

ACM/ICPC Template Manaual

QUST

hxk

August 12, 2018

Contents

0	Incl	ude 1
1	Mat	ch 2
	1.1	Prime
		1.1.1 Eratosthenes Sieve
		1.1.2 Eular Sieve
		1.1.3 Prime Factorization
		1.1.4 Miller Rabin
		1.1.5 Segment Sieve
	1.2	Eular phi
	1.2	1.2.1 Eular
		1.2.2 Sieve
	1.3	Basic Number Theory
	1.5	1.3.1 Extended Euclidean
	1 4	1.3.3 Multiplicative Inverse Modulo
	1.4	Modulo Linear Equation
		1.4.1 Chinese Remainder Theory
	1 -	1.4.2 ExCRT
	1.5	Combinatorics
		1.5.1 Combination
		1.5.2 Lucas
		1.5.3 Big Combination
		1.5.4 Polya
	1.6	Fast Power
	1.7	Mobius Inversion
		1.7.1 Mobius
		1.7.2 Number of Coprime-pair
		1.7.3 VisibleTrees
	1.8	Fast Transformation
		1.8.1 FFT
		1.8.2 NTT
		1.8.3 FWT
	1.9	Numerical Integration
		1.9.1 Adaptive Simpson's Rule
		1.9.2 Berlekamp-Massey
	1.10	Others
		Formula
2	Stri	ng Processing 16
		KMP
	2.2	ExtendKMP
	2.3	Manacher
	2.4	Aho-Corasick Automaton
	2.5	Suffix Array
	2.6	Suffix Automation
	2.7	HashString
3	Data	a Structure 22
	3.1	Binary Indexed Tree
		3.1.1 poj3468
	3.2	Segment Tree
	٠.٢	3.2.1 Single-point Update
		3.2.2 Interval Update
	2 2	1
	3.3	
	3.4	9
	3.5	Sparse Table
	3.6	Heavy-Light Decomposition

ACM/ICPC Template Manaual by hxk

	3.7	Link-Cut	Tree			 	 	 	 	 	 	 		 28
4	Gra	ph Theor	\mathbf{y}											30
	4.1	Union-Fir	d Set			 	 	 	 	 	 	 		 30
	4.2	Minimal S	Spanning Tree.			 	 	 	 	 	 	 		 30
			uskal											30
	4.3		Path											31
	1.0		kstra											31
														32
		_	fa											
	4.4	-												33
	4.5													35
			rjan											35
		4.5.2 LC	Armq			 	 	 	 	 	 	 		 35
	4.6	Depth-Fir	st Traversal			 	 	 	 	 	 	 		 37
		4.6.1 Bio	connected-Comp	onent		 	 	 	 	 	 	 		 37
			ongly Connecte											38
			AT		_									39
	4.7		h											41
	1.1		eury											41
	10		•											
	4.8	_	Graph Matchin	_										41
			ngry											41
			ngry(Multiple)											42
			hn-Munkres .											43
	4.9	Network 1	Flow			 	 	 	 	 	 	 		 44
		4.9.1 Ed	mondKarp			 	 	 	 	 	 	 		 45
		4.9.2 Di	nic			 	 	 	 	 	 	 		 46
		4.9.3 IS	AP			 	 	 	 	 	 	 		 47
		4.9.4 Mi	nCost MaxFlow	7		 	 	 	 	 				 49
_	~										 • •	 		
5		-	al Geometry											51
5	5.1	Basic Fun	ction			 	 	 	 	 	 	 	•	51
5		Basic Fun Position	$\begin{array}{cccccccccccccccccccccccccccccccccccc$			 	 • • •	 	 	 	 	 		 51 51
5	5.1	Basic Fun Position	ction			 	 • • •	 	 	 	 	 		 51
5	5.1	Basic Fun Position 5.2.1 Po	$\begin{array}{cccccccccccccccccccccccccccccccccccc$			 	 	 	 	 	 	 	•	 51 51
5	5.1	Basic Fun Position 5.2.1 Po 5.2.2 Lin	ction			 	 	 	 	 	 	 	•	 51 51 51
5	5.1	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Seg	ction			 	 • • •		 	 	 	 	• •	 51 51 51 51
5	5.1	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Sep 5.2.4 Lin	ction			 			 	 	 	 	• •	 51 51 51 51 52
5	5.1	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Seg 5.2.4 Lin 5.2.5 Po	ction			 			 	 	 	 		51 51 51 51 52 52 52
5	5.1	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Seg 5.2.4 Lin 5.2.5 Po 5.2.6 Po	ction			 			 	 	 	 		51 51 51 51 52 52 52 52
5	5.1 5.2	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Sep 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po	ction			 				 	 	 		51 51 51 52 52 52 52 52 52
5	5.1	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Seg 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon	ction							 				51 51 51 52 52 52 52 52 52 53
5	5.1 5.2	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Seg 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar	ction											51 51 51 52 52 52 52 52 52 53
5	5.1 5.2	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Seg 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po	ction											51 51 51 52 52 52 52 52 53 53
5	5.1 5.2	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Seg 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po	ction											51 51 51 51 52 52 52 52 52 53 53 53
5	5.1 5.2 5.3	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Se 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju	int-Point											51 51 51 52 52 52 52 52 53 53 53 53 54
5	5.1 5.2	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Se 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po	int-Point											51 51 51 52 52 52 52 52 53 53 53 53 54 54
5	5.1 5.2 5.3	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Se 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po	int-Point											51 51 51 51 52 52 52 52 52 53 53 53 53 54 54
5	5.1 5.2 5.3	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Seg 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po 5.4.1 Or	int-Point											51 51 51 52 52 52 52 52 53 53 53 53 54 54
5	5.1 5.2 5.3	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Seg 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po 5.4.1 Or 5.4.2 Or	ction											51 51 51 51 52 52 52 52 52 53 53 53 53 54 54
5	5.1 5.2 5.3	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Seg 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po 5.4.1 Or 5.4.2 Or 5.4.3 Ins	ction											51 51 51 52 52 52 52 52 53 53 53 53 54 54 54 54
5	5.15.25.3	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Sep 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po 5.4.1 Or 5.4.2 Or 5.4.3 Ins Circle	ction											51 51 51 52 52 52 52 52 53 53 53 54 54 54 54 54
	5.15.25.35.45.5	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Sep 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po 5.4.1 Or 5.4.2 Or 5.4.3 Ins Circle .	ction											51 51 51 52 52 52 52 53 53 53 54 54 54 54 54 54 54
	5.15.25.35.45.5Dyr	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Se 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po 5.4.1 Or 5.4.2 Or 5.4.3 Ins Circle 5.5.1 Circle Cambric Pro-	ction											51 51 51 52 52 52 52 52 53 53 53 54 54 54 54 54 54 54
	5.15.25.35.45.5	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Se 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po 5.4.1 Or 5.4.2 Or 5.4.2 Or 5.4.3 Ins Circle 5.5.1 Circle Subsequen	ction											51 51 51 52 52 52 52 52 53 53 53 54 54 54 54 54 54 54 54
	5.15.25.35.45.5Dyr	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Se 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po 5.4.1 Or 5.4.2 Or 5.4.2 Or 5.4.3 Ins Circle . 5.5.1 Circle Subsequer 6.1.1 Ma	ction											51 51 51 52 52 52 52 52 52 53 53 53 53 54 54 54 54 54 54 54 55 55
	5.15.25.35.45.5Dyr	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Se 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po 5.4.1 Or 5.4.2 Or 5.4.2 Or 5.4.3 Ins Circle . 5.5.1 Circle Subsequer 6.1.1 Ma	ction											51 51 51 52 52 52 52 52 53 53 53 54 54 54 54 54 54 54 54
	5.15.25.35.45.5Dyr	Basic Fun Position 5.2.1 Po 5.2.2 Lin 5.2.3 Se 5.2.4 Lin 5.2.5 Po 5.2.6 Po 5.2.7 Po Polygon 5.3.1 Ar 5.3.2 Po 5.3.3 Po 5.3.4 Ju Integer Po 5.4.1 Or 5.4.2 Or 5.4.2 Or 5.4.3 Ins Circle 5.5.1 Circle Subsequer 6.1.1 Ma 6.1.2 Lo	ction											51 51 51 52 52 52 52 52 52 53 53 53 53 54 54 54 54 54 54 54 55 55

ACM/ICPC Template Manaual by hxk

7 Oth		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
	Mo Algorithm	
7.4	BigNum	
	7.4.1 High-precision	
7.5	VIM	59
7.6	BASH	
	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
               <cstdio>
9 #include
               <vector>
10 #include
                <stack>
11 #include
                <queue>
12 #include
                <cmath>
13 #include
                  <set>
14 #include
                  <map>
15 using namespace std;
#define rep(i,a,b) for(int i=a;i<=b;i++)</pre>
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;
  /*********head***********/
29
30 int work(){
31
       return 0;
32
   }
33
   int main(){
34
35
   #ifdef superkunn
       freopen("input.txt","rt",stdin);
36
   #endif
37
38
       work();
       return 0;
39
40 }
```

1 Math

1.1 Prime

1.1.1 Eratosthenes Sieve

