LAPORAN FINAL PROJECT PERANCANGAN DAN ANALISIS ALGORITMA

SOAL SPOJ – THE WIT OF TENALI RAMAN

PENYUSUN

Rayhan Arvianta Bayuputra

NRP : 5025211217

DOSEN PENGAMPU

Rully Soelaiman, S.Kom., M.Kom.

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

SURABAYA

2022

A. Penjelasan Soal

Tenali Ramakhrisna adalah seorang Vikatakavi. Perdana menteri kemudian memberikan sebuah tas yang berisi N nomer dari permata dari 0 sampai N-1. Kemudian dia mengambil K permata dan meminta para prajurit untuk mendistribusikan kepada semua orang. Perdana menteri menghitung total dari permata yang ia bagikan. Dia menyuruh Tenali hanya membagikan yang jumlahnya bisa dibagi dengan N dan cari banyaknya pasangan kemungkinan dari permata yang dibagikan.

Hasil yang dikeluarkan sangat besar sehingga jawaban harus di modulo 1000000007.

Format input pada soal ini adalah baris pertama berisikan T yang berarti banyaknya test cases. Selanjutnya sebanyak T baris, mengandung nilai N dan K. N adalah banyaknya permata yang ada di dalam tas (1 <= N <= 1000000000). K adalah banyaknya permata yang dibagikan (1<= K <= Min(N,1000)). Artinya nilai K ini hanya berasal dari 1 – 1000 saja. Outputkan jumlah semua kemungkinan dari permata yang dibagikan.

Pada sample input diberikan contoh

1

7 4

Output : **5**

Pada sample input, terdapat N = 7 yang berarti di dalam tas berisi 7 permata dengan nomer 0, 1, 2, 3, 4, 5, 6 dan 4 permata harus dibagikan. Pasangan permata yang memungkinkan adalah {0, 1, 2, 4} karena jiska ditotal menghasilkan 7 yang artinya bisa dibagi dengan 7. Kemungkinan selanjutnya adalah {0, 3, 5, 6} {1, 2, 5, 6} {1, 3, 4, 6} {2, 3, 4, 5} karena jika setiap pasangan ditotal akan menghasilkan 14 yang artinya bisa dibagi dengan 7. Dari uraian diatas menghasilkan 5 kemungkinan pasangan permata sehingga outputkan 5.

B. Abstraksi Penyelesaian

Dari soal tersebut, dapat kita cari test case lain agar mendukung pencarian pola. Untuk mendapatkan test case, dapat kita jabarkan sebagai berikut :

n | k	1	2	3	4	5
1	[0]	-	-	-	-
2	[0]	0	-	-	-
3	[0]	[0, 1]	[0,1,2]	-	-
4	[0]	[1, 3]	[0,1,3]	0	0
5	[0]	[1, 4] [2, 3]	[0,1,4] [0,2,3]	[1,2,3,4]	[0,1,2,3,4]

Setelah melakukan uji test case, didapatkan tabel sebagai berikut

n | k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-
3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-
4	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-
5	1	2	2	1	1	-	-	-	-	-	-	-	-	-	-
6	1	2	4	3	1	0	-	-	-	-	-	-	-	-	-
7	1	3	5	5	3	1	1	-	-	-	-	-	-	-	-
8	1	3	7	9	7	3	1	0	-	-	-	-	-	-	-
9	1	4	10	14	14	10	4	1	1	-	-	-	-	-	-
10	1	4	12	22	26	20	12	5	1	0	-	-	-	-	-
11	1	5	15	30	42	42	30	15	5	1	1	-	-	-	-
12	1	5	19	42	66	76	66	43	19	5	1	0	-	-	-
13	1	6	22	55	99	132	132	99	55	22	6	1	1	-	-
14	1	6	26	73	143	212	246	217	143	70	26	7	1	0	-
15	1	7	31	91	201	335	429	429	335	201	91	31	7	1	1

Dengan melihat pola barisan bilangan yang sudah disediakan, dapat kita selesaikan permasalah ini dengan menggunakan rumus :

$$\left(\frac{1}{n}\ x \sum_{s \mid \gcd(n,k)}^{n} (-1)^{(\mathrm{k}-k/s)}\ x\ \Phi(s)\ x \binom{n/s}{k/s}\right)(\ mod\ 10^9 + 7)$$

Sumber : https://oeis.org/A267632

Dimana

- gcd (n,k) : FPB dari n dan k
- s : Faktor positif dari gcd (n,k)
- Σ : Jumlah perhitungan rumus untuk seluruh kemungkinan s
- (s) : Euler's Totient Function dari s
- Euler Totient : Banyaknya bilangan yang relatif prima dengan s
- Relatif prima : Pasangan bilangan yang FPB nya sama dengan 1
- $\binom{n/s}{k/s}$: (n/s) Kombinasi (k/s)

