Index

Useful Things

→**Fast I/O**

```
C++:
ios_base::sync_with_stdio(false),
cin.tie(nullptr), cout.tie(nullptr);
Python:
import sys
input = sys.stdin.readline
sys.stdout.write("------")
```

VSCode setup

```
{
   "key": "f5",
   "command":
"workbench.action.terminal.sendSequence",
   "args": {
      "text": "g++
${fileBasenameNoExtension}.cpp -o
          ${fileBasenameNoExtension} &&
          ./${fileBasenameNoExtension}
        < in.txt > out.txt\n"
   }
}
```

→ **Policy Based Data Structure**

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using o_set = tree<T,
null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
// find_by_order(k) - returns an iterator to
// the k-th largest element (0 indexed);
// order_of_key(k)-the number of elements in
// the set that are strictly smaller than k;
```

1.11 Gen Random Number

```
#define accuracy
chrono::steady_clock::now().time_since_epoch
().count()
mt19937 rng(accuracy);
ll rand(ll l, ll r) {
  uniform_int_distribution<ll> ludo(l, r);
  return ludo(rng);
}
```

4.8 GP Hash Table

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename p, typename q>
using ht = gp_hash_table<p, q>;
```

1 Formula

1.1 Area Formulas

Type	Area
Rectangle	length × width
Square	side × side
Triangle	0.5 × base × height
Parallelogram	base × height
Pyramid (excluding base)	0.5 × perimeter of base × slant height
Polygon	1. $\frac{1}{2}\left\|\sum_{i=1}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)\right\|$ 2. $a + \frac{b}{2} - 1$ (for ll coordinates)

a=#ll polls strictly inside polygon
b=#ll polls on sides polygon

1.2 Volume Formulas

Type	Volume
Cube	$side^3$
Rect Prism	$length \times width \times height$
Cylinder	$\pi \times radius^2 \times height$
Sphere	$\frac{4}{3} \times \pi \times radius^3$
Pyramid	$\frac{1}{3} \times base\ area \times height$

1.3 Surface Area Formulas

Type	Surface Area
Cube	6 × side × side
Rectangular Prism	2×(length×width + length×height + width × height)
Cylinder	2 × π × radius × (radius + height)
Sphere	$4 \times \pi \times radius^2$
Pyramid	base area + $\frac{1}{2}$ × perimeter of base × slant height

1.4 Triangles

Side lengths: a, b, c

Semiperimeter	$s = \frac{a+b+c}{2}$
Area	$A = \sqrt{(s(s-a)(s-b)(s-c))}$
Circumradius	$R = \frac{abc}{4A}$
Inradius	$r = \frac{A}{s}$
Length of median	$m_a = \frac{1}{2} * \sqrt{2b^2 + 2c^2 - a^2}$
Length of bisector	$sa = \sqrt{\frac{bc}{1-(\frac{a}{b+c})^2}}$

1.5 Trigonometry

sin law: $\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c} = \frac{1}{2R}$
cos law: $a^2 = b^2 + c^2 - 2bc\cos\alpha$
tan law: $\frac{a+b}{a-b} = \frac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$
$\sin(A + B) = \sin A\cos B + \cos A\sin B$
$\cos(A + B) = \cos A\cos B - \sin A\sin B$
$\sin(A - B) = \sin A\cos B - \cos A\sin B$
$\cos(A - B) = \cos A\cos B + \sin A\sin B$
$\tan(A + B) = \frac{(\tan A + \tan B)}{(1 - \tan A\tan B)}$
$\tan(A - B) = \frac{(\tan A - \tan B)}{(1 + \tan A\tan B)}$
$\sin 2\theta = 2\sin\theta\cos\theta$
$\cos 2\theta = \cos^2\theta - \sin^2\theta$
$\tan 2\theta = \frac{(2\tan\theta)}{(1 - \tan\text{^}2\,\theta)}$
$\sin(\frac{\theta}{2}) = \pm\sqrt{\frac{1-\cos\theta}{2}}$
$\cos(\frac{\theta}{2}) = \pm\sqrt{\frac{1+\cos\theta}{2}}$
$\tan(\frac{\theta}{2}) = \frac{(1-\cos\theta)}{\sin\theta}$
$\sin r + \sin w = 2\sin(\frac{r+w}{2})\cos(\frac{r-w}{2})$
$\cos r + \cos w = 2\cos(\frac{r+w}{2})\cos(\frac{r-w}{2})$
$(V + W)\tan(\frac{r-w}{2}) = (V - W)\tan(\frac{r+w}{2})$ where V, W are lengths of sides opposite
$a\cos x + b\sin x = r\cos(x - \varphi)$ $a\sin x - b\cos x = r\sin(x - \varphi)$ where $r = \sqrt{a^2 + b^2}$, $\varphi = atan2(b, a)$

1.6 Sum

$c^k + c^{k+1} + ... + c^n = c^{n+1} - c^k$
// (c - 1) for c ≠ 1

$1 + 2 + 3 + ... + n = \frac{n*(n+1)}{2}$

$1^2 + 2^2 + ... + n^2 = \frac{n*(n+0\,1)*(2n+1)}{6}$

$1^3 + 2^3 + ... + n^3 = \frac{n^2*(n+1)^2}{4}$

$1^4 + 2^4 + ... + n^4 = \frac{n*(n+1)*(2n+1)*(3n^2+3n-1}{30}$

sum of first n odd num = n^2

1.7 Pick's Theorem

A = I-(½)*B-1 [I-inside,B-boundary]

1.9 Stars and Bars

Number of solutions of x_1 + x_2 + x_3 + ... + x_k = n is
C(n-1, k-1) where x_i > 0 and
C(n+k-1, k-1) where x_i >= 0

1.7 Logarithmic Basic

$\log_b 1 = 0$	$\log_b b = 0$
$b^{\log_b a} = a$	$x^{\log_b y} = y^{\log_b x}$
$\log_a b = \frac{1}{\log_b a}$	$\log_a x = \frac{\log_b x}{\log_b a}$
$\log_b(AB) = \log_b A + \log_b B$	
$\log_b(\frac{A}{B}) = \log_b A - \log_b B$	
$\log_a c = \log_a b * \log_b c$	
$\log_b A^x = x\log_b A$	

1.8 Series

1.8.1 Catalan Series

Series: 1, 1, 2, 5, 14, 42, 132, 429, ...
Equation:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} \quad \text{for } n \geq 0.$$

$C_n = C_0 . C_{n-1-0} + C_1 . C_{n-1-1} + ... + C_k . C_{n-1-k} + ... + C_{n-1} . C$

1.8.2 Arithmetic Series

- $a_n = a + (n - 1) * d$
- $s_n = \frac{n}{2}(2 * a + (n - 1) * d)$

1.8.3 Geometric Series

- $a_n = a * r^{n-1}$
- $s_n = \frac{a(1-r^n)}{1-r}$

1.8.4 Derangement Series

Series : 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, 14684570

$$D_n = n!\sum_{k=0}^{n}\frac{(-1)^k}{k!}$$

$$D_n = floor\left[\frac{n!}{e} + \frac{1}{2}\right]$$

1.8.5 nth Fibo Golden Ratio

$$f_n = \left[\frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}\right]$$

1.8.6 Facts

- $ceil\left[\frac{a}{b}\right] = floor\left[\frac{a-1}{b}\right] + 1$
- $natural\ num\ sum = \frac{l+r}{2} * (r - l + 1)$
- $floor\left[\frac{floor\left[\frac{n}{a}\right]}{b}\right] = floor\left[\frac{n}{ab}\right]$

1.8.7 LCM

lcm(a,n)+lcm(n−a,n)=n^2/gcd(a,n)

SUM=n/2(∑d|n(ϕ(d)×d)+1)

1.12 Stress Sh

```
#!/usr/bin/env bash
wrong="solution"
correct="brute"
gen="gen"
g++ -g solution.cpp -DONPC -o "$wrong"
g++ -g brute.cpp -DONPC -o "$correct"
g++ -g gen.cpp -DONPC -o "$gen"

for ((testNum=0;testNum<$1;testNum++))
do
       ./$gen 2>/dev/null > stdinput
       ./$correct < stdinput 2>/dev/null >
outSlow
       ./$wrong < stdinput 2>/dev/null >
outWrong
       H1=`md5sum outWrong`
       H2=`md5sum outSlow`
       if !(cmp -s "outWrong" "outSlow")
       then
       echo "Error found!"
       echo "Input:"
       cat stdinput
       echo "Wrong Output:"
       cat outWrong
       echo "Slow Output:"
       cat outSlow
       exit
       fi
done
echo Passed $1 tests
# Usage: ./contest.sh times


string to_string(const string& s) {
       return '"' + s + '"'; }
string to_string(const char* s) {
       return to_string(string(s)); }
string to_string(const char c) {
       return "'" + string(1, c) + "'"; }
string to_string(bool b) {
       return b ? "true" : "false"; }
template <typename A, typename B>
string to_string(pair<A, B> p) {
    return "(" + to_string(p.first) + ", "
    + to_string(p.second) + ")";
}
template <typename A>
string to_string(A v) {
    string res = "{";
    for (const auto& x : v) {
        res += to_string(x) + ", ";
    }
    res += "}";
    return res;
}
void debug_out() { cerr << endl; }
template <typename Head, typename... Tail>
void debug_out(Head H, Tail... T) {
    cerr << " " << to_string(H);
    debug_out(T...);
}
#define dbg(...) cerr << __LINE__ << ": ["
<< #__VA_ARGS__ << "] = ",
debug_out(__VA_ARGS__)
```

2 Number Theory

2.1 Prime number under 1000

```
2   3   5   7   11  13  17  19  23  29  31
37  41  43  47  53  59  61  67  71  73  79
83  89  97  101 103 107 109 113 127 131 137
139 149 151 157 163 167 173 179 181 191 193
197 199 211 223 227 229 233 239 241 251 257
263 269 271 277 281 283 293 307 311 313 317
331 337 347 349 353 359 367 373 379 383 389
397 401 409 419 421 431 433 439 443 449 457
461 463 467 479 487 491 499 503 509 521 523
541 547 557 563 569 571 577 587 593 599 601
607 613 617 619 631 641 643 647 653 659 661
673 677 683 691 701 709 719 727 733 739 743
751 757 761 769 773 787 797 809 811 821 823
827 829 839 853 857 859 863 877 881 883 887
907 911 919 929 937 941 947 953 967 971 977
983 991 997
```

1.10 Most Number of divisors:

Maximum Value	Number with Most Divisors	Number of Divisors
10^3	83,160	128
10^6	720,720	240
10^7	9,609,600	640
10^8	98,280,000	672
10^9	735,134,400	1,344
10^10	7,242,460,800	2,688
10^11	73,346,256,000	5,376
10^12	936,966,912,400	10,752

1.10 Divisibility rules for a number:

7	A rule for 7 is to double the last digit, subtract it from the rest of the number, and check if the result is divisible by 7. Repeat if necessary.
8	The last three digits form a number divisible by 8.
11	The difference between the sum of digits of odd, even places is divisible by 11.

2.2 Leap year

```
bool isLeap(ll n){
    if (n%100==0)
        return (n % 400 == 0);
    else
        return (n % 4 == 0);
}
```

2.3 Num of Leap year in between

```
ll calNum(ll y) {
   return (y/4) - (y/100) +(y/400);
}
ll leapNum(ll l, ll r) {
      return calNum(r) - calNum(--l);
}
```

2.4 Prll Calendar of any year

```
ll dayNumber(ll day, ll month, ll year){
   ll t[]={0,3,2,5,0,3,5,1,4,6,2,4};
   year -= month < 3;
   return (year + year / 4 - year / 100 +
      year / 400 + t[month - 1] + day) % 7;
}
string getMonthName(ll monthNumber) {
    string months[]={"January", "February",
        "March","April","May","June","July",
        "August","September", "October",
        "November", "December"};
    return (months[monthNumber]);
}
ll numberOfDays(ll monthNumber, ll year){
    if (monthNumber==1 && isLeapYear(year))
        return 29;
    ll monthDays[] = {31, 28, 31, 30, 31,
              30, 31, 31, 30, 31, 30, 31};
    return (monthDays[monthNumber]);
}
void prllCalendar(ll year) {
    prllf("       Calendar - %d\n\n",year);
    ll days;
    ll current = dayNumber(1, 1, year);
    // i--> Iterate through all the months
    // j--> Iterate through all the days of
            the month - i
    for (ll i = 0; i < 12; i++) {
        days = numberOfDays(i, year);
        cout << "          |" <<
            getMonthName(i).c_str()
            << "|" << endl;
        prllf(" Sun Mon Tue Wed Thu Fri
                 Sat\n");
        ll k;
        for (k = 0; k < current; k++)
            prllf("    ");
        for (ll j = 1; j <= days; j++) {
            prllf("%4d", j);
            if (++k > 6) {
                k = 0; cout << endl;
            }
        }
        if (k)
            cout << endl;
        cout <<
        "-----------------------------\n";
        current = k;
    }
}
```

2.6 BINARY EXPONENTIATION:(a^b)

```
ll binaryExp(ll base, ll power, ll MOD =
mod) {
    ll res = 1;
    while (power) {
        if (power & 1)
            res = (res * base) % MOD;
        base = ((base%MOD)*(base%MOD))%MOD;
        power /= 2;
    }
    return res;
}
```

2.7 BINARY EXPONENTIATION:(a^b^c)

```
//function call:
binaryExp(a, binaryExp(b, c, mod-1), mod)
```

1.10 Divisor Count:

```
ll maxVal = 1e6 + 1;
vector<ll> countDivisor(maxVal, 0);
void countingDivisor(){
    for (ll i = 1; i < maxVal; i++)
        for(ll j= i; j<maxVal;j+= i)
            countDivisor[j]++;
}
// count the number of divisors of all
numbers in a range.
```

2.8 Check is prime number-O(sqrt(n))

```
bool prime(ll n){
    if (n<2) return false;
    if (n<=3) return true;
    if (!(n%2) || !(n%3)) return false;
    for (ll i=5; i*i<=n; i+=6){
        if (!(n%i) || !(n%(i+2)))
             return false;
    }
    return true;
}
```

2.9 Prime factorization-O(sqrt(n))

```
int spf[N];
void sieve() {
  for (int i = 2; i < N; i++) {
    spf[i] = i;
  }
  for (int i = 2; i * i < N; i++) {
    if (spf[i] == i) {
      for (int j = i * i; j < N; j += i) {
        spf[j] = min(spf[j], i);
      }
    }
  }
}
// smallest prime factor of a number.
ll factor(ll n){
    ll a;
    if (n%2==0)
        return 2;
    for (a=3; a<=sqrt(n); a+=2){
        if (n%a==0)
            return a;
    }
    return n;
}
// complete factorization
ll r;
while (n>1){
    r = factor(n);prllf("%d", r);n /= r;}
// some facts about spf
```

suppose you have a number N = 120;
you represent it as N = 2^3 * 3^1 * 5^2
Now from this representation we can easily calculate the number of divisors of number N. Let's see how it works:

(i). we can take 2^3 in 4 different ways like 2^0, 2^1, 2^2, 2^3. In the same way we can take 3^1 in 2 ways(3^0, 3^1) and 5^2 in 3 ways(5^0, 5^1, 5^2).
(ii). Total number of divisor is = 4*2*3

suppose, N = $p_1^a \times p_2^b \times p_3^c$
number_of_divisors = $(a + 1) * (b + 1) * (c + 1)$
As like calculating the number of divisors, we can also calculate the sum of all divisors.

sum_of_divisors

$$\sigma(N) = \frac{p_1^{a+1}-1}{p_1-1} * \frac{p_2^{b+1}-1}{p_2-1} * \frac{p_3^{c+1}-1}{p_3-1}$$

