queries with Fixed Length

The problem asks us to find the maximum values for each subarray of size  **d** and then print the minimum of these maximums. The main problem is to keep track of the maximum elements for each subarray of size **d** . Using a deque can easily solve this problem for us.

For each query, we get an integer **d** , and we'll try to maintain a deque of size  **d** where it contains the maximum element at one of its ends. We slide over the array, keeping a window of size **d**.  
Pretend we are at index , and the maximum element is at the end of the deque. When we add the **(i+1)th** element, we first delete all the elements from the deque that do not fall within the current window of size **d** . Then, we delete those elements that are less than the **(i+1)th**  element and push it accordingly so that the maximum element of this window is at the end of the deque.

We maintain the answer by taking the minimum of the answer with the maximum value for each subarray for a particular  and print it at the end when all the subarrays are processed.

### **C++**

#include <cstdio>

#include <cassert>

#include <deque>

using namespace std;

const int N = 1e5 + 100;

int a[N];

deque<int> dq;

int main() {

int n, cq;

scanf(" %d %d", &n, &cq);

for (int i = 0; i < n; i++) {

scanf(" %d", a + i);

}

for (int it = 0; it < cq; it++) {

int d;

scanf(" %d", &d);

dq.clear();

int best = 1 << 29;

for (int i = 0; i < n; i++) {

while (dq.size() && a[dq.back()] < a[i])

dq.pop\_back();

dq.push\_back(i);

while (dq.size() && dq.front() <= i - d)

dq.pop\_front();

if (i >= d - 1) {

assert(dq.size());

if (best > a[dq.front()]) {

best = a[dq.front()];

}

}

}

printf("%d\n", best);

}

return 0;

}

## 8. Game of Two Stacks

## **Observation**

This problem can be solved in linear time. We'll begin by taking as many integers as possible from stack  **A** without exceeding the sum. Once we've done this we'll start taking integers from **B** , but whenever the sum becomes larger than the limit, we'll put integers back into stack **A** . Make sure to update the answer (the number of integers) as the traversal through stack **B** takes place. Break the loop when you have put back all integers that was taken from **A** and it's not possible to take any more integers from **B**.

#include <bits/stdc++.h>

using namespace std;

int A[100002],B[100002];

int main(){

int t;

cin >> t;

while (t--) {

int n, m ,ms;

scanf("%d%d%d", &n, &m, &ms);

for(int A\_i = 0; A\_i < n; A\_i++){

scanf("%d",&A[A\_i]);

}

for(int B\_i = 0; B\_i < m; B\_i++){

scanf("%d",&B[B\_i]);

}

long long sum = 0;

int x = 0, y = 0;

while (x < n && sum + A[x] <= ms) {

sum += A[x++];

}

int ans = x;

while (y < m && x >= 0) {

sum += B[y++];

while (sum > ms && x > 0) {

sum -= A[--x];

}

if (sum <= ms && ans < x + y) {

ans = x + y;

}

}

printf("%d\n",ans);

}

return 0;

}

# 9.What's Next?

This challenge requires you to find the next permuatation of a binary number.

**For 30 Points of the Total Possible Score**

The elements in A are smaller than 104 for this subset of test cases, so we can build the array of bits. If we are using C++, we can simply call the next\_permuatation() function (other languages may or may not have another library function comparable to this) to get the next permutation. Then we compress it to get our result.

**For All Possible Points**

To achieve a maximum score, it is not possible to build the binary number as the elements in A can be as large as **1018**.

The next permutation **(D)** must be larger than the current number **(B).** Suppose the rightmost bit of the binary number is at position X (the least significant bit is at position 0). Flipping a 0-bit (making it a 1) at a position < X will never result in a larger number. Why? Because when we flip a bit at a position < X, we must flip a 1-bit (making it a 0) at some position > X to keep setCount(B) and setCount(D) equal, so this approach will always result ina D < B (which violates the requirements of the problem).

So, now we know we must always flip a **0-bit** (making it a 1) at some **position > =X.** How do we choose which one? We'll flip the first 0-bit occurring at a position >= X. Once we flip it, we must flip a 1-bit (making it a 0) to ensure that **setCount(B) =setCount(D).** Suppose we flip a 0-bit at position Y; then we must flip all 1-bits at positions < Y. Suppose we flipped all 1-bits from position **0 to Z**; then we just flip the first (Z — 1) 0-bits.