Dari rumus tersebut, dapat kita lihat bahwa digunakan Euler's Totient Function. Euler's Totient Function adalah banyaknya bilangan {1, 2, 3, … , n-1} yang relatif prima dengan n atau dengan kata lain bilangan yang memiliki FPB dengan n adalah 1. Berikut adalah contoh dari Euler's Totient Function :

Input	Euler Totient	Penjelasan
1	Φ(1) = 1	GCD(1,1) = 1
2	Φ(2) = 1	GCD(1,2) = 1
3	Φ(3) = 2	GCD(1,3) = 1, GCD(2,3) = 1
4	Φ(4) = 2	GCD(1,4) = 1, GCD(3,4) = 1
5	Φ(5) = 4	GCD(1,5) = 1, GCD(2,5) = 1, GCD(3,5) = 1, GCD(4,5) = 1
6	Φ(6) = 2	GCD(1,6) = 1, GCD(5,6) = 1

Sumber : https://www.geeksforgeeks.org/eulers-totient-function/

Untuk mengoptimasi Euler's Totient Function digunakan Sieve of Eratosthenes. Sieve of Eratosthenes adalah cara untuk menemukan semua bilangan prima antara 1 dan n. Selain itu karena kita menggunakan kombinasi di dalam rumus, maka kita memerlukan faktorial. Untuk dapat mengoptimasi faktorial, kita melakukan precompute dengan membuat array Factorial dan menghitungnya terlebih dahulu.

N	Faktorial
1	1! = 1
2	2! = 2
3	3! = 6
4	4! = 24
5	5! = 120
6	6! = 720
7	7! = 5040
8	8! = 40320

Pada rumus, kita gunakan juga kombinasi dari (n/s) dan (k,s). Banyaknya kombinasi dari sebuah himpunan dapat dihitung dengan menggunakan rumus berikut :

$$nCr = \frac{n!}{n!\,(n-r)!}$$

Karena constraint input yang sangat besar, dapat kita optimasi juga dengan menggunakan Fast I/O dan melakukan % MOD pada setiap operasi.

C. Pseudocode

Fungsi modex

Input : b, e
Output: res

```
1.  if (b == 0) then return 0
2.  if (b == 1) then return 1
3.  if (b == -1) then return (e & 1 ? -1 : 1)
4.
5.  res = 1
6.  while e do
7.      if (e & 1) then res = (res * b) % (10^9+7)
8.      b = (b * b) % (10^9+7)
9.      e >>= 1
10. return res
```

Fungsi init

```
1.  sieve[0] = sieve[1] = 1;
2.  for i ← 2 to sqrt 1000 do
3.      if (!sieve[i]) then
4.          for j ← i * i to 1000 do
5.              if (!sieve[j]) then
6.                  sieve[j] = i
7.  fact[0] = fact[1] = 1
8.  for i ← 2 to 1000 do
9.      fact [i] = (fact [i – 1] * i) % (10^9+7)
```

Fungsi totient

Input : x
Output: res

```
1.  last = 1
2.  res = temp = x
3.
4.  while (sieve[temp] > 0 && temp > 1) do
5.      if (sieve[temp] != last) then
6.          res /= sieve[temp]
7.          res *= (sieve[temp] – 1)
8.      last = sieve[temp]
9.      temp /= sieve[temp]
10. if (temp > 1 && temp != last) then
11.     res /= temp
12.     res *= (temp – 1)
13. return res
```

Fungsi kombin

Input : n, k
Output: res

1. res = 1
2. mn = min(n – k, k)
3. mx = max(n – k, k)
4. for i ← mx + 1 to n do
5. res = (res * i) % (10^9+7)
6. res = (res * modex(fact[mn], 10^9+7 – 2)) % (10^9+7)
7. return res

Fungsi calc

Input : n, k, s
Output: res

1. res1 = ((k - k / s) & 1 ? -1 : 1)
2. res2 = totient(s)
3. res3 = kombin(n / s, k / s)
4. res = (res1 * res2 * res3) % (10^9+7)
5. return res

Fungsi solve

1. ans = 0
2. fpb = __gcd(n, k)
3. for s ← 1 to sqrt fpb do
4. if (fpb % s == 0) then
5. ans = (ans + calc(n, k, s)) % (10^9+7)
6. ss = fpb / s
7. if (ss != s) then ans = (ans + calc(n, k, ss)) % (10^9+7)
8. ans = (ans * modex(n, 10^9+7 - 2)) % (10^9+7)
9. while (ans < 0) do ans += 10^9+7
10. print ans

