



ACM/ICPC Template Manual

QUST

hxx

August 12, 2018

Contents

0	Include	1
1	Math	2
1.1	Prime	2
1.1.1	Eratosthenes Sieve	2
1.1.2	Eular Sieve	2
1.1.3	Prime Factorization	2
1.1.4	Miller Rabin	3
1.1.5	Segment Sieve	3
1.2	Eular phi	4
1.2.1	Eular	4
1.2.2	Sieve	4
1.3	Basic Number Theory	4
1.3.1	Extended Euclidean	4
1.3.2	$ax+by=c$	5
1.3.3	Multiplicative Inverse Modulo	5
1.4	Modulo Linear Equation	5
1.4.1	Chinese Remainder Theory	5
1.4.2	ExCRT	6
1.5	Combinatorics	6
1.5.1	Combination	6
1.5.2	Lucas	7
1.5.3	Big Combination	7
1.5.4	Polya	8
1.6	Fast Power	8
1.7	Mobius Inversion	8
1.7.1	Mobius	8
1.7.2	Number of Coprime-pair	9
1.7.3	VisibleTrees	9
1.8	Fast Transformation	10
1.8.1	FFT	10
1.8.2	NTT	11
1.8.3	FWT	11
1.9	Numerical Integration	12
1.9.1	Adaptive Simpson's Rule	12
1.9.2	Berlekamp-Massey	12
1.10	Others	14
1.11	Formula	14
2	String Processing	16
2.1	KMP	16
2.2	ExtendKMP	16
2.3	Manacher	17
2.4	Aho-Corasick Automaton	18
2.5	Suffix Array	19
2.6	Suffix Automation	20
2.7	HashString	21
3	Data Structure	22
3.1	Binary Indexed Tree	22
3.1.1	poj3468	22
3.2	Segment Tree	23
3.2.1	Single-point Update	23
3.2.2	Interval Update	23
3.3	Splay Tree	24
3.4	Functional Segment Tree	26
3.5	Sparse Table	26
3.6	Heavy-Light Decomposition	27

3.7	Link-Cut Tree	28
4	Graph Theory	30
4.1	Union-Find Set	30
4.2	Minimal Spanning Tree	30
4.2.1	Kruskal	30
4.3	Shortest Path	31
4.3.1	Dijkstra	31
4.3.2	Spfa	32
4.4	Topo Sort	33
4.5	LCA	35
4.5.1	Tarjan	35
4.5.2	LCArmq	35
4.6	Depth-First Traversal	37
4.6.1	Biconnected-Component	37
4.6.2	Strongly Connected Component	38
4.6.3	2-SAT	39
4.7	Eular Path	41
4.7.1	Fleury	41
4.8	Bipartite Graph Matching	41
4.8.1	Hungry	41
4.8.2	Hungry(Multiple)	42
4.8.3	Kuhn-Munkres	43
4.9	Network Flow	44
4.9.1	EdmondKarp	45
4.9.2	Dinic	46
4.9.3	ISAP	47
4.9.4	MinCost MaxFlow	49
5	Computational Geometry	51
5.1	Basic Function	51
5.2	Position	51
5.2.1	Point-Point	51
5.2.2	Line-Line	51
5.2.3	Segment-Segment	52
5.2.4	Line-Segment	52
5.2.5	Point-Line	52
5.2.6	Point-Segment	52
5.2.7	Point on Segment	52
5.3	Polygon	53
5.3.1	Area	53
5.3.2	Point in Convex	53
5.3.3	Point in Polygon	53
5.3.4	Judge Convex	54
5.4	Integer Points	54
5.4.1	On Segment	54
5.4.2	On Polygon Edge	54
5.4.3	Inside Polygon	54
5.5	Circle	54
5.5.1	Circumcenter	54
6	Dynamic Programming	55
6.1	Subsequence	55
6.1.1	Max Sum	55
6.1.2	Longest Increase	55
6.1.3	Longest Common Increase	56
6.2	Digit Statistics	56

7	Others	57
7.1	Matrix	57
7.1.1	Matrix FastPow	57
7.1.2	Gauss Elimination	57
7.2	Tricks	57
7.2.1	Stack-Overflow	57
7.2.2	Fast-Scanner	58
7.2.3	Strok-Sscanf	58
7.3	Mo Algorithm	58
7.4	BigNum	59
7.4.1	High-precision	59
7.5	VIM	59
7.6	BASH	60
7.6.1	a.sh	60

0 Include

```
1  // #include <bits/stdc++.h>
2  #include <algorithm>
3  #include <iostream>
4  #include <cstring>
5  #include <string>
6  #include <cstdio>
7  #include <vector>
8  #include <cstdlib>
9  #include <vector>
10 #include <stack>
11 #include <queue>
12 #include <cmath>
13 #include <set>
14 #include <map>
15 using namespace std;
16 #define rep(i,a,b) for(int i=a;i<=b;i++)
17 #define per(i,a,b) for(int i=a;i>=b;i--)
18 #define clr(a,x) memset(a,x,sizeof(a))
19 #define pb push_back
20 #define mp make_pair
21 #define all(x) (x).begin(),(x).end()
22 #define fi first
23 #define se second
24 #define SZ(x) ((int)(x).size())
25 typedef unsigned long long ull;
26 typedef long long ll;
27 typedef vector<int> vi;
28 typedef pair<int,int> pii;
29 /*****head*****/
30 int work(){
31
32     return 0;
33 }
34 int main(){
35 #ifdef superkunn
36     freopen("input.txt","rt",stdin);
37 #endif
38     work();
39     return 0;
40 }
```

1 Math

1.1 Prime

1.1.1 Eratosthenes Sieve

$O(n \log \log n)$ maxn
 $notprime[i] = 0/1 \quad 0 \quad 1$

```

1  const int maxn = "Edit";
2  bool notprime[maxn] = {1, 1};    // 0 && 1
3  void GetPrime()
4  {
5      for (int i = 2; i < maxn; i++)
6          if (!notprime[i] && i <= maxn / i) //  $\sqrt{n}$ 
7              for (int j = i * i; j < maxn; j += i)
8                  notprime[j] = 1;
9  }
```

1.1.2 Euler Sieve

$O(n)$ phi[] prime[] tot
 n

```

1  const int maxn = "Edit";
2  bool vis[maxn];
3  int tot, phi[maxn], prime[maxn];
4  void CalPhi(int n)
5  {
6      clr(vis, 0);
7      phi[1] = 1;
8      tot = 0;
9      for (int i = 2; i < n; i++)
10     {
11         if (!vis[i])
12             prime[tot++] = i, phi[i] = i - 1;
13         for (int j = 0; j < tot; j++)
14         {
15             if (i * prime[j] > n) break;
16             vis[i * prime[j]] = 1;
17             if (i % prime[j] == 0)
18             {
19                 phi[i * prime[j]] = phi[i] * prime[j];
20                 break;
21             }
22             else
23                 phi[i * prime[j]] = phi[i] * (prime[j] - 1);
24         }
25     }
26 }
```

1.1.3 Prime Factorization

$fact[i][0]^{fact[i][1]}$ i

```

1 ll fact[100][2];
2 int getFactors(ll x)
3 {
4     int cnt = 0;
5     for (int i = 0; prime[i] <= x / prime[i]; i++)
6     {
7         fact[cnt][1] = 0;
8         if (x % prime[i] == 0)
9         {
10             fact[cnt][0] = prime[i];
11             while (x % prime[i] == 0) fact[cnt][1]++, x /= prime[i];
12             cnt++;
13         }
14     }
15     if (x != 1) fact[cnt][0] = x, fact[cnt++][1] = 1;
16     return cnt;
17 }

```

1.1.4 Miller Rabin

$O(s \log n) \quad 2^{63}, s$

```

1 bool Miller_Rabin(ll n, int s)
2 {
3     if (n == 2) return 1;
4     if (n < 2 || !(n & 1)) return 0;
5     int t = 0;
6     ll x, y, u = n - 1;
7     while ((u & 1) == 0) t++, u >>= 1;
8     for (int i = 0; i < s; i++)
9     {
10         ll a = rand() % (n - 1) + 1;
11         ll x = Pow(a, u, n);
12         for (int j = 0; j < t; j++)
13         {
14             ll y = Mul(x, x, n);
15             if (y == 1 && x != 1 && x != n - 1) return 0;
16             x = y;
17         }
18         if (x != 1) return 0;
19     }
20     return 1;
21 }

```

1.1.5 Segment Sieve

$[a, b)$

is_prime[i-a]=true i
 $a < b \leq 10^{12}, b - a \leq 10^6$

```

1 const int maxn = "Edit";
2 bool is_prime_small[maxn], is_prime[maxn];
3 int prime[maxn];
4 int segment_sieve(ll a, ll b)
5 {
6     int tot = 0;

```

```

7   for (ll i = 0; i * i < b; ++i)
8       is_prime_small[i] = true;
9   for (ll i = 0; i < b - a; ++i)
10      is_prime[i] = true;
11   for (ll i = 2; i * i < b; ++i)
12       if (is_prime_small[i])
13       {
14           for (ll j = 2 * i; j * j < b; j += i)
15               is_prime_small[j] = false;
16           for (ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
17               is_prime[j - a] = false;
18       }
19   for (ll i = 0; i < b - a; ++i)
20       if (is_prime[i]) prime[tot++] = i + a;
21   return tot;
22 }

```

1.2 Euler phi

1.2.1 Euler

```

1  ll Euler(ll n)
2  {
3      ll rt = n;
4      for (int i = 2; i * i <= n; i++)
5          if (n % i == 0)
6          {
7              rt -= rt / i;
8              while (n % i == 0) n /= i;
9          }
10     if (n > 1) rt -= rt / n;
11     return rt;
12 }

```

1.2.2 Sieve

```

1  const int N = "Edit";
2  int phi[N] = {0, 1};
3  void CalEuler()
4  {
5      for (int i = 2; i < N; i++)
6          if (!phi[i])
7              for (int j = i; j < N; j += i)
8              {
9                  if (!phi[j]) phi[j] = j;
10                 phi[j] = phi[j] / i * (i - 1);
11             }
12 }

```

1.3 Basic Number Theory

1.3.1 Extended Euclidean

```

1  ll exgcd(ll a, ll b, ll &x, ll &y)
2  {
3      ll d = a;
4      if (b) d = exgcd(b, a % b, y, x), y -= x * (a / b);

```



```

5     else x = 1, y = 0;
6     return d;
7 }

```

1.3.2 $ax+by=c$

$$\begin{matrix} : X = x + k * dx, Y = y - k * dy \\ x \quad , \quad 0 \end{matrix}$$

```

1 #define Mod(a, b) (((a) % (b) + (b)) % (b))
2 bool solve(ll a, ll b, ll c, ll& x, ll& y, ll& dx, ll& dy)
3 {
4     if (a == 0 && b == 0) return 0;
5     ll x0, y0;
6     ll d = exgcd(a, b, x0, y0);
7     if (c % d != 0) return 0;
8     dx = b / d, dy = a / d;
9     x = Mod(x0 * c / d, dx);
10    y = (c - a * x) / b;
11    // y = Mod(y0 * c / d, dy); x = (c - b * y) / a;
12    return 1;
13 }

```

1.3.3 Multiplicative Inverse Modulo

$$\text{exgcd } a \quad m \quad , \quad \text{gcd}(a, m) == 1.$$

```

1 ll inv(ll a, ll m)
2 {
3     ll x, y;
4     ll d = exgcd(a, m, x, y);
5     return d == 1 ? (x + m) % m : -1;
6 }

```

$$a < p \quad p \quad ,$$

```

1 ll inv(ll a, ll p) { return Pow(a, p - 2, p); }

1 for (int i = 2; i < n; i++) inv[i] = inv[p % i] * (p - p / i) % p;

```

1.4 Modulo Linear Equation

1.4.1 Chinese Remainder Theory

$$\begin{matrix} X = r_i(\text{mod } m_i); \quad m_i \\ X = re + k * mo \end{matrix}$$

```

1 void crt(ll r[], ll m[], ll n, ll &re, ll &mo)
2 {
3     mo = 1, re = 0;
4     for (int i = 0; i < n; i++) mo *= m[i];
5     for (int i = 0; i < n; i++)
6     {
7         ll x, y, tm = mo / m[i];
8         ll d = exgcd(tm, m[i], x, y);
9         re = (re + tm * x * r[i]) % mo;

```

```

10     }
11     re = (re + mo) % mo;
12 }

```

1.4.2 ExCRT

$$X = r_i \pmod{m_i}; m_i$$

$$X = re + k * mo;$$

```

1 bool excrt(ll r[], ll m[], ll n, ll &re, ll &mo)
2 {
3     ll x, y;
4     mo = m[0], re = r[0];
5     for (int i = 1; i < n; i++)
6     {
7         ll d = exgcd(mo, m[i], x, y);
8         if ((r[i] - re) % d != 0) return 0;
9         x = (r[i] - re) / d * x % (m[i] / d);
10        re += x * mo;
11        mo = mo / d * m[i];
12        re %= mo;
13    }
14    re = (re + mo) % mo;
15    return 1;
16 }

```

1.5 Combinatorics

1.5.1 Combination

$$0 \leq m \leq n \leq 1000$$

```

1 const int maxn = 1010;
2 ll C[maxn][maxn];
3 void CalComb()
4 {
5     C[0][0] = 1;
6     for (int i = 1; i < maxn; i++)
7     {
8         C[i][0] = 1;
9         for (int j = 1; j <= i; j++) C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
10    }
11 }

```

$$0 \leq m \leq n \leq 10^5, \quad p$$

```

1 const int maxn = 100010;
2 ll f[maxn];
3 ll inv[maxn]; //
4 void CalFact()
5 {
6     f[0] = 1;
7     for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
8     inv[maxn - 1] = Pow(f[maxn - 1], p - 2, p);
9     for (int i = maxn - 2; ~i; i--) inv[i] = inv[i + 1] * (i + 1) % p;
10 }
11 ll C(int n, int m) { return f[n] * inv[m] % p * inv[n - m] % p; }

