HKOI Training 2016/17
Mathematics in OI (II)

Alex Tung

18 March 2017

Prerequisites / “What you should know”

- (a subset of) Mathematics in OI (I)
  - G.C.D., L.C.M.
  - Euclidean algorithm, *extended* Euclidean algorithm
  - Modular arithmetic/Modular inverse
  - Chinese Remainder Theorem
  - Finding Primes (using Sieve of Eratosthenes)
- Fast exponentiation, aka “big mod” (i.e. $a^b \bmod m$)

Today's Content

1. Combinatorics
   - 1.1. C_r^n, P_r^n
   - 1.2. Pigeon-hole principle
2. Linear algebra
   - 2.1. Basic concepts
   - 2.2. Solving linear recurrences
3. More number theory
   - 3.1. Euler's phi (ϕ) function, Euler's theorem
   - 3.2. Mu (μ) function, Möbius inversion (!)

(!): special topic

Note

- Core content based in Jeffrey Hui's notes last year
- But this year's "special topics" are different

- Check last year's slides for:
  - The Josephus problem (!)
  - Lucas's Theorem (!)
  - Catalan number (!)

1.1. C_r^n, P_r^n

Definition 1. Let $0 \leq r \leq n$ be integers.

C_r^n is defined to be the number of ways to choose r objects from n different objects, where order does not matter

P_r^n is defined to be the number of ways to choose r objects from n different objects, where order matters

Theorem 2. $P_r^n = \frac{n!}{(n-r)!}$, $C_r^n = \frac{n!}{r!(n-r)!}$.

Proof. (For P_r^n)

The number of ways to choose the i-th object is (n + 1 – i).

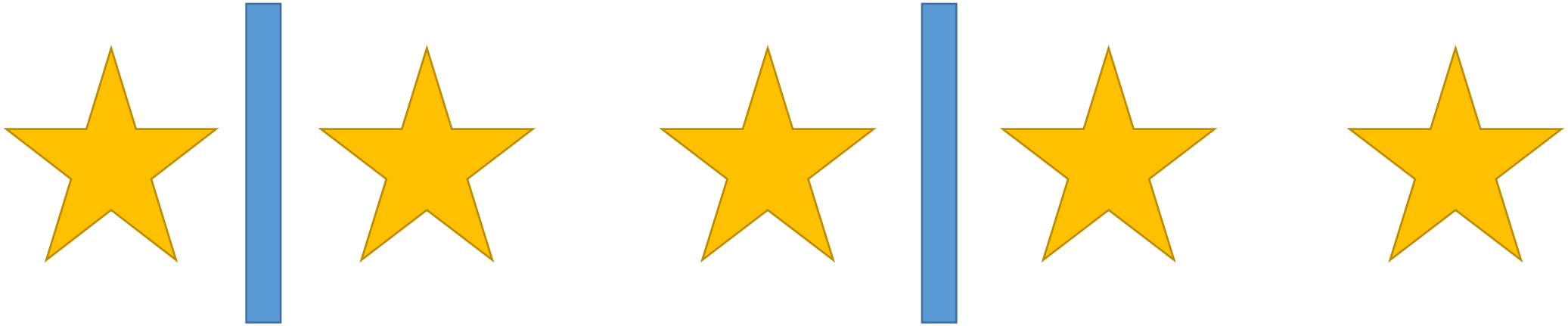
Multiplying, $P_r^n = n\times(n-1)\times\cdots\times(n+1-r) = \frac{n!}{(n-r)!}$

(For C_r^n)

Similar to P_r^n; if we care about order, then for each suitable subset of objects, there are $r!$ ways of choosing. So to neglect order, divide by $r!$ to get $C_r^n = \frac{n!}{r!(n-r)!}$.

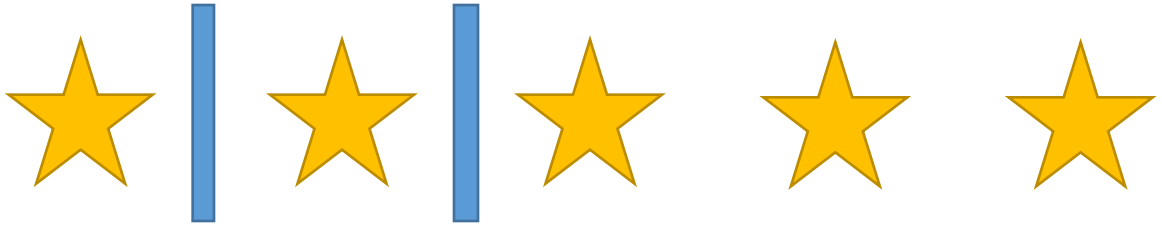
Exercise 3. How many solutions are there to the equation $x_1 + x_2 + \cdots + x_k = N$, where $x_i \in \mathbb{N}$ and $k \leq N$?