2.10 Seive

```
const ll N = 1e7 + 5;
ll isprime[N];
void sieveOfEratosthenes() {
    for (ll i = 2; i < N; i++)
        isprime[i] = 1;
    for (ll i = 4; i < N; i += 2)
        isprime[i] = 0;
    for (ll i = 3; i * i < N; i += 2) {
        if (isprime[i]) {
            for (ll j = i * i; j<N; j+=i*2)
                isprime[j] = 0;
        }
    }
}
```

2.11 Optimized Seive(upto 1e9)

```
vector<ll> sieve(const ll N, const ll Q =
17, const ll L = 1 << 15) {
  static const ll rs[] = {1, 7, 11, 13, 17,
19, 23, 29};
  struct P {
    P(ll p) : p(p) {}
    ll p; ll pos[8];
  };
  auto approx_prime_count = [] (const ll N)
-> ll {
    return N > 60184 ? N / (log(N) - 1.1)
                     : max(1., N / (log(N) -
1.11)) + 1;
  };
  const ll v = sqrt(N), vv = sqrt(v);
  vector<bool> isp(v + 1, true);
  for (ll i = 2; i <= vv; ++i) if (isp[i]) {
    for (ll j = i * i; j <= v; j += i)
isp[j] = false;
  }
  const ll rsize = approx_prime_count(N +
30);
  vector<ll> primes = {2, 3, 5}; ll psize =
3;
  primes.resize(rsize);

  vector<P> sprimes; size_t pbeg = 0;
  ll prod = 1;
  for (ll p = 7; p <= v; ++p) {
    if (!isp[p]) continue;
    if (p <= Q) prod *= p, ++pbeg,
primes[psize++] = p;
    auto pp = P(p);
    for (ll t = 0; t < 8; ++t) {
      ll j = (p <= Q) ? p : p * p;
      while (j % 30 != rs[t]) j += p << 1;
      pp.pos[t] = j / 30;
    }
    sprimes.push_back(pp);
  }

  vector<unsigned char> pre(prod, 0xFF);
  for (size_t pi = 0; pi < pbeg; ++pi) {
    auto pp = sprimes[pi]; const ll p =
pp.p;
    for (ll t = 0; t < 8; ++t) {
      const unsigned char m = ~(1 << t);
      for (ll i = pp.pos[t]; i < prod; i +=
p) pre[i] &= m;
    }
  }
  const ll block_size = (L + prod - 1) /
prod * prod;
  vector<unsigned char> block(block_size);
unsigned char* pblock = block.data();
  const ll M = (N + 29) / 30;

  for (ll beg = 0; beg < M; beg +=
block_size, pblock -= block_size) {
    ll end = min(M, beg + block_size);
    for (ll i = beg; i < end; i += prod) {
      copy(pre.begin(), pre.end(), pblock +
i);
    }
    if (beg == 0) pblock[0] &= 0xFE;
    for (size_t pi = pbeg; pi <
sprimes.size(); ++pi) {
      auto& pp = sprimes[pi];
      const ll p = pp.p;
      for (ll t = 0; t < 8; ++t) {
        ll i = pp.pos[t]; const unsigned
char m = ~(1 << t);
        for (; i < end; i += p) pblock[i] &=
m;
        pp.pos[t] = i;
      }
    }
    for (ll i = beg; i < end; ++i) {
      for (ll m = pblock[i]; m > 0; m &= m -
1) {
        primes[psize++] = i * 30 +
rs[__builtin_ctz(m)];
      }
    }
  }
  assert(psize <= rsize);
  while (psize > 0 && primes[psize - 1] > N)
--psize;
  primes.resize(psize);
  return primes;
}
```

2.15 nth prime number O(log(logn))

```
vector<ll> nth_prime;
const ll MX = 86200005;
bitset<MX> visited;
void optimized_prime(){
    nth_prime.push_back(2);
    for(ll i=3; i<MX; i+=2){
            if(visited[i]) continue;
            nth_prime.push_back(i);
            if(1ll*i*i > MX) continue;
            for(ll j = i*i; j< MX; j+= i+i)
                visited[j] = true;
    }
}
```

2.15 nCr(more space, less time)

```
const ll MAX = 1e7 + 5;
vector<ll> fact(MAX), ifact(MAX), inv(MAX);
void factorial() {
    inv[1] = fact[0] = ifact[0] = 1;
    for (ll i = 2; i < MAX; i++)
        inv[i]=inv[mod%i]*(mod-mod/i)%mod;
    for (ll i = 1; i < MAX; i++)
        fact[i] = (fact[i - 1] * i) % mod;
    for (ll i = 1; i < MAX; i++)
        ifact[i]=ifact[i-1]*inv[i] % mod;
}
ll nCr(ll n, ll r) {
    if (r < 0 || r > n)
        return 0;
    return (ll)fact[n] * ifact[r] % mod *
ifact[n - r] % mod;
}
```

2.16 nCr(less space, more time)

```
const ll MAX = 1e7+10;
vector<ll> fact(MAX), inv(MAX);
void factorial(){
    fact[0] = 1;
    for (ll i = 1; i < MAX; i++)
        fact[i] = (i * fact[i - 1]) % MOD;
}
binaryExp(ll a, ll n, ll M = MOD); //needs to implement
void inverse(){
    for (ll i = 0; i < MAX; ++i)
        inv[i] = bigmod(fact[i], MOD - 2);
}
ll nCr(ll a, ll b){
    if (a < b or a < 0 or b < 0)
        return 0;
    ll de = (inv[b] * inv[a - b]) % MOD;
    return (fact[a] * de) % MOD;
}
```

2.16.1 nCr mod m where m is not prime

```
ll C_mod_p(ll n, ll k, ll p) {
    if (k > n) return 0;
    vector<ll> fac(p);
    fac[0] = 1;
    for (int i = 1; i < p; i++) fac[i] = fac[i - 1] * i % p;

    ll res = 1;
    while (n || k) {
        ll ni = n % p, ki = k % p;
        if (ki > ni) return 0;
        res = res * fac[ni] % p * modInv(fac[ki], p) % p * modInv(fac[ni - ki], p) % p;
        n /= p;
        k /= p;
    }
    return res;
}
// compute nCr mod composite m (non-prime)
ll nCr_mod_m(ll n, ll k, ll m) {
    // Step 1: factorize m
    vector<int> primes;
    int tmp = m;
    for (int i = 2; i * i <= tmp; i++) {
        if (tmp % i == 0) {
            primes.push_back(i);
            while (tmp % i == 0) tmp /= i;
        }
    }
    if (tmp > 1) primes.push_back(tmp);

    // Step 2: compute result mod each prime
    vector<ll> rem, mod;
    for (int p : primes) {
        rem.push_back(C_mod_p(n, k, p));
        mod.push_back(p);
    }
    // Step 3: Chinese Remainder Theorem (combine)
    ll res = 0;
    for (int i = 0; i < (int)mod.size(); i++) {
        ll Mi = m / mod[i];
        ll invMi = power(Mi, mod[i] - 2, mod[i]); // modular inverse
        res = (res + rem[i] * Mi % m * invMi % m) % m;
    }

    return res;
}


// nCr ends here
```

2.17 Factorial mod

```
//n! mod p : Here P is mod value
//For binaryExp we call 1.6 function
ll factmod (ll n, ll p){
    ll res = 1;
    while (n > 1){
        res=(res*binaryExp(p-1,n/p,p))%p;
        for (ll i = 2; i <= n % p; ++i)
            res=(res*i) %p;
            n /= p;
    }
    return ll (res % p);
}
```

2.19 PHI of N

// the positive llegers less than or equal to n that are relatively prime to n.

if $n = p_1^{a_1} * p_2^{a_2} *...* p_k^{a_k}$ then

$$\phi(n) = n * (1 - \frac{1}{p_1}) * (1 - \frac{1}{2}) *...* (1 - \frac{1}{p_k})$$

```
ll phi(ll n) {
    ll result = n;
    for (ll i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
```

2.20 PHI of 1 to N

```
const int N = 1e6 + 9;
int phi[N];
int phiS[N];
void totient()
{
    for (int i = 1; i < N; i++)
        phi[i] = i;
    for (int i = 2; i < N; i++)
    {
        if (phi[i] == i)
        {
            for (int j = i; j < N; j += i)
                phi[j] -= phi[j] / i;
        }
    }
    phiS[0] = phi[0];
    for (int i = 1; i < N; i++)
        phiS[i] = phiS[i - 1] + phi[i];
}
```

x^n mod m=x^(nmodφ(m)) modm

Fact: Summation of phi of divisors of N is equal to N. For example N = 10. Divisors of 10 are 1, 2, 5, 10. Hence,

$\phi(1) + \phi(2) + \phi(5) + \phi(10) = 1 + 1 + 4 + 4 = 10$

2.22 Catalan numbers

```
void catalan(ll n){
    ll res = 1;
    cout << res << " ";
    for (ll i = 1; i < n; i++) {
        res = (res *(4 * i - 2)) / (i + 1);
           cout << res << " ";
    }
}
```

2.24 EXT_GCD

```
// return {x,y} such that ax+by=gcd(a,b)
ll extended_euclid(ll a, ll b, ll &x, ll &y)
{
  if (b == 0) {
    x = 1; y = 0;
    return a;
  }
  ll x1, y1;
  ll d = extended_euclid(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
ll inverse(ll a, ll m) {
  ll x, y;
  ll g = extended_euclid(a, m, x, y);
  if (g != 1) return -1;
  return (x % m + m) % m;
}
```

2.27 Large Mod

```
ll mod(string& num, ll a){
    ll res = 0;
    for (ll i = 0; i < num.length(); i++)
        res = (res*10 + num[i] - '0') % a;
     return res;
}
```

2.28 divisor of n!

```
ll factorialDivisors(ll n) {
    ll result = 1;
    for (ll i=0;i < allPrimes.size(); i++){
        ll p = allPrimes[i]; ll exp = 0;
        while (p <= n) {
            exp = exp + (n / p);
           p = p * allPrimes[i];
        }
        result = result * (exp + 1);
    }
    return result;
}
```

2.20 10-ary to m-ary

```
char a[16] =
      {'0','1','2','3','4','5','6','7',
       '8','9','A','B','C','D','E','F'};
string tenToM(ll n, ll m){
    ll temp=n;
    string result="";
    while (temp!=0){
        result=a[temp%m]+result;
        temp/=m;
    }
    return result;
}
```

2.21 m-ary to 10-ary

```
string num = "0123456789ABCDE";
ll mToTen(string n, ll m){
    ll multi=1;
    ll result=0;
    for (ll i=n.size()-1; i>=0; i--)    {
        result += num.find(n[i])*multi;
        multi*=m;
    }
    return result;
}
```

2.33 Count 1's from 0 to n

```
ll cntOnes(ll n) {
    ll cnt = 0;
    for(ll i=1;i<=n;i<<=1) {
        ll x = (n + 1) / (i << 1);
        cnt += x * i;
        if((n + 1) % i && n & i)
            cnt += (n + 1) % i;
    }
    return cnt;
}
```

2.30 Disarrangement Formula

```
ll disarrange(ll n) {
    if (n == 1) return 0;
    if (n == 2) return 1;
    return (n - 1) * (disarrange(n - 1) +
                      disarrange(n - 2));
}
//D(n) = (n!)/e
```

2.29 Find numbers in between [L, R] which are divisible by all Array elements

```
void solve(ll* arr, ll N, ll L, ll R) {
    ll LCM = arr[0];
    for (ll i = 1; i < N; i++) {
        LCM = (LCM * arr[i]) /
            (__gcd(LCM, arr[i]));
    }
    if ((LCM<L&& LCM * 2 > R) || LCM > R){
         return;
    }
    ll k = (L / LCM) * LCM;
    if (k < L) k = k + LCM;
    for (ll i = k; i <= R; i = i + LCM)
        cout << i << ' ';
}
```

2.31 MSLCM

```
//For a given number N, maximum sum LCM
indicates the set of numbers whose LCM is N
and summation is maximum. Let, MSLCM(N)
denote this maximum sum of numbers. Given
the value of N you will have to find the
value: ∑(i=2->n) MSLCM(i)
ll MSLCM(ll n) {
      ll l = 1, r, val, ret = 0;
      while (l <= n) {
            val = n / l,r = n / val;
            ret += val * ((l+r)*(r-l+1)/2);
            l = r+1;
      }
      return ret-1;
}
```

2.34 Millar Rabin

```
using u64 = ull64_t;
using u128 = __ull128_t;
u64 binpower(u64 base, u64 e, u64 mod) {
  u64 result = 1;
  base %= mod;
  while (e) {
      if (e & 1) result = (u128)result *
base % mod;
      base = (u128)base * base % mod;
      e >>= 1;  }
  return result;}
```

```
bool check_composite(u64 n, u64 a, u64 d, ll
s) {
  u64 x = binpower(a, d, n);
  if (x == 1 || x == n - 1) return false;
  for (ll r = 1; r < s; r++) {
      x = (u128)x * x % n;
      if (x == n - 1) return false;   }
  return true;};
bool MillerRabin(u64 n, ll iter = 5) {   //
returns true if n is probably prime, else
returns false.
  if (n < 4) return n == 2 || n == 3;
  ll s = 0;
  u64 d = n - 1;
  while ((d & 1) == 0) {
      d >>= 1;
      s++;  }
  for (ll i = 0; i < iter; i++) {
      ll a = 2 + rand() % (n - 3);
      if (check_composite(n, a, d, s))
return false;
  }
  return true;
}
```

3 Algorithms

3.1 Modular Operation

Addition:

```
ll mod_add(ll a, ll b, ll MOD = mod){
    a = a % MOD, b = b % MOD;
    return (((a + b) % MOD) + MOD) % MOD;
}
```

Subtraction:

```
ll mod_sub(ll a, ll b, ll MOD = mod){
    a = a % MOD, b = b % MOD;
    return (((a - b) % MOD) + MOD) % MOD;
}
```

Multiplication:

```
ll mod_mul(ll a, ll b, ll MOD = mod){
    a = a % MOD, b = b % MOD;
    return (((a * b) % MOD) + MOD) % MOD;
}
```

Division:

```
ll mminvprime(ll a, ll b) {
    return binaryExp(a, b - 2, b);
}
ll mod_div(ll a, ll b, ll MOD = mod) {
    a = a % MOD, b = b % MOD;
    return (mod_mul(a, mminvprime(b, MOD),
MOD) + MOD) % MOD;
}
```

3.1 KMP Algorithm-O(n+m)

```
vector<ll> createLPS(string pattern) {
    ll n = pattern.length(), idx = 0;
    vector<ll> lps(n);
    for (ll i = 1; i < n;) {
        if (pattern[idx] == pattern[i]) {
            lps[i] = idx + 1;
            idx++, i++;
        }
        else {
            if (idx != 0)
                idx = lps[idx - 1];
            else
                lps[i] = idx, i++;
        }
    }
    return lps;
}
ll kmp(string text, string pattern) {
    ll cnt_of_match = 0, i = 0, j = 0;
    vector<ll> lps = createLPS(pattern);
    while (i < text.length()) {
        if (text[i] == pattern[j])
            i++, j++;// i->text,j->pattern
        else {
            if (j != 0)
                j = lps[j - 1];
            else
                i++;
        }
        if (j == pattern.length()) {
            cnt_of_match++;
            // the index where match found
-> (i - pattern.length());
            j = lps[j - 1];
        }
    }
    return cnt_of_match;
}
```