```

1.5.2 Lucas

$1 \leq n, m \leq 1000000000, 1 < p < 100000, p$

```

1  const int maxp = 100010;
2  ll f[maxn];
3  ll inv[maxn]; //
4  void CalFact()
5  {
6      f[0] = 1;
7      for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
8      inv[maxn - 1] = Pow(f[maxn - 1], p - 2, p);
9      for (int i = maxn - 2; ~i; i--) inv[i] = inv[i + 1] * (i + 1) % p;
10 }
11 ll Lucas(ll n, ll m, ll p)
12 {
13     ll ret = 1;
14     while (n && m)
15     {
16         ll a = n % p, b = m % p;
17         if (a < b) return 0;
18         ret = ret * f[a] % p * inv[b] % p * inv[a - b] % p;
19         n /= p, m /= p;
20     }
21     return ret;
22 }
```

1.5.3 Big Combination

$0 \leq n \leq 10^9, 0 \leq m \leq 10^4, 1 \leq k \leq 10^9 + 7$

```

1  vector<int> v;
2  int dp[110];
3  ll Cal(int l, int r, int k, int dis)
4  {
5      ll res = 1;
6      for (int i = l; i <= r; i++)
7      {
8          int t = i;
9          for (int j = 0; j < v.size(); j++)
10             {
11                 int y = v[j];
12                 while (t % y == 0) dp[j] += dis, t /= y;
13             }
14         res = res * (ll)t % k;
15     }
16     return res;
17 }
18 ll Comb(int n, int m, int k)
19 {
20     clr(dp, 0);
21     v.clear();
22     int tmp = k;
23     for (int i = 2; i * i <= tmp; i++)
24         if (tmp % i == 0)
25             {
26                 int num = 0;
27                 while (tmp % i == 0) tmp /= i, num++;

```

```

28         v.pb(i);
29     }
30     if (tmp != 1) v.pb(tmp);
31     ll ans = Cal(n - m + 1, n, k, 1);
32     for (int j = 0; j < v.size(); j++) ans = ans * Pow(v[j], dp[j], k) % k;
33     ans = ans * inv(Cal(2, m, k, -1), k) % k;
34     return ans;
35 }

```

1.5.4 Polya

$$N * N^{\frac{m^8+17m^4+6m^2}{24}}, c^{n^2+2c^{\frac{n^2+3}{4}}+c^{\frac{n^2+1}{2}}+2c^{n\frac{n+1}{2}}+2c^{\frac{n(n+1)}{2}}}$$

```

1 // n c
2 ll solve(int c, int n)
3 {
4     if (n == 0) return 0;
5     ll ans = 0;
6     for (int i = 1; i <= n; i++) ans += Pow(c, __gcd(i, n));
7     if (n & 1) ans += n * Pow(c, n + 1 >> 1);
8     else ans += n / 2 * (1 + c) * Pow(c, n >> 1);
9     return ans / n / 2;
10 }

```

1.6 Fast Power

```

1 ll Mul(ll a, ll b, ll mod)
2 {
3     ll t = 0;
4     for (; b >= 1, a = (a << 1) % mod)
5         if (b & 1) t = (t + a) % mod;
6     return t;
7 }
8 ll Pow(ll a, ll n, ll mod)
9 {
10     ll t = 1;
11     for (; n; n >= 1, a = (a * a % mod))
12         if (n & 1) t = (t * a % mod);
13     return t;
14 }

```

1.7 Mobius Inversion

1.7.1 Mobius

$$F(n) = \sum_{d|n} f(d) \Rightarrow f(n) = \sum_{d|n} \mu(d) F\left(\frac{n}{d}\right)$$

$$F(n) = \sum_{n|d} f(d) \Rightarrow f(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) F(d)$$

```

1 ll ans;
2 const int maxn = "Edit";
3 int n, x, prime[maxn], tot, mu[maxn];
4 bool check[maxn];
5 void calmu()

```

```

6 {
7     mu[1] = 1;
8     for (int i = 2; i < maxn; i++)
9     {
10         if (!check[i]) prime[tot++] = i, mu[i] = -1;
11         for (int j = 0; j < tot; j++)
12         {
13             if (i * prime[j] >= maxn) break;
14             check[i * prime[j]] = true;
15             if (i % prime[j] == 0)
16             {
17                 mu[i * prime[j]] = 0;
18                 break;
19             }
20             else mu[i * prime[j]] = -mu[i];
21         }
22     }
23 }

```

1.7.2 Number of Coprime-pair

n ($n \leq 100000$), n

```

1 ll solve()
2 {
3     int b[100005];
4     ll _max, ans = 0;
5     clr(b, 0);
6     for (int i = 0; i < n; i++)
7     {
8         scanf("%d", &x);
9         if (x > _max) _max = x;
10        b[x]++;
11    }
12    for (int i = 1; i <= _max; i++)
13    {
14        int cnt = 0;
15        for (ll j = i; j <= _max; j += i) cnt += b[j];
16        ans += 1LL * mu[i] * cnt * cnt;
17    }
18    return (ans - b[1]) / 2;
19 }

```

1.7.3 VisibleTrees

$\gcd(x, y) = 1$, $x \leq n, y \leq m$

```

1 ll solve(int n, int m)
2 {
3     if (n < m) swap(n, m);
4     ll ans = 0;
5     for (int i = 1; i <= m; ++i) ans += (ll)mu[i] * (n / i) * (m / i);
6     return ans;
7 }

```

1.8 Fast Transformation

1.8.1 FFT

```

1  const double PI = acos(-1.0);
2  //
3  struct Complex
4  {
5      double x, y; // x+yi
6      Complex(double _x = 0.0, double _y = 0.0) { x = _x, y = _y; }
7      Complex operator-(const Complex& b) const { return Complex(x - b.x, y - b.y); }
8      Complex operator+(const Complex& b) const { return Complex(x + b.x, y + b.y); }
9      Complex operator*(const Complex& b) const { return Complex(x * b.x - y * b.y, x * b
        .y + y * b.x); }
10 };
11 /*
12  * FFT IFFT
13  * i (i )
14  * len 2
15  */
16 void change(Complex y[], int len)
17 {
18     for (int i = 1, j = len / 2; i < len - 1; i++)
19     {
20         if (i < j) swap(y[i], y[j]);
21         // ,i<j
22         //i +1,j +1, i j
23         int k = len / 2;
24         while (j >= k) j -= k, k /= 2;
25         if (j < k) j += k;
26     }
27 }
28 /*
29  * FFT
30  * len 2^k ,
31  * on==1 DFT,on== -1 IDFT
32  */
33 void fft(Complex y[], int len, int on)
34 {
35     change(y, len);
36     for (int h = 2; h <= len; h <= 1)
37     {
38         Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
39         for (int j = 0; j < len; j += h)
40         {
41             Complex w(1, 0);
42             for (int k = j; k < j + h / 2; k++)
43             {
44                 Complex u = y[k];
45                 Complex t = w * y[k + h / 2];
46                 y[k] = u + t, y[k + h / 2] = u - t;
47                 w = w * wn;
48             }
49         }
50     }
51     if (on == -1)
52         for (int i = 0; i < len; i++) y[i].x /= len;
53 }

```

1.8.2 NTT

$$P^G, G^P, G^{\frac{P-1}{n}} \quad w_n = e^{\frac{2i\pi}{n}} \quad P, G \quad 1.11$$

```

1  const int mod = 119 << 23 | 1;
2  const int G = 3;
3  int wn[20];
4  void getwn()
5  { //
6      for (int i = 0; i < 20; i++) wn[i] = Pow(G, (mod - 1) / (1 << i), mod);
7  }
8  void change(int y[], int len)
9  {
10     for (int i = 1, j = len / 2; i < len - 1; i++)
11     {
12         if (i < j) swap(y[i], y[j]);
13         int k = len / 2;
14         while (j >= k) j -= k, k /= 2;
15         if (j < k) j += k;
16     }
17 }
18 void ntt(int y[], int len, int on)
19 {
20     change(y, len);
21     for (int h = 2, id = 1; h <= len; h <<= 1, id++)
22     {
23         for (int j = 0; j < len; j += h)
24         {
25             int w = 1;
26             for (int k = j; k < j + h / 2; k++)
27             {
28                 int u = y[k] % mod;
29                 int t = 1LL * w * (y[k + h / 2] % mod) % mod;
30                 y[k] = (u + t) % mod, y[k + h / 2] = ((u - t) % mod + mod) % mod;
31                 w = 1LL * w * wn[id] % mod;
32             }
33         }
34     }
35     if (on == -1)
36     {
37         //
38         int inv = Pow(len, mod - 2, mod);
39         for (int i = 1; i < len / 2; i++) swap(y[i], y[len - i]);
40         for (int i = 0; i < len; i++) y[i] = 1LL * y[i] * inv % mod;
41     }
42 }

```

1.8.3 FWT

```

1  void fwt(int f[], int m)
2  {
3      int n = __builtin_ctz(m);
4      for (int i = 0; i < n; ++i)
5          for (int j = 0; j < m; ++j)
6              if (j & (1 << i))
7              {
8                  int l = f[j ^ (1 << i)], r = f[j];
9                  f[j ^ (1 << i)] = l + r, f[j] = l - r;

```

```

10         // or: f[j] += f[j ^ (1 << i)];
11         // and: f[j ^ (1 << i)] += f[j];
12     }
13 }
14 void ifwt(int f[], int m)
15 {
16     int n = __builtin_ctz(m);
17     for (int i = 0; i < n; ++i)
18         for (int j = 0; j < m; ++j)
19             if (j & (1 << i))
20             {
21                 int l = f[j ^ (1 << i)], r = f[j];
22                 f[j ^ (1 << i)] = (l + r) / 2, f[j] = (l - r) / 2;
23                 //
24                 // or: f[j] -= f[j ^ (1 << i)];
25                 // and: f[j ^ (1 << i)] -= f[j];
26             }
27 }

```

1.9 Numerical Integration

1.9.1 Adaptive Simpson's Rule

$$\int_a^b f(x)dx \approx \frac{b-a}{6}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$$

$$|S(a, c) + S(c, b) - S(a, b)|/15 < \epsilon$$

```

1 double F(double x) {}
2 double simpson(double a, double b)
3 { // Simpson
4     double c = a + (b - a) / 2;
5     return (F(a) + 4 * F(c) + F(b)) * (b - a) / 6;
6 }
7 double asr(double a, double b, double eps, double A)
8 { // Simpson ( ) [a,b] Simpson A
9     double c = a + (b - a) / 2;
10    double L = simpson(a, c), R = simpson(c, b);
11    if (fabs(L + R - A) <= 15 * eps) return L + R + (L + R - A) / 15.0;
12    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
13 }
14 double asr(double a, double b, double eps) { return asr(a, b, eps, simpson(a, b)); }

```

1.9.2 Berlekamp-Massey

```

1 const int N = 1 << 14;
2 ll res[N], base[N], _c[N], _md[N];
3 vector<int> Md;
4 void mul(ll* a, ll* b, int k)
5 {
6     for (int i = 0; i < k + k; i++) _c[i] = 0;
7     for (int i = 0; i < k; i++)
8         if (a[i])
9             for (int j = 0; j < k; j++) _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
10    for (int i = k + k - 1; i >= k; i--)
11        if (_c[i])
12            for (int j = 0; j < Md.size(); j++) _c[i - k + Md[j]] = (_c[i - k + Md[j]]
13            - _c[i] * _md[Md[j]]) % mod;
14    for (int i = 0; i < k; i++) a[i] = _c[i];

```



```

15 int solve(ll n, VI a, VI b)
16 {
17     ll ans = 0, pnt = 0;
18     int k = a.size();
19     assert(a.size() == b.size());
20     for (int i = 0; i < k; i++) _md[k - 1 - i] = -a[i];
21     _md[k] = 1;
22     Md.clear();
23     for (int i = 0; i < k; i++)
24         if (_md[i] != 0) Md.push_back(i);
25     for (int i = 0; i < k; i++) res[i] = base[i] = 0;
26     res[0] = 1;
27     while ((1LL << pnt) <= n) pnt++;
28     for (int p = pnt; p >= 0; p--)
29     {
30         mul(res, res, k);
31         if ((n >> p) & 1)
32         {
33             for (int i = k - 1; i >= 0; i--) res[i + 1] = res[i];
34             res[0] = 0;
35             for (int j = 0; j < Md.size(); j++) res[Md[j]] = (res[Md[j]] - res[k] * _md
[Md[j]]) % mod;
36         }
37     }
38     for (int i = 0; i < k; i++) ans = (ans + res[i] * b[i]) % mod;
39     if (ans < 0) ans += mod;
40     return ans;
41 }
42 VI BM(VI s)
43 {
44     VI C(1, 1), B(1, 1);
45     int L = 0, m = 1, b = 1;
46     for (int n = 0; n < s.size(); n++)
47     {
48         ll d = 0;
49         for (int i = 0; i <= L; i++) d = (d + (ll)C[i] * s[n - i]) % mod;
50         if (d == 0)
51             ++m;
52         else if (2 * L <= n)
53         {
54             VI T = C;
55             ll c = mod - d * Pow(b, mod - 2) % mod;
56             while (C.size() < B.size() + m) C.pb(0);
57             for (int i = 0; i < B.size(); i++) C[i + m] = (C[i + m] + c * B[i]) % mod;
58             L = n + 1 - L, B = T, b = d, m = 1;
59         }
60         else
61         {
62             ll c = mod - d * Pow(b, mod - 2) % mod;
63             while (C.size() < B.size() + m) C.pb(0);
64             for (int i = 0; i < B.size(); i++) C[i + m] = (C[i + m] + c * B[i]) % mod;
65             ++m;
66         }
67     }
68     return C;
69 }
70 int gao(VI a, ll n)
71 {
72     VI c = BM(a);