Solution. This looks abstract, but if we visualize the above as "stars and bars", it will be easy!



$\mathrm{N} = 5, \mathrm{k} = 3$

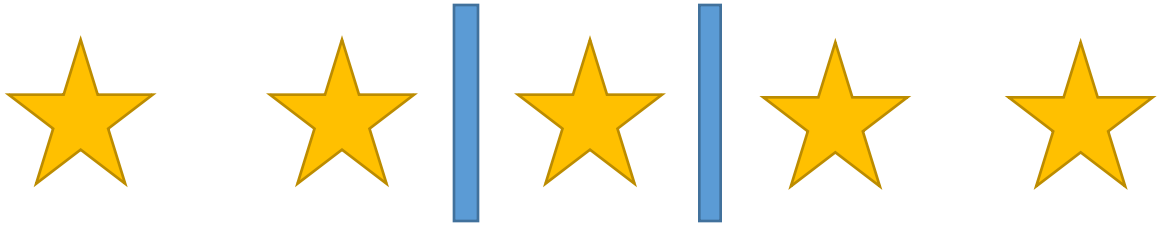
We are choosing which gaps to put bars on!



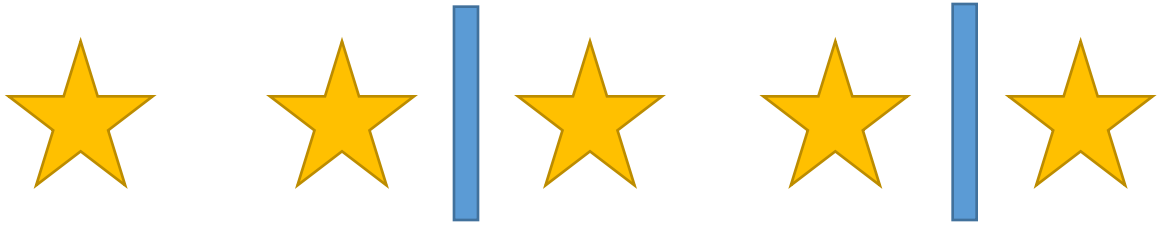
$x_1 = 1, x_2 = 1, x_3 = 3$

$x_1 = 1, x_2 = 2, x_3 = 2$

$x_1 = 1, x_2 = 3, x_3 = 1$



$x_1 = 2, x_2 = 1, x_3 = 2$



$x_1 = 2, x_2 = 2, x_3 = 1$

$x_1 = 3, x_2 = 1, x_3 = 1$

In general, there are $N-1$ gaps and $k-1$ bars.
So, answer $= C_{k-1}^{N-1}$.

Below we introduce several ways to calculate $C_r^n \pmod m$.

Lemma 4. For $0 < r < n$, $C_r^n = C_r^{n-1} + C_{r-1}^{n-1}$.

Proof. Either "direct explode", or note that
(#ways to choose r objects from n)
= (#ways to choose r objects from first n-1) <= skip n-th object
+ (#ways to choose r-1 objects from first n-1) <= choose n-th object

Algorithm 5. [Pascal's triangle] We can calculate $C_r^n \pmod m$ for all (r, n) with upper bound N

1. Input N
2. for i: 1...N

   pas[i][0] := pas[i][i] = 1 % m

   for j: 1...(i-1)

   pas[i][j] := (pas[i – 1][j] + pas[i – 1][j – 1]) % m
3. For each query (r, n), output pas[n][r]

Time complexity: $O(N^2 + Q)$, where Q = number of queries

Algorithm 6. If $m = p$ is a prime, we can calculate $C_r^n \pmod p$ for all (r, n) with upper bound N, using the idea of modular inverse

1. Input N
2. set fact[0] := 1, finv[0] := 1
3. for i: 1...N

   fact[i] := (fact[i – 1] * i) % p

   finv[i] := bigmod(fact[i], p – 2, p)

   bigmod(a, b, c) := $a^b \pmod c$
3. For each query (r, n),

   output fact[n] * finv[r] * finv[n – r] (mod p)

(CAUTION: may overflow, take mods where appropriate)

Time complexity: O(N log N + Q)

In fact, Algorithm 6 can be improved to O(N + Q). Do you see how? (Hint: find a more efficient way of calculating finv[].)

