Team notebook

HCMUS-PenguinSpammers

January 26, 2022

Contents

1 A Contest **1**
- 1.1 bashrc 1
- 1.2 hash 1
- 1.3 template 1
- 1.4 troubleshoot 1
- 1.5 vimrc 2

2 Combinatorial **2**
- 2.1 Factorial Approximate 2
- 2.2 Factorial 2
- 2.3 Fast Fourier Transform 2
- 2.4 Lucas Theorem 3
- 2.5 Others 3
- 2.6 Permutation To Int 3
- 2.7 Sigma Function 4

1 A Contest

1.1 bashrc

```
// alias c='g++ -Wall -Wconversion -Wfatal-errors
    -g -std=c++17 \
//     -fsanitize=undefined,address'
// xmodmap -e 'clear lock' -e 'keycode 66=less
    greater' #caps = <>
```

1.2 hash

```
// # Hashes a file, ignoring all whitespace and
    comments. Use for
// # verifying that code was correctly typed.
// cpp -dD -P -fpreprocessed | tr -d '[:space:]'|
    md5sum |cut -c-6
```

1.3 template

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
        ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
}
```

1.4 troubleshoot

```
// Pre-submit:
// Write a few simple test cases if sample is not
    enough.
// Are time limits close? If so, generate max
    cases.
// Is the memory usage fine?
// Could anything overflow?
// Make sure to submit the right file.

// Wrong answer:
// Print your solution! Print debug output, as
    well.
// Are you clearing all data structures between
    test cases?
// Can your algorithm handle the whole range of
    input?
// Read the full problem statement again.
// Do you handle all corner cases correctly?
// Have you understood the problem correctly?
// Any uninitialized variables?
// Any overflows?
// Confusing N and M, i and j, etc.?
// Are you sure your algorithm works?
// What special cases have you not thought of?
// Are you sure the STL functions you use work as
    you think?
// Add some assertions, maybe resubmit.
// Create some testcases to run your algorithm on.
// Go through the algorithm for a simple case.
// Go through this list again.
// Explain your algorithm to a teammate.
```

```
// Ask the teammate to look at your code.
// Go for a small walk, e.g. to the toilet.
// Is your output format correct? (including
    whitespace)
// Rewrite your solution from the start or let a
    teammate do it.

// Runtime error:
// Have you tested all corner cases locally?
// Any uninitialized variables?
// Are you reading or writing outside the range
    of any vector?
// Any assertions that might fail?
// Any possible division by 0? (mod 0 for example)
// Any possible infinite recursion?
// Invalidated pointers or iterators?
// Are you using too much memory?
// Debug with resubmits (e.g. remapped signals,
    see Various).

// Time limit exceeded:
// Do you have any possible infinite loops?
// What is the complexity of your algorithm?
// Are you copying a lot of unnecessary data?
    (References)
// How big is the input and output? (consider
    scanf)
// Avoid vector, map. (use arrays/unordered_map)
// What do your teammates think about your
    algorithm?

// Memory limit exceeded:
// What is the max amount of memory your
    algorithm should need?
// Are you clearing all data structures between
    test cases?
```

1.5 vimrc

```
// set cin aw ai is ts=4 sw=4 tm=50 nu noeb
    bg=dark ru cul
// sy on  |  im jk <esc>  |  im kj <esc>  |  no
    ; :
// " Select region and then type :Hash to hash
    your selection.
// " Useful for verifying that there aren't
    mistypes.
// ca Hash w !cpp -dD -P -fpreprocessed \| tr -d
    '[:space:]' \
//  \| md5sum \| cut -c-6
```

2 Combinatorial

2.1 Factorial Approximate

Approximate Factorial:

$$n! = \sqrt{2.\pi.n}.(\frac{n}{e})^n \tag{1}$$

2.2 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
n	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

2.3 Fast Fourier Transform