```
O(n \log \log n) maxn
   notprime[i] = 0/1 \ 0  1
1 const int maxn = "Edit";
   bool notprime[maxn] = {1, 1};
                                     // 0 && 1
   void GetPrime()
3
   {
4
        for (int i = 2; i < maxn; i++)</pre>
5
            if (!notprime[i] && i <= maxn / i) // √n</pre>
6
                for (int j = i * i; j < maxn; j += i)
7
                    notprime[j] = 1;
8
9
   }
   1.1.2 Eular Sieve
   O(n)
           phi[] prime[]
                           tot
     \mathbf{n}
1 const int maxn = "Edit";
2 bool vis[maxn];
3 int tot, phi[maxn], prime[maxn];
4 void CalPhi(int n)
5
        clr(vis, 0);
6
7
        phi[1] = 1;
8
        tot = 0;
9
        for (int i = 2; i < n; i++)
10
            if (!vis[i])
11
                prime[tot++] = i, phi[i] = i - 1;
12
            for (int j = 0; j < tot; j++)
13
14
                if (i * prime[j] > n) break;
15
                vis[i * prime[j]] = 1;
16
                if (i % prime[j] == 0)
17
18
                     phi[i * prime[j]] = phi[i] * prime[j];
19
20
21
                }
22
                else
                     phi[i * prime[j]] = phi[i] * (prime[j] - 1);
```

1.1.3 Prime Factorization

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

}

}

 $\frac{23}{24}$

25 26 }

```
ll fact[100][2];
   int getFactors(ll x)
2
3
        int cnt = 0;
4
        for (int i = 0; prime[i] <= x / prime[i]; i++)</pre>
5
6
            fact[cnt][1] = 0;
7
            if (x % prime[i] == 0)
8
9
                fact[cnt][0] = prime[i];
10
                while (x % prime[i] == 0) fact[cnt][1]++, x /= prime[i];
11
12
                cnt++;
            }
13
        }
14
        if (x != 1) fact[cnt][0] = x, fact[cnt++][1] = 1;
15
        return cnt;
16
17
   }
   1.1.4 Miller Rabin
              2^{63}
   O(s \log n)
   bool Miller_Rabin(ll n, int s)
2
   {
        if (n == 2) return 1;
3
        if (n < 2 | | !(n & 1)) return 0;
4
        int t = 0;
5
        ll x, y, u = n - 1;
6
        while ((u \& 1) == 0) t++, u >>= 1;
7
        for (int i = 0; i < s; i++)
8
9
10
            ll\ a = rand() \% (n - 1) + 1;
            11 x = Pow(a, u, n);
11
            for (int j = 0; j < t; j++)
12
13
                ll y = Mul(x, x, n);
14
                if (y == 1 \&\& x != 1 \&\& x != n - 1) return 0;
15
16
                x = y;
17
            if (x != 1) return 0;
18
19
20
        return 1;
21 }
   1.1.5 Segment Sieve
      [a,b)
   is_prime[i-a]=true i
   a < b \le 10^{12}, b - a \le 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;
```

```
for (ll i = 0; i * i < b; ++i)
7
            is_prime_small[i] = true;
8
       for (ll i = 0; i < b - a; ++i)
9
            is_prime[i] = true;
10
       for (ll i = 2; i * i < b; ++i)
11
            if (is_prime_small[i])
12
13
                for (ll j = 2 * i; j * j < b; j += i)
14
                    is_prime_small[j] = false;
15
                for (ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
16
                    is_prime[j - a] = false;
17
18
       for (ll i = 0; i < b - a; ++i)
19
20
           if (is_prime[i]) prime[tot++] = i + a;
21
       return tot;
   }
22
   1.2 Eular phi
   1.2.1 Eular
   ll Euler(ll n)
1
2
3
       ll rt = n;
       for (int i = 2; i * i <= n; i++)
4
           if (n \% i == 0)
5
6
7
                rt -= rt / i;
8
                while (n \% i == 0) n /= i;
9
       if (n > 1) rt -= rt / n;
10
       return rt;
11
12 }
   1.2.2 Sieve
1 const int N = "Edit";
   int phi[N] = \{0, 1\};
   void CalEuler()
3
   {
4
       for (int i = 2; i < N; i++)
5
            if (!phi[i])
6
                for (int j = i; j < N; j += i)
7
8
                    if (!phi[j]) phi[j] = j;
9
                    phi[j] = phi[j] / i * (i - 1);
10
                }
11
12 }
   1.3 Basic Number Theory
   1.3.1 Extended Euclidean
  ll exgcd(ll a, ll b, ll &x, ll &y)
1
2
   {
3
       if (b) d = exgcd(b, a \% b, y, x), y -= x * (a / b);
```

```
else x = 1, y = 0;
       return d;
7 }
   1.3.2 ax+by=c
      : X = x + k * dx, Y = y - k * dy
1 #define Mod(a, b) (((a) % (b) + (b)) % (b))
   bool solve(ll a, ll b, ll c, ll& x, ll& y, ll& dx, ll& dy)
3
       if (a == 0 \&\& b == 0) return 0;
4
5
       11 x0, y0;
6
       11 d = exgcd(a, b, x0, y0);
       if (c % d != 0) return 0;
7
       dx = b / d, dy = a / d;
8
       x = Mod(x0 * c / d, dx);
9
       y = (c - a * x) / b;
10
       // y = Mod(y0 * c / d, dy); x = (c - b * y) / a;
11
12
       return 1;
13 }
   1.3.3 Multiplicative Inverse Modulo
     exgcd a m , gcd(a, m) == 1.
1 ll inv(ll a, ll m)
2
   {
3
       11 x, y;
       ll d = exgcd(a, m, x, y);
       return d == 1 ? (x + m) % m : -1;
5
6 }
   a 
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
   X = r_i(modm_i); m_i
       X = re + k * mo
1 void crt(ll r[], ll m[], ll n, ll &re, ll &mo)
2
       mo = 1, re = 0;
3
       for (int i = 0; i < n; i++) mo *= m[i];</pre>
4
       for (int i = 0; i < n; i++)
5
6
           ll x, y, tm = mo / m[i];
7
           ll d = exgcd(tm, m[i], x, y);
8
           re = (re + tm * x * r[i]) % mo;
```

```
}
10
        re = (re + mo) \% mo;
11
   }
12
   1.4.2 ExCRT
   X = r_i(modm_i); m_i
       X = re + k * mo;
   bool excrt(ll r[], ll m[], ll n, ll &re, ll &mo)
1
2
3
        11 x, y;
        mo = m[0], re = r[0];
4
        for (int i = 1; i < n; i++)
5
6
7
            ll d = exgcd(mo, m[i], x, y);
            if ((r[i] - re) % d != 0) return 0;
8
            x = (r[i] - re) / d * x % (m[i] / d);
9
            re += x * mo;
10
            mo = mo / d * m[i];
11
            re %= mo;
12
13
        re = (re + mo) \% mo;
14
15
        return 1;
16 }
         Combinatorics
   1.5.1 Combination
   0 \leq m \leq n \leq 1000
   const int maxn = 1010;
1
   11 C[maxn][maxn];
2
  void CalComb()
3
   {
4
        C[0][0] = 1;
5
6
        for (int i = 1; i < maxn; i++)
7
            C[i][0] = 1;
8
            for (int j = 1; j \leftarrow i; j++) C[i][j] = (C[i-1][j-1] + C[i-1][j]) % mod;
9
10
   }
11
   0 \le m \le n \le 10^5, p
   const int maxn = 100010;
  ll f[maxn];
  ll inv[maxn]; //
   void CalFact()
4
5
        f[0] = 1;
6
7
        for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
        inv[maxn - 1] = Pow(f[maxn - 1], p - 2, p);
8
        for (int i = maxn - 2; \sim i; i--) inv[i] = inv[i + 1] * (i + 1) % p;
9
10
   ll C(int n, int m) { return f[n] * inv[m] % p * inv[n - m] % p; }
```

1.5.2 Lucas

```
1 \le n, m \le 1000000000, 1 
1 const int maxp = 100010;
2 11 f[maxn];
   ll inv[maxn]; //
3
   void CalFact()
5
6
        f[0] = 1;
        for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
7
        inv[maxn - 1] = Pow(f[maxn - 1], p - 2, p);
8
        for (int i = maxn - 2; \sim i; i--) inv[i] = inv[i + 1] * (i + 1) % p;
9
10 }
11 ll Lucas(ll n, ll m, ll p)
12 {
        ll ret = 1;
13
        while (n && m)
14
15
            ll a = n \% p, b = m \% p;
16
            if (a < b) return 0;
17
            ret = ret * f[a] % p * inv[b] % p * inv[a - b] % p;
18
19
            n \neq p, m \neq p;
20
21
        return ret;
22 }
   1.5.3 Big Combination
   0 \le n \le 10^9, 0 \le m \le 10^4, 1 \le k \le 10^9 + 7
1 vector<int> v;
   int dp[110];
3 ll Cal(int l, int r, int k, int dis)
   {
4
        ll res = 1;
5
        for (int i = 1; i <= r; i++)</pre>
6
7
8
            int t = i;
9
            for (int j = 0; j < v.size(); j++)</pre>
10
11
                int y = v[j];
12
                while (t % y == 0) dp[j] += dis, t /= y;
13
            res = res * (ll)t % k;
14
15
16
        return res;
17
   11 Comb(int n, int m, int k)
19
   {
        clr(dp, 0);
20
        v.clear();
21
22
        int tmp = k;
        for (int i = 2; i * i <= tmp; i++)</pre>
23
            if (tmp \% i == 0)
24
25
            {
26
                int num = 0;
27
                while (tmp % i == 0) tmp /= i, num++;
```