```

```

73     c.erase(c.begin());
74     for (int i = 0; i < c.size(); i++) c[i] = (mod - c[i]) % mod;
75     return solve(n, c, VI(a.begin(), a.begin() + c.size()));
76 }

```

1.10 Others

```

n, m
1 int josephus(int n, int m)
2 {
3     int r = 0;
4     for (int k = 1; k <= n; ++k) r = (r + m) % k;
5     return r + 1;
6 }

n^n
1 int leftmost(int n)
2 {
3     double m = n * log10((double)n);
4     double g = m - (ll)m;
5     return (int)pow(10.0, g);
6 }

n!
1 int count(ll n)
2 {
3     if (n == 1) return 1;
4     return (int)ceil(0.5 * log10(2 * M_PI * n) + n * log10(n) - n * log10(M_E));
5 }

```

1.11 Formula

1. $n = \prod_{i=1}^k p_i^{a_i}$,
 - (a) $f(n) = \prod_{i=1}^k (a_i + 1)$
 - (b) $g(n) = \prod_{i=1}^k (\sum_{j=0}^{a_i} p_i^j)$
2. $n \varphi(n) / 2$
3. $\gcd(n, i) = 1, \gcd(n, n - i) = 1 (1 \leq i \leq n)$
4. $D(n) = (n - 1)(D(n - 2) + D(n - 1)) = \sum_{i=2}^n \frac{(-1)^k n!}{k!} = \lfloor \frac{n!}{e} + 0.5 \rfloor$
5. $p \text{ is prime} \Rightarrow (p - 1)! \equiv -1 \pmod{p}$
6. $\gcd(a, n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1 \pmod{n}$
7. $\gcd(n, p) = 1 \Rightarrow a^n \equiv a^{n \% \varphi(p)} \pmod{p}$
8. $\pi(n), \lim_{n \rightarrow \infty} \pi(n) = \frac{n}{\ln n}$
9. $x = N = \log_{10}(n) + 1$
10. $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$
11. $a > 1, m, n > 0, \gcd(a^m - 1, a^n - 1) = a^{\gcd(m, n)} - 1$
12. $a > b, \gcd(a, b) = 1, \gcd(a^m - b^m, a^n - b^n) = a^{\gcd(m, n)} - b^{\gcd(m, n)}$

$$G = \gcd(C_n^1, C_n^2, \dots, C_n^{n-1}) = \begin{cases} n, & n \text{ is prime} \\ 1, & n \text{ has multy prime factors} \\ p, & n \text{ has single prime factor } p \end{cases}$$

$$\gcd(\text{Fib}(m), \text{Fib}(n)) = \text{Fib}(\gcd(m, n))$$

13. $\gcd(m, n) = 1, :$

(a) $m * n - m - n$

(b) $N = \frac{(m-1)(n-1)}{2}$

14. $(n+1)lcm(C_n^0, C_n^1, \dots, C_n^{n-1}, C_n^n) = lcm(1, 2, \dots, n+1)$

15. $p \text{ , } (x+y+\dots+w)^p \equiv x^p + y^p + \dots + w^p \pmod{p}$

16. $:1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012$

$$h(0) = h(1) = 1, h(n) = \frac{(4n-2)h(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$$

17. $:B_n = -\frac{1}{n+1} \sum_{i=0}^{n-1} C_{n+1}^i B_i$

$$\sum_{i=1}^n i^k = \frac{1}{k+1} \sum_{i=1}^{k+1} C_{k+1}^i B_{k+1-i} (n+1)^i$$

18. FFT

$r \cdot 2^k + 1$	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	119	23	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

2 String Processing

2.1 KMP

```

1 //MAXN
2 int nxt[MAXN];
3 void initkmp(char x[],int m){
4     int i=0,j=nxt[0]=-1;
5     while(i<m){
6         while(j!=-1&&x[i]!=x[j])j=nxt[j];
7         nxt[++i]=++j;
8     }
9 }
10 //x:pa y:tx
11 int kmp(char x[],int m,char y[],int n){
12     int i,j,ans;
13     i=j=ans=0;
14     initkmp(x,m);
15     while(i<n){
16         while(j!=-1&&y[i]!=x[j])j=nxt[j];
17         i++,j++;
18         if(j>=m){
19             ans++;
20             j=nxt[j];
21             //pos:i-m
22         }
23     }
24     return ans;
25 }

```

2.2 ExtendKMP

```

1 //next[i]:x[i...m-1] x[0...m-1]
2 //extend[i]:y[i...n-1] x[0...m-1]
3 const int N = "Edit";
4 int next[N], extend[N];
5 void pre_ekmp(char x[], int m)
6 {
7     next[0] = m;
8     int j = 0;
9     while (j + 1 < m && x[j] == x[j + 1]) j++;
10    next[1] = j;
11    int k = 1;
12    for (int i = 2; i < m; i++)
13    {
14        int p = next[k] + k - 1;
15        int L = next[i - k];
16        if (i + L < p + 1)
17            next[i] = L;
18        else
19        {
20            j = max(0, p - i + 1);
21            while (i + j < m && x[i + j] == x[j]) j++;
22            next[i] = j;
23            k = i;
24        }
25    }
26 }

```

```

27 void ekmp(char x[], int m, char y[], int n)
28 {
29     pre_ekmp(x, m, next);
30     int j = 0;
31     while (j < n && j < m && x[j] == y[j]) j++;
32     extend[0] = j;
33     int k = 0;
34     for (int i = 1; i < n; i++)
35     {
36         int p = extend[k] + k - 1;
37         int L = next[i - k];
38         if (i + L < p + 1)
39             extend[i] = L;
40         else
41         {
42             j = max(0, p - i + 1);
43             while (i + j < n && j < m && y[i + j] == x[j]) j++;
44             extend[i] = j, k = i;
45         }
46     }
47 }

```

2.3 Manacher

```

1  const int MAXN=3e5;//more than two times
2  char s[MAXN],str[MAXN];
3  int len1,len2,p[MAXN];
4  void init(){
5      str[0]='$';
6      str[1]='#';
7      rep(i,0,len1-1){
8          str[i*2+2]=s[i];
9          str[i*2+3]='#';
10     }
11     len2=len1*2+2;
12     str[len2]='*';
13 }
14 void manacher(){
15     int id=0,mx=0;
16     rep(i,1,len2-1){
17         if(mx>i)p[i]=min(p[2*id-i],mx-i);
18         else p[i]=1;
19         while(str[i+p[i]]==str[i-p[i]])p[i]++;
20         if(p[i]+i>mx){
21             mx=p[i]+i;
22             id=i;
23         }
24     }
25 }
26 int work(){
27     while(scanf("%s",s)!=EOF){
28         len1=strlen(s);
29         init();
30         manacher();
31         int ans=0;
32         rep(i,0,len2-1){
33             ans=max(ans,p[i]);
34         }

```

```

35     printf("%d\n",ans-1);
36 }
37 return 0;
38 }

```

2.4 Aho-Corasick Automaton

```

1  const int maxn = "Edit";
2  struct Trie
3  {
4      int ch[maxn][26], f[maxn], val[maxn];
5      int sz, rt;
6      int newnode() { clr(ch[sz], -1), val[sz] = 0; return sz++; }
7      void init() { sz = 0, rt = newnode(); }
8      inline int idx(char c) { return c - 'A'; }
9      void insert(const char* s)
10     {
11         int u = 0, n = strlen(s);
12         for (int i = 0; i < n; i++)
13         {
14             int c = idx(s[i]);
15             if (ch[u][c] == -1) ch[u][c] = newnode();
16             u = ch[u][c];
17         }
18         val[u]++;
19     }
20     void build()
21     {
22         queue<int> q;
23         f[rt] = rt;
24         for (int c = 0; c < 26; c++)
25         {
26             if (~ch[rt][c])
27                 f[ch[rt][c]] = rt, q.push(ch[rt][c]);
28             else
29                 ch[rt][c] = rt;
30         }
31         while (!q.empty())
32         {
33             int u = q.front();
34             q.pop();
35             // val[u] += val[f[u]];
36             for (int c = 0; c < 26; c++)
37             {
38                 if (~ch[u][c])
39                     f[ch[u][c]] = ch[f[u]][c], q.push(ch[u][c]);
40                 else
41                     ch[u][c] = ch[f[u]][c];
42             }
43         }
44     }
45     //
46     int query(const char* s)
47     {
48         int u = rt, n = strlen(s);
49         int res = 0;
50         for (int i = 0; i < n; i++)
51         {

```

```

52         int c = idx(s[i]);
53         u = ch[u][c];
54         int tmp = u;
55         while (tmp != rt)
56         {
57             res += val[tmp];
58             val[tmp] = 0;
59             tmp = f[tmp];
60         }
61     }
62     return res;
63 }
64 };

```

2.5 Suffix Array

```

1 //      , 0(nlogn)
2 const int maxn = "Edit";
3 char s[maxn];
4 int sa[maxn], t[maxn], t2[maxn], c[maxn], rank[maxn], height[maxn];
5 //n      , 0~m-1
6 void build_sa(int m, int n)
7 {
8     n++;
9     int *x = t, *y = t2;
10    //
11    for (int i = 0; i < m; i++) c[i] = 0;
12    for (int i = 0; i < n; i++) c[x[i] = s[i]]++;
13    for (int i = 1; i < m; i++) c[i] += c[i - 1];
14    for (int i = n - 1; ~i; i--) sa[--c[x[i]]] = i;
15    for (int k = 1; k <= n; k <= 1)
16    {
17        // sa
18        int p = 0;
19        for (int i = n - k; i < n; i++) y[p++] = i;
20        for (int i = 0; i < n; i++)
21            if (sa[i] >= k) y[p++] = sa[i] - k;
22        //
23        for (int i = 0; i < m; i++) c[i] = 0;
24        for (int i = 0; i < n; i++) c[x[y[i]]]++;
25        for (int i = 0; i < m; i++) c[i] += c[i - 1];
26        for (int i = n - 1; ~i; i--) sa[--c[x[y[i]]]] = y[i];
27        // say x
28        swap(x, y);
29        p = 1;
30        x[sa[0]] = 0;
31        for (int i = 1; i < n; i++)
32            x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k] ? p
- 1 : p++;
33        if (p >= n) break; //      ,sa      ,
34        m = p;           //
35    }
36    n--;
37    int k = 0;
38    for (int i = 0; i <= n; i++) rank[sa[i]] = i;
39    for (int i = 0; i < n; i++)
40    {
41        if (k) k--;

```

```

42     int j = sa[rank[i] - 1];
43     while (s[i + k] == s[j + k]) k++;
44     height[rank[i]] = k;
45 }
46 }
47
48 int dp[maxn][30];
49 void initrmq(int n)
50 {
51     for (int i = 1; i <= n; i++)
52         dp[i][0] = height[i];
53     for (int j = 1; (1 << j) <= n; j++)
54         for (int i = 1; i + (1 << j) - 1 <= n; i++)
55             dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
56 }
57 int rmq(int l, int r)
58 {
59     int k = 31 - __builtin_clz(r - l + 1);
60     return min(dp[l][k], dp[r - (1 << k) + 1][k]);
61 }
62 int lcp(int a, int b)
63 { //
64     a = rank[a], b = rank[b];
65     if (a > b) swap(a, b);
66     return rmq(a + 1, b);
67 }

```

2.6 Suffix Automation

```

1  const int maxn = "Edit";
2  struct SAM
3  {
4      int len[maxn << 1], link[maxn << 1], ch[maxn << 1][26];
5      int sz, rt, last;
6      int newnode(int x = 0)
7      {
8          len[sz] = x;
9          link[sz] = -1;
10         clr(ch[sz], -1);
11         return sz++;
12     }
13     void init() { sz = last = 0, rt = newnode(); }
14     void extend(int c)
15     {
16         int np = newnode(len[last] + 1);
17         int p;
18         for (p = last; ~p && ch[p][c] == -1; p = link[p]) ch[p][c] = np;
19         if (p == -1)
20             link[np] = rt;
21         else
22         {
23             int q = ch[p][c];
24             if (len[p] + 1 == len[q])
25                 link[np] = q;
26             else
27             {
28                 int nq = newnode(len[p] + 1);
29                 memcpy(ch[nq], ch[q], sizeof(ch[q]));

```



```

30         link[nq] = link[q], link[q] = link[np] = nq;
31         for (; ~p && ch[p][c] == q; p = link[p]) ch[p][c] = nq;
32     }
33 }
34 last = np;
35 }
36 int topcnt[maxn], topsam[maxn << 1];
37 void sort()
38 { //
39     clr(topcnt, 0);
40     for (int i = 0; i < sz; i++) topcnt[len[i]]++;
41     for (int i = 0; i < maxn - 1; i++) topcnt[i + 1] += topcnt[i];
42     for (int i = 0; i < sz; i++) topsam[--topcnt[len[i]]] = i;
43 }
44 };

```

2.7 HashString

```

1  const ll B1=1e7+7;
2  const ll B2=1e9+7;
3  char pa[10004];
4  char tx[1000006];
5  int work(){
6      int T;
7      scanf("%d",&T);
8      while(T--){
9          scanf("%s%s",pa,tx);
10         int pl=strlen(pa);
11         int tl=strlen(tx);
12         ll w=1;
13         rep(i,1,pl)w=(w*B1)%B2;
14         ll ph=0,th=0;
15         rep(i,0,pl-1){
16             ph=(ph*B1+pa[i])%B2;
17             th=(th*B1+tx[i])%B2;
18         }
19         int ans=0;
20         for(int i=0;i+pl<=tl;i++){
21             if(ph==th)ans++;
22             if(i+pl<tl)th=(th*B1+tx[i+pl]-tx[i]*w)%B2;
23         }
24         printf("%d\n",ans);
25     }
26     return 0;
27 }