Main

1. init()
2. input t
3. while (t--) do
4. input n, k
5. Solve (n, k)

D. Source Code

```
1. #include <cstdio>
2. #include <algorithm>
3. using namespace std;
4. #define ll long long
```

```
#define MOD 1000000007
#define MAXK 1000
#define ULL unsigned long long
#define gc getchar//_unlocked()

ll sieve[MAXK + 5], fact[MAXK + 5];

ULL read(){
    ULL value =0;bool ne=0;
    char c = gc();
    while(c==' ' or c=='\n') c =gc();
    if(c=='-'){ne = 1;c = gc();}
    while(c>='0' and c<='9'){
    value = (value<<3)+(value<<1)+c-'0';c=gc();
    }
    if(ne) value*=-1;
    return value;
}

ll modex(ll b, ll e){
    if (b == 0)
        return 0;
    if (b == 1)
        return 1;
    if (b == -1)
        return (e & 1 ? -1 : 1);
    ll res = 1;
    b %= MOD;
    while (e){
        if (e & 1) res = (res * b) % MOD;
        b = (b * b) % MOD;
        e >>= 1;
    }
    return res;
}

void init(){
    sieve[0] = sieve[1] = 1;
    for (int i = 2; i * i <= MAXK; ++i){
        if (!sieve[i]){
            for (int j = i * i; j <= MAXK; j += i){
                if (!sieve[j])
                    sieve[j] = i;
            }
        }
    }
```

```
51.     fact[0] = fact[1] = 1;
52.     for(ll i = 2; i <= MAXK; ++i){
53.         fact[i] = (fact[i - 1] * i) % MOD;
54.     }
55. }
56.
57. ll totient(ll x){
58.     ll res, temp, last = 1;
59.     res = temp = x;
60.     while (sieve[temp] > 0 && temp > 1){
61.         if (sieve[temp] != last){
62.             res /= sieve[temp];
63.             res *= (sieve[temp] - 1);
64.         }
65.         last = sieve[temp];
66.         temp /= sieve[temp];
67.     }
68.     if (temp > 1 && temp != last){
69.         res /= temp;
70.         res *= (temp - 1);
71.     }
72.     return res;
73. }
74.
75. ll kombin(ll n, ll k){
76.     ll res = 1;
77.     ll mn = min(n - k, k), mx = max(n - k, k);
78.     for (ll i = mx + 1; i <= n; ++i){
79.         res = (res * i) % MOD;
80.     }
81.     res = (res * modex(fact[mn], MOD - 2)) % MOD;
82.     return res;
83. }
84.
85. ll calc(ll n, ll k, ll s){
86.     ll res1 = ((k - k / s) & 1 ? -1 : 1), res2 = totient(s), res3 = kombin(n / s, k / s);
87.     ll res = (res1 * res2 * res3) % MOD;
88.     return res;
89. }
90.
91.  void solve(ll n, ll k){
92.      ll ans = 0, fpb = __gcd(n, k);
93.      for(ll s = 1; s * s <= fpb; ++s){
94.          if(fpb % s == 0){
95.              ans = (ans + calc(n, k, s)) % MOD;
96.              ll ss = fpb / s;
```

```
97.                 if (ss != s){
98.                     ans = (ans + calc(n, k, ss)) % MOD;
99.                 }
100.            }
101.        }
102.        ans = (ans * modex(n, MOD - 2)) % MOD;
103.        while (ans < 0)
104.            ans += MOD;
105.        printf("%lld\n", ans);
106. }
107.
108. int main(){
109.        init();
110.        int t;
111.        t = read();
112.        while(t--){
113.            ll n, k;
114.            n = read();
115.            k = read();
116.            solve(n, k);
117.        }
118.        return 0;
119. }
```

E. Penjelasan Code

Pada bagian in, kita akan membahas penjelasan dari source code yang sudah dipaparkan sebelumnya.

```
10. ll sieve[MAXK + 5], fact[MAXK + 5];
```

Pada line 10, array tersebut digunakan untuk menyimpan nilai Sieve of Eratosthenes dan Factorial dari suatu bilangan. Ukuran array yang diperlukan hanya hingga nilai maksimum dari k yaitu 1000. Sieve of Eratosthenes digunakan ketika menghitung Euler Totient, sedangkan Factorial digunakan untuk menghitung kombinasi.

```
12. ULL read(){
13.     ULL value =0;bool ne=0;
14.     char c = gc();
15.     while(c==' ' or c=='\n') c =gc();
16.     if(c=='-'){ne = 1;c = gc();}
17.     while(c>='0' and c<='9'){
18.     value = (value<<3)+(value<<1)+c-'0';c=gc();
19.     }
20.     if(ne) value*=-1;
21.     return value;
22. }
```