```
v.pb(i);
28
29
         if (tmp != 1) v.pb(tmp);
30
         ll ans = Cal(n - m + 1, n, k, 1);
31
         for (int j = 0; j < v.size(); j++) ans = ans * Pow(v[j], dp[j], k) % k;
32
         ans = ans * inv(Cal(2, m, k, -1), k) % k;
33
34
         return ans;
35
   }
    1.5.4 Polya
                     gcd(i, n)
    N*N\atop,\frac{m^8+17m^4+6m^2}{24},\frac{c^{n^2+3}}{m^4+11m^2}+2c^{\frac{n^2+1}{2}}+2c^{n\frac{n+1}{2}}+2c^{\frac{n(n+1)}{2}}
1 // n c
   ll solve(int c, int n)
3
         if (n == 0) return 0;
4
         11 \text{ ans} = 0;
5
         for (int i = 1; i \le n; i++) ans += Pow(c, __gcd(i, n));
6
         if (n & 1) ans += n * Pow(c, n + 1 >> 1);
else ans += n / 2 * (1 + c) * Pow(c, n >> 1);
7
9
         return ans / n / 2;
10 }
    1.6 Fast Power
   ll Mul(ll a, ll b, ll mod)
1
2
         11 t = 0;
3
         for (; b; b >>= 1, a = (a << 1) \% mod)
4
             if (b \& 1) t = (t + a) \% mod;
5
         return t;
6
7
8
   ll Pow(ll a, ll n, ll mod)
9
    {
10
         ll t = 1;
         for (; n; n >>= 1, a = (a * a % mod))
11
              if (n \& 1) t = (t * a % mod);
12
13
         return t;
14 }
         Mobius Inversion
    1.7.1 Mobius
    F(n) = \sum_{d|n} f(d) \Rightarrow f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})
    F(n) = \sum_{n|d} f(d) \Rightarrow f(n) = \sum_{n|d} \mu(\frac{d}{n}) F(d)
1 ll ans;
2 const int maxn = "Edit";
int n, x, prime[maxn], tot, mu[maxn];
4 bool check[maxn];
5 void calmu()
```

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

1.7.2 Number of Coprime-pair

```
n \quad (n \le 100000), \quad n
   ll solve()
1
2
   {
        int b[100005];
3
        11 \text{ _max}, \text{ ans } = 0;
4
        clr(b, 0);
5
        for (int i = 0; i < n; i++)
6
7
8
             scanf("%d", &x);
9
             if (x > _max) _max = x;
             b[x]++;
10
11
        for (int i = 1; i <= _max; i++)
12
13
             int cnt = 0;
14
             for (ll j = i; j \le \max; j += i) cnt += b[j];
15
             ans += 1LL * mu[i] * cnt * cnt;
16
17
        return (ans - b[1]) / 2;
18
   }
19
```

1.7.3 VisibleTrees

```
gcd(x,y) = 1 , x ≤ n,y ≤ 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  }</pre>
```

1.8 Fast Transformation

1.8.1 FFT

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

1.8.2 NTT

```
.G P G^{\frac{P-1}{n}} w_n = e^{\frac{2i\pi}{n}}
                                       P G 1.11
   const int mod = 119 << 23 | 1;</pre>
   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
  void change(int y[], int len)
8
9
10
        for (int i = 1, j = len / 2; i < len - 1; i++)
11
            if (i < j) swap(y[i], y[j]);</pre>
12
            int k = len / 2;
13
            while (j >= k) j -= k, k /= 2;
14
            if (j < k) j += k;
15
16
17
   }
18
   void ntt(int y[], int len, int on)
19
        change(y, len);
20
21
        for (int h = 2, id = 1; h <= len; h <<= 1, id++)
22
            for (int j = 0; j < len; <math>j += h)
23
24
25
                int w = 1;
                for (int k = j; k < j + h / 2; k++)
26
27
                     int u = y[k] \% mod;
28
                     int t = 1LL * w * (y[k + h / 2] % mod) % mod;
29
                     y[k] = (u + t) \% \mod, y[k + h / 2] = ((u - t) \% \mod + \mod) \% \mod;
30
                     w = 1LL * w * wn[id] % mod;
31
32
                }
            }
33
34
        if (on == -1)
35
36
37
            int inv = Pow(len, mod - 2, mod);
38
            for (int i = 1; i < len / 2; i++) swap(y[i], y[len - i]);
39
            for (int i = 0; i < len; i++) y[i] = 1LL * y[i] * inv % mod;
40
41
   }
42
   1.8.3 FWT
   void fwt(int f[], int m)
        int n = __builtin_ctz(m);
3
        for (int i = 0; i < n; ++i)
4
            for (int j = 0; j < m; ++j)
5
6
                if (j & (1 << i))
7
                 {
                     int l = f[j \land (1 << i)], r = f[j];
8
                     f[j \land (1 << i)] = l + r, f[j] = l - r;
9
```

```
// or: f[j] += f[j \land (1 << i)];
10
                      // and: f[j \land (1 << i)] += f[j];
11
12
13
   void ifwt(int f[], int m)
14
   {
15
        int n = __builtin_ctz(m);
16
        for (int i = 0; i < n; ++i)
17
             for (int j = 0; j < m; ++j)
18
                 if (j & (1 << i))
19
20
21
                      int l = f[j \land (1 << i)], r = f[j];
                      f[j \land (1 \lessdot i)] = (l + r) / 2, f[j] = (l - r) / 2;
22
23
                     // or: f[j] -= f[j \land (1 << i)];
24
                     // and: f[j \land (1 << i)] -= f[j];
25
                 }
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)]
    |\ddot{S}(a,c) + S(c,b) - S(a,b)|/15 < \epsilon
1 double F(double x) {}
   double simpson(double a, double b)
2
   { // Simpson
3
        double c = a + (b - a) / 2;
4
        return (F(a) + 4 * F(c) + F(b)) * (b - a) / 6;
5
6
   double asr(double a, double b, double eps, double A)
7
   { // Simpson ( )
                        [a,b] Simpson A
        double c = a + (b - a) / 2;
9
        double L = simpson(a, c), R = simpson(c, b); if (fabs(L + R - A) \ll 15 * eps) return L + R + (L + R - A) / 15.0;
10
11
        return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
12
13
   double asr(double a, double b, double eps) { return asr(a, b, eps, simpson(a, b)); }
   1.9.2 Berlekamp-Massey
   const int N = 1 \ll 14;
   11 res[N], base[N], _c[N], _md[N];
   vector<int> Md;
3
   void mul(ll* a, ll* b, int k)
4
   {
5
        for (int i = 0; i < k + k; i++) _{c}[i] = 0;
6
        for (int i = 0; i < k; i++)
7
8
             if (a[i])
                 for (int j = 0; j < k; j++) _{c[i + j]} = (_{c[i + j]} + a[i] * b[j]) % mod;
9
10
        for (int i = k + k - 1; i >= k; i--)
11
             if (_c[i])
                 for (int j = 0; j < Md.size(); j++) _c[i - k + Md[j]] = (_c[i - k + Md[j]]
12
        - _c[i] * _md[Md[j]]) % mod;
        for (int i = 0; i < k; i++) a[i] = _c[i];
13
14 }
```

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

```
c.erase(c.begin());
73
           for (int i = 0; i < c.size(); i++) c[i] = (mod - c[i]) % mod;
74
           return solve(n, c, VI(a.begin(), a.begin() + c.size()));
75
76 }
     1.10 Others
    n , , m
    int josephus(int n, int m)
 2
 3
           int r = 0;
           for (int k = 1; k \le n; ++k) r = (r + m) \% k;
 4
           return r + 1;
 5
    }
 6
    n^n
    int leftmost(int n)
 1
 2
           double m = n * log10((double)n);
 3
           double g = m - (11)m;
 4
 5
           return (int)pow(10.0, g);
    }
 6
    n!
 1
    int count(ll n)
 2
           if (n == 1) return 1;
 3
           return (int)ceil(0.5 * log10(2 * M_PI * n) + n * log10(n) - n * log10(M_E));
 4
 5
     1.11 Formula
            : n = \prod_{i=1}^{k} p_i^{a_i},
        1.
             (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
                     n\varphi(n)/2
        3. gcd(n, i) = 1, gcd(n, n - i) = 1(1 \le i \le n)
            D(n) = (n-1)(D(n-2) + D(n-1)) = \sum_{i=2}^{n} \frac{(-1)^{k} n!}{k!} = \left[\frac{n!}{e} + 0.5\right]
        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}
            : n \qquad \pi(n), \lim_{n \to \infty} \pi(n) = \frac{n}{\ln n}
        9. : x N = log 10(n) + 1
            n! \approx \sqrt{2\pi n} (\frac{n}{2})^n
       10.
       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, ..., 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(Fib(m), Fib(n)) = Fib(gcd(m, n))
```

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

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

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

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

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

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

ггі			
$r \ 2^k + 1$	r	k	g
3	1	1	$\frac{g}{2}$
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

```
//MAXN
   int nxt[MAXN];
2
   void initkmp(char x[],int m){
3
        int i=0, j=nxt[0]=-1;
4
        while(i<m){</pre>
5
            while(j!=-1&&x[i]!=x[j])j=nxt[j];
6
7
            nxt[++i]=++j;
        }
8
   }
9
  //x:pa y:tx
10
   int kmp(char x[],int m,char y[],int n){
        int i,j,ans;
12
        i=j=ans=0;
13
14
        initkmp(x,m);
        while(i<n){</pre>
15
            while(j!=-1&&y[i]!=x[j])j=nxt[j];
16
            i++,j++;
17
            if(j>=m){}
18
19
                 ans++;
                 j=nxt[j];
20
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;
       while (j + 1 < m \&\& x[j] == x[j + 1]) j++;
9
       next[1] = j;
10
       int k = 1;
11
       for (int i = 2; i < m; i++)
12
13
            int p = next[k] + k - 1;
14
            int L = next[i - k];
15
            if (i + L 
16
                next[i] = L;
17
            else
18
            {
19
                j = max(0, p - i + 1);
20
21
                while (i + j < m \&\& x[i + j] == x[j]) j++;
22
                next[i] = j;
                k = i;
23
            }
24
       }
25
26 }
```