3.1 Hashing

```
const ll N = 2e5 + 5;
const ll MOD1 = 127657753, MOD2 = 987654319;
const ll p1 = 137, p2 = 277;
ll ip1, ip2;
pair<ll, ll> pw[N], ipw[N];
void prec() {
    pw[0] = {1, 1};
    for (ll i = 1; i < N; i++) {
        pw[i].first = 1LL * pw[i - 1].first
                          * p1 % MOD1;
        pw[i].second = 1LL * pw[i-1].second
                          * p2 % MOD2;
    }
    ip1 = binaryExp(p1, MOD1 - 2, MOD1);
    ip2 = binaryExp(p2, MOD2 - 2, MOD2);
    ipw[0] = {1, 1};
    for (ll i = 1; i < N; i++) {
        ipw[i].first = 1LL * ipw[i-1].first
                          * ip1 % MOD1;
        ipw[i].second = 1LL*ipw[i-1].second
                          * ip2 % MOD2;
    }
}
struct Hashing {
    ll n;
    string s;                    // 0 - indexed
    vector<pair<ll, ll>> hs;     // 1 - indexed
    Hashing() {}
    Hashing(string _s) {
        n = _s.size();
        s = _s;
        hs.emplace_back(0, 0);
        for (ll i = 0; i < n; i++) {
            pair<ll, ll> p;
            p.first = (hs[i].first + 1LL *
            pw[i].first * s[i] %
            MOD1)%MOD1;
            p.second = (hs[i].second + 1LL *
            pw[i].second * s[i]
      %MOD2)%MOD2;
            hs.push_back(p);
        }
    }
    pair<ll, ll> get_hash(ll l, ll r) {
        // 1 - indexed
        assert(1 <= l && l <= r && r <= n);
        pair<ll, ll> ans;
        ans.first = (hs[r].first - hs[l -
1].first + MOD1) * 1LL * ipw[l - 1].first %
MOD1;
        ans.second = (hs[r].second - hs[l -
1].second + MOD2) * 1LL * ipw[l - 1].second
% MOD2;
        return ans;
    }
    pair<ll, ll> get_hash() {
        return get_hash(1, n); }
};
```

3.1 Biginteger Operation

```
struct BigInteger {
    string str;
    // Constructor to initialize
    // BigInteger with a string
    BigInteger(string s) { str = s; }
    // Overload + operator to add
    // two BigInteger objects
BigInteger operator+(const BigInteger& b) {
        string a = str, c = b.str;
        ll alen=a.length(),clen=c.length();
        ll n = max(alen, clen);
        if (alen > clen)
            c.insert(0, alen - clen, '0');
        else if (alen < clen)
            a.insert(0, clen - alen, '0');
        string res(n + 1, '0');
        ll carry = 0;
        for (ll i = n - 1; i >= 0; i--) {
            ll digit=(a[i] -'0')+(c[i]-'0')
                      +carry;
            carry = digit / 10;
            res[i + 1] = digit % 10 + '0';
        }
        if (carry == 1) {
            res[0] = '1';
            return BigInteger(res);
        }
        else
           return BigInteger(res.substr(1));
    }

    // Overload - operator to subtract
    // first check which number is greater
and then subtract
BigInteger operator-(const BigInteger& b) {
        string a = str;
        string c = b.str;
        ll alen=a.length(),clen=c.length();
        ll n = max(alen, clen);
        if (alen > clen)
            c.insert(0, alen - clen, '0');
        else if (alen < clen)
            a.insert(0, clen - alen, '0');
        if (a < c) {
            swap(a, c);
            swap(alen, clen);
        }
        string res(n, '0');
        ll carry = 0;
        for (ll i = n - 1; i >= 0; i--) {
            ll digit =(a[i]-'0')-(c[i]-'0')
                        - carry;
            if (digit<0) digit+=10,carry=1;
            else carry = 0;
            res[i] = digit + '0';
        }
        // remove leading zeros
        ll i = 0;
        while (i < n && res[i] == '0')i++;
        if (i == n)
            return BigInteger("0");
        return BigInteger(res.substr(i));
    }

    // Overload * operator to multiply
    // two BigInteger objects
 BigInteger operator*(const BigInteger& b) {
        string a = str, c = b.str;
        ll alen=a.length(),clen=c.length();
        ll n = alen + clen;
        string res(n, '0');
        for (ll i = alen - 1; i >= 0;i--) {
            ll carry = 0;
            for(ll j=clen-1; j>=0; j--) {
                ll digit = (a[i] - '0') *
```

```
        (c[j-'0')+(res[i+j+1]-'0')+carry;
                carry = digit / 10;
                res[i+j+1]=digit % 10 + '0';
            }
            res[i] += carry;
        }
        ll i = 0;
        while (i < n && res[i] == '0')
            i++;
        if (i == n)
            return BigInteger("0");
        return BigInteger(res.substr(i));
    }

    // Overload << operator to output
    // BigInteger object
    friend ostream& operator<<(ostream& out,
const BigInteger& b) {
        out << b.str;
        return out;
    }
};
```

3.2 Find rank k in array

```
ll find(ll l, ll r, ll k){
    ll i=0,j=0,x=0,t=0;
    if (l==r) return a[l];
    x=a[(l+r)/2];
    t=a[x];
    a[x]=a[r];
    a[r]=t;
    i=l-1;
    for (ll j=l; j<=r-1; j++)
        if (a[j]<=a[r]){
            i++;
            t=a[i];
            a[i]=a[j];
            a[j]=t;
        }
    i++;
    t=a[i]; a[i]=a[r]; a[r]=t;
    if (i==k) return a[i];
    if (i<k) return find(i+1, r,k);
    return find(l, i-1, k);
}
```

3.3 InfixToPostFix

```
bool delim(char c) { return c == ' '; }
bool is_op(char c) {
    return c == '+' || c == '-' || c == '*'
           || c == '/' || c == '^';
}
bool is_unary(char c) {
      return c == '+' || c == '-';
}
ll priority(char op) {
    if (op < 0) return 3;
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 4;
    return -1;
}

void process_op(string& output, char op) {
    if (op < 0) {
        switch (-op) {
            case '+':
                output += "+ ";
                break;
            case '-':
                output += "- ";
                break;
        }
    }
    else {
        switch (op) {
            case '+':
                output += "+ ";
                break;
            case '-':
                output += "- ";
                break;
            case '*':
                output += "* ";
                break;
            case '/':
                output += "/ ";
                break;
            case '^':
                output += "^ ";
                break;
        }
    }
}

string InfixToPostFix(string& s) {
    string output;
    stack<char> op;
    bool may_be_unary = true;
    for (ll i = 0; i <(ll)s.size(); i++){
        if (delim(s[i]))
            continue;
        if (s[i] == '(') {
            op.push('(');
            may_be_unary = true;
        }
        else if (s[i] == ')') {
            while (op.top() != '(') {
               process_op(output, op.top());
               op.pop();
            }
            op.pop();
            may_be_unary = false;
        }
        else if (is_op(s[i])) {
            char cur_op = s[i];
            if (may_be_unary &&
is_unary(cur_op))
                cur_op = -cur_op;
            while (!op.empty() &&
                   ((cur_op >= 0 &&
priority(op.top()) >= priority(cur_op)) ||
                   (cur_op < 0 &&
priority(op.top()) > priority(cur_op)))) {
                process_op(output,
op.top());
                op.pop();
            }
            op.push(cur_op);
            may_be_unary = true;
        }
        else {
            char number;
            while (i < (ll)s.size() &&
isalnum(s[i]))
                number = s[i++];
            --i;
            output.push_back(number);
            output.push_back(' ');
            may_be_unary = false;
        }
    }
    while (!op.empty()) {
        process_op(output, op.top());
        op.pop();
    }
    return output;
}
```

3.4 Expression Parsing

```
bool delim(char c) { return c == ' '; }
```

```
bool is_op(char c) { return c == '+' || c ==
'-' || c == '*' || c == '/'; }

bool is_unary(char c) { return c == '+' || c
== '-'; }

ll priority(char op) {
    if (op < 0)  // unary operator
        return 3;
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    return -1;
}

void process_op(stack<ll>& st, char op) {
    if (op < 0) {
        ll l = st.top();
        st.pop();
        switch (-op) {
            case '+':
                st.push(l);
                break;
            case '-':
                st.push(-l);
                break;
        }
    }
    else {
        ll r = st.top();
        st.pop();
        ll l = st.top();
        st.pop();
        switch (op) {
            case '+':
                st.push(l + r);
                break;
            case '-':
                st.push(l - r);
                break;
            case '*':
                st.push(l * r);
                break;
            case '/':
                st.push(l / r);
                break;
        }
    }
}

ll evaluate(string& s) {
    stack<ll> st;
    stack<char> op;
    bool may_be_unary = true;
    for (ll i = 0; i < (ll)s.size(); i++) {
        if (delim(s[i]))
            continue;

        if (s[i] == '(') {
            op.push('(');
            may_be_unary = true;
        }
        else if (s[i] == ')') {
            while (op.top() != '(') {
                process_op(st, op.top());
                op.pop();
            }
            op.pop();
            may_be_unary = false;
        }
        else if (is_op(s[i])) {
            char cur_op = s[i];
            if (may_be_unary &&
is_unary(cur_op))
                cur_op = -cur_op;
            while (!op.empty() &&
                   ((cur_op >= 0 &&
priority(op.top()) >= priority(cur_op)) ||
                    (cur_op < 0 &&
priority(op.top()) > priority(cur_op)))) {
                process_op(st, op.top());
                op.pop();
            }
            op.push(cur_op);
            may_be_unary = true;
        }
        else {
            ll number = 0;
            while (i < (ll)s.size() &&
isalnum(s[i]))
                number = number * 10 +
s[i++] - '0';
            --i;
            st.push(number);
            may_be_unary = false;
        }
    }

    while (!op.empty()) {
        process_op(st, op.top());
        op.pop();
    }
    return st.top();
}
```

3.7 Kadane's Algorithm O(n)

```
// return maximum subarray sum.
ll kadense(ll arr[], ll n) {
    ll mxsm = arr[0], curr_s = arr[0];
    for (ll i = 1; i < n; i++){
        curr_s = max(arr[i], curr_s+arr[i]);
        mxsm = max(mxsm, curr_s);
    }
    return mxsm;
}
```

4 Data Structure

4.1 SEGMENT TREE

```
class SEGMENT_TREE {
   public:
    vector<ll> v;
    vector<ll> seg;
    SEGMENT_TREE(ll n) {
        v.resize(n + 5);
        seg.resize(4 * n + 5);
```

```
    }
//! initially: ti = 1, low = 1, high = n
//(number of elements in the array);
    void build(ll ti, ll low, ll high) {
        if (low == high) {
            seg[ti] = v[low];
            return;
        }
        ll mid = (low + high) / 2;
        build(2 * ti, low, mid);
        build(2 * ti + 1, mid + 1, high);
        seg[ti] = (seg[2*ti]+seg[2*ti+1]);
    }
//! initially: ti = 1, low = 1, high = n
//(number of elements in the array),
//(ql & qr)=user input in 1 based index;
    ll find(ll ti, ll tl, ll tr, ll ql,
            ll qr) {
        if (tl > qr || tr < ql) {
            return 0;
        }
        if (tl >= ql and tr <= qr)
            return seg[ti];
        ll mid = (tl + tr) / 2;
        ll l = find(2*ti, tl, mid, ql, qr);
        ll r = find(2*ti+1,mid+1,tr,ql,qr);
        return (l + r);
    }
//! initially: ti = 1, tl = 1, tr = n
//(number of elements in the array),
//id = user input in 1 based indexing,
//val = updated value;
    void update(ll ti, ll tl, ll tr, ll
                id, ll val) {
        if (id > tr or id < tl)
            return;
        if (id == tr and id == tl) {
            seg[til] = val;
            return;
        }
        ll mid = (tl + tr) / 2;
        update(2 * ti, tl, mid, id, val);
        update(2*ti+1,mid + 1, tr, id, val);
        seg[til] = (seg[2*ti]+seg[2*ti +
1]);
    }
};
// use 1 based indexing for input and
//queries and update;
```

4.2 FENWICK TREE

```
// Sum
struct FenwickTree {
    vector<ll> bit;  // binary indexed tree
    ll n;
    FenwickTree(ll n) {
        this->n = n;
        bit.assign(n, 0);
    }
    FenwickTree(vector<ll>a):
                FenwickTree(a.size()) {
        for (size_t i=0; i < a.size(); i++)
            add(i, a[i]);
    }
    ll sum(ll r) {
        ll ret = 0;
        for (; r >= 0; r = (r&(r + 1)) - 1)
            ret += bit[r];
        return ret;
    }
    ll sum(ll l, ll r) {
        return sum(r) - sum(l - 1);
    }
    void add(ll idx, ll delta) {
        for (; idx<n; idx = idx | (idx + 1))
            bit[idx] += delta;
    }
};

// minimum
struct FenwickTreeMin {
    vector<ll> bit;
    ll n;
    const ll INF = (ll)1e9;
    FenwickTreeMin(ll n) {
        this->n = n;
        bit.assign(n, INF);
    }
    FenwickTreeMin(vector<ll> a) :
                  FenwickTreeMin(a.size())
          {
        for (size_t i=0; i < a.size(); i++)
            update(i, a[i]);
    }
    ll getmin(ll r) {
        ll ret = INF;
        for (; r>=0; r = (r & (r + 1)) - 1)
            ret = min(ret, bit[r]);
        return ret;
    }
    void update(ll idx, ll val) {
        for (; idx<n; idx = idx | (idx + 1))
            bit[idx] = min(bit[idx], val);
    }
};
```

4.3 SEGMENT TREE LAZY

```
const int N = 2e5 + 9;
int a[N];

struct ST
{
#define lc (n << 1)
#define rc ((n << 1) | 1)
    long long t[4 * N], lazy_add[4 * N],
lazy_set[4 * N];
    bool has_set[4 * N];

    ST()
    {
        memset(t, 0, sizeof t);
        memset(lazy_add, 0, sizeof
lazy_add);
        memset(lazy_set, 0, sizeof
lazy_set);
        memset(has_set, 0, sizeof has_set);
    }

    inline void apply_set(int n, int b, int
e, long long v)
    {
        t[n] = v * (e - b + 1);
        has_set[n] = true;
        lazy_set[n] = v;
```

```
        lazy_add[n] = 0;
    }

    inline void apply_add(int n, int b, int
e, long long v)
    {
        t[n] += v * (e - b + 1);
        if (has_set[n])
            lazy_set[n] += 0;
        lazy_add[n] += v;
    }

    inline void push(int n, int b, int e)
    {
        if (b == e)
        {
            if (has_set[n])
                has_set[n] = false;
            if (lazy_add[n] != 0)
                lazy_add[n] = 0;
            return;
        }
        int mid = (b + e) >> 1;
        if (has_set[n])
        {
            apply_set(lc, b, mid,
lazy_set[n]);
            apply_set(rc, mid + 1, e,
lazy_set[n]);
            has_set[n] = false;
        }
        if (lazy_add[n] != 0)
        {
            apply_add(lc, b, mid,
lazy_add[n]);
            apply_add(rc, mid + 1, e,
lazy_add[n]);
            lazy_add[n] = 0;
        }
    }

    inline void pull(int n)
    {
        t[n] = t[lc] + t[rc];
    }

    void build(int n, int b, int e)
    {
        has_set[n] = false;
        lazy_add[n] = 0;
        if (b == e)
        {
            t[n] = a[b];
            return;
        }
        int mid = (b + e) >> 1;
        build(lc, b, mid);
        build(rc, mid + 1, e);
        pull(n);
    }

    void add(int n, int b, int e, int i, int
j, long long v)
    {
        if (j < b || e < i)
            return;
        if (i <= b && e <= j)
        {
            apply_add(n, b, e, v);
            return;
        }
        push(n, b, e);
        int mid = (b + e) >> 1;
        add(lc, b, mid, i, j, v);
        add(rc, mid + 1, e, i, j, v);
        pull(n);
    }

    void setv(int n, int b, int e, int i,
int j, long long v)
    {
        if (j < b || e < i)
            return;
        if (i <= b && e <= j)
        {
            apply_set(n, b, e, v);
            return;
        }
        push(n, b, e);
        int mid = (b + e) >> 1;
        setv(lc, b, mid, i, j, v);
        setv(rc, mid + 1, e, i, j, v);
        pull(n);
    }

    long long query(int n, int b, int e, int
i, int j)
    {
        if (i > e || b > j)
            return 0;
        if (i <= b && e <= j)
            return t[n];
        push(n, b, e);
        int mid = (b + e) >> 1;
        return query(lc, b, mid, i, j) +
query(rc, mid + 1, e, i, j);
    }
};
```