Pada line 12-22, digunakan untuk melakukan Fast I/O sehingga dapat mempercepat dalam proses input.

```
24. ll modex(ll b, ll e){
25.     if (b == 0)
26.         return 0;
27.     if (b == 1)
28.         return 1;
29.     if (b == -1)
30.         return (e & 1 ? -1 : 1);
31.     ll res = 1;
32.     b %= MOD;
33.     while (e){
34.         if (e & 1) res = (res * b) % MOD;
35.         b = (b * b) % MOD;
36.         e >>= 1;
37.     }
38.     return res;
39. }
```

Pada line 24-39 adalah fungsi Modular Exponentiation yang digunakan nantinya untuk menghitung (b^e) (mod $10^9 + 7$). Agar lebih optimal, kita keluarkan beberapa kasus khusus, yaitu ketika b = 0 maka hasilnya adalah 0, ketika b = 1 maka hasilnya adalah -1, ketika e ganjil maka hasilnya -1, dan e genap maka hasilnya 1. Sebelum memulai perhitungan, kita lakukan modulo dengan MOD untuk mencegah terjadinya overflow.

```
41. void init(){
42.     sieve[0] = sieve[1] = 1;
43.     for (int i = 2; i * i <= MAXK; ++i){
44.         if (!sieve[i]){
45.             for (int j = i * i; j <= MAXK; j += i){
46.                 if (!sieve[j])
47.                     sieve[j] = i;
48.             }
49.         }
50.     }
51.     fact[0] = fact[1] = 1;
52.     for(ll i = 2; i <= MAXK; ++i){
53.         fact[i] = (fact[i - 1] * i) % MOD;
54.     }
55. }
```

Pada line 41-55, digunakan untuk menginitialisasi nilai awal Sieve of Eratosthenesis dan Factorial dari 0 hingga 1000. Loop i yang kita gunakan hanya berakhir hingga $\sqrt{k}$ dan j dimulai dari i^2. Hal tersebut dapat terjadi karena alasan berikut

Kelipatan 2
Kelipatan 3
Kelipatan 5

Kelipatan 7

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Perhatikan gambar di atas, jika kita memiliki angka dari 1-50, dengan 1 diberi warna abu-abu karena 1 merupakan kasus khusus dan tidak ikut dalam for loop perhitungan Sieve. Nantinya kita akan memberi warna pada angka-angka tersebut untuk mempermudah illustrasi. Angka yang tidak berwarna adalah angka-angka prima (sieve[i] = 0), angka yang berwarna merah adalah kelipatan 2 (sieve[i] = 2), angka yang berwarna oranye adalah kelipatan 3 (sieve[i] = 3), angka yang berwarna kuning adalah kelipatan 5 (sieve[i] = 5), dan seterusnya.

Kelipatan 2
Kelipatan 3
Kelipatan 5

Kelipatan 7

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Kemudian, kita lakukan loop mulai dari i = 2, karena sieve[2] = 0, maka kita lakukan loop j dari 4 hingga 50 dengan jarak antar loop adalah 2. Pada setiap loop j, kita lakukan sieve[j] = 2, artinya angka-angka yang berwarna merah adalah angka yang kelipatan 2.

Kelipatan 2

Kelipatan 3

Kelipatan 5

Kelipatan 7

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Selanjutnya ketika i = 3, karena sieve[3] = 0, maka kita lakukan loop dari 9 hingga 50 dengan jarak antar loop adalah 3. Perhatikan bahwa terdapat beberapa angka kelipatan 3 yang dilewati karena angka-angka tersebut sebelumnya sudah memiliki warna. Sebagai contoh angka 6 dan 12, kedua angka tersebut adalah kelipatan 2 dan sudah memiliki warna merah sebelumnya sehingga kita tidak perlu mewarnai dengan warna baru agar lebih efektif.

Inilah alasan mengapa kita memulai loop j dari i^2, bukan dari 2*i, karena dapat dipastikan bahwa seluruh bilangan komposit yang kurang dari i^2 pasti sudah memiliki warna.

Kelipatan 2

Kelipatan 3

Kelipatan 5

Kelipatan 7

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Hal yang sama terjadi ketika i = 5, seluruh kelipatan 5 yang kurang dari 25 sudah memiliki warna yaitu 10, 15, dan 20. Sehingga kita hanya perlu memulai loop j dari 25, tidak perlu dari 10.