1.2. Pigeon-hole principle

Proposition 7. Suppose there are N+1 (identical) balls, to be put in N (identical) boxes. Then we can find a box which contains no less than two balls.

Proof. [~~Common sense.~~] Suppose not, then the total number of balls will be AT MOST $1 \times N$, contradiction.

Proposition 7'. Suppose there are M (identical) balls, to be put in N (identical) boxes. Then we can find a box which contains no less than $\left\lceil \frac{M}{N} \right\rceil$ balls.

Proof. Suppose not, then the total number of balls will be AT MOST $\left(\left\lceil \frac{M}{N} \right\rceil - 1 \right) \times N < M$, contradiction.

Theorem 8. Let s[1...N] be an array of N integers. Then there exists a subarray s[L...R] with sum divisible by N.

Proof. Consider the partial sum of the array, ps[0...N],

where $ps[i] = s[1] + s[2] + ... + s[i]$ (Note: $ps[0] = 0$.)

There are (N+1) integers (balls) and N residue classes (boxes), so we can find some ps[i], ps[j] ($i < j$) such that $ps[i] \equiv ps[j] \pmod{N}$.

Take $L = i + 1$, $R = j$. Then $s[L] + ... + s[R] = ps[j] - ps[i]$, which is divisible by N. We are done.

I can only recall one problem which solution involves pigeon-hole principle:

CF 618F Double Knapsack

Break;

2.1. Basic concepts

Definition 1. An $N \times M$ (real) matrix T is "basically" an $N \times M$ array.

If $N = M$, we say T is a square matrix. In this case,

- The diagonal of T are the elements T[1][1], T[2][2], ..., T[N][N].
- If all its non-diagonal entries are 0, we say T is a diagonal matrix.

An identity matrix I_N is a $N \times N$ diagonal matrix whose diagonal entries are all ones.

Definition 2. [Basic operations]

(Addition) If T, U are $N \times M$ matrices, define T + U by

$$(\mathrm{T} + \mathrm{U})[\mathrm{i}][\mathrm{j}] = \mathrm{T}[\mathrm{i}][\mathrm{j}] + \mathrm{U}[\mathrm{i}][\mathrm{j}]$$

(Scalar multiplication) Define αT by $(\alpha T)[\mathrm{i}][\mathrm{j}] = \alpha \times \mathrm{T}[\mathrm{i}][\mathrm{j}]$

(Matrix multiplication) If T is a $N \times M$ matrix and U a $M \times K$ matrix, define $T \circ U$ by

$$(\mathrm{T} \circ \mathrm{U})[\mathrm{i}][\mathrm{j}] = \sum_{k=1}^{M} \mathrm{T}[\mathrm{i}][\mathrm{k}] * \mathrm{U}[\mathrm{k}][\mathrm{j}]$$

Matrix multiplication may be a bit confusing, so here is what the code may look like:

```
18  double T[105][105], U[105][105], T_times_U[105][105];
19
20  int N, M, K;
21
22  int main(){
23      scanf("%d %d %d", &N, &M, &K);
24      for(int i = 1; i <= N; i++)
25          for(int j = 1; j <= M; j++)
26              scanf("%lf", &T[i][j]);
27
28      for(int i = 1; i <= M; i++)
29          for(int j = 1; j <= K; j++)
30              scanf("%lf", &U[i][j]);
31
32      for(int i = 1; i <= N; i++)
33          for(int j = 1; j <= K; j++){
34              //calculate T_times_U[i][j] here
35              for(int k = 1; k <= M; k++)
36                  T_times_U[i][j] += T[i][k] * U[k][j];
37          }
38
39      for(int i = 1; i <= N; i++)
40          for(int j = 1; j <= K; j++)
41              printf("%.10lf%c", T_times_U[i][j], j == K ? '\n':' ');
42
43      return 0;
44  }
```

(*) means technical stuff
that we won't prove

Proposition 3(*). Matrix multiplication is "associative", that is, order of multiplication does not matter.