```

3 Data Structure

3.1 Binary Indexed Tree

```

1 //add(pos,a) sum(r)-sum(l-1)
2 //add(l,a) add(r+1,-a) sum(pos)
3 const int MAXN=100000;
4 struct BIT{
5     int n,c[MAXN<<1];
6     void init(int _n){n=_n;for(int i=0;i<=n;i++)c[i]=0;}
7     void add(int i,int v){for(;i<=n;i+=i&-i)c[i]+=v;}
8     int sum(int i){int s=0;for(;i>0;i-=i&-i)s+=c[i];return s;}
9 }bit;

```

3.1.1 poj3468

$$a_i = \sum_{j=1}^x d_j$$

$$\sum_{i=1}^x a_i = \sum_{i=1}^x \sum_{j=1}^i d_j = \sum_{i=1}^x (x-i+1)d_i$$

$$\sum_{i=1}^x a_i = (x+1) \sum_{i=1}^x d_i - \sum_{i=1}^x d_i \times i$$

```

1 const int MAXN=1e5+5;
2 int n,q,x,y,z;
3 long long c1[MAXN],c2[MAXN];
4 void add(int x,int y){
5     for(int i=x;i<=n;i+=i&(-i))c1[i]+=y,c2[i]+=1LL*x*y;
6 }
7 ll sum(int x){
8     ll ans(0);
9     for(int i=x;i>0;i-=i&(-i))ans+=1LL*(x+1)*c1[i]-c2[i];
10    return ans;
11 }
12 char op[5];
13 int work(){
14     scanf("%d%d",&n,&q);
15     int a1,a2;
16     a1=0;
17     rep(i,1,n){
18         scanf("%d",&a2);
19         add(i,a2-a1);
20         a1=a2;
21     }
22     while(q--){
23         scanf("%s",op);
24         if(op[0]=='Q'){
25             scanf("%d%d%d",&x,&y,&z);
26             printf("%lld\n",sum(y)-sum(x-1));
27         }else{
28             scanf("%d%d%d",&x,&y,&z);
29             add(x,z);
30             add(y+1,-z);
31         }
32     }
33     return 0;
34 }

```

3.2 Segment Tree

```

1  #define lson rt<<1
2  #define rson rt<<1|1
3  #define le l,m,lson
4  #define ri m+1,r,rson
5  #define mid m=(l+r)>>1

```

3.2.1 Single-point Update

```

1  const int MAXN=5e4+5;
2  int sum[MAXN<<2];
3  void push_up(int rt){
4      sum[rt]=sum[lson]+sum[rson];
5  }
6  void build(int l,int r,int rt){
7      if(l==r){
8          scanf("%d",&sum[rt]);
9          return;
10     }
11     int mid;
12     build(le);
13     build(ri);
14     push_up(rt);
15 }
16 void update(int p,int v,int l,int r,int rt){
17     if(l==r){
18         sum[rt]+=v;
19         return;
20     }
21     int mid;
22     if(p<=m)update(p,v,le);
23     else update(p,v,ri);
24     push_up(rt);
25 }
26 int query(int L,int R,int l,int r,int rt){
27     if(L<=l&&r<=R){
28         return sum[rt];
29     }
30     int mid;
31     int ret=0;
32     if(L<=m)ret+=query(L,R,le);
33     if(R>m)ret+=query(L,R,ri);
34     return ret;
35 }

```

3.2.2 Interval Update

```

1  const int MAXN=1e5+5;
2  ll lazy[MAXN<<2];
3  ll tree[MAXN<<2];
4  void push_up(int rt){
5      tree[rt]=tree[lson]+tree[rson];
6  }
7  void push_down(int rt,int m){
8      ll w=lazy[rt];
9      if(w){

```

```

10     lazy[lson]+=w;
11     lazy[rson]+=w;
12     tree[lson]+=w*(m-(m>>1));
13     tree[rson]+=w*(m>>1);
14     lazy[rt]=0;
15 }
16 }
17 void build(int l,int r,int rt){
18     lazy[rt]=0;
19     if(l==r){
20         scanf("%lld",&tree[rt]);
21         return;
22     }
23     int mid;
24     build(le);
25     build(ri);
26     push_up(rt);
27 }
28 void update(int L,int R,int v,int l,int r,int rt){
29     if(L<=l&&r<=R){
30         lazy[rt]+=v;
31         tree[rt]+=1ll*v*(r-l+1);
32         return;
33     }
34     push_down(rt,r-l+1);
35     int mid;
36     if(L<=m)update(L,R,v,le);
37     if(R>m)update(L,R,v,ri);
38     push_up(rt);
39 }
40 ll query(int L,int R,int l,int r,int rt){
41     if(L<=l&&r<=R){
42         return tree[rt];
43     }
44     push_down(rt,r-l+1);
45     int mid;
46     ll ret=0;
47     if(L<=m)ret+=query(L,R,le);
48     if(R>m)ret+=query(L,R,ri);
49     return ret;
50 }

```

3.3 Splay Tree

```

1 #define key_value ch[ch[rt][1]][0]
2 const int MAXN=1e5;
3 struct Splay{
4     int a[MAXN]; //0 base
5     int sz[MAXN],ch[MAXN][2],fa[MAXN];
6     int key[MAXN],rev[MAXN];
7     int rt,tot;
8     int stk[MAXN],top;
9     void push_up(int x){
10         sz[x]=sz[ch[x][0]]+sz[ch[x][1]]+1;
11     }
12     void push_down(int x){
13         if(rev[x]){
14             swap(ch[x][0],ch[x][1]);

```

```

15         if(ch[x][0])rev[ch[x][0]]^=1;
16         if(ch[x][1])rev[ch[x][1]]^=1;
17         rev[x]=0;
18     }
19 }
20 int newnode(int p=0,int k=0){
21     int x=top?stk[top--]:++tot;
22     fa[x]=p;
23     sz[x]=1;
24     ch[x][0]=ch[x][1]=0;
25     key[x]=k;
26     rev[x]=0;
27     return x;
28 }
29 int build(int l,int r,int p){
30     if(l>r)return 0;
31     int mid=(l+r)>>1;
32     int x=newnode(p,a[mid]);
33     ch[x][0]=build(l,mid-1,x);
34     ch[x][1]=build(mid+1,r,x);
35     push_up(x);
36     return x;
37 }
38 void init(int n){
39     tot=0,top=0;
40     rt=newnode(0,-1);
41     ch[rt][1]=newnode(rt,-1);
42     rep(i,0,n-1)a[i]=i+1;
43     key_value=build(0,n-1,ch[rt][1]);
44     push_up(ch[rt][1]);
45     push_up(rt);
46 }
47 void rotate(int x,int d){
48     int y=fa[x];
49     push_down(y);
50     push_down(x);
51     ch[y][d^1]=ch[x][d];
52     fa[ch[x][d]]=y;
53     if(fa[y])ch[fa[y]][ch[fa[y]][1]==y]=x;
54     fa[x]=fa[y];
55     ch[x][d]=y;
56     fa[y]=x;
57     push_up(y);
58 }
59 void splay(int x,int goal=0){
60     push_down(x);
61     while(fa[x]!=goal){
62         if(fa[fa[x]]==goal){
63             rotate(x,ch[fa[x]][0]==x);
64         }else{
65             int y=fa[x];
66             int d=ch[fa[y]][0]==y;
67             ch[y][d]==x?rotate(x,d^1):rotate(y,d);
68             rotate(x,d);
69         }
70     }
71     push_up(x);
72     if(goal==0)rt=x;
73 }

```

```

74     int kth(int r,int k){
75         push_down(r);
76         int t=sz[ch[r][0]]+1;
77         if(t==k)return r;
78         return t>k?kth(ch[r][0],k):kth(ch[r][1],k-t);
79     }
80     void select(int l,int r){
81         splay(kth(rt,1),0);
82         splay(kth(ch[rt][1],r-l+2),rt);
83     }
84 };

```

3.4 Functional Segment Tree

```

1  //poj 2104
2  const int MAXN=1e5+6;
3  int n,m,cnt,x,y,k,root[MAXN],a[MAXN];
4  struct node{int l,r,sum;}T[MAXN*40];
5  vi v;
6  int getid(int x){return lower_bound(all(v),x)-v.begin()+1;}
7  void update(int l,int r,int &x,int y,int pos){
8      x=++cnt;
9      T[x]=T[y];
10     T[x].sum++;
11     if(l==r)return;
12     int mid=(l+r)>>1;
13     if(mid>=pos)update(l,mid,T[x].l,T[y].l,pos);
14     else update(mid+1,r,T[x].r,T[y].r,pos);
15 }
16 int query(int l,int r,int x,int y,int k){
17     if(l==r)return l;
18     int sum=T[T[y].l].sum-T[T[x].l].sum;
19     int mid=(l+r)>>1;
20     if(sum>=k)return query(l,mid,T[x].l,T[y].l,k);
21     else return query(mid+1,r,T[x].r,T[y].r,k-sum);
22 }
23 int work(){
24     scanf("%d%d",&n,&m);
25     v.clear();
26     rep(i,1,n)scanf("%d",&a[i]),v.pb(a[i]);
27     sort(all(v)),v.erase(unique(all(v)),v.end());
28     cnt=0;
29     rep(i,1,n)update(1,n,root[i],root[i-1],getid(a[i]));
30     rep(i,1,m)scanf("%d%d%d",&x,&y,&k),printf("%d\n",v[query(1,n,root[x-1],root[y],k)-1]);
31     return 0;
32 }

```

3.5 Sparse Table

```

1  const int MAXN = "Edit";
2  int mmax[MAXN][30], mmin[MAXN][30];
3  int a[MAXN], n, k;
4  void init(){
5      for (int i = 1; i <= n; i++) mmax[i][0] = mmin[i][0] = a[i];
6      for (int j = 1; (1 << j) <= n; j++)
7          for (int i = 1; i + (1 << j) - 1 <= n; i++){

```

```

8         mmax[i][j] = max(mmax[i][j - 1], mmax[i + (1 << (j - 1))][j - 1]);
9         mmin[i][j] = min(mmin[i][j - 1], mmin[i + (1 << (j - 1))][j - 1]);
10    }
11 }
12 // op=0/1 return [l,r] max/min
13 int rmq(int l, int r, int op){
14     int k = 31 - __builtin_clz(r - l + 1);
15     if (op == 0)
16         return max(mmax[l][k], mmax[r - (1 << k) + 1][k]);
17     return min(mmin[l][k], mmin[r - (1 << k) + 1][k]);
18 }

2D

1 void init(){
2     for (int i = 0; (1 << i) <= n; i++){
3         for (int j = 0; (1 << j) <= m; j++){
4             if (i == 0 && j == 0) continue;
5             for (int row = 1; row + (1 << i) - 1 <= n; row++)
6                 for (int col = 1; col + (1 << j) - 1 <= m; col++)
7                     if (i){
8                         dp[row][col][i][j] = max(dp[row][col][i - 1][j];
9                         dp[row + (1 << (i - 1))][col][i - 1][j]);
10                    }else{
11                        dp[row][col][i][j] = max(dp[row][col][i][j - 1];
12                        dp[row][col + (1 << (j - 1))][i][j - 1]);
13                    }
14            }
15        }
16    }
17    int rmq(int x1, int y1, int x2, int y2){
18        int kx = 31 - __builtin_clz(x2 - x1 + 1);
19        int ky = 31 - __builtin_clz(y2 - y1 + 1);
20        int m1 = dp[x1][y1][kx][ky];
21        int m2 = dp[x2 - (1 << kx) + 1][y1][kx][ky];
22        int m3 = dp[x1][y2 - (1 << ky) + 1][kx][ky];
23        int m4 = dp[x2 - (1 << kx) + 1][y2 - (1 << ky) + 1][kx][ky];
24        return max(max(m1, m2), max(m3, m4));
25    }

```

3.6 Heavy-Light Decomposition

```

1 const int maxn = "Edit";
2 struct HLD
3 {
4     int n, dfs_clock;
5     int sz[maxn], top[maxn], son[maxn], dep[maxn], fa[maxn], id[maxn];
6     vector<int> G[maxn];
7     void init(int n)
8     {
9         this->n = n, clr(son, -1), dfs_clock = 0;
10        for (int i = 0; i < n; i++) G[i].clear();
11    }
12    void add_edge(int u, int v) { G[u].pb(v), G[v].pb(u); }
13    void dfs(int u, int p, int d)
14    {
15        dep[u] = d, fa[u] = p, sz[u] = 1;
16        for (auto& v : G[u])
17            {

```

```

18         if (v == p) continue;
19         dfs(v, u, d + 1);
20         sz[u] += sz[v];
21         if (son[u] == -1 || sz[v] > sz[son[u]]) son[u] = v;
22     }
23 }
24 void link(int u, int t)
25 {
26     top[u] = t, id[u] = ++dfs_clock;
27     if (son[u] == -1) return;
28     link(son[u], t);
29     for (auto& v : G[u])
30         if (v != son[u] && v != fa[u]) link(v, v);
31 }
32 // ,
33 int query_path(int u, int v)
34 {
35     int ret = 0;
36     while (top[u] != top[v])
37     {
38         if (dep[top[u]] < dep[top[v]]) swap(u, v);
39         ret += query(id[top[u]], id[u]);
40         u = fa[top[u]];
41     }
42     if (dep[u] > dep[v]) swap(u, v);
43     ret += query(id[u], id[v]);
44 }
45 };