```
/**
 * Fast Fourier Transform.
 * Useful to compute convolutions.
 * computes:
 *    C(f star g)[n] = sum_m(f[m] * g[n - m])
 * for all n.
 * test: icpc live archive, 6886 - Golf Bot
 * */
using namespace std;
#include <bits/stdc++.h>
#define D(x) cout << #x " = " << (x) << endl
#define endl '\n'

const int MN = 262144 << 1;
int d[MN + 10], d2[MN + 10];


const double PI = acos(-1.0);

struct cpx {
  double real, image;
  cpx(double _real, double _image) {
    real = _real;
    image = _image;
  }
  cpx(){}
};

cpx operator + (const cpx &c1, const cpx &c2) {
  return cpx(c1.real + c2.real, c1.image +
      c2.image);
}

cpx operator - (const cpx &c1, const cpx &c2) {
  return cpx(c1.real - c2.real, c1.image -
      c2.image);
}

cpx operator * (const cpx &c1, const cpx &c2) {
  return cpx(c1.real*c2.real - c1.image*c2.image,
      c1.real*c2.image + c1.image*c2.real);
}

int rev(int id, int len) {
  int ret = 0;
  for (int i = 0; (1 << i) < len; i++) {
    ret <<= 1;
    if (id & (1 << i)) ret |= 1;
  }
  return ret;
}
```

```
cpx A[1 << 20];

void FFT(cpx *a, int len, int DFT) {
  for (int i = 0; i < len; i++)
    A[rev(i, len)] = a[i];
  for (int s = 1; (1 << s) <= len; s++) {
    int m = (1 << s);
    cpx wm = cpx(cos( DFT * 2 * PI / m), sin(DFT
        * 2 * PI / m));
    for(int k = 0; k < len; k += m) {
      cpx w = cpx(1, 0);
      for(int j = 0; j < (m >> 1); j++) {
        cpx t = w * A[k + j + (m >> 1)];
        cpx u = A[k + j];
        A[k + j] = u + t;
        A[k + j + (m >> 1)] = u - t;
        w = w * wm;
      }
    }
  }
  if (DFT == -1) for (int i = 0; i < len; i++)
      A[i].real /= len, A[i].image /= len;
  for (int i = 0; i < len; i++) a[i] = A[i];
  return;
}

cpx in[1 << 20];

void solve(int n) {
  memset(d, 0, sizeof d);
  int t;
  for (int i = 0; i < n; ++i) {
    cin >> t;
    d[t] = true;
  }
  int m;
  cin >> m;
  vector<int> q(m);
  for (int i = 0; i < m; ++i)
    cin >> q[i];

  for (int i = 0; i < MN; ++i) {
    if (d[i])
      in[i] = cpx(1, 0);
    else
      in[i] = cpx(0, 0);
  }

  FFT(in, MN, 1);
  for (int i = 0; i < MN; ++i) {
    in[i] = in[i] * in[i];
  }
  FFT(in, MN, -1);

  int ans = 0;
  for (int i = 0; i < q.size(); ++i) {
    if (in[q[i]].real > 0.5 || d[q[i]]) {
      ans++;
    }
  }
  cout << ans << endl;
}

int main() {
  ios_base::sync_with_stdio(false);cin.tie(NULL);
  int n;
  while (cin >> n)
    solve(n);
  return 0;
}
```

2.4 Lucas Theorem

For non-negative integers m and n and a prime p, the following congruence relation holds: :

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod p,$$

where :

$$m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0,$$

and :

$$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$$

are the base p expansions of m and n respectively. This uses the convention that $\binom{m}{n} = 0$ if $m \leq n$.

2.5 Others

Cycles Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp\left(\sum_{n\in S} \frac{x^n}{n}\right)$$

Derangements Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor \frac{n!}{e} \right.$$

Burnside's lemma Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|} \sum_{g\in G} |X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G = Z_n$ to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n} \sum_{k|n} f(k)\phi(n/k).$$

2.6 Permutation To Int

```
/**
 * Description: Permutation -> integer
     conversion. (Not order preserving.)
 * Integer -> permutation can use a lookup table.
 * Time: O(n)
 **/

int permToInt(vector<int>& v) {
        int use = 0, i = 0, r = 0;
```

```
    for(int x : v) r = r * ++i +
        __builtin_popcount(use & -(1<<x)),
           use |= 1 << x;                  //
               (note: minus, not ~!)
    return r;
}
```

2.7 Sigma Function

The Sigma Function is defined as:

$$\sigma_x(n) = \sum_{d|n} d^x$$

when $x = 0$ is called the divisor function, that counts the number of positive divisors of n.

Now, we are interested in find

$$\sum_{d|n} \sigma_0(d)$$

If n is written as prime factorization:

$$n = \prod_{i=1}^{k} P_i^{e_k}$$

We can demonstrate that:

$$\sum_{d|n} \sigma_0(d) = \prod_{i=1}^{k} g(e_k + 1)$$

where $g(x)$ is the sum of the first x positive numbers:

$$g(x) = (x * (x + 1))/2$$