```
void ekmp(char x[], int m, char y[], int n)
27
   {
28
       pre_ekmp(x, m, next);
29
       int j = 0;
30
       while (j < n \&\& j < m \&\& x[j] == y[j]) j++;
31
       extend[0] = j;
32
       int k = 0;
33
       for (int i = 1; i < n; i++)
34
35
            int p = extend[k] + k - 1;
36
37
            int L = next[i - k];
38
            if (i + L 
                extend[i] = L;
39
            else
40
            {
41
                j = max(0, p - i + 1);
42
                while (i + j < n \& j < m \& y[i + j] == x[j]) j++;
43
                extend[i] = j, k = i;
44
            }
45
46
       }
   }
47
   2.3 Manacher
   const int MAXN=3e5;//more than two times
   char s[MAXN],str[MAXN];
```

```
3
   int len1,len2,p[MAXN];
4
   void init(){
        str[0]='$';
5
        str[1]='#';
6
7
        rep(i,0,len1-1){
            str[i*2+2]=s[i];
8
9
            str[i*2+3]='#';
10
        len2=len1*2+2;
11
        str[len2]='*';
12
13
   }
   void manacher(){
14
15
        int id=0, mx=0;
        rep(i,1,len2-1){
16
            if(mx>i)p[i]=min(p[2*id-i],mx-i);
17
            else p[i]=1;
18
            while(str[i+p[i]]==str[i-p[i]])p[i]++;
19
20
            if(p[i]+i>mx){
21
                mx=p[i]+i;
22
                id=i;
23
            }
        }
24
25
   }
   int work(){
        while(scanf("%s",s)!=E0F){
27
            len1=strlen(s);
28
            init();
29
            manacher();
30
            int ans=0;
31
            rep(i,0,len2-1){
32
                ans=max(ans,p[i]);
33
34
            }
```

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

51

{

```
int c = idx(s[i]);
52
                u = ch[u][c];
53
                int tmp = u;
54
                while (tmp != rt)
55
56
                     res += val[tmp];
57
                     val[tmp] = 0;
58
                     tmp = f[tmp];
59
                }
60
            }
61
62
            return res;
63
        }
  };
64
   2.5 Suffix Array
           , O(nlogn)
   //
   const int maxn = "Edit";
   char s[maxn];
4 int sa[maxn], t[maxn], t2[maxn], c[maxn], rank[maxn], height[maxn];
5 //n
              0 \sim 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;
        for (int i = 0; i < n; i++) c[x[i] = s[i]]++;
12
        for (int i = 1; i < m; i++) c[i] += c[i - 1];
13
        for (int i = n - 1; \sim i; i--) sa[--c[x[i]]] = i; for (int k = 1; k <= n; k <<= 1)
14
15
16
17
            // sa
18
            int p = 0;
            for (int i = n - k; i < n; i++) y[p++] = i;
19
            for (int i = 0; i < n; i++)
20
21
                if (sa[i] >= k) y[p++] = sa[i] - k;
22
23
            for (int i = 0; i < m; i++) c[i] = 0;
            for (int i = 0; i < n; i++) c[x[y[i]]]++;
24
            for (int i = 0; i < m; i++) c[i] += c[i - 1];
25
            for (int i = n - 1; \sim i; i--) sa[--c[x[y[i]]]] = y[i];
26
            // say
27
            swap(x, y);
28
            p = 1;
29
            x[sa[0]] = 0;
30
31
            for (int i = 1; i < n; i++)
                x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k] ? p
32
         -1:p++;
            if (p >= n) break; //
33
                                       ,sa
34
            m = p;
        }
35
36
        n--;
37
        int k = 0;
        for (int i = 0; i <= n; i++) rank[sa[i]] = i;</pre>
38
        for (int i = 0; i < n; i++)
39
40
            if (k) k--;
41
```

```
int j = sa[rank[i] - 1];
42
            while (s[i + k] == s[j + k]) k++;
43
            height[rank[i]] = k;
44
45
       }
   }
46
47
   int dp[maxn][30];
48
   void initrmq(int n)
49
50
        for (int i = 1; i <= n; i++)
51
            dp[i][0] = height[i];
52
53
       for (int j = 1; (1 << j) <= n; j++)
            for (int i = 1; i + (1 << j) - 1 <= n; i++)
54
                dp[i][j] = min(dp[i][j-1], dp[i+(1 << (j-1))][j-1]);
55
56
   int rmq(int 1, int r)
57
58
   {
       int k = 31 - \_builtin\_clz(r - l + 1);
59
60
       return min(dp[l][k], dp[r - (1 << k) + 1][k]);
61 }
62 int lcp(int a, int b)
63 { //
       a = rank[a], b = rank[b];
64
65
       if (a > b) swap(a, b);
66
       return rmq(a + 1, b);
67 }
   2.6 Suffix Automation
   const int maxn = "Edit";
2
   struct SAM
3
   {
4
       int len[maxn << 1], link[maxn << 1], ch[maxn << 1][26];</pre>
       int sz, rt, last;
5
       int newnode(int x = 0)
6
7
8
            len[sz] = x;
9
            link[sz] = -1;
10
            clr(ch[sz], -1);
            return sz++;
11
12
13
       void init() { sz = last = 0, rt = newnode(); }
       void extend(int c)
14
15
16
            int np = newnode(len[last] + 1);
17
            int p;
            for (p = last; \sim p \&\& ch[p][c] == -1; p = link[p]) ch[p][c] = np;
18
            if (p == -1)
19
                link[np] = rt;
20
21
            else
22
            {
                int q = ch[p][c];
23
                if (len[p] + 1 == len[q])
24
                    link[np] = q;
25
26
                else
27
                {
                    int nq = newnode(len[p] + 1);
28
29
                    memcpy(ch[nq], ch[q], sizeof(ch[q]));
```

```
link[nq] = link[q], link[q] = link[np] = nq;
30
                     for (; \sim p \& ch[p][c] == q; p = link[p]) ch[p][c] = nq;
31
                }
32
33
            last = np;
34
35
        int topcnt[maxn], topsam[maxn << 1];</pre>
36
37
        void sort()
        { //
38
            clr(topcnt, 0);
39
            for (int i = 0; i < sz; i++) topcnt[len[i]]++;</pre>
40
41
            for (int i = 0; i < maxn - 1; i++) topcnt[i + 1] += topcnt[i];
            for (int i = 0; i < sz; i++) topsam[--topcnt[len[i]]] = i;
42
        }
43
   };
44
   2.7 HashString
1 const ll B1=1e7+7;
   const ll B2=1e9+7;
3
   char pa[10004];
   char tx[1000006];
   int work(){
5
6
        int T;
        scanf("%d",&T);
7
        while(T--){
8
            scanf("%s%s",pa,tx);
9
            int pl=strlen(pa);
10
            int tl=strlen(tx);
11
12
            ll w=1;
            rep(i,1,pl)w=(w*B1)%B2;
13
            ll ph=0,th=0;
14
            rep(i,0,pl-1){
15
                ph=(ph*B1+pa[i])%B2;
16
                th=(th*B1+tx[i])%B2;
17
18
            }
19
            int ans=0;
            for(int i=0;i+pl<=tl;i++){</pre>
20
21
                if(ph==th)ans++;
22
                if(i+pl<tl)th=(th*B1+tx[i+pl]-tx[i]*w)%B2;</pre>
23
            printf("%d\n",ans);
24
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;
    struct BIT{
4
         int n,c[MAXN<<1];</pre>
5
         void init(int _n){n=_n;for(int i=0;i<=n;i++)c[i]=0;}</pre>
6
         void add(int i,int v){for(;i<=n;i+=i&-i)c[i]+=v;}</pre>
         int sum(int i){int s=0;for(;i>0;i-=i&-i)s+=c[i];return s;}
    }bit;
    3.1.1 poj3468
    a_i = \sum_{i=1}^x d_i
    \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){
         for(int i=x;i<=n;i+=i&(-i))c1[i]+=y,c2[i]+=1LL*x*y;</pre>
5
6
    11 sum(int x){
7
         ll ans(0);
8
         for(int i=x;i;i-=i&(-i))ans+=1LL*(x+1)*c1[i]-c2[i];
9
         return ans;
10
    }
11
12 char op[5];
    int work(){
13
         scanf("%d%d",&n,&q);
14
         int a1,a2;
15
16
         a1=0;
17
         rep(i,1,n)
              scanf("%d",&a2);
18
              add(i,a2-a1);
19
              a1=a2;
20
21
         while(q--){
22
              scanf("%s",op);
23
              if(op[0]=='Q'){
    scanf("%d%d%d",&x,&y,&z);
    printf("%lld\n",sum(y)-sum(x-1));
24
25
26
              }else{
27
                   scanf("%d%d%d",&x,&y,&z);
28
29
                   add(x,z);
30
                   add(y+1,-z);
31
              }
         }
32
         return 0;
33
    }
34
```

3.2 Segment Tree