**CODE:**

#include <bits/stdc++.h>

#define pb push\_back

#define nl puts ("")

#define sp printf ( " " )

#define phl printf ( "hello\n" )

#define ff first

#define ss second

#define POPCOUNT \_\_builtin\_popcountll

#define RIGHTMOST \_\_builtin\_ctzll

#define LEFTMOST(x) (63-\_\_builtin\_clzll((x)))

#define MP make\_pair

#define FOR(i,x,y) for(vlong i = (x) ; i <= (y) ; ++i)

#define ROF(i,x,y) for(vlong i = (y) ; i >= (x) ; --i)

#define CLR(x,y) memset(x,y,sizeof(x))

#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())

#define MIN(a,b) ((a)<(b)?(a):(b))

#define MAX(a,b) ((a)>(b)?(a):(b))

#define NUMDIGIT(x,y) (((vlong)(log10((x))/log10((y))))+1)

#define SQ(x) ((x)\*(x))

#define ABS(x) ((x)<0?-(x):(x))

#define ODD(x) (((x)&1)==0?(0):(1))

using namespace std;

typedef long long vlong;

typedef unsigned long long uvlong;

typedef pair < int, int > pii;

typedef pair < vlong, vlong > pll;

typedef vector<pii> vii;

typedef vector<int> vi;

const vlong inf = 2147383647;

const double pi = 2 \* acos ( 0.0 );

const double eps = 1e-9;

vlong arr[1000];

vlong brr[1000];

void solution() {

int kase;

scanf ( "%d", &kase );

while ( kase-- ) {

int n;

scanf ( "%d", &n );

CLR(arr,0);

CLR(brr,0);

FOR(i,1,n) {

scanf ( "%lld", &arr[i] );

}

if ( n & 1 ) {

///Last is 1

///So we update last - 1 position

vlong temp = arr[n];

arr[n-1]--;

if ( arr[n-1] < 0 ) arr[n-1] = 0;

arr[n] = 1;

arr[n+1] = 1;

arr[n+2] = temp - 1;

}

else {

vlong temp = arr[n-1];

arr[n-2]--;

if ( arr[n-2] < 0 ) arr[n-2] = 0;

arr[n-1] = 1;

arr[n]++;

arr[n+1] = temp - 1;

}

if ( arr[0] ) {

ROF(i,1,100) {

arr[i] = arr[i-1];

}

}

///Now compress the array

int cur = 1;

FOR(i,0,n+10) {

if ( arr[i] == 0 ) {

continue;

}

if ( ODD(cur) && ODD(i) ) {

brr[cur] = arr[i];

cur++;

}

else if ( !ODD(cur) && !ODD(i) ) {

brr[cur] = arr[i];

cur++;

}

else {

///Add it to previous

brr[cur-1] += arr[i];

}

}

printf ( "%d\n", cur - 1 );

FOR(i,1,cur-1) {

if ( i > 1 ) sp;

printf ( "%lld", brr[i] );

}

nl;

}

}

int main () {

solution();

return 0;

}

# 10. Coprime Paths

#include <bits/stdc++.h>

#define sd(x) scanf("%d",&x)

#define sd2(x,y) scanf("%d%d",&x,&y)

#define sd3(x,y,z) scanf("%d%d%d",&x,&y,&z)

#define fi first

#define se second

#define pb push\_back

#define mp make\_pair

#define foreach(it, v) for(auto it=(v).begin(); it != (v).end(); ++it)

#define \_ ios\_base::sync\_with\_stdio(false);cin.tie(NULL);cout.tie(NULL);

#define \_\_ freopen("input.txt","r",stdin);freopen("output.txt","w",stdout);

#define tr(...) cout<<\_\_FUNCTION\_\_<<' '<<\_\_LINE\_\_<<" = ";trace(#\_\_VA\_ARGS\_\_, \_\_VA\_ARGS\_\_)

using namespace std;

typedef long long ll;

typedef long double ld;

typedef pair<int,int> pii;

template<typename S, typename T>

ostream& operator<<(ostream& out,pair<S,T> const& p){out<<'('<<p.fi<<", "<<p.se<<')';return out;}

template<typename T>

ostream& operator<<(ostream& out,vector<T> const& v){

int l=v.size();for(int i=0;i<l-1;i++)out<<v[i]<<' ';if(l>0)out<<v[l-1];return out;}

template<typename T>

void trace(const char\* name, T&& arg1){cout<<name<<" : "<<arg1<<endl;}

template<typename T, typename... Args>

void trace(const char\* names, T&& arg1, Args&&... args){

const char\* comma = strchr(names + 1, ',');cout.write(names, comma-names)<<" : "<<arg1<<" | ";trace(comma+1,args...);}

const ld PI = 3.1415926535897932384626433832795;

const int LOGN = 15;

const int N = 1 << LOGN;

const int M = 1 << LOGN;

int n, m;

int a[N];

vector<int> g[N];

vector<pair<pii, int> > q;

int type[M], lc[M];

int p[LOGN][N], lvl[N];

int LCA(int x, int y){

if(lvl[x] < lvl[y]) swap(x,y);

int tmp = 1;

while((1<<tmp) <= lvl[x]) tmp++;

tmp--;

for(int i = tmp; i >= 0; i--)

if(lvl[x] - (1<<i) >= lvl[y]) x = p[i][x];

if(x == y) return y;

for(int i = tmp; i >= 0; i--)

if(p[i][x] > 0 and p[i][x] != p[i][y]) x = p[i][x], y = p[i][y];

return p[0][x];