4.5 TRIE

```
const ll N = 26;
class Node {
 public:
  ll EoW;
  Node* child[N];
  Node() {
      EoW = 0;
      for (ll i = 0; i < N; i++) child[i] =
NULL;
  }};

void insert(Node* node, string s) {
  for (size_t i = 0; i < s.size(); i++) {
      ll r = s[i] - 'A';
      if (node->child[r] == NULL)
node->child[r] = new Node();
      node = node->child[r];  }
  node->EoW += 1;}
ll search(Node* node, string s) {
  for (size_t i = 0; i < s.size(); i++) {
      ll r = s[i] - 'A';
      if (node->child[r] == NULL) return 0;
  }  return node->EoW;}

void prll(Node* node, string s = "") {
  if (node->EoW) cout << s << "\n";
```

```
  for (ll i = 0; i < N; i++) {
      if (node->child[i] != NULL) {
      char c = i + 'A';
      prll(node->child[i], s + c);
      }  }}

bool isChild(Node* node) {
  for (ll i = 0; i < N; i++)
      if (node->child[i] != NULL) return
true;
  return false;}

bool isJunc(Node* node) {
  ll cnt = 0;
  for (ll i = 0; i < N; i++) {
      if (node->child[i] != NULL) cnt++;
  }
  if (cnt > 1) return true;
  return false;
}

ll trie_delete(Node* node, string s, ll k =
0) {
  if (node == NULL) return 0;
  if (k == (ll)s.size()) {
      if (node->EoW == 0) return 0;
      if (isChild(node)) {
      node->EoW = 0;
      return 0;
      }
      return 1;
  }
  ll r = s[k] - 'A';
  ll d = trie_delete(node->child[r], s, k +
1);
  ll j = isJunc(node);
  if (d) delete node->child[r];
  if (j) return 0;
  return d;
}
void delete_trie(Node* node) {
  for (ll i = 0; i < 15; i++) {
      if (node->child[i] != NULL)
delete_trie(node->child[i]);
  }
  delete node;
}
```

4.6 DSU

```
class DisjollSet{
    vector<ll> par, sz, minElmt, maxElmt,
cntElmt;

   public:
    DisjollSet(ll n){
        par.resize(n + 1);
        sz.resize(n + 1, 1);
        minElmt.resize(n + 1);
        maxElmt.resize(n + 1);
        cntElmt.resize(n + 1, 1);
        for (ll i = 1; i <= n; i++)
            par[i]=minElmt[i]=maxElmt[i]=i;
    }
    ll findUPar(ll u) {
        if (u == par[u])
            return u;
        return par[u] = findUPar(par[u]);
    }
    void unionBySize(ll u, ll v){
        ll pU = findUPar(u);
        ll pV = findUPar(v);
        if (pU == pV)
            return;
        if (sz[pU] < sz[pV])
            swap(pU, pV);
        par[pV] = pU;
        sz[pU] += sz[pV];
        cntElmt[pU] += cntElmt[pV];
        minElmt[pU] = min(minElmt[pU],
                          minElmt[pV]);
        maxElmt[pU] = max(maxElmt[pU],
                          maxElmt[pV]);
    }
    ll getMinElementIntheSet(ll u){
        return minElmt[findUPar(u)];
    }
    ll getMaxElementIntheSet(ll u){
        return maxElmt[findUPar(u)];
    }
    ll getNumofElementIntheSet(ll u){
        return cntElmt[findUPar(u)];
    }
};
```

4.6 HLD

```
ll par[N], sub_tree_sz[N], heavy[N],
wt_from_parent[N], depth[N], head[N],
position[N];
vector<pair<ll, ll>> gd[N];

// HLD part start
ll dfs(ll node, ll p) {
    par[node] = p;
    sub_tree_sz[node] = 1;
    heavy[node] = -1;

    for (auto [v, w] : gd[node]) {
        if (v == p)
            continue;
        depth[v] = depth[node] + 1;
        wt_from_parent[v] = w;
        sub_tree_sz[node] += dfs(v, node);
        if (heavy[node] == -1 ||
sub_tree_sz[v] > sub_tree_sz[heavy[node]]) {
            heavy[node] = v;
        }
    }
    return sub_tree_sz[node];
}
ll pos;
void decompose(ll node, ll hd) {
    head[node] = hd;
    position[node] = ++pos;
    if (heavy[node] != -1) {
        decompose(heavy[node], hd);
    }
    for (auto [v, w] : gd[node]) {
        if (v != par[node] && v !=
heavy[node]) {
            decompose(v, v);
        }
    }
}

// HLD part end


// in main function
ll n, m;
cin >> n;
SEGMENT_TREE seg(n); // Lazy if needed
vector<ll>edge_u(n),edge_v(n), edge_node(n);

for (int i = 1; i < n; i++) {
    ll u, v, wt = 1;
    cin >> u >> v >> wt;
    gd[u].push_back({v, wt});
```

```
    gd[v].push_back({u, wt});
    edge_u[i] = u;
    edge_v[i] = v;
}

dfs(1, -1);
pos = 0;
decompose(1, 1);

for (int i = 1; i <= n; i++) {
    // seg.v[position[i]] = val[i];  // for
node value
    seg.v[position[i]] = wt_from_parent[i];
// for edge value
}

// work on a specific edge
for (int i = 1; i < n; i++) {
    ll u = edge_u[i], v = edge_v[i];
    edge_node[i] = (depth[u] > depth[v]) ? u
: v;
}

seg.build(1, 1, n);

auto updatePath = [&](ll u, ll v, ll x) {
    while (head[u] != head[v]) {
        if (depth[head[u]] < depth[head[v]])
            swap(u, v);
        seg.update(1, 1, n,
position[head[u]], position[u], x);
        u = par[head[u]];
    }
    if (depth[u] > depth[v])
        swap(u, v);
    // edge value
    if (u != v) {
        seg.update(1, 1, n, position[u] + 1,
position[v], x);
    }
    // node value
    // seg.update(1, 1, n, position[u],
position[v], x);
};

auto querypath = [&](ll u, ll v) {
    ll ans = -inf;
    while (head[u] != head[v]) {
        if (depth[head[u]] < depth[head[v]])
            swap(u, v);
        ans = max(ans, seg.query(1, 1, n,
position[head[u]], position[u]));
        u = par[head[u]];
    }
    if (depth[u] > depth[v])
        swap(u, v);
    // upward + downward
    if (u != v) {
        ans = max(ans, seg.query(1, 1, n,
position[u] + 1, position[v]));
    }
    // only upward
    // ans = max(ans, seg.query(1, 1, n,
position[u],
    //
position[v]));  // for node value
    return ans;
};


seg.update(1, 1, n, position[edge_node[s]],
position[edge_node[s]], x); // single point
update. if path update need call update path
cout << querypath(x, s) << '\n';
```

5 Dynamic Programming

5.0 Knapsack

```
ll n, W, w[N], v[N], dp[N][100005];
ll rec(ll i, ll weight)
{    if (i == n + 1) return 0;
    if (dp[i][weight] != -1) return
dp[i][weight]; ll ans = rec(i + 1, weight);
    if (weight + w[i] <= W)  ans = max(ans,
rec(i + 1, weight + w[i]) + v[i]);
    return dp[i][weight] = ans;}
```

5.1 LCS O(n*m)

```
 //LCS DP Table and LCS Length

vector<vector<ll>>dp(s.size()+1,vector<ll>(t
.size()+1));
    for(ll i=1;i<=s.size();i++){
        for(ll j=1;j<=t.size();j++){

if(s[i-1]==t[j-1])dp[i][j]=dp[i-1][j-1]+1;
            else
dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
        }
    }
    cout<<dp[s.size()][t.size()]<<"\n";

    //Any LCS String
    string a;
    ll i=s.size();
    ll j=t.size();
    while(i>0 && j>0){
        if(s[i-1]==t[j-1]){
            a+=s[i-1];
            i--,j--;
```

```
        }
        else if(dp[i][j-1]>dp[i-1][j])j--;
        else i--;
    }
    reverse(a.begin(),a.end());
    cout<<a<<"\n";

    //Lexicographically smallest LCS

vector<vector<string>>dpp(s.size()+1,vector<
string>(t.size()+1));
    for(ll i=1;i<=s.size();i++){
        for(ll j=1;j<=t.size();j++){

if(s[i-1]==t[j-1])dpp[i][j]=dpp[i-1][j-1]+s[
i-1];
            else
if(dp[i][j-1]<dp[i-1][j])dpp[i][j]=dpp[i-1][
j];
            else
if(dp[i][j-1]>dp[i-1][j])dpp[i][j]=dpp[i][j-
1];
            else
dpp[i][j]=min(dpp[i][j-1],dpp[i-1][j]);
        }
    }
    cout<<dpp[s.size()][t.size()]<<"\n";

    //number of distinct LCS sequences
    ll MOD=1e9+7;

vector<vector<ll>>cnt(s.size()+1,vector<ll>(
t.size()+1,1));
    for(ll i=1;i<=s.size();i++){
        for(ll j=1;j<=t.size();j++){

if(s[i-1]==t[j-1])cnt[i][j]=cnt[i-1][j-1];
            else
if(dp[i][j-1]<dp[i-1][j])cnt[i][j]=cnt[i-1][
j];
            else
if(dp[i][j-1]>dp[i-1][j])cnt[i][j]=cnt[i][j-
1];
            else{

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

if(dp[i][j]==dp[i-1][j-1])cnt[i][j]-=cnt[i-1
][j-1];

cnt[i][j]=(cnt[i][j]+MOD)%MOD;
            }   }   }
```

Many problems can be solved using LCS techniques.

- Longest Increasing Substring
  To solve this, we just care about when two char equals. Rest of the things
  should be neglected.
- Longest Palindromic Subsequence(LPS)
  To solve this, we just take a new string which is the reverse of the original string. Then just call the LCS function to find LPS.
- Minimum insertions to make a string palindrome
  To solve this, we just basically do string length - LPS.
  Why this? Let's take an example:
  string s = aabca;
  Let's say **aca** is our LPS. Now we find how many char we need to insert to make the string palindrome while our LPS is fixed.
  a ab c a now to make the string palindrome we just need to insert the reverse of **ab** after c. So the new string looks like **a ab c ba a**
- Minimum Number of Deletions and Insertions to make the string equals
  To solve this we just find the LCS of those string then just do:
  n + m - 2 * LCS.length()
  where n, m = strings length

5.2 MCM O(n^3)

```
const ll N = 1005;
vector<ll> v;
ll dp[N][N], mark[N][N];
ll MCM(ll i, ll j) {
    if (i == j)
        return dp[i][j] = 0;
    if (dp[i][j] != -1)
        return dp[i][j];
    ll mn = INT_MAX;
    for (ll k = i; k < j; k++) {
        ll x = mn;
        mn = min(mn, MCM(i, k) + MCM(k + 1,
j) + v[i - 1] * v[k] * v[j]);
        if (x != mn)
            mark[i][j] = k;
    }
    return dp[i][j] = mn;
}

void prll_order(ll i, ll j) {
    if (i == j)
        cout << "X" << i;
    else {
        cout << "(";
        prll_order(i, mark[i][j]);
        prll_order(mark[i][j] + 1, j);
        cout << ")";
    }
}
// memset(dp, -1, sizeof dp);
// prll_order(1, n);
```

5.3 Length of LIS O(nlogn)

```
vector<ll> v = {7, 3, 5, 3, 6, 2, 9, 8};
vector<ll> seq;
/*here we basically check is the current
element from v is greater than the last
element of the sequence.
 if it is then push it to the seq array and
if not then replace that index value.
let's take an example: v = 7 3 5 3 6 2 9 8
1st iteration seq = 7;
2nd iteration seq = 3;
3rd iteration seq = 3 5;
4th iteration seq = 3 3;
5th iteration seq = 3 3 6;
6th iteration seq = 2 3 6;
7th iteration seq = 2 3 6 9;
8th iteration seq = 2 3 6 8;
*/
for (auto i : v) {
    auto id = lower_bound(seq.begin(),
seq.end(), i);
    if (id == seq.end())
```

```
        seq.push_back(i);
    else
        seq[id - seq.begin()] = i;
}
cout << seq.size() << endl;
```

5.4 LCIS O(n * m)

```
ll a[100]= {0}, b[100]= {0}, f[100]= {0};
ll n=0, m=0;
ll main(void){
    cin >> n;
    for (ll i=1; i<=n; i++) cin >> a[i];
    cin >> m;
    for (ll i=1; i<=m; i++) cin >> b[i];
    for (ll i=1; i<=n; i++){
        ll k=0;
        for (ll j=1; j<=m; j++){
            if (a[i]>b[j] && f[j]>k)
                k=f[j];
            else if (a[i]==b[j] && k+1>f[j])
                f[j]=k+1;
        }
    }
    ll and=0;
    for (ll i=1; i<=m; i++)
        if (f[i]>ans) ans=f[i];
    cout << and << endl;
    return 0;
}
```

5.5 Maximum submatrix

```
ll a[150][150]= {0};
ll c[200]= {0};
ll maxarray(ll n){
    ll b=0, sum=-100000000;
    for (ll i=1; i<=n; i++){
        if (b>0) b+=c[i];
        else b=c[i];
        if (b>sum) sum=b;
    }
    return sum;
}

ll maxmatrix(ll n){
    ll sum=-100000000, max=0;
    for (ll i=1; i<=n; i++){
        for (ll j=1; j<=n; j++)
            c[j]=0;
        for (ll j=i; j<=n; j++){
            for (ll k=1; k<=n; k++)
                c[k]+=a[j][k];
            max=maxarray(n);
            if (max>sum) sum=max;
        }
    }
    return sum;
}
ll main(void){
    ll n=0;
    cin >> n;
    for (ll i=1; i<=n; i++)
        for (ll j=1; j<=n; j++)
            cin >> a[i][j];
    cout << maxmatrix(n);
    return 0;
}
```

5.6 SOS DP

```
// sum over subsets
  for (int i = 0; i < B; i++) {
    for (int mask = 0; mask < (1 << B);
mask++) {
      if ((mask & (1 << i)) != 0) {
        f[mask] += f[mask ^ (1 << i)];
      }
    }
  }

  // sum over supersets
  for (int i = 0; i < B; i++) {
    for (int mask = (1 << B) - 1 ; mask >= 0
; mask--) {
      if ((mask & (1 << i)) == 0) g[mask] +=
g[mask ^ (1 << i)] ;
    }
  }
//submask
for (int mask = 1; mask < (1 << 5); mask++)
    {
        for (int submask = mask; submask >
0; submask = ((submask - 1) & mask))
        {
            int subset = mask ^ submask;

        }
    }
```