We postpone the definition of matrix inverse and invertibility to 2.2, where it will be useful.

2.2. Solving linear recurrences

Motivating example: Fibonacci numbers

$F_0 = 0, F_1 = 1, F_{n+2} = F_{n+1} + F_n$

If you study olympiad maths, you may know some magic stuff like:

- Solve $x^2 = x + 1$ -> Roots are $\alpha = \frac{1+\sqrt{5}}{2}$ and $\beta = \frac{1-\sqrt{5}}{2}$
- $F_n = A\alpha^n + B\beta^n$, for some constants A, B
- Plug in first few terms, get $F_n = \frac{1}{\sqrt{5}}\alpha^n - \frac{1}{\sqrt{5}}\beta^n$

This looks wonderful, but we usually care about values **mod m**, which makes this formula (more often than not) inapplicable.

Instead, we use the *matrix form* of the formula:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$$

[Why true though? Note $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 \times F_n + 1 \times F_{n-1} \\ 1 \times F_n + 0 \times F_{n-1} \end{pmatrix}$.]

Then,

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} F_1 \\ F_0 \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$$

So we want to perform fast exponentiation for matrices.

Algorithm 4. [Big mod for square matrices]

1. Input N, T[1...N][1...N], n, m. Wish to calculate T^n mod m.
2. Set ans := I_N.
3. Calculate $T, T^2, T^4, T^8, \dots, T^{2^k}$, for sufficiently large k.
4. If the 2^i-bit of n is 1, multiply ans by T^{2^i}.
5. T^n mod m = ans

One possible (slightly different) implementation

```cpp
#include <bits/stdc++.h>

#define FI(i,a,b) for(int i=(a);i<=(b);i++)

using namespace std;

int N, T[105][105], n, m;

int ans[105][105], T_power[105][105], temp[105][105];

int main(){
    scanf("%d", &N);
    FI(i, 1, N) FI(j, 1, N)
        scanf("%d", &T[i][j]);

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

    FI(i, 1, N) ans[i][i] = 1; //ans initialised as I_N

    FI(i, 1, N) FI(j, 1, N) T_power[i][j] = T[i][j];

```

```
21
22      while(n){
23          if(n % 2 == 1){ //2^i-th bit = 1
24              FI(i, 1, N) FI(j, 1, N) temp[i][j] = 0;
25
26              FI(i, 1, N) FI(j, 1, N) FI(k, 1, N)
27                  temp[i][j] = (temp[i][j] + (long long) T_power[i][k] * ans[k][j]) % m;
28
29              FI(i, 1, N) FI(j, 1, N) ans[i][j] = temp[i][j];
30          }
31
32          n /= 2;
33
34          //calculate T_power^2
35          FI(i, 1, N) FI(j, 1, N) temp[i][j] = 0;
36
37          FI(i, 1, N) FI(j, 1, N) FI(k, 1, N)
38              temp[i][j] = (temp[i][j] + (long long) T_power[i][k] * T_power[k][j]) % m;
39
40          FI(i, 1, N) FI(j, 1, N) T_power[i][j] = temp[i][j];
41      }
42
43      FI(i, 1, N) FI(j, 1, N) printf("%d%c", ans[i][j], j == N ? '\n':' ');
44
45      return 0;
46  }
```

Algorithm 5. [Computing $F_n \bmod m$]

1. Input n, m
2. Define $T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$.
3. Use Algorithm 4 to compute $U = T^n \bmod m$.
4. answer = U[1][1] (1-based)

Time complexity: O(log n) :)

Remark. The approach works for any linear recurrence of the form $a_{N+k} = \sum_{i=0}^{k-1} c_i \times a_{N+i}$. Try to write the corresponding transition matrix!

Practice problems:

SPOJ Fibonacci Sum

CF 446C DZY Loves Fibonacci Numbers

Next, we discuss why the two approaches ("matrix exponentiation" v.s. "solving characteristic equation") are basically the same.

Definition 6. Let T be a $N \times N$ square matrix.

Say T is *invertible* if there exists another matrix U, such that

$$T \circ U = U \circ T = I_N.$$

In this case, *inverse* is unique (!) and we write $U = T^{-1}$.

Say T is *diagonalizable* if we can write $T = UDU^{-1}$, where D is diagonal and U is invertible.

Proposition 7. $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ is diagonalizable.