```
1 #define lson rt<<1</pre>
2 #define rson rt<<1|1</pre>
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
   const int MAXN=5e4+5;
1
   int sum[MAXN<<2];</pre>
   void push_up(int rt){
3
        sum[rt]=sum[lson]+sum[rson];
4
5
   void build(int l,int r,int rt){
6
7
        if(l==r){
            scanf("%d",&sum[rt]);
8
            return;
9
10
        int mid;
11
12
        build(le);
13
        build(ri);
        push_up(rt);
14
15
   void update(int p,int v,int l,int r,int rt){
16
        if(l==r){
17
            sum[rt]+=v;
18
19
            return;
        }
20
21
        int mid;
        if(p<=m)update(p,v,le);</pre>
22
23
        else update(p,v,ri);
        push_up(rt);
24
25
   int query(int L,int R,int l,int r,int rt){
26
        if(L<=1&&r<=R){
27
            return sum[rt];
28
29
        int mid;
30
        int ret=0;
31
        if(L<=m)ret+=query(L,R,le);</pre>
32
33
        if(R>m)ret+=query(L,R,ri);
        return ret;
34
35 }
   3.2.2 Interval Update
1 const int MAXN=1e5+5;
2 11 lazy[MAXN<<2];</pre>
3 ll tree[MAXN<<2];</pre>
   void push_up(int rt){
4
        tree[rt]=tree[lson]+tree[rson];
5
   }
6
   void push_down(int rt,int m){
7
        ll w=lazy[rt];
8
9
        if(w){
```

```
lazy[lson]+=w;
10
            lazy[rson]+=w;
11
            tree[lson]+=w*(m-(m>>1));
12
            tree[rson]+=w*(m>>1);
13
14
            lazy[rt]=0;
        }
15
16
   }
   void build(int l,int r,int rt){
17
        lazy[rt]=0;
18
19
        if(l==r){
            scanf("%lld",&tree[rt]);
20
21
            return;
        }
22
        int mid;
23
        build(le);
24
        build(ri);
25
        push_up(rt);
26
   }
27
   void update(int L,int R,int v,int l,int r,int rt){
28
29
        if(L<=l&&r<=R){
            lazy[rt]+=v;
30
            tree[rt]+=111*v*(r-1+1);
31
            return;
32
33
34
        push_down(rt,r-l+1);
        int mid;
35
        if(L<=m)update(L,R,v,le);</pre>
36
        if(R>m)update(L,R,v,ri);
37
        push_up(rt);
38
39
   1l query(int L,int R,int l,int r,int rt){
40
        if(L<=1&&r<=R){
41
            return tree[rt];
42
43
        push_down(rt,r-l+1);
44
        int mid;
45
46
        ll ret=0;
        if(L<=m)ret+=query(L,R,le);</pre>
47
        if(R>m)ret+=query(L,R,ri);
48
49
        return ret;
50
   }
   3.3 Splay Tree
   #define key_value ch[ch[rt][1]][0]
   const int MAXN=1e5;
   struct Splay{
3
        int a[MAXN];//0 base
4
        int sz[MAXN], ch[MAXN][2], fa[MAXN];
5
        int key[MAXN], rev[MAXN];
6
7
        int rt, tot;
        int stk[MAXN],top;
8
9
        void push_up(int x){
            sz[x]=sz[ch[x][0]]+sz[ch[x][1]]+1;
10
11
        void push_down(int x){
12
13
            if(rev[x]){
14
                swap(ch[x][0],ch[x][1]);
```

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

```
int kth(int r,int k){
74
75
            push_down(r);
            int t=sz[ch[r][0]]+1;
76
77
            if(t==k)return r;
            return t>k?kth(ch[r][0],k):kth(ch[r][1],k-t);
78
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;
int n,m,cnt,x,y,k,root[MAXN],a[MAXN];
4 struct node{int 1,r,sum;}T[MAXN*40];
5 vi v;
   int getid(int x){return lower_bound(all(v),x)-v.begin()+1;}
   void update(int l,int r,int &x,int y,int pos){
7
8
       x=++cnt;
       T[x]=T[y];
9
10
       T[x].sum++;
       if(l==r)return;
11
12
       int mid=(l+r)>>1;
13
       if(mid>=pos)update(l,mid,T[x].l,T[y].l,pos);
       else update(mid+1,r,T[x].r,T[y].r,pos);
14
15
   }
   int query(int l,int r,int x,int y,int k){
16
       if(l==r)return l;
17
18
       int sum=T[T[y].l].sum-T[T[x].l].sum;
19
       int mid=(l+r)>>1;
       if(sum>=k)return query(l,mid,T[x].l,T[y].l,k);
20
21
       else return query(mid+1,r,T[x].r,T[y].r,k-sum);
   }
22
23
   int work(){
       scanf("%d%d",&n,&m);
24
       v.clear();
25
       rep(i,1,n)scanf("%d",&a[i]),v.pb(a[i]);
26
27
       sort(all(v)), v.erase(unique(all(v)), v.end());
28
       rep(i,1,n)update(1,n,root[i],root[i-1],qetid(a[i]));
29
       rep(i,1,m) scanf("%d%d%d",&x,&y,&k),printf("%d\n",v[query(1,n,root[x-1],root[y],k)
30
       -1]);
31
       return 0;
32 }
   3.5 Sparse Table
1 const int MAXN = "Edit";
  int mmax[MAXN][30], mmin[MAXN][30];
3 int a[MAXN], n, k;
4
   void init(){
        for (int i = 1; i \le n; i++) mmax[i][0] = mmin[i][0] = a[i];
5
6
       for (int j = 1; (1 << j) <= n; j++)
            for (int i = 1; i + (1 << j) - 1 <= n; i++){}
```

```
mmax[i][j] = max(mmax[i][j - 1], mmax[i + (1 << (j - 1))][j - 1]);
8
                mmin[i][j] = min(mmin[i][j - 1], mmin[i + (1 << (j - 1))][j - 1]);
9
10
11
   // op=0/1 return [l,r] max/min
12
   int rmq(int l, int r, int op){
13
        int k = 31 - \_builtin\_clz(r - l + 1);
14
        if (op == 0)
15
            return max(mmax[l][k], mmax[r - (1 << k) + 1][k]);
16
        return min(mmin[l][k], mmin[r - (1 \ll k) + 1][k]);
17
   }
18
   2D
   void init(){
1
        for (int i = 0; (1 << i) <= n; i++){
2
            for (int j = 0; (1 << j) <= m; j++){
3
                if (i == 0 \&\& j == 0) continue;
4
                for (int row = 1; row + (1 << i) - 1 <= n; row++)
5
6
                     for (int col = 1; col + (1 << j) - 1 <= m; col++)
7
                         if (i){
                             dp[row][col][i][j] = max(dp[row][col][i - 1][j];
8
                             dp[row + (1 << (i - 1))][col][i - 1][j]);
9
                         }else{
10
                             dp[row][col][i][j] = max(dp[row][col][i][j - 1];
11
                             dp[row][col + (1 << (j - 1))][i][j - 1]);
12
                         }
13
14
            }
        }
15
16
   }
   int rmq(int x1, int y1, int x2, int y2){
17
        int kx = 31 - \_builtin\_clz(x2 - x1 + 1);
        int ky = 31 - __builtin_clz(y2 - y1 + 1);
19
20
        int m1 = dp[x1][y1][kx][ky];
        int m2 = dp[x2 - (1 << kx) + 1][y1][kx][ky];
21
       int m3 = dp[x1][y2 - (1 << ky) + 1][kx][ky];
int m4 = dp[x2 - (1 << kx) + 1][y2 - (1 << ky) + 1][kx][ky];
22
23
        return max(max(m1, m2), max(m3, m4));
24
   }
25
   3.6 Heavy-Light Decomposition
   const int maxn = "Edit";
   struct HLD
2
   {
3
        int n, dfs_clock;
4
        int sz[maxn], top[maxn], son[maxn], dep[maxn], fa[maxn], id[maxn];
5
        vector<int> G[maxn];
6
        void init(int n)
7
8
        {
            this->n = n, clr(son, -1), dfs_clock = 0;
9
            for (int i = 0; i < n; i++) G[i].clear();
10
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
            dep[u] = d, fa[u] = p, sz[u] = 1;
15
16
            for (auto& v : G[u])
17
            {
```

```
if (v == p) continue;
18
                dfs(v, u, d + 1);
19
                sz[u] += sz[v];
20
                if (son[u] == -1 \mid | sz[v] > sz[son[u]]) son[u] = v;
21
22
23
24
        void link(int u, int t)
25
            top[u] = t, id[u] = ++dfs\_clock;
26
            if (son[u] == -1) return;
27
28
            link(son[u], t);
29
            for (auto& v : G[u])
                if (v != son[u] && v != fa[u]) link(v, v);
30
        }
31
        //
32
        int query_path(int u, int v)
33
34
            int ret = 0;
35
            while (top[u] != top[v])
36
37
                if (dep[top[u]] < dep[top[v]]) swap(u, v);</pre>
38
                ret += query(id[top[u]], id[u]);
39
                u = fa[top[u]];
40
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
        { //
            for (int i = 1; i <= n; i++) scanf("%d", val + i);</pre>
9
10
        inline bool isroot(int x) { return ch[fa[x]][0] != x && ch[fa[x]][1] != x; }
11
12
        inline bool get(int x) { return ch[fa[x]][1] == x; }
        void pushdown(int x)
13
        {
14
            if (!rev[x]) return;
15
            swap(ch[x][0], ch[x][1]);
16
            if (ch[x][0]) rev[ch[x][0]] ^= 1;
17
18
            if (ch[x][1]) rev[ch[x][1]] ^= 1;
            rev[x] ^= 1;
19
20
        }
        void pushup(int x) { sum[x] = val[x] + sum[ch[x][0]] + sum[ch[x][1]]; }
21
        void rotate(int x)
22
23
            int y = fa[x], z = fa[fa[x]], d = get(x);
24
```

if (!isroot(y)) ch[z][get(y)] = x;

25

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