```

3.7 Link-Cut Tree

```

1  const int maxn = "Edit";
2  struct LCT
3  {
4      int val[maxn], sum[maxn]; //
5      int rev[maxn], ch[maxn][2], fa[maxn];
6      int stk[maxn];
7      inline void init(int n)
8      { //
9          for (int i = 1; i <= n; i++) scanf("%d", val + i);
10     }
11     inline bool isroot(int x) { return ch[fa[x]][0] != x && ch[fa[x]][1] != x; }
12     inline bool get(int x) { return ch[fa[x]][1] == x; }
13     void pushdown(int x)
14     {
15         if (!rev[x]) return;
16         swap(ch[x][0], ch[x][1]);
17         if (ch[x][0]) rev[ch[x][0]] ^= 1;
18         if (ch[x][1]) rev[ch[x][1]] ^= 1;
19         rev[x] ^= 1;
20     }
21     void pushup(int x) { sum[x] = val[x] + sum[ch[x][0]] + sum[ch[x][1]]; }
22     void rotate(int x)
23     {
24         int y = fa[x], z = fa[fa[x]], d = get(x);
25         if (!isroot(y)) ch[z][get(y)] = x;

```



```

26     fa[x] = z;
27     ch[y][d] = ch[x][d ^ 1], fa[ch[y][d]] = y;
28     ch[x][d ^ 1] = y, fa[y] = x;
29     pushup(y), pushup(x);
30 }
31 void splay(int x)
32 {
33     int top = 0;
34     stk[++top] = x;
35     for (int i = x; !isroot(i); i = fa[i]) stk[++top] = fa[i];
36     for (int i = top; i; i--) pushdown(stk[i]);
37     for (int f; !isroot(x); rotate(x))
38         if (!isroot(f = fa[x])) rotate(get(x) == get(f) ? f : x);
39 }
40 void access(int x)
41 {
42     for (int y = 0; x; y = x, x = fa[x]) splay(x), ch[x][1] = y, pushup(x);
43 }
44 int find(int x) { access(x), splay(x); while (ch[x][0]) x = ch[x][0]; return x; }
45 void makeroot(int x) { access(x), splay(x), rev[x] ^= 1; }
46 void link(int x, int y) { makeroot(x), fa[x] = y, splay(x); }
47 void cut(int x, int y) { makeroot(x), access(y), splay(y), fa[x] = ch[y][0] = 0; }
48 void update(int x, int v) { val[x] = v, access(x), splay(x); }
49 int query(int x, int y) { makeroot(y), access(x), splay(x); return sum[x]; }
50 };

```

4 Graph Theory

4.1 Union-Find Set

```

1  const int MAXN=1e6+5;
2  struct DSU{
3      int p[MAXN];
4      void init(int n){for(int i=0;i<=n;i++)p[i]=i;}
5      int findp(int x){return x==p[x]?x:p[x]=findp(p[x]);}
6      void unite(int x,int y){x=findp(x);y=findp(y);if(x==y)return;p[y]=x;}
7      bool same(int x,int y){return findp(x)==findp(y);}
8  }dsu;

```

4.2 Minimal Spanning Tree

4.2.1 Kruskal

```

1  //poj 1258
2  #include<cstdio>
3  #include<algorithm>
4  using namespace std;
5  const int MAXE=1e5+5;
6  const int MAXN=1e5+5;
7  struct DSU{
8      int p[MAXN];
9      void init(int n){for(int i=0;i<=n;i++)p[i]=i;}
10     int findp(int x){return x==p[x]?x:p[x]=findp(p[x]);}
11     void unite(int x,int y){x=findp(x);y=findp(y);if(x==y)return;p[y]=x;}
12     bool same(int x,int y){return findp(x)==findp(y);}
13 }dsu;
14 struct edge{int u,v,cost;}es[MAXE];
15 bool cmp(const edge &x,const edge &y){return x.cost<y.cost;}
16 int V,E;
17 int kruskal(){
18     sort(es,es+E,cmp);
19     dsu.init(V);
20     int res=0;
21     for(int i=0;i<E;i++){
22         if(!dsu.same(es[i].u,es[i].v)){
23             dsu.unite(es[i].u,es[i].v);
24             res+=es[i].cost;
25         }
26     }
27     return res;
28 }
29 int main(){
30     while(~scanf("%d",&V)){
31         E=0;
32         for(int i=1;i<=V;i++){
33             for(int j=1;j<=V;j++){
34                 int w;
35                 scanf("%d",&w);
36                 if(i==j)continue;
37                 es[E].u=i;
38                 es[E].v=j;
39                 es[E].cost=w;
40                 E++;
41             }
42         }
43     }

```

```

42     }
43     printf("%d\n",kruskal());
44 }
45 return 0;
46 }

```

4.3 Shortest Path

4.3.1 Dijkstra

```

1  //cf 610 A
2  #include<bits/stdc++.h>
3  using namespace std;
4  const int INF=1e9;
5  const int MAXV=5e3+50;
6  const int MAXE=1e5+50;
7  int V;
8  struct edge{int to,cost;};
9  vector<edge> G[MAXV];
10 typedef pair<int,int> P;
11 int d[MAXV];
12 void dijkstra(int s){
13     priority_queue<P,vector<P>,greater<P> > que;
14     fill(d,d+V+1,INF);
15     d[s]=0;
16     que.push(P(0,s));
17     while(!que.empty()){
18         P t=que.top();
19         que.pop();
20         int v=t.second;
21         if(d[v]<t.first)continue;
22         for(int i=0;i<G[v].size();i++){
23             edge e=G[v][i];
24             if(d[e.to]>d[v]+e.cost){
25                 d[e.to]=d[v]+e.cost;
26                 que.push(P(d[e.to],e.to));
27             }
28         }
29     }
30 }
31 int mat[405][405];
32 int main(){
33     int n,m;
34     scanf("%d%d",&n,&m);
35     V=n;
36     for(int i=1;i<=m;i++){
37         int u,v;
38         scanf("%d%d",&u,&v);
39         G[u].push_back(edge{v,1});
40         G[v].push_back(edge{u,1});
41         mat[u][v]=mat[v][u]=1;
42     }
43     dijkstra(1);
44     int ans;
45     if(d[n]==INF){
46         printf("-1");
47         return 0;
48     }
49     ans=d[n];

```

```

50     for(int i=1;i<=n;i++)G[i].clear();
51     for(int i=1;i<=n;i++){
52         for(int j=1;j<=n;j++){
53             if(i==j)continue;
54             if(mat[i][j]==0){
55                 G[i].push_back(edge{j,1});
56             }
57         }
58     }
59     dijkstra(1);
60     if(d[n]==INF){
61         printf("-1");
62         return 0;
63     }
64     printf("%d",max(ans,d[n]));
65     return 0;
66 }

```

4.3.2 Spfa

```

1  //poj 3259
2  #include<cstdio>
3  #include<iostream>
4  #include<algorithm>
5  #include<queue>
6  #include<cstring>
7  using namespace std;
8  const int INF=1e9;
9  const int MAXV=500+5;
10 const int MAXE=2700+5;
11 int tot;
12 int head[MAXV];
13 struct node{
14     int to,cost,next;
15 }edge[MAXE<<1];
16 int d[MAXV];
17 queue<int> que;
18 bool inq[MAXV];
19 int qtime[MAXV];
20 void init(){
21     tot=0;
22     memset(head,-1,sizeof(head));
23 }
24 void add_edge(int u,int v,int x){
25     edge[tot].to=v;
26     edge[tot].cost=x;
27     edge[tot].next=head[u];
28     head[u]=tot++;
29 }
30 bool spfa(int n){
31     memset(d,-1,sizeof(d));
32     d[1]=0;
33     while(!que.empty())que.pop();
34     memset(inq,0,sizeof(inq));
35     memset(qtime,0,sizeof(qtime));
36     que.push(1);
37     inq[1]=1;
38     qtime[1]++;

```

```

39     while(!que.empty()){
40         int u=que.front();
41         que.pop();
42         inq[u]=0;
43         for(int i=head[u];i!=-1;i=edge[i].next){
44             int v=edge[i].to;
45             int w=edge[i].cost;
46             if(d[v]==-1||d[u]+w<d[v]){
47                 d[v]=d[u]+w;
48                 if(!inq[v]){
49                     inq[v]=1;
50                     que.push(v);
51                     qtime[v]++;
52                     if(qtime[v]>n){
53                         return false;
54                     }
55                 }
56             }
57         }
58     }
59     return true;
60 }
61 int main(){
62     int kase;
63     scanf("%d",&kase);
64     while(kase--){
65         init();
66         int n,m,w;
67         scanf("%d%d%d",&n,&m,&w);
68         while(m--){
69             int u,v,x;
70             scanf("%d%d%d",&u,&v,&x);
71             add_edge(u,v,x);
72             add_edge(v,u,x);
73         }
74         while(w--){
75             int u,v,x;
76             scanf("%d%d%d",&u,&v,&x);
77             add_edge(u,v,-x);
78         }
79         if(!spfa(n)){
80             puts("YES");
81         }else{
82             puts("NO");
83         }
84     }
85     return 0;
86 }

```

4.4 Topo Sort

```

1 //cf 915D
2 const int MAXN=505;
3 const int MAXM=1e5+5;
4 int n,m;
5 int tot;
6 int head[MAXN],cur[MAXN],idec[MAXN];
7 struct Edge{

```

```

8     int v,nxt;
9     Edge(){}
10    Edge(int v,int nxt):v(v),nxt(nxt){}
11 }edge[MAXM];
12 void init(){
13     tot=0;
14     clr(head,-1);
15 }
16 void add_edge(int u,int v){
17     edge[tot]=Edge(v,head[u]);
18     head[u]=tot++;
19 }
20 int que[MAXN];
21 int st,ed;
22 bool topsort(int x){
23     int nst=1,ned=0;
24     rep(i,1,n)cur[i]=idec[i];
25     cur[x]--;
26     que[++ned]=x;
27     while(nst<=ned){
28         int u=que[nst++];
29         for(int i=head[u];i!=-1;i=edge[i].nxt){
30             int v=edge[i].v;
31             if(--cur[v]==0)que[++ned]=v;
32         }
33     }
34     if(ned+ed==n)return true;
35     else return false;
36 }
37 int work(){
38     scanf("%d%d",&n,&m);
39     init();
40     while(m--){
41         int u,v;
42         scanf("%d%d",&u,&v);
43         add_edge(u,v);
44         idec[v]++;
45     }
46     st=1,ed=0;
47     rep(i,1,n){
48         if(idec[i]==0)que[++ed]=i;
49     }
50     while(st<=ed){
51         int u=que[st++];
52         for(int i=head[u];i!=-1;i=edge[i].nxt){
53             int v=edge[i].v;
54             if(--idec[v]==0)que[++ed]=v;
55         }
56     }
57     if(ed==n){
58         puts("YES");
59         return 0;
60     }
61     rep(i,1,n){
62         if(idec[i]==1){
63             if(topsort(i)){
64                 puts("YES");
65                 return 0;
66             }

```

```

67     }
68 }
69 puts("NO");
70 return 0;
71 }

```

4.5 LCA

4.5.1 Tarjan

Tarjan

$O(n + q)$

```

1  const int maxn = "Edit";
2  int par[maxn];           //
3  int ans[maxn];          //
4  vector<int> G[maxn];     //
5  vector<PII> query[maxn]; //
6  bool vis[maxn];         //
7  inline void init(int n)
8  {
9      for (int i = 1; i <= n; i++)
10     {
11         G[i].clear(), query[i].clear();
12         par[i] = i, vis[i] = 0;
13     }
14 }
15 inline void add_edge(int u, int v) { G[u].pb(v); }
16 inline void add_query(int id, int u, int v)
17 {
18     query[u].pb(mp(v, id));
19     query[v].pb(mp(u, id));
20 }
21 void tarjan(int u)
22 {
23     vis[u] = 1;
24     for (auto& v : G[u])
25     {
26         if (vis[v]) continue;
27         tarjan(v);
28         unite(u, v);
29     }
30     for (auto& q : query[u])
31     {
32         int &v = q.X, &id = q.Y;
33         if (!vis[v]) continue;
34         ans[id] = find(v);
35     }
36 }

```

4.5.2 LCArmq

```

1  #include<bits/stdc++.h>
2  #define MAXV 100005
3  #define MAXLOGV 32
4  using namespace std;
5  int N,M,Q;
6  int st[MAXLOGV][MAXV];

```

```

7  vector<int> G[MAXV];
8  int root;
9  int vs[MAXV*2];
10 int depth[MAXV*2];
11 int id[MAXV];
12 void dfs(int v,int p,int d,int &k){
13     id[v]=k;
14     vs[k]=v;
15     depth[k++]=d;
16     for(int i=0;i<G[v].size();i++){
17         if(G[v][i]!=p){
18             dfs(G[v][i],v,d+1,k);
19             vs[k]=v;
20             depth[k++]=d;
21         }
22     }
23 }
24 int getMin(int x, int y){
25     return depth[x]<depth[y]?x:y;
26 }
27
28 void rmq_init(int n){
29     for(int i=0;i<n;++i) st[0][i]=i;
30     for(int i=1;1<=i<n;++i)
31         for(int j=0;j+(1<=i)-1<n;++j)
32             st[i][j]=getMin(st[i-1][j],st[i-1][j+(1<=(i-1))]);
33 }
34 void init(int V){
35     int k=0;
36     dfs(root,-1,0,k);
37     rmq_init(V*2-1);
38 }
39 int query(int l, int r){
40     int k=31-__builtin_clz(r-l+1);
41     return getMin(st[k][l],st[k][r-(1<=k)+1]);
42 }
43 int lca(int u,int v){
44     if(u==v) return u;
45     return vs[query(min(id[u],id[v]),max(id[u],id[v]))];
46 }
47 int dis(int u,int v){
48     return depth[id[u]]+depth[id[v]]-2*depth[id[lca(u,v)]];
49 }
50 int main()
51 {
52     scanf("%d%d",&N,&M);
53     for(int i=0;i<M;i++){
54         int x,y;
55         scanf("%d%d",&x,&y);
56         G[x].push_back(y);
57         G[y].push_back(x);
58     }
59     root=0;
60     init(N);
61     scanf("%d",&Q);
62     while(Q--){
63         int x,y;
64         scanf("%d%d",&x,&y);
65         printf("%d\n",lca(x,y));