5.7 All possible SubArraySum in O(1)

```
bitset<100005> bs = 1;
    for (auto i : a)
    {
        bs |= (bs << i); // if previous 1
value pos is possible now ith bit or ith sm
is also possible
    }
    cout << bs.count() - 1 << endl;
    for (ll i = 1; i <= 100003; i++)
        if (bs[i])
            cout << i << " ";
    cout << endl;
```

5.8 Range DP

```
ll f(ll st,ll ed){
    if(st>ed)return 0;
    if(st==ed)return dp[st][ed]=1;
    if(dp[st][ed]!=-1)return dp[st][ed];

    ll cnt=1+f(st+1,ed);
    for(ll i=st+1;i<=ed;i++){
        if(a[st]==a[i]){

cnt=min(cnt,f(st+1,i-1)+f(i,ed));
        }    }
    return dp[st][ed]=cnt;}
```

5.9 Bitmask DP

```
check: (mask & (1<<i))
set: (mask | (1<<i))
clear: if(ith bit 1) → (mask - (1<<i))
ll n,mod=1e9+7;
vector<vector<ll>>a(21,vector<ll>(21));
vector<vector<ll>>dp(21,vector<ll>((1<<21),-
1));
ll f(ll m,ll mask){
    if(m==n)return 1;
    if(dp[m][mask]!=-1)return dp[m][mask];
    ll cnt=0;
    for(ll i=0;i<n;i++){
        if(a[m][i] && !(mask&(1<<i))){
            cnt+=f(m+1,(mask|(1<<i)));
            cnt%=mod;
        }    }
    return dp[m][mask]=cnt;}
//call by f(0,0)
```

6 Graph Theory

6.1 SPFA – Optimal BF O(V * E)

```
ll q[3001]= {0};// queue for node
```

```
it d[1001]= {0}; // record shortest path
from start to ith node
bool f[1001]= {0};
ll a[1001][1001]= {0}; // adjacency list
ll w[1001][1001]= {0}; // adjacency matrix
ll main(void) {
  ll n=0, m=0;
  cin >> n >> m;
  for (ll i=1; i<=m; i++){
  ll x=0, y=0, z=0;
cin >> x >> y >> z;
// node x to node y has weight z
a[x][0]++;
 a[x][a[x][0]]=y;
  w[x][y]=z;
/*
// for undirected graph
 a[x][0]++;
 a[y][a[y][0]]=x;
w[y][x]=z;
*/
}
ll s=0, e=0;
cin >> s >> e; // s: start, e: end
SPFA(s);
cout << d[e] << endl;
return 0;
}
void SPFA(ll v0){
    ll t,h,u,v;
    for (ll i=0; i<1001; i++) d[i]=INT_MAX;
    for (ll i=0; i<1001; i++) f[i]=false;
    d[v0]=0;
    h=0;
    t=1;
    q[1]=v0;
    f[v0]=true;
    while (h!=t){
        h++;
        if (h>3000) h=1;
        u=q[h];
        for (ll j=1; j<=a[u][0]; j++){
            v=a[u][j];
            if (d[u]+w[u][v]<d[v]) // change
to > if calculating longest path
            {
                d[v]=d[u]+w[u][v];
                if (!f[v]){
                    t++;
                    if (t>3000) t=1;
                    q[t]=v;
                    f[v]=true;
                }
            }
        }
        f[u]=false;
    }
}
```

6.2 Dijkstra O(V + ElogV)

```
typedef pair<ll, ll> pairi;
ll N = 20000 + 5;
vector<vector<pairi>> adj(N);
vector<ll> dis(N, inf), parent(N);

void dijkstra(ll src) {
    priority_queue<pairi, vector<pairi>,
                   greater<pairi>> pq;
    dis[src] = 0;
    pq.push({0, src});
    while (pq.size()) {
        auto top = pq.top();
        pq.pop();
        for (auto i : adj[top.second]) {
            ll v = i.first;
            ll wt = i.second;
            if (dis[v]>dis[top.second]+wt) {
                dis[v]=dis[top.second]+wt;
                pq.push({dis[v], v});
                 parent[v] = top.second
            }
        }
    }
}
ll node = n;
while (parent[node] != node){
  path.push_back(node);
  node = parent[node]; }
path.push_back(1);
```

6.3 BellmanFord O(V.E)

```
vector<ll> dist;
vector<ll> parent;
vector<vector<pair<ll, ll>>> adj;
// resize the vectors from main function

void bellmanFord(ll num_of_nd, ll src) {
    dist[src] = 0;
    for (ll step=0;step<num_of_nd;step) {
        for (ll i = 1; i<=num_of_nd; i++) {
            for (auto it : adj[i]) {
                ll u = i;
                ll v = it.first;
                ll wt = it.second;
                if (dist[u] != inf &&
               ((dist[u] + wt) < dist[v]))
            {
                    if(step==num_of_nd - 1){
                        cout << "Negative
                              cycle
                  found\n";
                        return;
                    }
                    dist[v] = dist[u] + wt;
                    parent[v] = u;
                } } } }
    for (ll i = 1; i <= num_of_nd; i++)
        cout << dist[i] << " ";
    cout << endl;
}
```

6.4 Floyd-Warshall algorithm O(n^3)

```
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

typedef vector<ll> VI;
typedef vector<VI> VVI;

bool FloydWarshall (VVT &w, VVI &prev){
  ll n = w.size();
  prev = VVI (n, VI(n, -1));

  for (ll k = 0; k < n; k++){
    for (ll i = 0; i < n; i++){
      for (ll j = 0; j < n; j++){
        if (w[i][j] > w[i][k] + w[k][j]){
          w[i][j] = w[i][k] + w[k][j];
          prev[i][j] = k;}}}}

  // check for negative weight cycles
  for(ll i=0;i<n;i++)
    if (w[i][i] < 0) return false;
  return true;
}
```

6.5 Topological sort

```
map<string, vector<string>> adj;
map<string, ll> degree;
set<string> nodes;
vector<string> ans;
```

```
// adj: graph input, degree: cnt indegree,
// node: unique nodes, ans: path
ll c = 0;
void topo_sort() {
    queue<string> qu;
// traverse all the nodes and check if its
degree is 0 or not..
    for (string i : nodes) {
        if (degree[i] == 0) qu.push(i);
    }
    while (!qu.empty()) {
        string top = qu.front();
        qu.pop();
        ans.push_back(top);
        for (string i : adj[top]) {
            degree[i]--;
            if (degree[i] == 0) {
                qu.push(i);}}}}
```

6.6 Kruskal O(ElogE)

```
typedef pair<ll, ll> edge;
class Graph {
    vector<pair<ll, edge>> G, T;
    vector<ll> parent;
    ll cost = 0;

   public:
    Graph(ll n) {
        for (ll i = 0; i < n; i++)
            parent.push_back(i);
    }

    void add_edges(ll u,ll v,ll wt) {
        G.push_back({wt, {u, v}});
    }

    ll find_set(ll n) {
        if (n == parent[n])
            return n;
        else
            return find_set(parent[n]);
    }

    void union_set(ll u, ll v) {
        parent[u] = parent[v];
    }

    void kruskal() {
        sort(G.begin(), G.end());
        for (auto it : G) {
         ll uRep=find_set(it.second.first);
        ll vRep=find_set(it.second.second);
            if (uRep != vRep) {
                cost += it.first;
                T.push_back(it);
                union_set(uRep, vRep);
            }     }    }

    ll get_cost() { return cost; }
    void prll() {
        for (auto it : T)
            cout << it.second.first << " "
                 << it.second.second <<
            "->"
                 << it.first << endl;
    }
};

// g.add_edges(u, v, wt);
// g.kruskal();
```

6.7 Prim – MST O(ElogV)

```
typedef pair<ll, ll> pii;

class Prims {
    map<ll, vector<pii>> graph;
    map<ll, ll> visited;

   public:
    void addEdge(ll u, ll v, ll w) {
        graph[u].push_back({v, w});
        graph[v].push_back({u, w});
    }

    vector<ll> path(pii start) {
       vector<ll> ans;
       priority_queue<pii, vector<pii>,
                        greater<pii>> pq;
                        // cost vs node
       pq.push({start.second, start.first});
       while (!pq.empty()) {
            pair<ll, ll> curr = pq.top();
            pq.pop();
            if (visited[curr.second])
                continue;
            visited[curr.second] = 1;
            ans.push_back(curr.second);
            for (auto i:graph[curr.second]){
               if (visited[i.first])
                  continue;
               pq.push({i.second, i.first});
            }       }
       return ans;
    }};
```

6.8 Eulerian circuit O(V+E)

```
unordered_map<ll, ll> Start, End, Val;
unordered_map<ll, pair<ll, ll>> Range;
ll start = 0;
void dfs(ll node){
    visited[node] = true;
    Start[node] = start++;
    for (auto child : adj[node]){
        if (!visited[child])
            dfs(child);
    }
    End[node] = start - 1;
}
dfs(1);
vector<ll> FlatArray(start + 5);
for (auto i : Start){
    FlatArray[i.second] = Val[i.first];
    Range[i.first]=
                  {i.second,
            End[i.first]};
}
```

6.9 LCA

```
ll n, l;
vector<vector<ll>> adj;

ll timer;
vector<ll> tin, tout;
vector<vector<ll>> up;

void dfs(ll v, ll p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (ll i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (ll u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(ll u, ll v)
```

```
{
    return tin[u] <= tin[v] && tout[u] >=
tout[v];
}

ll lca(ll u, ll v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (ll i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(ll root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<ll>(l + 1));
    dfs(root, root);
}
```

6.10 Min cost max flow

```
struct Edge{
    ll from, to, capacity, cost;
};
vector<vector<ll>> adj, cost, capacity;
const ll INF = 1e9;
void shortest_paths(ll n, ll v0, vector<ll>&
d, vector<ll>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<ll> q;
    q.push(v0);
    p.assign(n, -1);
    while (!q.empty()) {
        ll u = q.front();
        q.pop();
        inq[u] = false;
        for (ll v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] >
d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
            }}}}}
ll min_cost_flow(ll N, vector<Edge> edges,
ll K, ll s, ll t) {
    adj.assign(N, vector<ll>());
    cost.assign(N, vector<ll>(N, 0));
    capacity.assign(N, vector<ll>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }
    ll flow = 0;
    ll cost = 0;
    vector<ll> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;
        // find max flow on that path
        ll f = K - flow;
        ll cur = t;
        while (cur != s) {
            f = min(f,
capacity[p[cur]][cur]);
            cur = p[cur];
        }
        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```

6.11 SCC

```
unordered_map<ll, vector<ll>> adj, InvAdj;
stack<ll> order;
unordered_map<ll, bool> visited;
unordered_map<ll, vector<ll>> all_scc;
unordered_map<ll, ll> compId;
void dfs_for_start(ll curr){
    visited[curr] = 1;
    for (auto i : adj[curr])
        if (!visited[i])
            dfs_for_start(i);
    order.push(curr);
}
vector<ll> curr_comp;
void dfs_for_scc(ll curr){
    visited[curr] = 1;
    for (auto i : InvAdj[curr])
        if (!visited[i])
            dfs_for_scc(i);
    curr_comp.push_back(curr);
}
inline void scc(){
    ll n, e, u, v;
    cin >> n >> e;
    for (ll i = 0; i < e; i++){
        cin >> u >> v;
        adj[u].push_back(v);
        InvAdj[v].push_back(u);
    }
    for (ll i = 1; i <= n; i++)
        if (!visited[i])
            dfs_for_start(i);
    visited.clear();
    while (!order.empty()){
        if (!visited[order.top()]){
            curr_comp.clear();
            dfs_for_scc(order.top());
            ll sz = all_scc.size() + 1;
            all_scc[sz] = curr_comp;
            for (auto i : curr_comp)
                compId[i] = sz;
        }
        order.pop();
    }
}
no. of ways and min cost of connecting the
sccs
const ll MOD = 1e9 + 7, N = 1e5 + 2, INF =
1e18 + 2;
ll n, m, comp[N];
```

```
vector<ll> adj[N], rev[N];
bitset<N> vis;
void DFS1(ll u, stack<ll> &TS){
    vis[u] = true;
    for (ll v : adj[u])
        if (!vis[v])
            DFS1(v, TS);
    TS.push(u);
}
void DFS2(ll u, const ll scc_no, ll
&min_cost, ll &ways, vector<ll> &cost){
    vis[u] = true;
    comp[u] = scc_no;
    for (ll v : rev[u])
        if (!vis[v]){
            if (min_cost == cost[v])
                ++ways;
            else if (min_cost > cost[v]){
                ways = 1;
                min_cost = cost[v];
            }
            DFS2(v, scc_no, min_cost, ways,
                 cost);
        }
}
signed main(){
    FIO cin >> n;
    vector<ll> cost(n + 1);
    for (ll i = 1; i <= n; ++i)
        cin >> cost[i];
    cin >> m;
    while (m--){
        ll u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        rev[v].push_back(u);
    }
    ll tot = 0, ways = 1;
    stack<ll> TS;
    for (ll i = 1; i <= n; ++i)
        if (!vis[i])
            DFS1(i, TS);
    vis.reset();
    ll scc_no = 0;
    while (!TS.empty()){
        ll u = TS.top();
        TS.pop();
        if (!vis[u]){
            ll tmp_cst = cost[u], tmp_ways =
1;
            DFS2(u, ++scc_no, tmp_cst,
                 tmp_ways, cost);
            tot += tmp_cst;
            ways = (ways * tmp_ways) % MOD;
        }
    }
    cout << tot << ' ' << ways;
}//TC: O(V+E)
```

6.12 Biparitite

```
const ll N=1000;
ll adj[N][N];
ll n,e;
bool isBicolored(ll s){
    ll colorArray[n];
    for(ll i=0;i<n;i++)
        colorArray[i]=-1; //init no color;
    queue<ll>q;
    q.push(s);
    colorArray[s]=1; //assigning first color
    while(!q.empty()){
        ll senior = q.front();
        q.pop();
        if(adj[senior][senior]==1)
            return false;
        for(ll i=0;i<n;i++){
            ll junior=i;
            if(adj[senior][junior]==1){

if(colorArray[junior]==colorArray[senior])
//successor(child/junior) having same color
                    return false;

///if(colorArray[junior]!=-1) continue;
///not same color but have a color
                else
if(colorArray[junior]==-1){        ///No
color assigned
                    q.push(junior);

colorArray[junior]=!colorArray[senior];
///assigning diff color
 }}}} return true;}
```

6.13 Two farthest node

```
vector<ll>adj[30001];
map<pair<ll,ll>,ll>weight;
map<ll,ll>vis,dis;
void dfs(ll node)
{
    vis[node]=1;
    for(ll i=0;i<adj[node].size();i++){
        ll child=adj[node][i];
        if(vis[child]==1) continue;

dis[child]+=dis[node]+weight[{node,child}];
        dfs(child);
    }
}
void reset()
{
    for(ll i=0;i<30001;i++){
        adj[i].clear();
    }
    dis.clear(),weight.clear(),vis.clear();
}
ll main()
{
    ll t; cin>>t;
    for(ll p=1;p<=t;p++)
    {
        ll n,u,v,w; cin>>n;
        for(ll i=0;i<n-1;i++){
            cin>>u>>v>>w;
            adj[u].push_back(v);
            adj[v].push_back(u);

            weight[{u,v}]=w;
            weight[{v,u}]=w;
        }
        dfs(0);
        ll max_dis=0,farthestVertex;
        map<ll,ll>::iterator i;
        for(i=did.begin();i!=did.end();i++){
            if(i->second>max_dis){
                max_dis=i->second;
                farthestVertex=i->first;
            }
        }

        vis.clear();
        dis.clear();

        dfs(farthestVertex);
        max_dis=0;
        for(i=did.begin();i!=did.end();i++){
            if(i->second>max_dis){
                max_dis=i->second;
            }
```