```
const int MAXN=1e6+5;
struct DSU{
   int p[MAXN];
   void init(int n){for(int i=0;i<=n;i++)p[i]=i;}
   int findp(int x){return x==p[x]?x:p[x]=findp(p[x]);}
   void unite(int x,int y){x=findp(x);y=findp(y);if(x==y)return;p[y]=x;}
   bool same(int x,int y){return findp(x)==findp(y);}
}dsu;</pre>
```

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;
  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;}</pre>
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;}
        bool same(int x,int y){return findp(x)==findp(y);}
12
   }dsu;
13
   struct edge{int u,v,cost;}es[MAXE];
   bool cmp(const edge &x,const edge &y){return x.cost<y.cost;}</pre>
16
  int V,E;
17
   int kruskal(){
        sort(es,es+E,cmp);
18
        dsu.init(V);
19
20
        int res=0;
        for(int i=0;i<E;i++){</pre>
21
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
   int main(){
29
        while(~scanf("%d",&V)){
30
31
            E=0;
            for(int i=1;i<=V;i++){</pre>
32
                 for(int j=1;j<=V;j++){</pre>
33
34
                     int w;
                     scanf("%d",&w);
35
36
                     if(i==j)continue;
                     es[E].u=i;
37
                     es[E].v=j;
38
                     es[E].cost=w;
39
40
                     E++;
                }
41
```

```
42
            printf("%d\n",kruskal());
43
44
45
        return 0;
   }
46
   4.3
         Shortest Path
   4.3.1 Dijkstra
   //cf 610 A
  #include<bits/stdc++.h>
3 using namespace std;
   const int INF=1e9;
   const int MAXV=5e3+50;
   const int MAXE=1e5+50;
6
7
   int V;
   struct edge{int to,cost;};
8
   vector<edge> G[MAXV];
   typedef pair<int, int> P;
10
   int d[MAXV];
11
12
   void dijkstra(int s){
13
        priority_queue<P,vector<P>,greater<P> > que;
        fill(d,d+V+1,INF);
14
        d[s]=0;
15
16
        que.push(P(0,s));
17
        while(!que.empty()){
18
            P t=que.top();
19
            que.pop();
            int v=t.second;
20
            if(d[v]<t.first)continue;</pre>
21
            for(int i=0;i<G[v].size();i++){</pre>
22
                edge e=G[v][i];
23
24
                 if(d[e.to]>d[v]+e.cost){
                     d[e.to]=d[v]+e.cost;
25
                     que.push(P(d[e.to],e.to));
26
27
                }
            }
28
        }
29
30
   }
31
   int mat[405][405];
   int main(){
32
        int n,m;
33
        scanf("%d%d",&n,&m);
34
35
        V=n;
        for(int i=1;i<=m;i++){</pre>
36
            int u,v;
37
            scanf("%d%d",&u,&v);
38
            G[u].push_back(edge{v,1});
39
            G[v].push_back(edge{u,1});
40
            mat[u][v]=mat[v][u]=1;
41
        }
42
        dijkstra(1);
43
        int ans;
44
        if(d[n]==INF){
45
            printf("-1");
46
            return 0;
47
48
        ans=d[n];
49
```

```
for(int i=1;i<=n;i++)G[i].clear();</pre>
50
        for(int i=1;i<=n;i++){</pre>
51
            for(int j=1;j<=n;j++){
   if(i==j)continue;</pre>
52
53
                 if(mat[i][j]==0){
54
                     G[i].push_back(edge{j,1});
55
                 }
56
            }
57
        }
58
        dijkstra(1);
59
60
        if(d[n]==INF){
            printf("-1");
61
            return 0;
62
        }
63
        printf("%d",max(ans,d[n]));
64
        return 0;
65
   }
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];</pre>
16 int d[MAXV];
17 queue<int> que;
18 bool inq[MAXV];
19 int qtime[MAXV];
20
   void init(){
        tot=0:
21
22
        memset(head,-1,sizeof(head));
23
   void add_edge(int u,int v,int x){
24
        edge[tot].to=v;
25
26
        edge[tot].cost=x;
        edge[tot].next=head[u];
27
        head[u]=tot++;
28
29
   }
   bool spfa(int n){
30
        memset(d,-1,sizeof(d));
31
32
        d\Gamma17=0;
33
        while(!que.empty())que.pop();
        memset(inq,0,sizeof(inq));
34
        memset(qtime,0,sizeof(qtime));
35
        que.push(1);
36
        ina[1]=1;
37
38
        qtime[1]++;
```

```
while(!que.empty()){
39
            int u=que.front();
40
            que.pop();
41
            inq[u]=0;
42
            for(int i=head[u];i!=-1;i=edge[i].next){
43
                 int v=edge[i].to;
44
                 int w=edge[i].cost;
45
                 if(d[v]==-1|[d[u]+w<d[v]){
46
                     d[v]=d[u]+w;
47
                     if(!inq[v]){
48
49
                         inq[v]=1;
50
                         que.push(v);
                         qtime[v]++;
51
52
                         if(qtime[v]>n){
53
                              return false;
                         }
54
                     }
55
                 }
56
57
            }
58
        }
59
        return true;
   }
60
   int main(){
61
62
        int kase;
        scanf("%d",&kase);
63
        while(kase--){
64
            init();
65
            int n,m,w;
66
            scanf("%d%d%d",&n,&m,&w);
67
            while(m--){
68
                 int u,v,x;
scanf("%d%d%d",&u,&v,&x);
69
70
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);
                 add_edge(u,v,-x);
77
78
            if(!spfa(n)){
79
                 puts("YES");
80
            }else{
81
82
                 puts("N0");
83
            }
84
85
        return 0;
   }
86
         Topo Sort
    4.4
1 //cf 915D
2 const int MAXN=505;
3 const int MAXM=1e5+5;
4 int n,m;
5 int tot;
   int head[MAXN], cur[MAXN], idec[MAXN];
   struct Edge{
```

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

```
67
            }
68
       puts("NO");
69
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]; //
  bool vis[maxn];
                             //
   inline void init(int n)
7
8
9
       for (int i = 1; i <= n; i++)
10
            G[i].clear(), query[i].clear();
11
            par[i] = i, vis[i] = 0;
12
       }
13
14
   inline void add_edge(int u, int v) { G[u].pb(v); }
15
  inline void add_query(int id, int u, int v)
   {
17
       query[u].pb(mp(v, id));
18
       query[v].pb(mp(u, id));
19
20
  }
21 void tarjan(int u)
22
   {
23
       vis[u] = 1;
24
       for (auto& v : G[u])
25
            if (vis[v]) continue;
26
27
            tarjan(v);
28
            unite(u, v);
29
       for (auto& q : query[u])
30
31
            int &v = q.X, &id = q.Y;
32
            if (!vis[v]) continue;
33
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];
```

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

```
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];
   stack<PII> s;
6
   void init(int n)
7
        for (int i = 0; i < n; i++) G[i].clear();</pre>
8
9
   inline void add_edge(int u, int v) { G[u].pb(v), G[v].pb(u); }
10
   int dfs(int u, int fa)
12
   {
13
        int lowu = pre[u] = ++dfs_clock;
        int child = 0;
14
        for (auto& v : G[u])
15
16
            PII e = mp(u, v);
17
            if (!pre[v])
18
19
            {
20
                // v
21
                s.push(e);
22
                child++;
                int lowv = dfs(v, u);
23
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();
                         if (bccno[x.X] != bcc_cnt)
34
                             bcc[bcc\_cnt].pb(x.X), bcc[x.X] = bcc\_cnt;
35
                         if (bccno[x.Y] != bcc_cnt)
36
                             bcc[bcc\_cnt].pb(x.Y), bcc[x.Y] = bcc\_cnt;
37
38
                         if (x.X == u \&\& x.Y == v) break;
                     }
39
                }
40
41
            else if (pre[v] < pre[u] && v != fa)</pre>
42
43
44
                s.push(e);
                lowu = min(lowu, pre[v]); //
45
46
            }
47
        if (fa < 0 && child == 1) iscut[u] = 0;</pre>
48
49
        return lowu;
50
   void find_bcc(int n)
```

```
52 {
53
       clr(pre, 0), clr(iscut, 0), clr(bccno, 0);
54
55
       dfs_clock = bcc_cnt = 0;
56
       for (int i = 0; i < n; i++)
            if (!pre[i]) dfs(i, -1);
57
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;//
   const int MAXM = 5005;//
   struct Edge{
7
8
       int to,next;
9
   } edge[MAXM];
10 int head[MAXN],tot;
int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong 1~scc
12 int Index,top;
13 int scc;//
14 bool Instack[MAXN];
   void init(){
15
16
       tot = 0;
17
       memset(head,-1,sizeof(head));
18
   }
19
   void addedge(int u,int v){
       edge[tot].to = v;
20
21
       edge[tot].next = head[u];
22
       head[u] = tot++;
23
   }
24
   void Tarjan(int u){
       int v;
25
       Low[u] = DFN[u] = ++Index;
26
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;
            if( !DFN[v] ){
31
                Tarjan(v);
32
                if( Low[u] > Low[v] )Low[u] = Low[v];
33
34
            else if(Instack[v] && Low[u] > DFN[v])
35
                Low[u] = DFN[v];
36
37
       if(Low[u] == DFN[u]){
38
            scc++;
39
            do{
40
                V = Stack[--top];
41
                Instack[v] = false;
42
43
                Belong[v] = scc;
44
           while( v != u);
45
46
       }
47
   void solve(int N){
```