```



```

66     }
67     return 0;
68 }

```

4.6 Depth-First Traversal

4.6.1 Biconnected-Component

```

1  // bccno
2  const int maxn = "Edit";
3  int pre[maxn], iscut[maxn], bccno[maxn], dfs_clock, bcc_cnt;
4  vector<int> G[maxn], bcc[maxn];
5  stack<PII> s;
6  void init(int n)
7  {
8      for (int i = 0; i < n; i++) G[i].clear();
9  }
10 inline void add_edge(int u, int v) { G[u].pb(v), G[v].pb(u); }
11 int dfs(int u, int fa)
12 {
13     int lowu = pre[u] = ++dfs_clock;
14     int child = 0;
15     for (auto& v : G[u])
16     {
17         PII e = mp(u, v);
18         if (!pre[v])
19         {
20             // v
21             s.push(e);
22             child++;
23             int lowv = dfs(v, u);
24             lowu = min(lowu, lowv); // low
25             if (lowv >= pre[u])
26             {
27                 iscut[u] = true;
28                 bcc_cnt++;
29                 bcc[bcc_cnt].clear(); // !bcc 1
30                 for (;;)
31                 {
32                     PII x = s.top();
33                     s.pop();
34                     if (bccno[x.X] != bcc_cnt)
35                         bcc[bcc_cnt].pb(x.X), bcc[x.X] = bcc_cnt;
36                     if (bccno[x.Y] != bcc_cnt)
37                         bcc[bcc_cnt].pb(x.Y), bcc[x.Y] = bcc_cnt;
38                     if (x.X == u && x.Y == v) break;
39                 }
40             }
41         }
42         else if (pre[v] < pre[u] && v != fa)
43         {
44             s.push(e);
45             lowu = min(lowu, pre[v]); //
46         }
47     }
48     if (fa < 0 && child == 1) iscut[u] = 0;
49     return lowu;
50 }
51 void find_bcc(int n)

```

```

52 {
53     // s ,
54     clr(pre, 0), clr(iscut, 0), clr(bccno, 0);
55     dfs_clock = bcc_cnt = 0;
56     for (int i = 0; i < n; i++)
57         if (!pre[i]) dfs(i, -1);
58 }

```

4.6.2 Strongly Connected Component

```

1 //cf 999 E
2 #include<bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 const int MAXN = 5005;//
6 const int MAXM = 5005;//
7 struct Edge{
8     int to,next;
9 } edge[MAXN];
10 int head[MAXN],tot;
11 int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN]; //Belong 1~scc
12 int Index,top;
13 int scc;//
14 bool Instack[MAXN];
15 void init(){
16     tot = 0;
17     memset(head,-1,sizeof(head));
18 }
19 void addedge(int u,int v){
20     edge[tot].to = v;
21     edge[tot].next = head[u];
22     head[u] = tot++;
23 }
24 void Tarjan(int u){
25     int v;
26     Low[u] = DFN[u] = ++Index;
27     Stack[top++] = u;
28     Instack[u] = true;
29     for(int i = head[u]; i != -1; i = edge[i].next){
30         v = edge[i].to;
31         if( !DFN[v] ){
32             Tarjan(v);
33             if( Low[u] > Low[v] )Low[u] = Low[v];
34         }
35         else if(Instack[v] && Low[u] > DFN[v])
36             Low[u] = DFN[v];
37     }
38     if(Low[u] == DFN[u]){
39         scc++;
40         do{
41             v = Stack[--top];
42             Instack[v] = false;
43             Belong[v] = scc;
44         }
45         while( v != u);
46     }
47 }
48 void solve(int N){

```

```

49     memset(DFN,0,sizeof(DFN));
50     memset(Instack,0,sizeof(Instack));
51     Index = scc = top = 0;
52     for(int i = 1; i <= N; i++)
53         if(!DFN[i])
54             Tarjan(i);
55 }
56 int u[MAXM],v[MAXM],in[MAXN],vis[MAXN];
57 int n,m,s;
58 void dfs(int x){
59     Belong[x]=Belong[s];
60     vis[x]=true;
61     for(int i=head[x];i!=-1;i=edge[i].next){
62         int e=edge[i].to;
63         if(!vis[e])dfs(e);
64     }
65 }
66 int main(){
67     scanf("%d%d%d",&n,&m,&s);
68     init();
69     for(int i=1;i<=m;i++){
70         scanf("%d%d",&u[i],&v[i]);
71         addedge(u[i],v[i]);
72     }
73     solve(n);
74     dfs(s);
75     int ans=0;
76     for(int i=1;i<=m;i++){
77         if(Belong[u[i]]!=Belong[v[i]]){
78             in[Belong[v[i]]]++;
79         }
80     }
81     set<int> ss;
82     for(int i=1;i<=n;i++){
83         ss.insert(Belong[i]);
84     }
85     set<int>::iterator it;
86     for(it=ss.begin();it!=ss.end();it++){
87         if(*it!=Belong[s]){
88             if(in[*it]==0){
89                 ans++;
90             }
91         }
92     }
93     printf("%d",ans);
94     return 0;
95 }

```

4.6.3 2-SAT

```

1 //hdu 3062
2 #include<bits/stdc++.h>
3 using namespace std;
4 const int MAXV=1e4;
5 int V;
6 vector<int> G[MAXV];
7 vector<int> rG[MAXV];
8 vector<int> vs;

```

```

9  bool used[MAXV];
10 int Belong[MAXV];
11 void init(int x){
12     V=x;
13     for(int i=0;i<MAXV;i++){
14         G[i].clear();
15         rG[i].clear();
16     }
17 }
18 void add_edge(int u,int v){
19     G[u].push_back(v);
20     rG[v].push_back(u);
21 }
22 void dfs(int v){
23     used[v]=true;
24     for(int i=0;i<G[v].size();i++){
25         if(!used[G[v][i]]) dfs(G[v][i]);
26     }
27 }
28 void rdfs(int v,int k){
29     used[v]=true;
30     Belong[v]=k;
31     for(int i=0;i<rG[v].size();i++){
32         if(!used[rG[v][i]]) rdfs(rG[v][i],k);
33     }
34 }
35 int scc(){
36     memset(used,0,sizeof(used));
37     vs.clear();
38     for(int v=1;v<=V;v++){//from 1 to V
39         if(!used[v]) dfs(v);
40     }
41     int k=0;
42     memset(used,0,sizeof(used));
43     for(int i=vs.size()-1;i>=0;i--){
44         if(!used[vs[i]]) rdfs(vs[i],k++);
45     }
46     return k;
47 }
48 bool judge(){
49     for(int i=1;i<V;i+=2){
50         if(Belong[i]==Belong[i+1])return false;
51     }
52     return true;
53 }
54 int main(){
55     int n,m;
56     while(scanf("%d%d",&n,&m)!=EOF){
57         init(2*n);
58         for(int i=1;i<=m;i++){
59             int a1,a2,c1,c2;
60             scanf("%d%d%d%d",&a1,&a2,&c1,&c2);
61             add_edge(((a1*2+c1))+1,((a2*2+c2)^1)+1);
62             add_edge(((a2*2+c2))+1,((a1*2+c1)^1)+1);
63         }
64         scc();
65         printf("%s\n",judge()?"YES":"NO");
66     }
67     return 0;
68 }

```

4.7 Euler Path

- :
- : ()
- :
- : (,),
- G
- G
- G () 0 2.
- G
- G
- G
- G
- G
- u 1,v 1, (u ,v)
- G
- G
- G

4.7.1 Fleury

```

,
1  const int maxn = "Edit";
2  int G[maxn][maxn];
3  int deg[maxn][maxn];
4  vector<int> Ans;
5  inline void init() { clr(G, 0), clr(deg, 0); }
6  inline void AddEdge(int u, int v) { deg[u]++, deg[v]++, G[u][v]++, G[v][u]++; }
7  void Fleury(int s)
8  {
9      for (int i = 0; i < n; i++)
10         if (G[s][i])
11             {
12                 G[s][i]--, G[i][s]--;
13                 Fleury(i);
14             }
15     Ans.pb(s);
16 }

```

4.8 Bipartite Graph Matching

4.8.1 Hungry

```

1  //poj3041
2  const int MAXV=1e3+5;
3  struct BM{
4      int V;
5      vi G[MAXV];
6      int match[MAXV];
7      bool vis[MAXV];
8      void init(int x){

```

```

9      V=x;
10     rep(i,1,V)G[i].clear();
11 }
12 void add_edge(int u,int v){
13     G[u].pb(v);
14     G[v].pb(u);
15 }
16 bool dfs(int u){
17     vis[u]=true;
18     for(int i=0;i<(int)G[u].size();i++){
19         int v=G[u][i];
20         int w=match[v];
21         if(w==-1||(!vis[w]&&dfs(w))){
22             match[u]=v;
23             match[v]=u;
24             return true;
25         }
26     }
27     return false;
28 }
29 int matching(){
30     int ret=0;
31     clr(match,-1);
32     rep(i,1,V){
33         if(match[i]==-1){
34             clr(vis,0);
35             if(dfs(i))ret++;
36         }
37     }
38     return ret;
39 }
40 }bm;
41 int work(){
42     int n,k;
43     scanf("%d%d",&n,&k);
44     bm.init(2*n);
45     while(k--){
46         int u,v;
47         scanf("%d%d",&u,&v);
48         bm.add_edge(u,n+v);
49     }
50     printf("%d",bm.matching());
51     return 0;
52 }

```

4.8.2 Hungry(Multiple)

```

1  const int maxn = "Edit";
2  const int maxm = "Edit";
3  int uN, vN;      //u,v ,
4  int g[maxn][maxm]; //
5  int linker[maxm][maxn];
6  bool used[maxm];
7  int num[maxm]; //
8  bool dfs(int u)
9  {
10     for (int v = 0; v < vN; v++)
11         if (g[u][v] && !used[v])

```

```

12     {
13         used[v] = true;
14         if (linker[v][0] < num[v])
15         {
16             linker[v][++linker[v][0]] = u;
17             return true;
18         }
19         for (int i = 1; i <= num[0]; i++)
20             if (dfs(linker[v][i]))
21             {
22                 linker[v][i] = u;
23                 return true;
24             }
25     }
26     return false;
27 }
28 int hungary()
29 {
30     int res = 0;
31     for (int i = 0; i < vN; i++) linker[i][0] = 0;
32     for (int u = 0; u < uN; u++)
33     {
34         clr(used, 0);
35         if (dfs(u)) res++;
36     }
37     return res;
38 }

```

4.8.3 Kuhn-Munkres

```

1  const int maxn = "Edit";
2  int nx, ny; //
3  int g[maxn][maxn]; //
4  int linker[maxn], lx[maxn], ly[maxn]; //y ,x,y
5  int slack[N];
6  bool visx[N], visy[N];
7  bool dfs(int x)
8  {
9      visx[x] = true;
10     for (int y = 0; y < ny; y++)
11     {
12         if (visy[y]) continue;
13         int tmp = lx[x] + ly[y] - g[x][y];
14         if (tmp == 0)
15         {
16             visy[y] = true;
17             if (linker[y] == -1 || dfs(linker[y]))
18             {
19                 linker[y] = x;
20                 return true;
21             }
22         }
23         else if (slack[y] > tmp)
24             slack[y] = tmp;
25     }
26     return false;
27 }
28 int KM()

```

```

29 {
30     clr(linker, -1), clr(ly, 0);
31     for (int i = 0; i < nx; i++)
32     {
33         lx[i] = -INF;
34         for (int j = 0; j < ny; j++)
35             if (g[i][j] > lx[i]) lx[i] = g[i][j];
36     }
37     for (int x = 0; x < nx; x++)
38     {
39         clr(slack, 0x3f);
40         for (;;)
41         {
42             clr(visx, 0), clr(visy, 0);
43             if (dfs(x)) break;
44             int d = INF;
45             for (int i = 0; i < ny; i++)
46                 if (!visy[i] && d > slack[i]) d = slack[i];
47             for (int i = 0; i < nx; i++)
48                 if (visx[i]) lx[i] -= d;
49             for (int i = 0; i < ny; i++)
50                 if (visy[i])
51                     ly[i] += d;
52             else
53                 slack[i] -= d;
54         }
55     }
56     int res = 0;
57     for (int i = 0; i < ny; i++)
58         if (~linker[i]) res += g[linker[i]][i];
59     return res;
60 }

```

4.9 Network Flow

```

1 struct Edge
2 {
3     int from, to, cap, flow;
4     Edge(int u, int v, int c, int f)
5         : from(u), to(v), cap(c), flow(f) {}
6 };

1 struct Edge
2 {
3     int from, to, cap, flow, cost;
4     Edge(int u, int v, int c, int f, int w)
5         : from(u), to(v), cap(c), flow(f), cost(w) {}
6 };

```

, , ,

: S T , S X , Y T , X Y , , , , ,

m n , , $\{p_1, p_2\}$

S , X , Y , x , S , 1 , 1 , T , x , x
 S , T , m , $O(\log m)$
 k , k
 w $[u, v)$ $u \rightarrow v$, 1 , $-w$ $i \rightarrow i + 1$, k , 0 ,
 $G()$,
 s t , s , $S - \{s\}$