```
        }
        court<<"Case "<<p<<":
"<<max_dis<<"\n";
        reset();
    }
}
```

6.14 0-1 BFS

```
vector<ll> d(n, INF);d[s] = 0;deque<ll> q;
q.push_front(s);
while (!q.empty()) {
    ll v = q.front(); q.pop_front();
    for (auto edge : adj[v]) {
        ll u = edge.first;
        ll w = edge.second;
        if (d[v] + w < d[u]) {
            d[u] = d[v] + w;
   if (w == 1)  q.push_back(u);
   else q.push_front(u);
   }}}
```

6.15 2D Convex Hull

```
template <typename P_, typename Q_>
class CH2D{
public:
    vector<pair<P_, Q_>> polls,hull;
    auto area(pair<P_, Q_> O, pair<P_, Q_>
P, pair<P_, Q_> Q) -> P_ {

 return (P.first - O.first) * (Q.second -
O.second) - (P.second - O.second) * (Q.first
- O.first);   } // AxB->area, AxBxC=Ax(B.C)

void add_poll(P_ x, Q_ y) {
polls.push_back({x, y});  }

    CH2D(vector<pair<P_, Q_>> polls) :
polls(polls) {}

void monotone_chain() {

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

  polls.erase(unique(polls.begin(),
polls.end()), polls.end());

  P_ n = polls.size();

  if (n < 3) { hull = polls;return; }

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

   while (hull.size() > 1 &&
area(hull[hull.size() - 2], hull.back(),
polls[i]) < 0) hull.pop_back();

      hull.push_back(polls[i]);}

      auto lower_hull_length = hull.size();

  for (auto i = n - 2; i >= 0; i--){

  while (hull.size() > lower_hull_length &&

  area(hull[hull.size() - 2], hull.back(),
polls[i]) < 0) hull.pop_back();

  hull.push_back(polls[i]); }

   hull.pop_back(); }

 vector<pair<P_, Q_>> get_hull(){return
hull; }};
```

6.14 Depth and width of tree

```
ll l[100]= {0}, ll r[100]= {0};
stack<ll> mystack;
ll n = 0, w = 0, d = 0;
ll depth(ll n){
    if (l[n]==0 && r[n]==0)
        return 1;
    in depthl=depth(l[n]);
    ll depthr=depth(r[n]);
    ll dep=depthl>depthr ? depthl:depthr;
    return dep+1;
}
void width(ll n){
    if (n<=d){
        ll t=0,x;
        stack<ll> tmpstack;
        while (!mystack.empty()){
            x=mystack.top();
            mystack.pop();
            if (x!=0){
                t++;
                tmpstack.push(l[x]);
                tmpstack.push(r[x]);
            }
        }
        w=w>t?w:t;
        mystack=tmpstack;
        width(n+1);
    }
}
ll main(void){
    cin >> n;
    for (ll i=1; i<=n; i++)
        cin >> l[i] >> r[i];
    d=depth(1);
    mystack.push(1);
    width(1);
    cout << w << " " << d << endl;
    return 0;
}
```

6.15 Max Pos and Next Greater

```
const ll MXX = 1e5 + 5;
ll mxtree[4 * MXX], arr[MXX];
void maxtree(ll idx, ll left, ll right)
{
  if (left == right) mxtree[idx] = left;
  else {
    ll mid = (left + right) / 2;
    maxtree(idx * 2, left, mid);
    maxtree(idx * 2 + 1, mid + 1, right);
    ll left = mxtree[idx * 2];
    ll right = mxtree[idx * 2 + 1];
    if (arr[left] < arr[right]) mxtree[idx]
= right;
    else mxtree[idx] = left;
  }
}
ll mxPos(ll idx, ll tleft, ll tright, ll
qleft, ll qright)
{
  if (qleft > qright) return -1;
  if (qleft == tleft and qright == tright)
return mxtree[idx];
  ll tmid = (tleft + tright) / 2;
  ll left = mxPos(idx * 2, tleft, tmid,
qleft, min(qright, tmid));
  ll right = mxPos(idx * 2 + 1, tmid + 1,
tright, max(qleft, tmid + 1), qright);
  ll ans;
  if (left == -1) ans = right;
  else if (right == -1) ans = left;
  else if (arr[left] < arr[right]) ans =
right;
```

```
  else ans = left;
  return ans;
}
ll main() {
  ll t = 1, n, q, a, b;
  cin >> t;
  while (t--){
    cin >> n >> q;
    for (ll i = 0; i < n; i++)cin >> arr[i];
    stack<ll> stk;
    ll nge[n];
    stk.push(0);
    for (ll i = 1; i < n; i++){
      while (stk.size() and arr[stk.top()] <
arr[i]){
        nge[stk.top()] = i;
        stk.pop();
      }
      stk.push(i);
    }
    while (stk.size()){
      nge[stk.top()] = n;
      stk.pop();
    }
    ll ans[n];
    ans[n - 1] = 0;
    for (ll i = n - 2; i >= 0; i--){
      ll tmp = nge[i];
      if (tmp == n) ans[i] = 0;
      else ans[i] = ans[tmp] + 1;
    }
    maxtree(1, 0, n - 1);
    for (ll i = 0; i < q; i++){
      cin >> a >> b;
      if (a > b) swap(a, b);
      cout << ans[mxPos(1, 0, n - 1, a - 1,
b - 1)] << "\n";
    } } }
```

7 Random Staff

7.4 Knight Moves

```
ll X[8]={2,1,-1,-2,-2,-1,1,2};
ll Y[8]={1,2,2,1,-1,-2,-2,-1};
```

7.6 Matrix Exponentiation

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

LL arr[60][60],res[60][60],tmp[60][60],m;

void matMul (LL a[][60], LL b[][60], LL mod)
{
    for(ll i=0; i<m; i++)
        for(ll j=0; j<m; j++)
        {
            tmp[i][j] = 0;
            for(ll k=0; k<m; k++)
            {
                tmp[i][j] +=
(a[i][k]*b[k][j])%mod;
                tmp[i][j] %= mod;
            }        }}

void power(LL n, LL mod)
{
    for(ll i=0; i<m; i++)
        for(ll j=0; j<m; j++)
            if(i==j) res[i][j] = 1;
            else res[i][j] = 0;

    while(n)    {
        if(n&1)        {
            matMul(res,arr,mod);
            for(ll i=0; i<m; i++)
                for(ll j=0; j<m; j++)
res[i][j] = tmp[i][j];
            n--;        }
        else        {
            matMul(arr,arr,mod);
            for(ll i=0; i<m; i++)
                for(ll j=0; j<m; j++)
arr[i][j] = tmp[i][j];
            n/=2;        }    }}
```

7.8 sqrt decomposition(MO's Algo)

```
// https://www.spoj.com/problems/DQUERY/
#include <bits/stdc++.h>
using namespace std;
const ll SIZE_1 = 1e6 + 10, SIZE_2 = 3e4 +
10;
class query{
public:
    ll l, r, indx;
};

ll block_size, cnt = 0;
ll frequency[SIZE_1], a[SIZE_2];
void add(ll indx){
    ++frequency[a[indx]];
    if (frequency[a[indx]] == 1)
        ++cnt;
}
void sub(ll indx){
    --frequency[a[indx]];
    if (frequency[a[indx]] == 0)
        --cnt;
}
bool comp(query a, query b){
    if (a.l / block_size == b.l /
block_size)
        return a.r < b.r;
    return a.l / block_size < b.l /
block_size;
}
signed main(){
    ll n; cin >> n;
    for(ll i = 0; i < n; ++i) cin>>a[i];

    ll q; cin >> q;
    ll ans[q] = {};
    query Qur[q];
    for (ll i = 0; i < q; ++i){
        ll l, r; cin>>l>>r;

        Qur[i].l = l - 1;
        Qur[i].r = r - 1;
        Qur[i].indx = i;
    }
    block_size = sqrt(n); // sqrt(q) dileo
hobe, but n is more accurate
    sort(Qur, Qur + q, comp);

    ll ML = 0, MR = -1;
    for(ll i = 0; i < q; ++i) {
        ll L = Qur[i].l;
        ll R = Qur[i].r;

        // fixing right poller
        while (MR < R) add(++MR);
        while (MR > R) sub(MR--);
        // fixing left poller
        while (ML < L) sub(ML++);
```

```
        while (ML > L) add(--ML);

        ans[Qur[i].indx] = cnt;
    }
    for (ll i = 0; i < q; ++i)
        cout << and[i] << '\n';
}//sqrt(n)
```

7.9 Meet in the middle

```
#include <bits/stdc++.h>
using namespace std;
ll les_equal(vector<ll> &s, ll key){
    ll size = s.size();
    ll lo = 0, hi = size - 1, ans = 0;

    while (hi >= lo){
        ll mid = lo + (hi - lo) / 2;
        if (s[mid] <= key){
            ans = max(ans, mid);
            lo = mid + 1;
        }
        else hi = mid - 1;
    }
    return ans;
}
signed main(){
    FIO ll n, n1, n2, t;
    cin >> n >> t;

    n1 = (n + 1) / 2;
    n2 = n / 2;

    ll a1[n1]; for(ll &i: a1) cin>>i;
    ll a2[n2]; for(ll &i: a2) cin>>i;

    vector<ll> set1, set2;
    for(ll mask=0; mask < (1<<n1); ++mask){
        ll temp_sum = 0;
        for (ll i = 0; i < n1; ++i){
            ll f = 1 << i;
            if (f & mask)
                temp_sum += a1[i];
        }
        set1.push_back(temp_sum);
    }
    for(ll mask=0; mask < (1<<n2); ++mask){
        ll temp_sum = 0;
        for (ll i = 0; i < n2; ++i){
            ll f = 1 << i;
            if (f & mask)
                temp_sum += a2[i];
        }
        set2.push_back(temp_sum);
    }
    sort(set2.begin(), set2.end());

    // for(auto itr: set2) cout<<itr<<' ';
    // cout<<'\n';
    // for(auto itr: set1) cout<<itr<<' ';
    // cout<<'\n';

    ll and = 0;
    for (auto it : set1){
        ll left = t - it;
        if (left < 0) continue;

        ll indx = les_equal(set2, left);
        ll temp_sum_set2 = (indx != -1 ? (it
+ set2[indx]) : 0);
        if (temp_sum_set2 <= t)
            ans = max(ans, temp_sum_set2);
    }
    cout<<ans;
}//TC: O(2^(LK+1))
```

7.10 PIE(inclusion - exclusion)

```
//count the numbers between 1 and n
(inclusive) that are not divisible by any of
the llegers in the given array a
//|A₁ ∪ A₂ ∪ ... ∪ A□| = Σ|Aᵢ| - Σ|Aᵢ ∩ A□|
+ Σ|Aᵢ ∩ A□ ∩ A□| - ... + (-1)^(n-1)|A₁ ∩ A₂
∩ ... ∩ A□|
#include <bits/stdc++.h>
using namespace std;

inline ll LCM(ll a, ll b){
    return a * b / __gcd(a, b);
}

ll PIE(ll div[], ll n, ll num){
    ll sum = 0;
    for(ll msk=1; msk < (1<<n); ++msk){
        ll bit_cnt = 0;
        ll cur_lcm = 1;
        for (ll i = 0; i < n; ++i){
            if (msk & (1 << i)){
                ++bit_cnt;
                cur_lcm = LCM(cur_lcm,
div[i]);
            }
        }

        ll cur = num / cur_lcm;
        if (bit_cnt & 1) sum += cur;
        else sum -= cur;
    }
    return num - sum;
}

signed main(){
    ll n, m;
    while (cin >> n >> m){
        ll a[m];
        for(ll &i : a)cin >> i;
        cout << PIE(a, m, n) << '\n'; }}
```

7.12 Binary Search

```
ll lo=0, hi=mx;  ///mx=max possible ans
while(lo<hi){
    ll mid=(lo+hi+1)>>1;
    if(condition)  ///valid condition->and
can be greater than or equal mid
        lo=mid;
    else
        hi=mid-1;  ///ans is less than mid
}
///or
while(lo<hi){
    ll mid=(lo+hi)>>1;
    if(condition)  ///valid condition->and
can be less than or equal mid
        hi=mid;
    else
        lo=mid+1;  ///ans is greater than
mid
}

ll lo=0, hi=mx, esp=maxError;
while((hi-lo)>esp){
    ll mid=(lo+hi+esp)/2.0;
    if(condition) lo=mid;
    else      hi=mid-esp;
}
 while((hi-lo)>esp){
    ll mid=(lo+hi)/2.0;
    if(condition)  hi=mid;
    else       lo=mid+esp;
}
```

7.12.1 Ternary Search

```
double ternary_search(double l, double r) {
```

```
    double eps = 1e-9;//error limit
    while (r - l > eps) {
        double m1 = l + (r - l) / 3,m2 = r -
(r - l) / 3;
        double f1 = f(m1),f2 =
f(m2);//evaluates the function at m1, m2
        if (f1 < f2)l = m1;
        elser = m2;
    }
    return f(l);//return the maximum of f(x)
in [l, r]
}
```

7.13 Generating Permutations

```
ll length, perm_left_to_prll;
bool placed[10000];
vector<char>perm;

void generate_permutations(ll curr_length){
    if(perm_left_to_prll==0) return;
    if(curr_length==length){
        for(ll i=0;i<length;i++){
            cout<<perm[i];        }
        cout<<"\n";
        perm_left_to_prll--;
        return;    }
    for(char ch='A';ch<('A'+length);ch++){
        if(!placed[ch-'A']){
            perm.push_back(ch);
            placed[ch-'A']=true;

generate_permutations(curr_length+1);
            perm.pop_back();
            placed[ch-'A']=false;
        }    }}

ll main(){
    ioi;
    ll t; cin>>t;
    for(ll tc=1;tc<=t;tc++){
        cin>>length>>perm_left_to_prll;
        court<<"Case "<<tc<<":\n";
        generate_permutations(0);
    }
}
```