Kelipatan 2
Kelipatan 3
Kelipatan 5

Kelipatan 7

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Setelah loop i = 7 selesai, bisa diperhatikan bahwa seluruh angka komposit sudah memiliki warna, serta seluruh angka yang tidak memiliki warna adalah angka prima. Dengan demikian, kita tidak perlu melanjutkan loop *i* untuk mencari kelipatan dari 11, 13, 17, dst. Inilah alasan mengapa loop *i* akan berhenti ketika mencapai $\sqrt{k}$. Seperti contoh di atas, ketika *k* bernilai 50, maka loop *i* hanya berjalan hingga $i = 7$, karena $\sqrt{50} \approx 7$. Berdasarkan hasil di atas, maka didapatkan bahwa sieve[i] menyimpan faktor prima terkecil dari *i*. Sebagai contoh, kita perhatikan angka 36.

$$\mathbf{36 = 2^2 \times 3^2}$$

Berdasarkan faktorisasi prima dari 36, didapatkan bahwa faktor prima terkecil dari 36 adalah 2. Dengan demikian, sieve[36] bernilai 2, bisa dilihat pada gambar di atas bahwa angka 36 berwarna merah. Kita ambil contoh lain yaitu angka 45.

$$\mathbf{45 = 3^2 \times 5}$$

Berdasarkan faktorisasi prima dari 45, didapatkan bahwa faktor prima terkecil dari 45 adalah 3, maka sieve[45] bernilai 3. Sieve[i] menyimpan faktor prima terkecil dar *i* karena nantinya akan digunakan untuk perhitungan Totient. Selanjutnya, untuk perhitungan factorial dilakukan dengan cara loop biasa, yaitu fact[i] = fact[i – 1] * i, tetapi jangan lupa untuk modulo dengan *MOD* pada setiap perhitungan untuk menghindari *overflow*.

```
ll totient(ll x){
    ll res, temp, last = 1;
    res = temp = x;
    while (sieve[temp] > 0 && temp > 1){
        if (sieve[temp] != last){
            res /= sieve[temp];
            res *= (sieve[temp] - 1);
        }
        last = sieve[temp];
        temp /= sieve[temp];
    }
    if (temp > 1 && temp != last){
        res /= temp;
        res *= (temp - 1);
}
```

Selanjutnya adalah perhitungan Euler's Totient Function. Cara yang umum digunakan untuk menghitung Totient(m) adalah dengan loop dari 2 hingga √m seperti pada gambar berikut :

```
ll totient(ll m){
    ll res, temp;
    res = temp = m;
    for(ll i = 2, i * i <= m && temp > 1; ++i){
        if (temp % i == 0){
            res /= i;
            res *= (i - 1);

            while (temp % i == 0)
                temp /= i;
        }
    }
    if (temp > 1){
        res /= temp;
        res *= (temp - 1);
    }
   return res;
}
```

Pertama kita loop i dari 2 hingga √m. Ketika menemukan i yang habis membagi temp, maka i merupakan salah satu faktor prima dari m, lalu lakukan perhitungan rumus Totient, yaitu kalikan res dengan (i-1) / i. Kemudian temp dibagi dengan i secara berulang hingga tidak bisa dibagi lagi. Setelah itu lanjutkan loop i untuk mencari faktor prima m yang lainnya. Apabila terdapat sisa, maka sisa tersebut adalah faktor prima yang terakhir. Berikut adalah contoh proses perhitungan Totient untuk m = 5733.

$5733 = 3^2 \times 7 \times 91$

	i	res	temp
LOOP	2	5733	5733
	3	3822	1911
	…	…	637
	7	3276	…
	…	…	91
	75	3276	…
SISA		3240	1

Pertama kita lakukan loop dari i = 2, kemudian kita menemukan faktor prima pertama dari 5733 yaitu ketika i = 3. Maka kita lakukan perhitungan Totient dan didapatkan nilai res menjadi 3822. Lalu, bagi temp dengan 3 hinga temp bernilai 637, pembagian berhenti karena 637 sudah tidak bisa dibagi lagi dengan 3.

Kemudian lanjutkan loop hingga menemukan faktor prima yang lain yaitu ketika i = 7. Maka kita lakukan perhitungan Totient dan didapatkan nilai res menjadi 3267. Lalu bagi temp dengan 7 sehingga bernilai 91. Setelah itu, lanjutkan loop hingga mencapai batas.

Karena $\sqrt{5733} \approx 75$, maka i berhenti hingga mencapai 75. Namun, ternyata temp masih bersisa, artinya temp tersebut adalah faktor prima terakhir dari 5733, yaitu 91. Maka lakukan perhitungan Totient dan didapatkan hasil akhir yaitu 3240.