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

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

```
4.7 Eular Path
```

7 8

9

10

11

12 13

14

15

3

4

5

6

7

```
(
                            )
                    ( ,
                             ),
           G
           - G
           - G
                (
                      )
                          0 \ 2.
           G
           - G
           - G
           G
           - G
                   _{1,v}
                         1,
                                (u ,v )
           — u
           \mathbf{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;
   inline void init() { clr(G, 0), clr(deg, 0); }
inline void AddEdge(int u, int v) { deg[u]++, deg[v]++, G[u][v]++, G[v][u]++; }
   void Fleury(int s)
   {
        for (int i = 0; i < n; i++)
            if (G[s][i])
            {
                 G[s][i]--, G[i][s]--;
                 Fleury(i);
        Ans.pb(s);
   }
16
   4.8 Bipartite Graph Matching
   4.8.1 Hungry
1 //poj3041
   const int MAXV=1e3+5;
   struct BM{
        int V;
        vi G[MAXV];
        int match[MAXV];
        bool vis[MAXV];
        void init(int x){
```

```
V=x;
9
             rep(i,1,V)G[i].clear();
10
11
        void add_edge(int u,int v){
12
             G[u].pb(v);
13
             G[v].pb(u);
14
15
        bool dfs(int u){
16
             vis[u]=true;
17
             for(int i=0;i<(int)G[u].size();i++){</pre>
18
19
                  int v=G[u][i];
20
                  int w=match[v];
                  if(w==-1||(!vis[w]&&dfs(w))){
21
22
                      match[u]=v;
23
                      match[v]=u;
24
                      return true;
                  }
25
             }
26
27
             return false;
28
29
        int matching(){
             int ret=0;
30
             clr(match,-1);
31
32
             rep(i,1,V){
33
                  if(match[i]==-1){
                      clr(vis,0);
34
                      if(dfs(i))ret++;
35
                  }
36
             }
37
38
             return ret;
        }
39
40
   }bm;
    int work(){
41
42
        int n,k;
        scanf("%d%d",&n,&k);
43
        bm.init(2*n);
44
45
        while(k--){
             int u,v;
46
             scanf("%d%d",&u,&v);
47
48
             bm.add_edge(u,n+v);
49
        printf("%d",bm.matching());
50
        return 0;
51
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];
   bool used[maxm];
6
   int num[maxm]; //
7
   bool dfs(int u)
8
9
   {
        for (int v = 0; v < vN; v++)
    if (g[u][v] && !used[v])</pre>
10
11
```

```
{
12
                used[v] = true;
13
                if (linker[v][0] < num[v])</pre>
14
15
                     linker[v][++linker[v][0]] = u;
16
                     return true;
17
18
                for (int i = 1; i <= num[0]; i++)</pre>
19
                     if (dfs(linker[v][i]))
20
21
                     {
22
                         linker[v][i] = u;
23
                         return true;
24
25
            }
26
        return false;
27
   int hungary()
28
29
   {
30
        int res = 0;
        for (int i = 0; i < vN; i++) linker[i][0] = 0;
31
        for (int u = 0; u < uN; u++)
32
33
            clr(used, 0);
34
35
            if (dfs(u)) res++;
36
37
        return res;
38 }
   4.8.3 Kuhn-Munkres
1 const int maxn = "Edit";
   int nx, ny;
3 int g[maxn][maxn];
4 int linker[maxn], lx[maxn], ly[maxn]; //y
                                                   ,x,y
5 int slack[N];
  bool visx[N], visy[N];
7 bool dfs(int x)
8
   {
9
        visx[x] = true;
10
        for (int y = 0; y < ny; y++)
11
            if (visy[y]) continue;
12
13
            int tmp = lx[x] + ly[y] - g[x][y];
            if (tmp == 0)
14
15
            {
                visy[y] = true;
16
                if (linker[y] == -1 || dfs(linker[y]))
17
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
   int KM()
```

```
29
   {
        clr(linker, -1), clr(ly, 0);
30
        for (int i = 0; i < nx; i++)
31
32
            lx[i] = -INF;
33
            for (int j = 0; j < ny; j++)
34
35
                 if (g[i][j] > lx[i]) lx[i] = g[i][j];
36
        for (int x = 0; x < nx; x++)
37
38
39
            clr(slack, 0x3f);
40
            for (;;)
41
                 clr(visx, 0), clr(visy, 0);
42
                 if (dfs(x)) break;
43
                 int d = INF;
44
                 for (int i = 0; i < ny; i++)
45
                     if (!visy[i] && d > slack[i]) d = slack[i];
46
                 for (int i = 0; i < nx; i++)
47
                     if (visx[i]) lx[i] -= d;
48
                 for (int i = 0; i < ny; i++)
49
                     if (visy[i])
50
                         ly[i] += d;
51
52
                     else
53
                         slack[i] -= d;
54
            }
        }
55
        int res = 0;
56
        for (int i = 0; i < ny; i++)
57
            if (~linker[i]) res += g[linker[i]][i];
58
59
        return res;
60 }
   4.9 Network Flow
   struct Edge
1
2
        int from, to, cap, flow;
3
        Edge(int u, int v, int c, int f)
4
            : from(u), to(v), cap(c), flow(f) {}
5
   };
1 struct Edge
2
   {
        int from, to, cap, flow, cost;
3
        Edge(int u, int v, int c, int f, int w)
            : from(u), to(v), cap(c), flow(f), cost(w) {}
6 };
                                 , \quad Y \qquad T \quad \  , \qquad X \quad Y
              S 	ext{ } T, 	ext{ } S 	ext{ } X
           m
                 n
                                                        \{p_1, p_2\}
```

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

```
if (!a[t]) break;
47
                for (int u = t; u != s; u = edges[p[u]].from)
48
49
                    edges[p[u]].flow += a[t];
50
                    edges[p[u] ^1].flow -= a[t];
51
52
53
                flow += a[t];
            }
54
            return flow;
55
56
       }
   };
57
   4.9.2 Dinic
   const int maxn = "Edit";
2
   struct Dinic
3
   {
       4
5
6
       vector<int> G[maxn]; // ,G[i][j] i j e
7
       bool vis[maxn];
                             //BFS
       int d[maxn];
                             // i
8
       int cur[maxn];
                             //
9
       void init(int n)
10
       {
11
12
            this->n = n;
13
            for (int i = 0; i < n; i++) G[i].clear();</pre>
14
            edges.clear();
15
       void AddEdge(int from, int to, int cap)
16
17
            edges.pb(Edge(from, to, cap, 0));
18
19
            edges.pb(Edge(to, from, 0, 0));
20
           m = edges.size();
           G[from].pb(m - 2);
21
22
           G[to].pb(m - 1);
23
       bool BFS()
24
25
26
            clr(vis, 0);
27
            clr(d, 0);
28
            queue<int> q;
29
            q.push(s);
           d[s] = 0;
30
            vis[s] = 1;
31
           while (!q.empty())
32
33
            {
34
                int x = q.front();
35
                q.pop();
                for (int i = 0; i < G[x].size(); i++)
36
37
                    Edge& e = edges[G[x][i]];
38
39
                    if (!vis[e.to] && e.cap > e.flow)
40
                        vis[e.to] = 1;
41
                        d[e.to] = d[x] + 1;
42
43
                        q.push(e.to);
                    }
44
```

```
}
45
            }
46
            return vis[t];
47
48
        int DFS(int x, int a)
49
50
            if (x == t | | a == 0) return a;
51
            int flow = 0, f;
52
            for (int& i = cur[x]; i < G[x].size(); i++)</pre>
53
54
            {
55
56
                Edge& e = edges[G[x][i]];
                if (d[x] + 1 == d[e.to] \&\& (f = DFS(e.to, min(a, e.cap - e.flow))) > 0)
57
58
                    e.flow += f;
59
                    edges[G[x][i] \land 1].flow -= f;
60
                    flow += f;
61
                    a -= f;
62
                    if (a == 0) break;
63
                }
64
            }
65
            return flow;
66
67
        int Maxflow(int s, int t)
68
69
70
            this -> s = s;
            this->t = t;
71
            int flow = 0;
72
            while (BFS())
73
74
                clr(cur, 0);
75
76
                flow += DFS(s, INF);
77
            return flow;
78
79
        }
   };
80
   4.9.3 ISAP
   const int maxn = "Edit";
2
   struct ISAP
   {
3
       4
5
6
        bool vis[maxn];
                              //BFS
7
        int d[maxn];
                              // i
8
        int cur[maxn];
                              //
9
                              //
        int p[maxn];
10
        int num[maxn];
                              //
11
        void init(int n)
12
13
        {
            this -> n = n;
14
            for (int i = 0; i < n; i++) G[i].clear();</pre>
15
            edges.clear();
16
17
        void AddEdge(int from, int to, int cap)
18
19
```