}

void makeTable(){

for(int i = 1; i < LOGN; i++){

for(int j = 1; j <= n; j++){

p[i][j] = p[i-1][p[i-1][j]];

}

}

}

int st[N], ex[N], tme;

void dfs(int x, int pr){

st[x] = tme++, p[0][x] = pr;

foreach(it, g[x]){

if(\*it == pr) continue;

lvl[\*it] = lvl[x] + 1;

dfs(\*it, x);

}

ex[x] = tme++;

}

int root;

bool cmp(const pair<pii, int> &p1, const pair<pii, int> &p2){

int x = p1.fi.fi / root, y = p2.fi.fi / root;

if(x == y){

return p1.fi.se < p2.fi.se;

}

return x < y;

}

int line[2\*N];

int active[N];

ll ans[M];

ll curans;

const int MAXNUM = 10 \* 1000 \* 1000;

int f[MAXNUM];

vector<int> pr[MAXNUM];

vector<int> primes;

const int L = sqrt(MAXNUM);

const int K = 3;

void getPrimes(){

primes.pb(2);

for(int i = 3; i <= L; i += 2){

bool ok = true;

for(auto &x : primes){

if(i%x == 0){

ok = false; break;

}

}

if(ok) primes.pb(i);

}

for(int i = 1; i <= n; i++){

if(!pr[a[i]].empty()) continue;

int num = a[i];

for(auto &x: primes){

if(num % x == 0) pr[a[i]].pb(x);

while(num % x == 0) num /= x;

}

if(num > 1) pr[a[i]].pb(num);

int sz = pr[a[i]].size();

assert(sz <= K);

}

}

void rem(int x);

void add(int x);

void add(int x){

int val = line[x];

if(active[val]){

rem(x);

}

else{

active[val] = 1;

int sz = pr[a[val]].size();

for(int i = 0; i < (1 << sz); i++){

int num = 1, par = 0;

for(int j = 0; j < sz; j++){

if(i&(1<<j)) num \*= pr[a[val]][j], par ^= 1;

}

if(par) curans -= f[num]++;

else curans += f[num]++;

}

}

}

void rem(int x){

int val = line[x];

if(active[val]){

active[val] = 0;

int sz = pr[a[val]].size();

for(int i = 0; i < (1 << sz); i++){

int num = 1, par = 0;

for(int j = 0; j < sz; j++){

if(i&(1<<j)) num \*= pr[a[val]][j], par ^= 1;

}

if(par) curans += --f[num];

else curans -= --f[num];

}

}

else{

add(x);

}

}

int main(){

sd2(n,m);

assert(1 <= n and n <= 25000);

assert(1 <= m and m <= 25000);

for(int i = 1; i <= n; i++){

sd(a[i]);

assert(a[i] <= 10000000);

}

getPrimes();

for(int i = 1; i < n; i++){

int u, v; sd2(u,v);

g[u].pb(v);

g[v].pb(u);

}

dfs(1, 1);

makeTable();

for(int i = 0; i < m; i++){

int u, v; sd2(u,v);

if(st[u] > st[v]) swap(u,v);

int lca = LCA(u,v);

lc[i] = lca;

if(lca == u){

type[i] = 1;

q.pb(mp(mp(st[u], st[v]), i));

}

else{

type[i] = 2;

q.pb(mp(mp(ex[u], st[v]), i));

}

}

root = sqrt(2\*n);

sort(q.begin(), q.end(), cmp);

for(int i = 1; i <= n; i++){

line[st[i]] = i;

line[ex[i]] = i;

}

for(int i = q[0].fi.fi; i <= q[0].fi.se; i++){

add(i);

}

int xx = q[0].fi.fi, yy = q[0].fi.se;

for(int i = 0; i < m; i++){

int nx = q[i].fi.fi, ny = q[i].fi.se;

while(xx < nx) rem(xx++);

while(xx > nx) add(--xx);

while(yy < ny) add(++yy);

while(yy > ny) rem(yy--);

ll tmp = 0;

if(type[q[i].se] == 2){

int node = lc[q[i].se];

vector<int> &pd = pr[a[node]];

int sz = pd.size();

for(int i = 0; i < (1 << sz); i++){

int num = 1, par = 0;

for(int j = 0; j < sz; j++){

if(i&(1<<j)) num \*= pd[j], par ^= 1;

}

if(par) tmp -= f[num];

else tmp += f[num];

}

}

ans[q[i].se] = curans + tmp;

}

for(int i = 0; i < m; i++){

assert(ans[i] >= 0);

printf("%lld\n", ans[i]);

}

return 0;

}