Dari cara diatas, untuk menghitung Totient(m) diperlukan loop sebanyak $\sqrt{m}$. Tentu cara ini tidak efisien. Untuk mengoptimasi perhitungan Totient, kita bisa gunakan Sieve of Eratosthenesis seperti berikut :

```
66. ll totient(ll x){
67.     ll res, temp, last = 1;
68.     res = temp = x;
69.     while (sieve[temp] > 0 && temp > 1){
70.         if (sieve[temp] != last){
71.             res /= sieve[temp];
72.             res *= (sieve[temp] - 1);
73.         }
74.         last = sieve[temp];
75.         temp /= sieve[temp];
76.     }
77.     if (temp > 1 && temp != last){
78.         res /= temp;
79.         res *= (temp - 1);
80. }
```

Dengan contoh yang sama yaitu x = 5733, berikut adalah proses perhitungannya :

	res	last	temp	sieve[temp]
LOOP	5733	1	5733	3
	3822	3	1911	3
	3822	3	637	7
	3276	7	91	0
SISA	3240	0	1	1

Perhatikan bahwa total perhitungan yang dilakukan hanya 4, jauh lebih efektif dibandingkan cara yang pertama. Kita ambil contoh lain untuk x = 52272

$$52272 = 2^4 \times 3^3 \times 11^2$$

	i	last	temp	sieve[temp]
LOOP	52272	1	52272	2
	26136	2	26136	2
	26136	2	13068	2
	26136	2	6534	2
	26136	2	3267	3
	17424	3	1089	3
	17424	3	363	3
	17424	3	121	11
	15840	11	11	0
SISA	15840	11	1	1

Berdasarkan gambar di atas, dapat diperhatikan bahwa total perhtiungan hanya sebanyak 9x dengan optimasi dari Sieve. Apabila kit amenggunakan cara loop, kita akan membutuhkan hingga √x, maka dibutuhkan loop sebanyak 228x.

```
84. ll kombin(ll n, ll k){
85.     ll res = 1;
86.     ll mn = min(n - k, k), mx = max(n - k, k);
87.     for (ll i = mx + 1; i <= n; ++i){
88.         res = (res * i) % MOD;
89.     }
90.     res = (res * modex(fact[mn], MOD - 2)) % MOD;
91.     return res;
92. }
```

Fungsi diatas digunakan untuk menghitung n kombinasi k. Karena nilai maksimum dari k adalah 1000, maka dalam rumus perhitungan kombinasi dapat kita lakukan sedikit optimasi. Perhatikan persamaan berikut :

$$\binom{n}{k} = \frac{n!}{k!\,x\,(n-k)!} = \frac{n\,x\,(n-1)\,x\,(n-2)\,x\,\dots\,x\,2\,x\,1}{k!\,x\,(n-k)!}$$

Solusi pertama

$$\binom{n}{k} = \frac{n\,x\,(n-1)\,x\,(n-2)\,x\,\dots\,x\,(k+1)\,x\,k\,x\,(k-1)\,x\,\dots\,x\,2\,x\,1}{k!\,x\,(n-k)!}$$

$$\binom{n}{k} = \frac{n\,x\,(n-1)\,x\,(n-2)\,x\,\dots\,x\,(k+1)\,x\,\cancel{k\,x\,(k-1)\,x\,\dots\,x\,2\,x\,1}}{\cancel{k!}\,x\,(n-k)!}$$

$$\binom{n}{k} = \frac{n\,x\,(n-1)\,x\,(n-2)\,x\,\dots\,x\,(k+1)}{(n-k)!}$$

Solusi kedua

$$\binom{n}{k} = \frac{n\,x\,(n-1)\,x\,(n-2)\,x\,\dots\,x\,(n-k+1)\,x\,(n-k)\,x\,(n-k-1)\,x\,\dots\,x\,2\,x\,1}{k!\,x\,(n-k)!}$$

$$\binom{n}{k} = \frac{n\,x\,(n-1)\,x\,(n-2)\,x\,\dots\,x\,(n-k+1)\,x\,\cancel{(n-k)\,x\,(n-k-1)\,x\,\dots\,x\,2\,x\,1}}{k!\,x\,\cancel{(n-k)!}}$$

$$\binom{n}{k} = \frac{n\,x\,(n-1)\,x\,(n-2)\,x\,\dots\,x\,(n-k+1)}{k!}$$

Berdasarkan 2 solusi tersebut, langkah selanjutnya adalah menghitung hasil perkalian pada pembilang dengan menggunakan loop. Maka dari itu, kita dapat mencari nilai minimum dan

maksimum dari k dan (n-k) agar loop yang terjadi tidak terlalu lama. Sebagai contoh, misalkan kita ingin menghitung 1000 kombinasi 2, maka terdapat 2 cara yaitu