Proof. (See right)

Input interpretation:

diagonalize	$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

Result:

$$M = S.J.S^{-1}$$

where

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$S = \begin{pmatrix} \frac{1}{2}\left(1-\sqrt{5}\right) & \frac{1}{2}\left(1+\sqrt{5}\right) \\ 1 & 1 \end{pmatrix}$$

$$J = \begin{pmatrix} \frac{1}{2}\left(1-\sqrt{5}\right) & 0 \\ 0 & \frac{1}{2}\left(1+\sqrt{5}\right) \end{pmatrix}$$

$$S^{-1} = \begin{pmatrix} -\frac{1}{\sqrt{5}} & \frac{1}{10}\left(5+\sqrt{5}\right) \\ \frac{1}{\sqrt{5}} & \frac{1}{10}\left(5-\sqrt{5}\right) \end{pmatrix}$$

Proposition 8(*). If $T = UDU^{-1}$, then $T^n = UD^nU^{-1}$.

So, if you write out a diagonalization of $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ and rearrange terms, you would get the "magic" $F_n = \frac{1}{\sqrt{5}}\alpha^n - \frac{1}{\sqrt{5}}\beta^n$.

Break;

3.1 Euler's ϕ function

Definition 1. Let N be a positive integer. $\phi(N)$ is defined to be the number of integers k in $[1, N]$ such that $\gcd(\mathrm{k}, \mathrm{N}) = 1$.

Examples:

$\phi(1) = 1$ $(\mathrm{k} = 1)$

$\phi(7) = 6$ $(\mathrm{k} = 1, 2, 3, 4, 5, 6)$

$\phi(12) = 4$ $(\mathrm{k} = 1, 5, 7, 11)$

[ϕ function is also called the totient function.]

Proposition 2. We have the following properties:

- $\phi(p) = p - 1$, where p is a prime
- $\phi(p^k) = p^{k-1}(p-1)$, where p is a prime

Proof. Follows from the simple observation that

$$\gcd(l, p) > 1 \Leftrightarrow l \text{ is a multiple of } p.$$

Definition 3. Let f be a function defined on $\mathbb{N}$ (the set of positive integers). f is said to be *multiplicative* if, for all **coprime** positive integers n and m we have $f(nm) = f(n)f(m)$.

Proposition 4(*). ϕ is multiplicative.

[Proof idea: use Chinese Remainder Theorem.]

Example:

$$\phi(3) = 2 \ (\mathrm{k} = 1, 2)$$

$$\phi(4) = 2 \ (\mathrm{k} = 1, 3)$$

$$\phi(3\times4) = 4 = 2\times2 \ (\mathrm{k} = 1, 5, 7, 11)$$

WARNING: $\phi(nm) = \phi(n)\phi(m)$ may not hold for arbitrary n, m!

(Counter-example: $n = m = 2$)

Algorithm 5. [Computing $\phi(N)$]

1. Input N
2. Write $N = \prod_{i=1}^{k} p_i^{a_i}$ (prime factorisation of N)
3. Compute $\phi(N) = \prod_{i=1}^{k} [p_i^{a_i - 1}(p_i - 1)]$

Proof of correctness.

$$\begin{aligned}\phi(N) &= \phi\left(\prod_{i=1}^{k} p_i^{a_i}\right) \\ &= \prod_{i=1}^{k} \phi(p_i^{a_i}) && \text{(Prop. 4)} \\ &= \prod_{i=1}^{k} [p_i^{a_i - 1}(p_i - 1)] && \text{(Prop. 2)}\end{aligned}$$

Example:

1. Input N: 360
2. $360 = 2^3 \times 3^2 \times 5^1$
3. So, $\phi(360) = [2^{3-1} \times (2-1)] \times [3^{2-1} \times (3-1)] \times [5^{1-1} \times (5-1)]$

$\Rightarrow \phi(360) = 4 \times 6 \times 4 = 96$

Algorithm 5b. [Computing $\phi(N)$, "shortcut" method]

1. Input N
2. Initialise ans := N
3. For each **distinct** prime factor p_i of N, multiply ans by $\frac{p_i-1}{p_i}$
4. $\phi(N) = ans$

Proof of correctness.

$$\phi(N) = \prod_{i=1}^{k}[p_i^{a_i-1}(p_i - 1)] \quad \text{(Prop. 2)}$$

$$= \prod_{i=1}^{k}[p_i^{a_i} \times \frac{p_i-1}{p_i}]$$

$$= N \times \prod_{i=1}^{k} \frac{p_i-1}{p_i}$$

Example:

1. Input N: 360
2. Initialise ans = 360
3. $360 = 2^3 \times 3^2 \times 5^1$

$$p_1 = 2\text{: ans} = 360 \times \frac{2-1}{2} = 180$$

$$p_1 = 3\text{: ans} = 180 \times \frac{3-1}{3} = 120$$

$$p_1 = 5\text{: ans} = 120 \times \frac{5-1}{5} = 96$$

$$\Rightarrow \phi(360) = ans = 96$$

Theorem 6(*). [Euler's Theorem] Let a and n be positive coprime integers. Then $a^{\phi(n)} \equiv 1 \pmod{n}$.

Corollary 6'(*). [Fermat's little Theorem] Let p be a prime and a an integer. Then $a^{p} \equiv a \pmod{p}$.

Algorithm 7. [Modular inverse]

1. Input a, n (positive coprime integers)
2. Compute $\phi(n)$
3. Use fast exponentiation to get $a^{-1} \equiv a^{\phi(n)-1} \pmod{n}$

Remark. Algorithm 7 is particularly useful when you are asked to do something "mathy" modulo a prime (typically 1,000,000,007).

Alternatively, you can use extended Euclidean algorithm to find a^{-1}.

Practice problems:

CF327C Magic Five

CF300C Beautiful Numbers (study 3.1 first)

CF622F The Sum of the k-th Powers

(Hint: Lagrange interpolation polynomial)

Before moving on to 3.2, we introduce a lemma here.

Lemma 8. Let N be a positive integer. Then $N = \sum_{d|n} \phi(d)$.

Example: $\mathrm{N} = 12$

$$\phi(1) + \phi(2) + \phi(3) + \phi(4) + \phi(6) + \phi(12)$$
$$= 1 + 1 + 2 + 2 + 2 + 4 = 12$$

Proof. Note that (for $d|n$) among integers in $[1, N]$, there are exactly $\phi(d)$ of them with $\gcd(\mathrm{N}, \mathrm{k}) = \frac{N}{d}$.

(See next page for illustration.)

d=gcd(N, k)	{k}	$\{\frac{k}{d}\}$	$\frac{N}{d}$	$\phi(\frac{N}{d})$
1	1, 5, 7, 11	1, 5, 7, 11	12	4
2	2, 10	1, 5	6	2
3	3, 9	1, 3	4	2
4	4, 8	1, 2	3	2
6	6	1	2	1
12	12	1	1	1

3.2. Mu (μ) function, Möbius inversion (!)

Definition 9. Let $N = \prod_{i=1}^{k} p_i^{a_i}$ be a positive integer.

$$\mu(N) = \begin{cases} (-1)^k, & N \text{ is square}-\text{free} \\ 0, & \text{otherwise} \end{cases}$$

(We say N is square-free if it is NOT divisible by any perfect equares other than 1, i.e. $a_i = 1$ for all i.)

Examples: $\mu(1) = 1, \mu(3) = -1, \mu(4) = 0, \mu(6) = 1$

GOAL: to introduce and apply the Möbius inversion formula

Proposition 10(*). μ is multiplicative.

[Proof idea: Consider different cases, check by definition.]

Definition 11. Let f and g be multiplicative functions. Then the Dirichlet convolution $(f * g)$ of f and g is defined by

$$(f * g)(n) = \sum_{d|n} f(d) g(\frac{n}{d})$$

Proposition 12(*). Let f and g be multiplicative. Then $(f * g)$ is again multiplicative.

Theorem 13(*). [Möbius inversion formula] Let f and g be ~~multiplicative~~ **arithmetic functions (meaning that they are defined on the positive integers)**, and μ the Möbius function. Then:

$$\forall n, f(n) = \sum_{d|n} g(d) \Leftrightarrow \forall n, g(n) = \sum_{d|n} f(d)\mu(\frac{n}{d})$$

This may look scary at first, so let's look at some examples.

Example (1): $f = id$ (i.e. $f(n) = n$), $g = \phi$

By Lemma 8, $n = \sum_{d|n} \phi(d)$.