4.9.1 EdmondKarp

```

1  const int maxn = "Edit";
2  struct EdmondsKarp // O(v*E*E)
3  {
4      int n, m;
5      vector<Edge> edges; //
6      vector<int> G[maxn]; // G[i][j] i j e
7      int a[maxn]; // i
8      int p[maxn]; // p
9      void init(int n)
10     {
11         for (int i = 0; i < n; i++) G[i].clear();
12         edges.clear();
13     }
14     void AddEdge(int from, int to, int cap)
15     {
16         edges.pb(Edge(from, to, cap, 0));
17         edges.pb(Edge(to, from, 0, 0)); //
18         m = edges.size();
19         G[from].pb(m - 2);
20         G[to].pb(m - 1);
21     }
22     int Maxflow(int s, int t)
23     {
24         int flow = 0;
25         for (;;)
26         {
27             clr(a, 0);
28             queue<int> q;
29             q.push(s);
30             a[s] = INF;
31             while (!q.empty())
32             {
33                 int x = q.front();
34                 q.pop();
35                 for (int i = 0; i < G[x].size(); i++)
36                 {
37                     Edge& e = edges[G[x][i]];
38                     if (!a[e.to] && e.cap > e.flow)
39                     {
40                         p[e.to] = G[x][i];
41                         a[e.to] = min(a[x], e.cap - e.flow);
42                         q.push(e.to);
43                     }
44                 }
45                 if (a[t]) break;
46             }

```

```

47         if (!a[t]) break;
48         for (int u = t; u != s; u = edges[p[u]].from)
49         {
50             edges[p[u]].flow += a[t];
51             edges[p[u] ^ 1].flow -= a[t];
52         }
53         flow += a[t];
54     }
55     return flow;
56 }
57 };

```

4.9.2 Dinic

```

1  const int maxn = "Edit";
2  struct Dinic
3  {
4      int n, m, s, t;          // , ( ),
5      vector<Edge> edges;      // edge[e] edge[e^1]
6      vector<int> G[maxn];    // ,G[i][j] i j e
7      bool vis[maxn];         //BFS
8      int d[maxn];            // i
9      int cur[maxn];          //
10     void init(int n)
11     {
12         this->n = n;
13         for (int i = 0; i < n; i++) G[i].clear();
14         edges.clear();
15     }
16     void AddEdge(int from, int to, int cap)
17     {
18         edges.pb(Edge(from, to, cap, 0));
19         edges.pb(Edge(to, from, 0, 0));
20         m = edges.size();
21         G[from].pb(m - 2);
22         G[to].pb(m - 1);
23     }
24     bool BFS()
25     {
26         clr(vis, 0);
27         clr(d, 0);
28         queue<int> q;
29         q.push(s);
30         d[s] = 0;
31         vis[s] = 1;
32         while (!q.empty())
33         {
34             int x = q.front();
35             q.pop();
36             for (int i = 0; i < G[x].size(); i++)
37             {
38                 Edge& e = edges[G[x][i]];
39                 if (!vis[e.to] && e.cap > e.flow)
40                 {
41                     vis[e.to] = 1;
42                     d[e.to] = d[x] + 1;
43                     q.push(e.to);
44                 }

```

```

45     }
46 }
47 return vis[t];
48 }
49 int DFS(int x, int a)
50 {
51     if (x == t || a == 0) return a;
52     int flow = 0, f;
53     for (int& i = cur[x]; i < G[x].size(); i++)
54     {
55         //
56         Edge& e = edges[G[x][i]];
57         if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a, e.cap - e.flow))) > 0)
58         {
59             e.flow += f;
60             edges[G[x][i] ^ 1].flow -= f;
61             flow += f;
62             a -= f;
63             if (a == 0) break;
64         }
65     }
66     return flow;
67 }
68 int Maxflow(int s, int t)
69 {
70     this->s = s;
71     this->t = t;
72     int flow = 0;
73     while (BFS())
74     {
75         clr(cur, 0);
76         flow += DFS(s, INF);
77     }
78     return flow;
79 }
80 };

```

4.9.3 ISAP

```

1  const int maxn = "Edit";
2  struct ISAP
3  {
4      int n, m, s, t;          // , ( ),
5      vector<Edge> edges;      // edges[e] edges[e^1]
6      vector<int> G[maxn];     // ,G[i][j] i j e
7      bool vis[maxn];         //BFS
8      int d[maxn];            // i
9      int cur[maxn];          //
10     int p[maxn];             //
11     int num[maxn];           //
12     void init(int n)
13     {
14         this->n = n;
15         for (int i = 0; i < n; i++) G[i].clear();
16         edges.clear();
17     }
18     void AddEdge(int from, int to, int cap)
19     {

```

```

20     edges.pb(Edge(from, to, cap, 0));
21     edges.pb(Edge(to, from, 0, 0));
22     int m = edges.size();
23     G[from].pb(m - 2);
24     G[to].pb(m - 1);
25 }
26 int Augument()
27 {
28     int x = t, a = INF;
29     while (x != s)
30     {
31         Edge& e = edges[p[x]];
32         a = min(a, e.cap - e.flow);
33         x = edges[p[x]].from;
34     }
35     x = t;
36     while (x != s)
37     {
38         edges[p[x]].flow += a;
39         edges[p[x] ^ 1].flow -= a;
40         x = edges[p[x]].from;
41     }
42     return a;
43 }
44 void BFS()
45 {
46     clr(vis, 0);
47     clr(d, 0);
48     queue<int> q;
49     q.push(t);
50     d[t] = 0;
51     vis[t] = 1;
52     while (!q.empty())
53     {
54         int x = q.front();
55         q.pop();
56         int len = G[x].size();
57         for (int i = 0; i < len; i++)
58         {
59             Edge& e = edges[G[x][i]];
60             if (!vis[e.from] && e.cap > e.flow)
61             {
62                 vis[e.from] = 1;
63                 d[e.from] = d[x] + 1;
64                 q.push(e.from);
65             }
66         }
67     }
68 }
69 int Maxflow(int s, int t)
70 {
71     this->s = s;
72     this->t = t;
73     int flow = 0;
74     BFS();
75     clr(num, 0);
76     for (int i = 0; i < n; i++)
77         if (d[i] < INF) num[d[i]]++;
78     int x = s;

```

```

79     clr(cur, 0);
80     while (d[s] < n)
81     {
82         if (x == t)
83         {
84             flow += Augument();
85             x = s;
86         }
87         int ok = 0;
88         for (int i = cur[x]; i < G[x].size(); i++)
89         {
90             Edge& e = edges[G[x][i]];
91             if (e.cap > e.flow && d[x] == d[e.to] + 1)
92             {
93                 ok = 1;
94                 p[e.to] = G[x][i];
95                 cur[x] = i;
96                 x = e.to;
97                 break;
98             }
99         }
100         if (!ok) //Retreat
101         {
102             int m = n - 1;
103             for (int i = 0; i < G[x].size(); i++)
104             {
105                 Edge& e = edges[G[x][i]];
106                 if (e.cap > e.flow) m = min(m, d[e.to]);
107             }
108             if (--num[d[x]] == 0) break; //gap
109             num[d[x] = m + 1]++;
110             cur[x] = 0;
111             if (x != s) x = edges[p[x]].from;
112         }
113     }
114     return flow;
115 }
116 };

```

4.9.4 MinCost MaxFlow

```

1  const int maxn = "Edit";
2  struct MCMF
3  {
4      int n, m;
5      vector<Edge> edges;
6      vector<int> G[maxn];
7      int inq[maxn]; //
8      int d[maxn]; //bellmanford
9      int p[maxn]; //
10     int a[maxn]; //
11     void init(int n)
12     {
13         this->n = n;
14         for (int i = 0; i < n; i++) G[i].clear();
15         edges.clear();
16     }
17     void AddEdge(int from, int to, int cap, int cost)

```

```

18     {
19         edges.pb(Edge(from, to, cap, 0, cost));
20         edges.pb(Edge(to, from, 0, 0, -cost));
21         m = edges.size();
22         G[from].pb(m - 2);
23         G[to].pb(m - 1);
24     }
25     bool BellmanFord(int s, int t, int& flow, ll& cost)
26     {
27         for (int i = 0; i < n; i++) d[i] = INF;
28         clr(inq, 0);
29         d[s] = 0;
30         inq[s] = 1;
31         p[s] = 0;
32         a[s] = INF;
33         queue<int> q;
34         q.push(s);
35         while (!q.empty())
36         {
37             int u = q.front();
38             q.pop();
39             inq[u] = 0;
40             for (int i = 0; i < G[u].size(); i++)
41             {
42                 Edge& e = edges[G[u][i]];
43                 if (e.cap > e.flow && d[e.to] > d[u] + e.cost)
44                 {
45                     d[e.to] = d[u] + e.cost;
46                     p[e.to] = G[u][i];
47                     a[e.to] = min(a[u], e.cap - e.flow);
48                     if (!inq[e.to])
49                     {
50                         q.push(e.to);
51                         inq[e.to] = 1;
52                     }
53                 }
54             }
55         }
56         if (d[t] == INF) return false; //
57         flow += a[t];
58         cost += (ll)d[t] * (ll)a[t];
59         for (int u = t; u != s; u = edges[p[u]].from)
60         {
61             edges[p[u]].flow += a[t];
62             edges[p[u] ^ 1].flow -= a[t];
63         }
64         return true;
65     }
66     int MincostMaxflow(int s, int t, ll& cost)
67     {
68         int flow = 0;
69         cost = 0;
70         while (BellmanFord(s, t, flow, cost));
71         return flow;
72     }
73 };

```

5 Computational Geometry

5.1 Basic Function

```

1  #define zero(x) ((fabs(x) < eps ? 1 : 0))
2  #define sgn(x) (fabs(x) < eps ? 0 : ((x) < 0 ? -1 : 1))
3
4  struct point
5  {
6      double x, y;
7      point(double a = 0, double b = 0) { x = a, y = b; }
8      point operator-(const point& b) const { return point(x - b.x, y - b.y); }
9      point operator+(const point& b) const { return point(x + b.x, y + b.y); }
10     //
11     bool operator==(point& b) { return zero(x - b.x) && zero(y - b.y); }
12     // ( )
13     double operator*(const point& b) const { return x * b.x + y * b.y; }
14     // ( )
15     double operator^(const point& b) const { return x * b.y - y * b.x; }
16     // P a
17     point rotate(point b, double a)
18     {
19         double dx, dy;
20         (*this - b).split(dx, dy);
21         double tx = dx * cos(a) - dy * sin(a);
22         double ty = dx * sin(a) + dy * cos(a);
23         return point(tx, ty) + b;
24     }
25     // a b
26     void split(double& a, double& b) { a = x, b = y; }
27 };
28 struct line
29 {
30     point s, e;
31     line() {}
32     line(point ss, point ee) { s = ss, e = ee; }
33 };

```

5.2 Position

5.2.1 Point-Point

```

1  double dist(point a, point b) { return sqrt((a - b) * (a - b)); }

```

5.2.2 Line-Line

```

1  // <0, *> ; <1, *> ; <2, P> P;
2  pair<int, point> spoint(line l1, line l2)
3  {
4      point res = l1.s;
5      if (sgn((l1.s - l1.e) ^ (l2.s - l2.e)) == 0)
6          return mp(sgn((l1.s - l2.e) ^ (l2.s - l2.e)) != 0, res);
7      double t = ((l1.s - l2.s) ^ (l2.s - l2.e)) / ((l1.s - l1.e) ^ (l2.s - l2.e));
8      res.x += (l1.e.x - l1.s.x) * t;
9      res.y += (l1.e.y - l1.s.y) * t;
10     return mp(2, res);
11 }

```

5.2.3 Segment-Segment

```
1 bool segxseg(line l1, line l2)
2 {
3     return
4         max(l1.s.x, l1.e.x) >= min(l2.s.x, l2.e.x) &&
5         max(l2.s.x, l2.e.x) >= min(l1.s.x, l1.e.x) &&
6         max(l1.s.y, l1.e.y) >= min(l2.s.y, l2.e.y) &&
7         max(l2.s.y, l2.e.y) >= min(l1.s.y, l1.e.y) &&
8         sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e - l1.e) ^ (l1.s - l1.e)) <= 0 &&
9         sgn((l1.s - l2.e) ^ (l2.s - l2.e)) * sgn((l1.e - l2.e) ^ (l2.s - l2.e)) <= 0;
10 }
```

5.2.4 Line-Segment

```
1 //l1 ,l2
2 bool segxline(line l1, line l2)
3 {
4     return sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e - l1.e) ^ (l1.s - l1.e)) <=
5         0;
6 }
```

5.2.5 Point-Line

```
1 double pointtoline(point p, line l)
2 {
3     point res;
4     double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
5     res.x = l.s.x + (l.e.x - l.s.x) * t, res.y = l.s.y + (l.e.y - l.s.y) * t;
6     return dist(p, res);
7 }
```

5.2.6 Point-Segment

```
1 double pointtosegment(point p, line l)
2 {
3     point res;
4     double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
5     if (t >= 0 && t <= 1)
6         res.x = l.s.x + (l.e.x - l.s.x) * t, res.y = l.s.y + (l.e.y - l.s.y) * t;
7     else
8         res = dist(p, l.s) < dist(p, l.e) ? l.s : l.e;
9     return dist(p, res);
10 }
```

5.2.7 Point on Segment

```
1 bool PointOnSeg(point p, line l)
2 {
3     return
4         sgn((l.s - p) ^ (l.e - p)) == 0 &&
5         sgn((p.x - l.s.x) * (p.x - l.e.x)) <= 0 &&
6         sgn((p.y - l.s.y) * (p.y - l.e.y)) <= 0;
7 }
```


5.3 Polygon

5.3.1 Area

```

1 double area(point p[], int n)
2 {
3     double res = 0;
4     for (int i = 0; i < n; i++) res += (p[i] ^ p[(i + 1) % n]) / 2;
5     return fabs(res);
6 }

```

5.3.2 Point in Convex

```

1 //      ,      (      <0 >0)
2 //      : [0,n)
3 // -1 :
4 // 0 :
5 // 1 :
6 int PointInConvex(point a, point p[], int n)
7 {
8     for (int i = 0; i < n; i++)
9         if (sgn((p[i] - a) ^ (p[(i + 1) % n] - a)) < 0)
10             return -1;
11         else if (PointOnSeg(a, line(p[i], p[(i + 1) % n])))
12             return 0;
13     return 1;
14 }