$$\binom{1000}{2} = \frac{1000\ x\ 999\ x\ 998\ x\ \dots\ x\ 3\ x\ 2\ x\ 1}{2!\ x\ 998!}$$

$$\binom{1000}{2} = \frac{1000\ x\ 999\ x\ 998\ x\ \dots\ x\ 3\ x\ \cancel{2\ x\ 1}}{\cancel{2!}\ x\ 998!}$$

$$\binom{1000}{2} = \frac{1000\ x\ 999\ x\ 998\ x\ \dots\ x\ 3}{998!}$$

Atau

$$\binom{1000}{2} = \frac{1000\ x\ 999\ x\ 998\ x\ \dots\ x\ 3\ x\ 2\ x\ 1}{2!\ x\ 998!}$$

$$\binom{1000}{2} = \frac{1000\ x\ 999\ x\ \cancel{998\ x\ \dots\ x\ 3\ x\ 2\ x\ 1}}{2!\ x\ \cancel{998!}}$$

$$\binom{1000}{2} = \frac{1000\ x\ 999}{2!}$$

Berdasarkan kedua cara di atas, tentunya loop akan bekerja jauh lebih cepat apabila kita menggunakan cara kedua. Kemudian karena perhitungan harus dimodulo dengan MOD, maka kita perlu menggunakan Modular Multiplicative Inverse untuk menghitung penyebut dari bentuk di atas dengan menggunakan sifat :

$$\boxed{\frac{1}{A}\ (mod\ B) \equiv 1\ x\ A^{-1}(mod\ B) \equiv A^{\Phi(B)}\ x\ A^{-1}(mod\ B) \equiv\ A^{\Phi(B)-1}(mod\ B)}$$

Berdasarkan Euler's Totient, dikatakan bahwa $A^{\Phi(B)}(mod\ B) \equiv 1$, dengan syarat gcd(A,B) = 1. Pada soal, hasil perhitungan selalu dimodulo dengan MOD (10^9 + 7), karena MOD adalah bilangan prima, maka $\Phi(\text{MOD}) = MOD - 1$, serta dapat dipastikan bahwa k! relatif prima dengan MOD. Sehingga untuk menghitung penyebut dari kombinasi kita perlu menghitung bentuk berikut dengan menggunakan Modular Exponentiation

$$(k!)^{MOD-2}\ (mod\ MOD)$$

```
115.ll calc(ll n, ll k, ll s){
116.     ll res1 = ((k - k / s) & 1 ? -1 : 1), res2 = totient(s), res3 = kombin(n / s, k / s);
117.     ll res = (res1 * res2 * res3) % MOD;
118.     return res;
119.}
```

Fungsi di atas digunakan untuk menghitung bentuk persamaan berikut untuk suatu nilai s :

$$(-1)^{(\mathrm{k}-k/s)}\ x\ \Phi(s)\ x\ \binom{n/s}{k/s}$$

```
96. void solve(ll n, ll k){
97.     ll ans = 0, fpb = __gcd(n, k);
98.     for(ll s = 1; s * s <= fpb; ++s){
99.         if(fpb % s == 0){
100.             ans = (ans + calc(n, k, s)) % MOD;
101.             ll ss = fpb / s;
102.             if (ss != s){
103.                 ans = (ans + calc(n, k, ss)) % MOD;
104.             }
105.         }
106.     }
107.     ans = (ans * modex(n, MOD - 2)) % MOD;
108.     while (ans < 0)
109.         ans += MOD;
110.     printf("%lld\n", ans);
111.}
```

Prosedur di atas berguna untuk menghitung rumus berikut :

$$\left(\frac{1}{n}\ x\ \sum_{s\ |\ \gcd(n,k)}^{n} (-1)^{(\mathrm{k}-k/s)}\ x\ \Phi(s)\ x\binom{n/s}{k/s})(\ mod\ 10^9 + \ 7)\right)$$

Pertama, kita cari FPB dari n dan k, kemudian kita lakukan loop s dari 1 hingga √FPB untuk mengetahui seluruh faktor positif dari FPB. Alasan mengapa loop hanya dilakukan hingga √FPB adalah sebagai berikut :

Misal nilai FPB adalah 144, maka

	144		
s	1	144	ss
	2	72	

	3	48	
	4	36	
	6	24	
	8	18	
	12	12	

Tabel diatas menunjukkan seluruh pasangan 2 bilangan bulat positif yang dapat membentuk 144, dengan kata lain angka-angka tersebut adalah semua faktor positif dari 144. Berdasarkan tabel tersebut, ketika s bernilai 1 maka ss bernilai 144, ketika s bernilai 2 maka ss bernilai 72, ketika s bernilai 3 maka ss bernilai 48, dst.