7.15 HLD

```
const ll N = 1e5 + 9, LG = 18, inf = 1e9 +
9;
struct ST {
#define lc (n << 1)
#define rc ((n << 1) | 1)
  ll t[4 * N], lazy[4 * N];
  ST() {
    fill(t, t + 4 * N, -inf);
    fill(lazy, lazy + 4 * N, 0);
  }
  inline void push(ll n, ll b, ll e) {
    if(lazy[n] == 0) return;
    t[n] = t[n] + lazy[n];
    if(b != e) {
      lazy[lc] = lazy[lc] + lazy[n];
      lazy[rc] = lazy[rc] + lazy[n];
    }
    lazy[n] = 0;
  }
  inline ll combine(ll a, ll b) {
    return max(a, b); //merge left and right
queries
  }
  inline void pull(ll n) {
    t[n] = max(t[lc], t[rc]); //merge lower
nodes of the tree to get the parent node
  }
  void build(ll n, ll b, ll e) {
    if(b == e) {  t[n] = 0;  return;    }
    ll mid = (b + e) >> 1;
    build(lc, b, mid);  build(rc, mid + 1,
e); pull(n);  }
  void upd(ll n, ll b, ll e, ll i, ll j, ll
v) {
    push(n, b, e);
    if(j < b || e < i) return;
    if(i <= b && e <= j) {
      lazy[n] += v;
      push(n, b, e);
      return;
    }
    ll mid = (b + e) >> 1;
    upd(lc, b, mid, i, j, v);
    upd(rc, mid + 1, e, i, j, v);
    pull(n);
  }
  ll query(ll n, ll b, ll e, ll i, ll j) {
    push(n, b, e);
    if(i > e || b > j) return -inf;
    if(i <= b && e <= j) return t[n];
    ll mid = (b + e) >> 1;
    return combine(query(lc, b, mid, i, j),
query(rc, mid + 1, e, i, j));
  }
} t;

vector<ll> g[N];
ll par[N][LG + 1], dep[N], sz[N];
void dfs(ll u, ll p = 0) {
  par[u][0] = p;
  dep[u] = dep[p] + 1;
  sz[u] = 1;
  for (ll i = 1; i <= LG; i++) par[u][i] =
par[par[u][i - 1]][i - 1];
  if (p) g[u].erase(find(g[u].begin(),
g[u].end(), p));
  for (auto &v : g[u]) if (v != p) {
      dfs(v, u);
      sz[u] += sz[v];
      if(sz[v] > sz[g[u][0]]) swap(v,
g[u][0]);
    }
}
ll lca(ll u, ll v) {
  if (dep[u] < dep[v]) swap(u, v);
  for (ll k = LG; k >= 0; k--) if
(dep[par[u][k]] >= dep[v]) u = par[u][k];
  if (u == v) return u;
  for (ll k = LG; k >= 0; k--) if (par[u][k]
!= par[v][k]) u = par[u][k], v = par[v][k];
  return par[u][0];
}
ll kth(ll u, ll k) {
  assert(k >= 0);
  for (ll i = 0; i <= LG; i++) if (k & (1 <<
i)) u = par[u][i];
  return u;
}
ll T, head[N], st[N], en[N];
void dfs_hld(ll u) {
  st[u] = ++T;
  for (auto v : g[u]) {
    head[v] = (v == g[u][0] ? head[u] : v);
    dfs_hld(v);
  }
  en[u] = T;
}
ll n;
ll query_up(ll u, ll v) {
  ll ans = -inf;
  while(head[u] != head[v]) {
```

```
    ans = max(ans, t.query(1, 1, n,
st[head[u]], st[u]));
    u = par[head[u]][0];
  }
  ans = max(ans, t.query(1, 1, n, st[v],
st[u]));
  return ans;
}
ll query(ll u, ll v) {
  ll l = lca(u, v);
  ll ans = query_up(u, l);
  if (v != l) ans = max(ans, query_up(v,
kth(v, dep[v] - dep[l] - 1)));
  return ans;
}
//  first build the tree by calling dfs(1)
//  then call dfs_hld(1)
//  then call t.build(1, 1, n)
//  then call query(u, v) to get the answer
//  then call t.upd(1, 1, n, st[u], en[u],
v) to update the subtree of u with value v
//  then call t.query(1, 1, n, st[u], en[u])
to get the value of the subtree of u
//  then call t.query(1, 1, n, st[1], en[1])
to get the value of the whole tree
```

7.16 GRUNDY

```
ll calculateGrundy(ll n, vector<ll> &grundy,
const vector<ll> &moves)
{    if (grundy[n] != -1)  return grundy[n];
 unordered_set<ll> s; for (ll move : moves)
{    if (n >= move)  {
s.insert(calculateGrundy(n - move, grundy,
moves));  } }  ll g = 0; while (s.count(g))
        g++;  return grundy[n] = g;}
vector<ll> computeGrundy(ll maxN, const
vector<ll> &moves){    vector<ll>
grundy(maxN + 1, -1);
    grundy[0] = 0; for (ll i = 1; i <= maxN;
++i) { calculateGrundy(i, grundy, moves);
}  return grundy;}
```

7.17 Gaussian Elimination

```
class GaussianElimination{
public:
GaussianElimination(vector<vector<double>>
matrix, vector<double> results)
         : matrix(matrix), results(results),
n(matrix.size()) {}
void solve()  { fElim(); bSub(); }
vector<vector<double>> matrix;
vector<double> results, solution; ll n;
void fElim()    {
for (ll i = 0; i < n; ++i) {
  ll maxRow = i;
  for (ll k = i + 1; k < n; ++k)
if(abs(matrix[k][i])>abs(matrix[maxRow][i]))
maxRow = k;
swap(matrix[i], matrix[maxRow]);
swap(results[i], results[maxRow]);
for (ll k = i + 1; k < n; ++k) {
double factor = matrix[k][i] / matrix[i][i];
 for (ll j = i; j < n; ++j)
   matrix[k][j] -= factor * matrix[i][j];
   results[k] -= factor * results[i]; }   }
}
void bSub()  {
solution.resize(n);
for (ll i = n - 1; i >= 0; --i)        {
 solution[i] = results[i];
 for (ll j = i + 1; j < n; ++j)
  solution[i] -= matrix[i][j] * solution[j];
            solution[i] /= matrix[i][i];   }
}};
```

7.18 Calculating nth element of recurrence relation in O(logN)

- (f1 f2)(a b, c d) = (f3,f4)
- find a,b,c,d first then (f1 f2)(a b, c d)^n-1=(fn fn+1)

7.19 Manacher Algorithm

```
struct Manacher
{
  vector<ll> p[2];
  string s;
  // p[1][i] = (max odd length palindrome
centered at i) / 2 [floor division]
  // p[0][i] = same for even, it considers
the right center
  // e.g. for s = "abbabba", p[1][3] = 3,
p[0][2] = 2
  Manacher(string s)
  {
    this->s = s;
    ll n = s.size();
    p[0].resize(n + 1);
    p[1].resize(n);
    for (ll z = 0; z < 2; z++)
    {
      for (ll i = 0, l = 0, r = 0; i < n;
i++)      {
        ll t = r - i + !z;
        if (i < r)
          p[z][i] = min(t, p[z][l + t]);
        ll L = i - p[z][i], R = i + p[z][i]
- !z;
        while (L >= 1 && R + 1 < n && s[L -
1] == s[R + 1])
          p[z][i]++, L--, R++;
        if (R > r)
          l = L, r = R;
      }    }  }
  bool is_palindrome(ll l, ll r)
  {
    ll mid = (l + r + 1) / 2, len = r - l +
1;
    return 2 * p[len % 2][mid] + len % 2 >=
len;  }
  string get_palin(ll i, bool odd = true) {
    ll len = p[odd][i];
    return s.substr(i - len, 2 * len + 1 -
!odd);
  }};
```

7.20 Two Line Intersection

```
ll cross(ll x1,ll y1,ll x2,ll y2,ll x3,ll
y3){return (x2-x1)*(y3-y1)-(y2-y1)*(x3-x1);}

bool intersect(ll x1,ll y1,ll x2,ll y2,ll
x3,ll y3,ll x4,ll y4){ ll
c1=cross(x1,y1,x2,y2,x3,y3),c2=cross(x1,y1,x
2,y2,x4,y4),c3=cross(x3,y3,x4,y4,x1,y1),c4=c
ross(x3,y3,x4,y4,x2,y2); if((!c1 &&
min(x1,x2)<=x3 && x3<=max(x1,x2) &&
min(y1,y2)<=y3 && y3<=max(y1,y2))|(!c2 &&
min(x1,x2)<=x4 && x4<=max(x1,x2) &&
min(y1,y2)<=y4 && y4<=max(y1,y2))|(!c3 &&
min(x3,x4)<=x1 && x1<=max(x3,x4) &&
min(y3,y4)<=y1 && y1<=max(y3,y4))|(!c4 &&
min(x3,x4)<=x2 && x2<=max(x3,x4) &&
```

```
min(y3,y4)<=y2 && y2<=max(y3,y4))) return
true; return (c1>0)!=(c2>0)&&(c3>0)!=(c4>0);
}
```

7.21 Count Simple Cycle

```
void findNumberOfSimpleCycles(
    int N, vector<vector<int>> adj)
{
    int ans = 0;
    int dp[(1 << N)][N];
    memset(dp, 0, sizeof dp);
    for (int mask = 0;
         mask < (1 << N); mask++)
    {
        int nodeSet =
__builtin_popcountll(mask);
        int firstSetBit =
__builtin_ffsl(mask);
        if (nodeSet == 1)
            dp[mask][firstSetBit] = 1;
        else
        {
            for (int j = firstSetBit + 1;
                 j < N; j++)
            {
                if ((mask & (1 << j)))
                {
                    int newNodeSet = mask ^
(1 << j);
                    for (int k = 0; k < N;
k++)
                    {
                        if ((newNodeSet & (1
<< k)) && adj[k][j])
                        {
                            dp[mask][j] +=
dp[newNodeSet][k];
                            if
(adj[j][firstSetBit] && nodeSet > 2)
                                ans +=
dp[mask][j]; }   }   }    }  }    }
    cout << ans << endl;}
```

7.19 Linear Recurrance

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
    int n = tr.size();
    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1);
        for (int i = 0; i <= n; ++i)
            for (int j = 0; j <= n; ++j)
                res[i + j] = (res[i + j] +
a[i] * b[j]) % mod;
        for (int i = 2 * n; i > n; --i)
            for (int j = 0; j < n; ++j)
                res[i - 1 - j] = (res[i - 1
- j] + res[i] * tr[j]) % mod;
        res.resize(n + 1);
        return res;
    };
    Poly pol(n + 1), e(pol);
    pol[0] = e[1] = 1;
    for (++k; k; k /= 2) {
        if (k % 2)
            pol = combine(pol, e);
        e = combine(e, e);
    }
    ll res = 0;
    for (int i = 0; i < n; ++i)
        res = (res + pol[i + 1] * S[i]) %
mod;
    return res;
}
```

7.20 2D prefix sum

```
pref[i][j] = a[i][j] + pref[i - 1][j] +
pref[i][j - 1] - pref[i - 1][j - 1];
Sum of region = pref[row2 + 1][col2 + 1] -
pref[row2 + 1][col1] - pref[row1][col2 + 1]
+ pref[row1][col1];
```

7.21 Bezout's Identity

1. gcd(a, b) = g → there exist x, y such that ax + by = g
2. All integers of the form ax+by are exactly the multiples of g.
3. Adding or subtracting multiples doesn't change gcd
4. a≡b(mod g)⟺g∣(a−b)
5. If gcd(a, b) = 1 ⇒ any integer can be formed
6. If gcd(a, b) = g ⇒ any multiple of g can be formed
7. gcd(a,b)=gcd(a−b,b)=gcd(a,b−a)
8. If gcd(a, b) = g ⇒ gcd(a/g, b/g) = 1
9. gcd(ka,kb)=kgcd(a,b)
10. If gcd(a,m) = 1,then Bézout gives ax+my=1 ⇒ ax≡1(mod m), so x is the modular inverse of a mod m. **(Important when mod is needed and m is not prime)**

7.22 CRT

```
class CRT
{
    typedef long long vlong;
    typedef pair<vlong, vlong> pll;
    vector<pll> equations;
public:
    void clear(){
        equations.clear();
    }
    vlong extended_euclid(vlong a, vlong b,
vlong &x, vlong &y)
    {
        if (b == 0)
        {
            x = 1;
            y = 0;
            return a;
        }
        vlong x1, y1;
        vlong d = extended_euclid(b, a % b,
x1, y1);
```

```
        x = y1;
        y = x1 - y1 * (a / b);
        return d;
    }
    vlong inverse(vlong a, vlong m)
    {
        vlong x, y;
        vlong g = extended_euclid(a, m, x,
y);
        if (g != 1)
            return -1;
        return (x % m + m) % m;
    }

    /** Add equation of the form x = r (mod
m)*/
    void addEquation(vlong r, vlong m)
    {
        equations.push_back({r, m});
    }
    pll solve()
    {
        if (equations.size() == 0)
            return {-1, -1};

        vlong a1 = equations[0].first;
        vlong m1 = equations[0].second;
        a1 %= m1;
        for (int i = 1; i <
equations.size(); i++)
        {
            vlong a2 = equations[i].first;
            vlong m2 = equations[i].second;

            vlong g = __gcd(m1, m2);
            if (a1 % g != a2 % g)
                return {-1, -1};
            vlong p, q;
            extended_euclid(m1 / g, m2 / g,
p, q);

            vlong mod = m1 / g * m2;
            vlong x = ((__int128)a1 * (m2 /
g) % mod * q % mod + (__int128)a2 * (m1 / g)
% mod * p % mod) % mod;
            a1 = x;
            if (a1 < 0)
                a1 += mod;
            m1 = mod;
        }
        return {a1, m1};
    }
};
```

7.23 Intersect two arithmetic progression

```
using T = __int128;
// ax + by = __gcd(a, b)
// returns __gcd(a, b)
T extended_euclid(T a, T b, T &x, T &y)
{
    T xx = y = 0;
    T yy = x = 1;
    while (b)
    {
        T q = a / b;
        T t = b;
        b = a % b;
        a = t;
        t = xx;
        xx = x - q * xx;
        x = t;
        t = yy;
        yy = y - q * yy;
        y = t;
    }
    return a;
}
pair<T, T> CRT(T a1, T m1, T a2, T m2)
{
    T p, q;
    T g = extended_euclid(m1, m2, p, q);
    if (a1 % g != a2 % g)
        return make_pair(0, -1);
    T m = m1 / g * m2;
    p = (p % m + m) % m;
    q = (q % m + m) % m;
    return make_pair((p * a2 % m * (m1 / g)
% m + q * a1 % m * (m2 / g) % m) % m, m);
}
// intersecting AP of two APs: (a1 + d1x)
and (a2 + d2x)
pair<ll, ll> intersect(ll a1, ll d1, ll a2,
ll d2)
{
    auto x = CRT(a1 % d1, d1, a2 % d2, d2);
    ll a = x.first, d = x.second;
    if (d == -1)
        return {0, 0}; // empty
    ll st = max(a1, a2);
    a = a < st ? a + ((st - a + d - 1) / d)
: a; // while (a < st) a += d;
    return {a, d};
}
```

7.24 Find nth value in a recurrence relation in O(logn)

[1, 1; 1, 0]^(n-1) = [F(n), F(n-1); F(n-1), F(n-2)]

```
// Function to multiply two 2x2 matrices
void multiply(vector<vector<int>>& mat1,

vector<vector<int>>& mat2) {
    // Perform matrix multiplication
    int x = mat1[0][0] * mat2[0][0] +
mat1[0][1] * mat2[1][0];
    int y = mat1[0][0] * mat2[0][1] +
mat1[0][1] * mat2[1][1];
    int z = mat1[1][0] * mat2[0][0] +
mat1[1][1] * mat2[1][0];
    int w = mat1[1][0] * mat2[0][1] +
mat1[1][1] * mat2[1][1];

    // Update matrix mat1 with the result
    mat1[0][0] = x;
```

```
    mat1[0][1] = y;
    mat1[1][0] = z;
    mat1[1][1] = w;
}

// Function to perform matrix exponentiation
void matrixPower(vector<vector<int>>& mat1,
int n) {
    // Base case for recursion
    if (n == 0 || n == 1) return;

    // Initialize a helper matrix
    vector<vector<int>> mat2 = {{1, 1}, {1,
0}};

    // Recursively calculate mat1^(n/2)
    matrixPower(mat1, n / 2);

    // Square the matrix mat1
    multiply(mat1, mat1);

    // If n is odd, multiply by the helper
matrix mat2
    if (n % 2 != 0) {
        multiply(mat1, mat2);
    }
}

// Function to calculate the nth Fibonacci
number
// using matrix exponentiation
int nthFibonacci(int n) {
    if (n <= 1) return n;

    // Initialize the transformation matrix
    vector<vector<int>> mat1 = {{1, 1}, {1,
0}};

    // Raise the matrix mat1 to the power of
(n - 1)
    matrixPower(mat1, n - 1);

    // The result is in the top-left cell of
the matrix
    return mat1[0][0];
}
```

7.21 All solution of ax+by = c