```

5.3.3 Point in Polygon

```

1 //      ,poly[]      3, 0~n-1
2 // -1 :
3 // 0 :
4 // 1 :
5 int PointInPoly(point p, point poly[], int n)
6 {
7     int cnt;
8     line ray, side;
9     cnt = 0;
10    ray.s = p;
11    ray.e.y = p.y;
12    ray.e.x = -1000000000000.0; // -INF,
13    for (int i = 0; i < n; i++)
14    {
15        side.s = poly[i], side.e = poly[(i + 1) % n];
16        if (PointOnSeg(p, side)) return 0;
17        //
18        if (sgn(side.s.y - side.e.y) == 0)
19            continue;
20        if (PointOnSeg(side.s, ray))
21            cnt += (sgn(side.s.y - side.e.y) > 0);
22        else if (PointOnSeg(side.e, ray))
23            cnt += (sgn(side.e.y - side.s.y) > 0);
24        else if (segxseg(ray, side))
25            cnt++;
26    }
27    return cnt % 2 == 1 ? 1 : -1;
28 }

```

5.3.4 Judge Convex

```

1 //
2 // 1~n-1
3 bool isconvex(point poly[], int n)
4 {
5     bool s[3];
6     clr(s, 0);
7     for (int i = 0; i < n; i++)
8     {
9         s[sgn((poly[(i + 1) % n] - poly[i]) ^ (poly[(i + 2) % n] - poly[i])) + 1] = 1;
10        if (s[0] && s[2]) return 0;
11    }
12    return 1;
13 }

```

5.4 Integer Points

5.4.1 On Segment

```

1 int OnSegment(line l) { return __gcd(fabs(l.s.x - l.e.x), fabs(l.s.y - l.e.y)) + 1; }

```

5.4.2 On Polygon Edge

```

1 int OnEdge(point p[], int n)
2 {
3     int i, ret = 0;
4     for (i = 0; i < n; i++)
5         ret += __gcd(fabs(p[i].x - p[(i + 1) % n].x), fabs(p[i].y - p[(i + 1) % n].y));
6     return ret;
7 }

```

5.4.3 Inside Polygon

```

1 int InSide(point p[], int n)
2 {
3     int i, area = 0;
4     for (i = 0; i < n; i++)
5         area += p[(i + 1) % n].y * (p[i].x - p[(i + 2) % n].x);
6     return (fabs(area) - OnEdge(n, p)) / 2 + 1;
7 }

```

5.5 Circle

5.5.1 Circumcenter

```

1 point waixin(point a, point b, point c)
2 {
3     double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1 * a1 + b1 * b1) / 2;
4     double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2 * a2 + b2 * b2) / 2;
5     double d = a1 * b2 - a2 * b1;
6     return point(a.x + (c1 * b2 - c2 * b1) / d, a.y + (a1 * c2 - a2 * c1) / d);
7 }

```

6 Dynamic Programming

6.1 Subsequence

6.1.1 Max Sum

```

1 // a n,
2 int MaxSeqSum(int a[], int n)
3 {
4     int rt = 0, cur = 0;
5     for (int i = 0; i < n; i++)
6         cur += a[i], rt = max(cur, rt), cur = max(0, cur);
7     return rt;
8 }

```

6.1.2 Longest Increase

```

1 // 1, LIS(), lis[]
2 const int N = "Edit";
3 int len, a[N], b[N], f[N];
4 int Find(int p, int l, int r)
5 {
6     while (l <= r)
7     {
8         int mid = (l + r) >> 1;
9         if (a[p] > b[mid])
10             l = mid + 1;
11         else
12             r = mid - 1;
13     }
14     return f[p] = l;
15 }
16 int LIS(int lis[], int n)
17 {
18     int len = 1;
19     f[1] = 1, b[1] = a[1];
20     for (int i = 2; i <= n; i++)
21     {
22         if (a[i] > b[len])
23             b[++len] = a[i], f[i] = len;
24         else
25             b[Find(i, 1, len)] = a[i];
26     }
27     for (int i = n, t = len; i >= 1 && t >= 1; i--)
28         if (f[i] == t) lis[--t] = a[i];
29     return len;
30 }
31
32 // ( 0 , )
33 int dp[N];
34 int LIS(int a[], int n)
35 {
36     clr(dp, 0x3f);
37     for (int i = 0; i < n; i++) *lower_bound(dp, dp + n, a[i]) = a[i];
38     return lower_bound(dp, dp + n, INF) - dp;
39 }

```

6.1.3 Longest Common Increase

```

1 // 1
2 int LCIS(int a[], int b[], int n, int m)
3 {
4     clr(dp, 0);
5     for (int i = 1; i <= n; i++)
6     {
7         int ma = 0;
8         for (int j = 1; j <= m; j++)
9         {
10             dp[i][j] = dp[i - 1][j];
11             if (a[i] > b[j]) ma = max(ma, dp[i - 1][j]);
12             if (a[i] == b[j]) dp[i][j] = ma + 1;
13         }
14     }
15     return *max_element(dp[n] + 1, dp[n] + 1 + m);
16 }

```

6.2 Digit Statistics

```

1 int a[20];
2 ll dp[20][state];
3 ll dfs(int pos, /*state*/, bool lead /* */, bool limit /* */)
4 {
5     // , , 0, pos== -1
6     if (pos == -1) return 1;
7     /* 1, , ,
8     pos, */
9     if (!limit && !lead && dp[pos][state] != -1) return dp[pos][state];
10    /* , */
11    int up = limit ? a[pos] : 9; // limit up
12    ll ans = 0;
13    for (int i = 0; i <= up; i++) // , ans
14    {
15        if () ...
16        else if () ...
17        ans += dfs(pos - 1, /* */, lead && i == 0, limit && i == a[pos])
18        //
19        /* i,
20        , state i */
21    }
22    // ,
23    if (!limit && !lead) dp[pos][state] = ans;
24    /* , lead, lead */
25    return ans;
26 }
27 }
28 ll solve(ll x)
29 {
30     int pos = 0;
31     do //
32         a[pos++] = x % 10;
33     while (x /= 10);
34     return dfs(pos - 1 /* */, /* */, true, true);
35     // , 0
36 }

```

7 Others

7.1 Matrix

7.1.1 Matrix FastPow

```

1  typedef vector<ll> vec;
2  typedef vector<vec> mat;
3  mat mul(mat& A, mat& B)
4  {
5      mat C(A.size(), vec(B[0].size()));
6      for (int i = 0; i < A.size(); i++)
7          for (int k = 0; k < B.size(); k++)
8              if (A[i][k]) //
9                  for (int j = 0; j < B[0].size(); j++)
10                     C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % mod;
11     return C;
12 }
13 mat Pow(mat A, ll n)
14 {
15     mat B(A.size(), vec(A.size()));
16     for (int i = 0; i < A.size(); i++) B[i][i] = 1;
17     for (; n >= 1; A = mul(A, A))
18         if (n & 1) B = mul(B, A);
19     return B;
20 }

```

7.1.2 Gauss Elimination

```

1  void gauss()
2  {
3      int now = 1, to;
4      double t;
5      for (int i = 1; i <= n; i++, now++)
6      {
7          /*for (to = now; !a[to][i] && to <= n; to++);
8          // ,
9          if (to != now)
10             for (int j = 1; j <= n + 1; j++)
11                 swap(a[to][j], a[now][j]);*/
12         t = a[now][i];
13         for (int j = 1; j <= n + 1; j++) a[now][j] /= t;
14         for (int j = 1; j <= n; j++)
15             if (j != now)
16             {
17                 t = a[j][i];
18                 for (int k = 1; k <= n + 1; k++) a[j][k] -= t * a[now][k];
19             }
20     }
21 }

```

7.2 Tricks

7.2.1 Stack-Overflow

```

1  #pragma comment(linker, "/STACK:1024000000,1024000000")

```

7.2.2 Fast-Scanner

```

1  template <class T>
2  inline bool scan_d(T &ret){
3      char c;
4      int sgn;
5      if (c = getchar(), c == EOF) return 0; //EOF
6      while (c != '-' && (c < '0' || c > '9')) c = getchar();
7      sgn = (c == '-') ? -1 : 1;
8      ret = (c == '-') ? 0 : (c - '0');
9      while (c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c - '0');
10     ret *= sgn;
11     return 1;
12 }
13 inline void out(int x){
14     if(x<0){
15         putchar('-');
16         x=-x;
17     }
18     if (x > 9) out(x / 10);
19     putchar(x % 10 + '0');
20 }

```

7.2.3 Strok-Sscanf

```

1  // get some integers in a line
2  gets(buf);
3  int v;
4  char *p = strtok(buf, " ");
5  while (p){
6      sscanf(p, "%d", &v);
7      p = strtok(NULL, " ");
8  }

```

7.3 Mo Algorithm

```

1  //cf 671 E
2  #include <bits/stdc++.h>
3  using namespace std;
4  typedef long long ll;
5  const int MAXN=1<<20;
6  struct node{
7      int l,r,id;
8  }Q[MAXN];
9  int n,m,k;
10 int block;
11 int a[MAXN];
12 int pre[MAXN];
13 ll cnt[MAXN];
14 ll ANS,ans[MAXN];
15 bool cmp(node x,node y){
16     if(x.l/block==y.l/block)return x.r<y.r;
17     else return x.l/block<y.l/block;
18 }
19 void add(int x){
20     ANS+=cnt[pre[x]^k];
21     cnt[pre[x]]++;

```

```

22 }
23 void del(int x){
24     cnt[pre[x]]--;
25     ANS-=cnt[pre[x]^k];
26 }
27 int main(){
28     scanf("%d%d%d",&n,&m,&k);
29     block=(int)sqrt(n);
30     pre[0]=0;
31     for(int i=1;i<=n;i++){
32         scanf("%d",&a[i]);
33         pre[i]=a[i]^pre[i-1];
34     }
35     for(int i=1;i<=m;i++){
36         scanf("%d%d",&Q[i].l,&Q[i].r);
37         Q[i].id=i;
38     }
39     sort(Q+1,Q+1+m,cmp);
40     ANS=0;
41     memset(cnt,0,sizeof(cnt));
42     cnt[0]=1;
43     int L=1,R=0;
44     for(int i=1;i<=m;i++){
45         while(L>Q[i].l){L--;add(L-1);};
46         while(L<Q[i].l){del(L-1);L++;}
47         while(R<Q[i].r){R++;add(R);};
48         while(R>Q[i].r){del(R);R--};
49         ans[Q[i].id]=ANS;
50     }
51     for(int i=1;i<=m;i++){
52         printf("%lld\n",ans[i]);
53     }
54     return 0;
55 }

```

7.4 BigNum

7.4.1 High-precision

```
1 java
```

7.5 VIM

```

1 syntax on
2 set nu
3 set tabstop=4
4 set shiftwidth=4
5 set cin
6 set mouse=a
7
8 map<F2> :call SetTitle(<CR>
9 func SetTitle()
10 let l = 0
11 let l = l + 1 | call setline(l,'#include <algorithm>')
12 let l = l + 1 | call setline(l,'#include <iostream>')
13 let l = l + 1 | call setline(l,'#include <cstring>')
14 let l = l + 1 | call setline(l,'#include <string>')
15 let l = l + 1 | call setline(l,'#include <cstdio>')

```

```

16 let l = l + 1 | call setline(l,'#include    <vector>')
17 let l = l + 1 | call setline(l,'#include    <cstdio>')
18 let l = l + 1 | call setline(l,'#include    <vector>')
19 let l = l + 1 | call setline(l,'#include    <stack>')
20 let l = l + 1 | call setline(l,'#include    <queue>')
21 let l = l + 1 | call setline(l,'#include    <cmath>')
22 let l = l + 1 | call setline(l,'#include    <set>')
23 let l = l + 1 | call setline(l,'#include    <map>')
24 let l = l + 1 | call setline(l,'using namespace std;')
25 let l = l + 1 | call setline(l,'#define rep(i,a,b) for(int i=a;i<=b;i++)')
26 let l = l + 1 | call setline(l,'#define per(i,a,b) for(int i=a;i>=b;i--)')
27 let l = l + 1 | call setline(l,'#define clr(a,x) memset(a,x,sizeof(a))')
28 let l = l + 1 | call setline(l,'#define pb push_back')
29 let l = l + 1 | call setline(l,'#define mp make_pair')
30 let l = l + 1 | call setline(l,'#define all(x) (x).begin(),(x).end()')
31 let l = l + 1 | call setline(l,'#define fi first')
32 let l = l + 1 | call setline(l,'#define se second')
33 let l = l + 1 | call setline(l,'#define SZ(x) ((int)(x).size())')
34 let l = l + 1 | call setline(l,'typedef unsigned long long ull;')
35 let l = l + 1 | call setline(l,'typedef long long ll;')
36 let l = l + 1 | call setline(l,'typedef vector<int> vi;')
37 let l = l + 1 | call setline(l,'typedef pair<int,int> pii;')
38 let l = l + 1 | call setline(l,'/*****head*****/')
39 let l = l + 1 | call setline(l,'int work(){')
40 let l = l + 1 | call setline(l,'')
41 let l = l + 1 | call setline(l,'    return 0;')
42 let l = l + 1 | call setline(l,'}')
43 let l = l + 1 | call setline(l,'int main(){')
44 let l = l + 1 | call setline(l,'#ifdef superkunn')
45 let l = l + 1 | call setline(l,'    freopen("input.txt","rt",stdin);')
46 let l = l + 1 | call setline(l,'#endif')
47 let l = l + 1 | call setline(l,'    work();')
48 let l = l + 1 | call setline(l,'    return 0;')
49 let l = l + 1 | call setline(l,'}')
50 endfunc

```

7.6 BASH

7.6.1 a.sh