Berdasarkan pengamatan tersebut, ketika kita menghitung calc(n,k,s), kita juga bisa sekaligus menghitung calc(n, k, ss) agar dapat menghemat banyaknya proses loop. Dengan demikian, kita hanya perlu melakukan loop hingga $\sqrt{144}$ yaitu 12.

Setelah proses loop selesai, maka kita sudah mendapatkan hasil dari $\sum$.

Selanjutnya, kita akan mengalikan dengan n^{-1} menggunakan Modular Multiplicative Inverse sehingga kita mendapatkan hasil akhir. Untuk mencegah terjadinya kesalahan, maka proses pada baris 108 dan 109 perlu dilakukan.

Sebagai contoh, apabila hasil akhir yang kita dapatkan adalah -1000054391, setelah dimodulo dengan MOD, maka didapatkan hasil -54384 karena

$$1000054391\ (mod\ MOD) \equiv 54384\ (mod\ MOD)$$

Sehingga

$$-1000054391\ (mod\ MOD) \equiv -54384\ (mod\ MOD)$$

Maka berdasarkan sifat

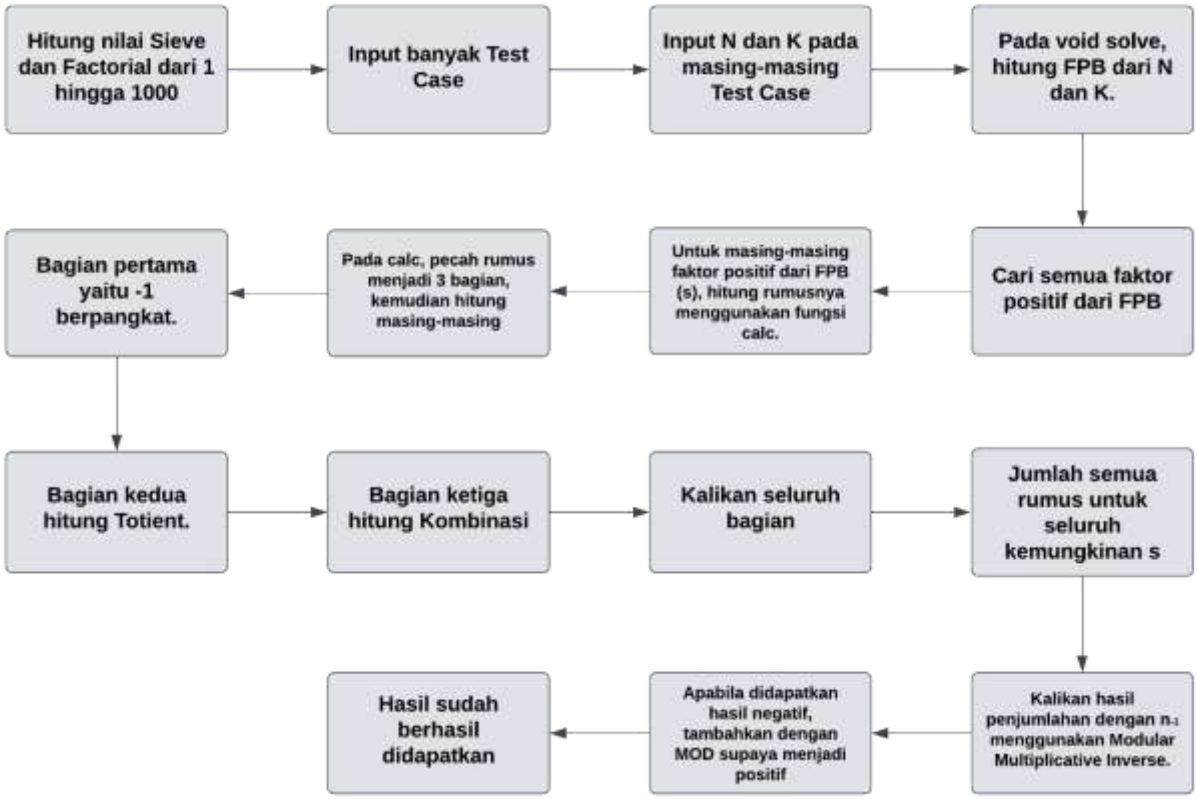
$$-A\ (mod\ MOD) \equiv B - A(mod\ MOD) \equiv\ -A + B\ (mod\ B)$$

Dapat kita lakukan

$$-54384\ (mod\ MOD) \equiv -54384 + MOD\ (mod\ MOD)\ \equiv 999945623\ (mod\ MOD)$$

Ingat bahwa MOD = $10^9 + 7$.

F. Alur Program



G. Bukti Screenshot AC