```
// a*x+b*y=c. returns valid x and y if
possible.
// all solutions are of the form (x0 + k * b
/ g, y0 - k * b / g)
bool find_any_solution (ll a, ll b, ll c, ll
&x0, ll &y0, ll &g) {
  if (a == 0 and b == 0) {
    if (c) return false;
    x0 = y0 = g = 0;
    return true;
  }
  g = extended_euclid (abs(a), abs(b), x0,
y0);
  if (c % g != 0) return false;
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 *= -1;
  if (b < 0) y0 *= -1;
  return true;
}
void shift_solution(ll &x, ll &y, ll a, ll
b, ll cnt) {
  x += cnt * b;
  y -= cnt * a;
}
// returns the number of solutions where x
is in the range[minx, maxx] and y is in the
range[miny, maxy]
ll find_all_solutions(ll a, ll b, ll c, ll
minx, ll maxx, ll miny,ll maxy) {
  ll x, y, g;
  if (find_any_solution(a, b, c, x, y, g) ==
0) return 0;
  if (a == 0 and b == 0) {
    assert(c == 0);
    return 1LL * (maxx - minx + 1) * (maxy -
miny + 1);
  }
  if (a == 0) {
    return (maxx - minx + 1) * (miny <= c /
b and c / b <= maxy);
  }
  if (b == 0) {
    return (maxy - miny + 1) * (minx <= c /
a and c / a <= maxx);
  }
  a /= g, b /= g;
  ll sign_a = a > 0 ? +1 : -1;
  ll sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x) /
b);
  if (x < minx) shift_solution(x, y, a, b,
sign_b);
  if (x > maxx) return 0;
  ll lx1 = x;
  shift_solution(x, y, a, b, (maxx - x) /
b);
  if (x > maxx) shift_solution (x, y, a, b,
-sign_b);
  ll rx1 = x;
  shift_solution(x, y, a, b, -(miny - y) /
a);
  if (y < miny) shift_solution (x, y, a, b,
-sign_a);
  if (y > maxy) return 0;
  ll lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y) /
a);
  if (y > maxy) shift_solution(x, y, a, b,
sign_a);
  ll rx2 = x;
  if (lx2 > rx2) swap (lx2, rx2);
  ll lx = max(lx1, lx2);
  ll rx = min(rx1, rx2);
  if (lx > rx) return 0;
```

```
  return (rx - lx) / abs(b) + 1;
}

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int t, cs = 0; cin >> t;
  while (t--) {
    ll a, b, c, x1, x2, y1, y2; cin >> a >>
b >> c >> x1 >> x2 >> y1 >> y2;
    cout << "Case " << ++cs << ": " <<
find_all_solutions(a, b, -c, x1, x2, y1, y2)
<< '\n';
  }
  return 0;
}
```

7.22 mint and all soln of linear eq

```
template <const int32_t MOD>
struct modint {
  int32_t value;
  modint() = default;
  modint(int32_t value_) : value(value_) {}
  inline modint<MOD> operator + (modint<MOD>
other) const { int32_t c = this->value +
other.value; return modint<MOD>(c >= MOD ? c
- MOD : c); }
  inline modint<MOD> operator - (modint<MOD>
other) const { int32_t c = this->value -
other.value; return modint<MOD>(c <    0 ? c
+ MOD : c); }
  inline modint<MOD> operator * (modint<MOD>
other) const { int32_t c =
(int64_t)this->value * other.value % MOD;
return modint<MOD>(c < 0 ? c + MOD : c); }
  inline modint<MOD> & operator +=
(modint<MOD> other) { this->value +=
other.value; if (this->value >= MOD)
this->value -= MOD; return *this; }
  inline modint<MOD> & operator -=
(modint<MOD> other) { this->value -=
other.value; if (this->value <    0)
this->value += MOD; return *this; }
  inline modint<MOD> & operator *=
(modint<MOD> other) { this->value =
(int64_t)this->value * other.value % MOD; if
(this->value < 0) this->value += MOD; return
*this; }
  inline modint<MOD> operator - () const {
return modint<MOD>(this->value ? MOD -
this->value : 0); }
  modint<MOD> pow(uint64_t k) const {
modint<MOD> x = *this, y = 1; for (; k; k
>>= 1) { if (k & 1) y *= x; x *= x; } return
y; }
  modint<MOD> inv() const { return pow(MOD -
2); }  // MOD must be a prime
  inline modint<MOD> operator /
(modint<MOD> other) const { return *this *
other.inv(); }
  inline modint<MOD> operator /=
(modint<MOD> other)       { return *this *=
other.inv(); }
  inline bool operator == (modint<MOD>
other) const { return value == other.value;
}
  inline bool operator != (modint<MOD>
other) const { return value != other.value;
}
  inline bool operator < (modint<MOD> other)
const { return value < other.value; }
  inline bool operator > (modint<MOD> other)
const { return value > other.value; }
};
template <int32_t MOD> modint<MOD> operator
* (int64_t value, modint<MOD> n) { return
modint<MOD>(value) * n; }
template <int32_t MOD> modint<MOD> operator
* (int32_t value, modint<MOD> n) { return
modint<MOD>(value % MOD) * n; }
template <int32_t MOD> istream & operator >>
(istream & in, modint<MOD> &n) { return in
>> n.value; }
template <int32_t MOD> ostream & operator <<
(ostream & out, modint<MOD> n) { return out
<< n.value; }

using mint = modint<mod>;

struct Combi{
  int n; vector<mint> facts, finvs, invs;
  Combi(int _n): n(_n), facts(_n),
finvs(_n), invs(_n){
    facts[0] = finvs[0] = 1;
    invs[1] = 1;
      for (int i = 2; i < n; i++) invs[i] =
invs[mod % i] * (-mod / i);
    for(int i = 1; i < n; i++){
      facts[i] = facts[i - 1] * i;
      finvs[i] = finvs[i - 1] * invs[i];
    }
  }
  inline mint fact(int n) { return facts[n];
}
  inline mint finv(int n) { return finvs[n];
}
  inline mint inv(int n) { return invs[n]; }
  inline mint ncr(int n, int k) { return n <
k ? 0 : facts[n] * finvs[k] * finvs[n-k]; }
};
Combi C(N);

// returns the number of solutions to the
equation
// x_1 + x_2 + ... + x_n = s and 0 <= l <=
x_i <= r
mint yo(int n, int s, int l, int r) {
  if (s < l * n) return 0;
  s -= l * n;
  r -= l;
  mint ans = 0;
```

```
  for (int k = 0; k <= n; k++) {
    mint cur = C.ncr(s - k - k * r + n - 1 +
1, n - 1 + 1) * C.ncr(n, k);
    if (k & 1) ans -= cur;
    else ans += cur;
  }
  return ans;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout << yo(3, 3, 0, 1) << '\n';
  return 0;
}
```

7.33 subset sum in sqrt(n)

```
    // Sum of elements <= N implies that
every element is <= N
    vector<int> freq(N + 1, 0);
    for (int i = 0; i < N; i++) {
        int x;
        cin >> x;
        freq[x]++;
    }

    vector<pair<int, int>> compressed;
    for (int i = 1; i <= N; i++) {
        if (freq[i] > 0)
compressed.emplace_back(i, freq[i]);
    }

    vector<int> dp(N + 1, 0);
    dp[0] = 1;

    for (const auto &[w, k] : compressed) {
        vector<int> ndp = dp;

        for (int p = 0; p < w; p++) {
            int sum = 0;

            for (int multiple = p, count =
0; multiple <= N; multiple += w, count++) {
                if (count > k) {
                    sum -= dp[multiple - w *
count];
                    count--;
                }

                if (sum > 0) ndp[multiple] =
1;
                sum += dp[multiple];
            }
        }

        swap(dp, ndp);
    }

    cout << "Possible subset sums are:\n";
    for (int i = 0; i <= N; i++) {
        if (dp[i] > 0) cout << i << " ";
    }
}
```

7.44 berkeley

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

const int N = 3e5 + 9, mod = 1e9 + 7;
template <int32_t MOD>
struct modint {
  int32_t value;
  modint() = default;
  modint(int32_t value_) : value(value_) {}
  inline modint<MOD> operator + (modint<MOD>
other) const { int32_t c = this->value +
other.value; return modint<MOD>(c >= MOD ? c
- MOD : c); }
  inline modint<MOD> operator - (modint<MOD>
other) const { int32_t c = this->value -
other.value; return modint<MOD>(c <    0 ? c
+ MOD : c); }
  inline modint<MOD> operator * (modint<MOD>
other) const { int32_t c =
(int64_t)this->value * other.value % MOD;
return modint<MOD>(c < 0 ? c + MOD : c); }
  inline modint<MOD> & operator +=
(modint<MOD> other) { this->value +=
other.value; if (this->value >= MOD)
this->value -= MOD; return *this; }
  inline modint<MOD> & operator -=
(modint<MOD> other) { this->value -=
other.value; if (this->value <    0)
this->value += MOD; return *this; }
  inline modint<MOD> & operator *=
(modint<MOD> other) { this->value =
(int64_t)this->value * other.value % MOD; if
(this->value < 0) this->value += MOD; return
*this; }
  inline modint<MOD> operator - () const {
return modint<MOD>(this->value ? MOD -
this->value : 0); }
  modint<MOD> pow(uint64_t k) const {
    modint<MOD> x = *this, y = 1;
    for (; k; k >>= 1) {
      if (k & 1) y *= x;
      x *= x;
    }
    return y;
  }
  modint<MOD> inv() const { return pow(MOD -
2); }  // MOD must be a prime
  inline modint<MOD> operator /
(modint<MOD> other) const { return *this *
other.inv(); }
  inline modint<MOD> operator /=
(modint<MOD> other)       { return *this *=
other.inv(); }
  inline bool operator == (modint<MOD>
other) const { return value == other.value;
}
```

```
  inline bool operator != (modint<MOD>
other) const { return value != other.value;
}
  inline bool operator < (modint<MOD> other)
const { return value < other.value; }
  inline bool operator > (modint<MOD> other)
const { return value > other.value; }
};
template <int32_t MOD> modint<MOD> operator
* (int64_t value, modint<MOD> n) { return
modint<MOD>(value) * n; }
template <int32_t MOD> modint<MOD> operator
* (int32_t value, modint<MOD> n) { return
modint<MOD>(value % MOD) * n; }
template <int32_t MOD> istream & operator >>
(istream & in, modint<MOD> &n) { return in
>> n.value; }
template <int32_t MOD> ostream & operator <<
(ostream & out, modint<MOD> n) { return out
<< n.value; }

using mint = modint<mod>;

vector<mint> BerlekampMassey(vector<mint> S)
{
  int n = (int)S.size(), L = 0, m = 0;
  vector<mint> C(n), B(n), T;
  C[0] = B[0] = 1;
  mint b = 1;
  for(int i = 0; i < n; i++) {
    ++m; mint d = S[i];
    for(int j = 1; j <= L; j++) d += C[j] *
S[i - j];
    if (d == 0) continue;
    T = C; mint coef = d * b.inv();
    for(int j = m; j < n; j++) C[j] -= coef
* B[j - m];
    if (2 * L > i) continue;
    L = i + 1 - L; B = T; b = d; m = 0;
  }
  C.resize(L + 1); C.erase(C.begin());
  for(auto &x: C)  x *= -1;
  return C;
}
vector<mint> combine (int n, vector<mint>
&a, vector<mint> &b, vector<mint> &tr) {
  vector<mint> res(n * 2 + 1, 0);
  for (int i = 0; i < n + 1; i++) {
    for (int j = 0; j < n + 1; j++) res[i +
j] += a[i] * b[j];
  }
  for (int i = 2 * n; i > n; --i) {
    for (int j = 0; j < n; j++) res[i - 1 -
j] += res[i] * tr[j];
  }
  res.resize(n + 1);
  return res;
};
// transition -> for(i = 0; i < x; i++) f[n]
+= tr[i] * f[n-i-1]
// S contains initial values, k is 0 indexed
mint LinearRecurrence(vector<mint> &S,
vector<mint> &tr, long long k) {
  int n = S.size(); assert(n ==
(int)tr.size());
  if (n == 0) return 0;
  if (k < n) return S[k];
  vector<mint> pol(n + 1), e(pol);
  pol[0] = e[1] = 1;
  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(n, pol, e, tr);
    e = combine(n, e, e, tr);
  }
  mint res = 0;
  for (int i = 0; i < n; i++) res += pol[i +
1] * S[i];
  return res;
}
int prime[] = {2, 3, 5, 7, 11, 13, 17, 19},
ok[20];
int dp[2000][20];
int yo(int i, int last) {
  if (i == 0) return 1;
  int &ret = dp[i][last];
  if (ret != -1) return ret;
  ret = 0;
  for (int k = 1; k <= 9; k++) {
    if (!last || ok[last + k]) ret = (ret +
yo(i - 1, k)) % mod;
  }
  return ret;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  memset(dp, -1, sizeof dp);
  for (int i = 0; i < 8; i++) ok[prime[i]] =
1;
  vector<mint> S; S.push_back(4); mint sum =
4;
  for (int i = 2; i < 100; i++) sum += yo(i,
0), S.push_back(sum);
  auto tr = BerlekampMassey(S);
  S.resize((int)tr.size());
  int q; cin >> q;
  while (q--) {
    int n; cin >> n; --n;
    cout << LinearRecurrence(S, tr, n) <<
'\n';
  }
  return 0;
}
```

Segment Tree:(Hasnat)

```
class SEGMENT_TREE {
   public:
    vector<ll> v;
    vector<ll> seg;
    vector<ll> lazy;
    SEGMENT_TREE(ll n) {
        v.resize(n + 5, 0);
```

```
        seg.resize(4 * n + 5, 0);
        lazy.resize(4 * n + 5, 0);
    }
    void pull(ll ti) { seg[ti] = (seg[2 *
ti] & seg[2 * ti + 1]); }
    void push(ll ti, ll tl, ll tr) {
        if (lazy[ti] == 0)
            return;
        seg[ti] |= lazy[ti];
        if (tl != tr) {
            lazy[2 * ti] |= lazy[ti];
            lazy[2 * ti + 1] |= lazy[ti];
        }
        lazy[ti] = 0;
    }
    //! llially: ti = 1, low = 1, high =
n(number of elements in the array);
    void build(ll ti, ll low, ll high) {
        lazy[ti] = 0;
        if (low == high) {
            seg[ti] = v[low];
            return;
        }
        ll mid = (low + high) / 2;
        build(2 * ti, low, mid);
        build(2 * ti + 1, mid + 1, high);
        pull(ti);
    }
    //! llially: ti = 1, low = 1, high =
n(number of elements in the array), (ql
    //! & qr) = user input in 1 based
indexing;
    ll query(ll ti, ll tl, ll tr, ll ql, ll
qr) {
        push(ti, tl, tr);
        if (tl > qr || tr < ql) {
            return (1LL << 32) - 1;
        }
        if (tl >= ql and tr <= qr)
            return seg[ti];
        ll mid = (tl + tr) / 2;
        ll l = query(2 * ti, tl, mid, ql,
qr);
        ll r = query(2 * ti + 1, mid + 1,
tr, ql, qr);
        return (l & r);
    }
    //! llially: ti = 1, tl = 1, tr =
n(number of elements in the array), id =
    //! user input in 1 based indexing, val
= updated value;
    void update(ll ti, ll tl, ll tr, ll idL,
ll idR, ll val) {
        push(ti, tl, tr);
        if (idR < tl or tr < idL)
            return;
        if (idL <= tl and tr <= idR) {
            lazy[ti] |= val;
            push(ti, tl, tr);
            return;
        }
        ll mid = (tl + tr) / 2;
        update(2 * ti, tl, mid, idL, idR,
val);
        update(2 * ti + 1, mid + 1, tr, idL,
idR, val);
        pull(ti);
    }
    // use 1 based indexing for input and
queries and update;
};
```