C. Square Subsets

time limit per test: 4 seconds
memory limit per test: 256 megabytes

Petya was late for the lesson too. The teacher gave him an additional task. For some array a Petya should find the number of different ways to select non-empty subset of elements from it in such a way that their product is equal to a square of some integer.

Two ways are considered different if sets of indexes of elements chosen by these ways are different.

Since the answer can be very large, you should find the answer modulo $10^9 + 7$.

Input
First line contains one integer n ($1 \le n \le 10^5$) — the number of elements in the array.

Second line contains n integers a_i ($1 \le a_i \le 70$) — the elements of the array.

Output
Print one integer — the number of different ways to choose some elements so that their product is a square of a certain integer modulo $10^9 + 7$.

Examples

input	Copy
4 1 1 1 1	
output	Copy
15	

input	Copy
4 2 2 2 2	
output	Copy
7	

input	Copy
5 1 2 4 5 8	
output	Copy
7	

Note
In first sample product of elements chosen by any way is 1 and $1 = 1^2$. So the answer is $2^4 - 1 = 15$.

In second sample there are six different ways to choose elements so that their product is 4, and only one way so that their product is 16. So the answer is $6 + 1 = 7$.

→ **Problem tags**

bitmasks combinatorics dp math *2000

No tag edit access

Problem Breakdown

Lets understand the problem first

We're given:

- n (up to 1e5)
- an array a[1..n], each a[i] is between 1 and 70

We need to count how many non-empty subsets of indices we can choose such that:

The product of the chosen numbers is a perfect square.

Answer modulo MOD = 1e9+7.

Two subsets are different if their index sets are different
(so [a1, a2] and [a2, a1] is the same subset; order doesn't matter).

DP definition

dp[mask] = number of ways to choose a subset from the values we have processed so far such that the XOR of their masks is exactly 'mask'.

Initially, before processing anything, there is exactly one subset: the empty subset, with XOR = 0.

So:

```
dp[0] = 1;
dp[others] = 0;
```

Now, for each value v from 1 to 70 with cnt[v] = c > 0:

1. Compute mv = mask[v].

2. Compute:

```
evenWays = powmod(2, c-1)
oddWays = powmod(2, c-1)
```

Create a new DP array next_dp[mask] initially 0.

Transition for each mask:

If we take an even number of v's, XOR doesn't change:

```
next_dp[mask] += dp[mask] * evenWays;
```

If we take an odd number of v's, XOR flips by mv:

```
next_dp[mask ^ mv] += dp[mask] * oddWays;
```

Code

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

const long long MOD = 1000000007LL;

long long modPow(long long base, long long exp) {
    long long res = 1;
    base %= MOD;
    while (exp > 0) {
        if (exp & 1) res = (res * base) % MOD;
        base = (base * base) % MOD;
        exp >>= 1;
    }
    return res;
}
```

We need 2^{c-1} mod MOD a lot.
Use fast exponentiation:

```cpp
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;

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

    // Step 1: list of primes up to 70
    vector<int> primes = {
        2, 3, 5, 7, 11, 13, 17, 19, 23,
        29, 31, 37, 41, 43, 47, 53, 59, 61, 67
    };
    int P = (int)primes.size(); // 19
    int MAXMASK = 1 << P;

    // Step 2: precompute mask for each v in [1..70]
    int valMask[71];
    for (int v = 1; v <= 70; v++) {
        int x = v;
        int mask = 0;
        for (int i = 0; i < P; i++) {
            int p = primes[i];
            int cnt = 0;
            while (x % p == 0) {
                x /= p;
                cnt ^= 1; // toggle parity
            }
            if (cnt & 1) {
                mask |= (1 << i);
            }
        }
        valMask[v] = mask;
    }

    // Step 3: count occurrences of each value
    int cnt[71] = {0};
    for (int i = 0; i < n; i++) {
        cnt[a[i]]++;
    }

    // Step 4: DP over masks
    vector<long long> dp(MAXMASK, 0), next_dp(MAXMASK);
    dp[0] = 1; // empty subset

    // Process each value from 1 to 70
    for (int v = 1; v <= 70; v++) {
        int c = cnt[v];
        if (c == 0) continue;

        long long evenWays = modPow(2, c - 1); // number of even subsets from c
        long long oddWays  = evenWays;         // same as even

        int mv = valMask[v];

        fill(next_dp.begin(), next_dp.end(), 0);

        for (int mask = 0; mask < MAXMASK; mask++) {
            if (dp[mask] == 0) continue;

            long long cur = dp[mask];

            // pick even number of v's: XOR unchanged
            long long addEven = (cur * evenWays) % MOD;
            next_dp[mask] = (next_dp[mask] + addEven) % MOD;

            // pick odd number of v's: XOR flips by mv
            long long addOdd = (cur * oddWays) % MOD;
            int newMask = mask ^ mv;
            next_dp[newMask] = (next_dp[newMask] + addOdd) % MOD;
        }

        dp.swap(next_dp);
    }

    // dp[0] = number of subsets (including empty) whose product is a perfect square
    long long ans = dp[0] - 1; // exclude empty subset
    if (ans < 0) ans += MOD;
    cout << ans % MOD << "\n";

    return 0;
}
```