Now by Möbius inversion formula, $\phi(n) = \sum_{d|n} d \times \mu(\frac{n}{d})$

Check when $n = 12$:

$$\sum_{d|12} d \times \mu(\frac{12}{d})$$
$$= 1 \times \mu(12) + 2 \times \mu(6) + 3 \times \mu(4) + 4 \times \mu(3) + 6 \times \mu(2) + 12 \times \mu(1)$$
$$= 1 \times 0 + 2 \times 1 + 3 \times 0 + 4 \times (-1) + 6 \times (-1) + 12 \times (1)$$
$$= 4 = \phi(12)$$

Example (2): $f = \sigma, g = id$ ($\sigma(n) =$ sum of factors of n)

$\sigma(n) = \sum_{d|n} d$ by definition.

Now by Möbius inversion formula, $n = \sum_{d|n} \sigma(d) \times \mu(\frac{n}{d})$

Check when n $= 12$:

$\Sigma_{d|12} \sigma(d) \times \mu(\frac{12}{d})$

$= \sigma(1) \times \mu(12) + \sigma(2) \times \mu(6) + \sigma(3) \times \mu(4) + \sigma(4) \times \mu(3) + \sigma(6) \times \mu(2) + \sigma(12) \times \mu(1)$

$= 1 \times 0 + 3 \times 1 + 4 \times 0 + 7 \times (-1) + 12 \times (-1) + 28 \times 1$

$= 12$

Exercise 14. Given N $(1 \leq N \leq 10^6)$, find $\sum_{i=1}^{N} LCM(i, N)$. Note that there are multiple queries in one test case!

[Source: SPOJ LCMSUM]

Solution. Recall LCM(a, b) $= \frac{a \times b}{GCD(a,b)}$. So,

$$\sum_{i=1}^{N} LCM(i, N) = N \times \sum_{i=1}^{N} \frac{i}{\gcd(i,N)}$$

$$= N \times \sum_{d|N} \left[\frac{1}{d} \times \sum_{\substack{i \leq N \\ (i,N)=d}} i\right]$$

$$= N \times \sum_{d|N} \left[\frac{1}{\cancel{d}} \times \sum_{\substack{i \leq N/d \\ (i,N/d)=1}} \cancel{d} \times i\right]$$

$$= N \times \sum_{d|N} g(\frac{N}{d}), \text{ where } \mathrm{g(k)} = \sum_{\substack{e \le k \\ (e,k)=1}} e$$

$$= N \times \sum_{d|N} g(d)$$

In other words, g(k) is the sum of elements e in $[1, k]$ such that $\gcd(e, k) = 1$.

If we know how to calculate g, we would be done.

Note that $\frac{N \times (N+1)}{2} = \sum_{d|N} \frac{N}{d} g(d)$, as each integer from 1 to N appears exactly once on RHS. So simplifying we get

$$N + 1 = \sum_{d|N} \frac{2 \times g(d)}{d}$$

$$N + 1 = \sum_{d|N} \frac{2 \times g(d)}{d}$$

By Möbius inversion formula,

$$\frac{2g(N)}{N} = \sum_{d|N} (d + 1) \times \mu(\frac{N}{d}).$$

This gives an efficient way of calculating g.

~~[Update: we actually cannot use Möbius inversion formula here, as e functions are NOT multiplicative. It is purely a coincidence that it works here.~~

A simpler method exists.

Simply note that, <u>for k > 1</u>, $\gcd(e,k) = 1 \Leftrightarrow \gcd(\mathrm{k} - \mathrm{e}, \mathrm{k}) = 1$, so

$$\mathrm{g(k)} = \sum_{\substack{e \le k \\ (e,k)=1}} e$$

$$= \phi(k) \times \text{average}(\mathrm{e} \mid e \le k \text{ and } (\mathrm{e}, \mathrm{k}) = 1)$$

$$= \phi(k) \times \frac{k}{2}$$

So we just need to calculate $\phi(k)$ quickly for all k.

Update 2 (21/3): I gave the inversion formula another thought and realised that it works even if the functions are NOT multiplicative. Sorry for the confusion.

Still, the previous slide suggests a simpler method for manipulating this particular summation.

Remark. We can calculate μ and ϕ using a sieve-like method.

Practice problems:

SPOJ GCDEX

CF 645F Cowslip Collections

The End

PM Session: Mini Competition II :)

Reference

- A dance with Mobius function https://www.quora.com/profile/Surya-Kiran/Posts/A-Dance-with-Mobius-Function