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

```
clr(cur, 0);
79
             while (d[s] < n)
80
81
                 if(x == t)
82
83
                      flow += Augumemt();
84
85
                      X = S;
86
                 int ok = 0;
87
                 for (int i = cur[x]; i < G[x].size(); i++)</pre>
88
89
90
                      Edge& e = edges[G[x][i]];
                      if (e.cap > e.flow && d[x] == d[e.to] + 1)
91
92
93
                          ok = 1;
                          p[e.to] = G[x][i];
94
95
                          cur[x] = i;
                          x = e.to;
96
97
                          break;
                      }
98
99
                 if (!ok) //Retreat
100
101
                      int m = n - 1;
102
103
                      for (int i = 0; i < G[x].size(); i++)</pre>
104
                          Edge& e = edges[G[x][i]];
105
                          if (e.cap > e.flow) m = min(m, d[e.to]);
106
107
                      if (--num[d[x]] == 0) break; //gap
108
                      num[d[x] = m + 1] ++;
109
                      cur[x] = 0;
110
                      if (x != s) x = edges[p[x]].from;
111
                 }
112
             }
113
             return flow;
114
115
         }
116 };
    4.9.4 MinCost MaxFlow
    const int maxn = "Edit";
 2
    struct MCMF
    {
 3
 4
         int n, m;
         vector<Edge> edges;
 5
         vector<int> G[maxn];
 6
         int inq[maxn]; //
 7
                         //bellmanford
         int d[maxn];
 8
         int p[maxn];
 9
                         //
10
         int a[maxn];
                         //
         void init(int n)
11
12
13
             this->n = n;
             for (int i = 0; i < n; i++) G[i].clear();</pre>
14
15
             edges.clear();
16
17
         void AddEdge(int from, int to, int cap, int cost)
```

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

5 Computational Geometry

5.1 Basic Function

```
#define zero(x) ((fabs(x) < eps ? 1 : 0))
   #define sqn(x) (fabs(x) < eps ? 0 : ((x) < 0 ? -1 : 1))
4 struct point
5
       double x, y;
6
       point(double a = 0, double b = 0) { x = a, y = b; }
7
       point operator-(const point& b) const { return point(x - b.x, y - b.y); }
8
       point operator+(const point& b) const { return point(x + b.x, y + b.y); }
9
10
       bool operator==(point& b) { return zero(x - b.x) && zero(y - b.y); }
11
12
       // ( )
       double operator*(const point& b) const { return x * b.x + y * b.y; }
13
       // ( )
14
       double operator^(const point& b) const { return x * b.y - y * b.x; }
15
       point rotate(point b, double a)
17
18
           double dx, dy;
19
           (*this - b).split(dx, dy);
20
           double tx = dx * cos(a) - dy * sin(a);
21
           double ty = dx * sin(a) + dy * cos(a);
22
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 = \{
       point s, e;
30
31
       line() {}
       line(point ss, point ee) { s = ss, e = ee; }
32
   };
33
   5.2 Position
   5.2.1 Point-Point
double dist(point a, point b) { return sqrt((a - b) * (a - b)); }
   5.2.2 Line-Line
1 // <0, *> ; <1, *> ; <2, P>
  pair<int, point> spoint(line l1, line l2)
2
3
       point res = l1.s;
4
       if (sgn((11.s - 11.e) \wedge (12.s - 12.e)) == 0)
5
           return mp(sqn((l1.s - l2.e) ^ (l2.s - l2.e)) != 0, res);
6
       double t = ((11.s - 12.s) \wedge (12.s - 12.e)) / ((11.s - 11.e) \wedge (12.s - 12.e));
7
       res.x += (l1.e.x - l1.s.x) * t;
8
       res.y += (l1.e.y - l1.s.y) * t;
9
10
       return mp(2, res);
11 }
```

5.2.3 Segment-Segment

```
1 bool segxseg(line l1, line l2)
2
   {
3
       return
4
           max(11.s.x, 11.e.x) >= min(12.s.x, 12.e.x) &&
5
            max(12.s.x, 12.e.x) >= min(11.s.x, 11.e.x) &&
            max(11.s.y, 11.e.y) >= min(12.s.y, 12.e.y) &&
6
            max(12.s.y, 12.e.y) >= min(11.s.y, 11.e.y) &&
7
            sgn((l2.s - l1.e) \land (l1.s - l1.e)) * sgn((l2.e-l1.e) \land (l1.s - l1.e)) <= 0 &&
8
            sgn((11.s - 12.e) \wedge (12.s - 12.e)) * sgn((11.e-12.e) \wedge (12.s - 12.e)) <= 0;
9
10 }
   5.2.4 Line-Segment
1 //11 ,12
2 bool segxline(line l1, line l2)
3
       return sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e - l1.e) ^ (l1.s - l1.e)) <=
4
       0;
5 }
   5.2.5 Point-Line
1 double pointtoline(point p, line l)
2
       point res;
3
       double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
4
       res.x = 1.s.x + (1.e.x - 1.s.x) * t, res.y = 1.s.y + (1.e.y - 1.s.y) * t;
5
       return dist(p, res);
6
7
  }
   5.2.6 Point-Segment
   double pointtosegment(point p, line l)
2
3
       point res:
       double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
4
       if (t >= 0 && t <= 1)
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
7
       else
            res = dist(p, l.s) < dist(p, l.e) ? l.s : l.e;
8
9
       return dist(p, res);
10 }
   5.2.7 Point on Segment
   bool PointOnSeg(point p, line l)
1
2
3
       return
            sgn((1.s - p) \wedge (1.e-p)) == 0 \&\&
4
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; for (int i = 0; i < n; i++) res $+= (p[i] \land p[(i + 1) \% n]) / 2;$ 4 return fabs(res); 6 } 5.3.2 Point in Convex < 0 > 0)// : [0,n) 3 // -1: 4 // 0 : 5 // 1 6 int PointInConvex(point a, point p∏, int n) 7 { for (int i = 0; i < n; i++) 8 if $(sgn((p[i] - a) \land (p[(i + 1) \% n] - a)) < 0)$ 9 10 return -1; else if (PointOnSeg(a, line(p[i], p[(i + 1) % n]))) 11 return 0; 1213 return 1; 14 } 5.3.3 Point in Polygon 3, 0~n-1 1 // ,poly[] 2 // -1: 3 // 0 : 4 // 1 5 int PointInPoly(point p, point poly[], int n) { 6 int cnt; 7 line ray, side; 8 9 cnt = 0;10 ray.s = p;11 ray.e.y = p.y; ray.e.x = -100000000000.0; // -INF, 12 for (int i = 0; i < n; i++) 13 14 side.s = poly[i], side.e = poly[(i + 1) % n]; 15 if (PointOnSeg(p, side)) return 0; 16 17 if (sgn(side.s.y - side.e.y) == 0)18 19 continue; if (PointOnSeg(sid e.s, r ay)) 20 21 cnt += (sgn(side.s.y - side.e.y) > 0);22else if (PointOnSeg(side.e, ray)) cnt += (sgn(side.e.y - side.s.y) > 0);23 else if (segxseg(ray, side)) 2425 cnt++;

26 27

28 }

return cnt % 2 == 1 ? 1 : -1;

```
5.3.4 Judge Convex
1 //
2 // 1~n-1
3 bool isconvex(point poly[], int n)
4
       bool s[3];
5
       clr(s, 0);
6
       for (int i = 0; i < n; i++)
7
8
           s[sgn((poly[(i + 1) % n] - poly[i]) ^ (poly[(i + 2) % n] - poly[i])) + 1] = 1;
9
           if (s[0] && s[2]) return 0;
10
11
12
       return 1;
13 }
   5.4 Integer Points
   5.4.1 On Segment
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
       int i, ret = 0;
3
       for (i = 0; i < n; i++)
4
           ret += \__gcd(fabs(p[i].x - p[(i + 1) % n].x), fabs(p[i].y - p[(i + 1) % n].y));
5
       return ret;
6
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++)
           area += p[(i + 1) % n].y * (p[i].x - p[(i + 2) % n].x);
5
       return (fabs(area) - OnEdge(n, p)) / 2 + 1;
6
   }
7
   5.5 Circle
   5.5.1 Circumcenter
   point waixin(point a, point b, point c)
2
       double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1 * a1 + b1 * b1) / 2;
3
       double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2 * a2 + b2 * b2) / 2;
4
       double d = a1 * b2 - a2 * b1;
5
       return point(a.x + (c1 * b2 - c2 * b1) / d, a.y + (a1 * c2 - a2 * c1) / d);
6
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 }</pre>
```

6.1.2 Longest Increase

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

6.1.3 Longest Common Increase

```
1
2 int LCIS(int a□, int b□, int n, int m)
3 {
       clr(dp, 0);
4
       for (int i = 1; i <= n; i++)
5
6
7
            int ma = 0;
            for (int j = 1; j <= m; j++)
8
9
                dp[i][j] = dp[i - 1][j];
10
                if (a[i] > b[j]) ma = max(ma, dp[i - 1][j]);
11
                if (a[i] == b[j]) dp[i][j] = ma + 1;
12
13
14
       return *max_element(dp[n] + 1, dp[n] + 1 + m);
15
16 }
   6.2 Digit Statistics
   int a[20];
   11 dp[20][state];
   ll dfs(int pos, /*state */, bool lead /* */, bool limit /* */)
3
4
       // , , 0, pos==-1
5
       if (pos == -1) return 1;
6
7
           1,
8
            pos,
       if (!limit && !lead && dp[pos][state] != -1) return dp[pos][state];
9
10
       int up = limit ? a[pos] : 9; // limit
11
12
       11 \text{ ans} = 0;
       for (int i = 0; i \le up; i++) //,
13
14
            if () ...
15
           else if () ...
16
           ans += dfs(pos - 1, /* */, lead && i == 0, limit && i == a[pos])
17
18
           /*
19
                 , state i */
20
       }
21
22
       if (!limit && !lead) dp[pos][state] = ans;
23
24
             lead, lead
25
26
       return ans;
27
  ll solve(ll x)
28
29
   {
       int pos = 0;
30
       do //
31
32
           a[pos++] = x \% 10;
33
       while (x \neq 10);
                                 */, /* */, true, true);
       return dfs(pos - 1 /*
34
35
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
        mat C(A.size(), vec(B[0].size()));
5
        for (int i = 0; i < A.size(); i++)</pre>
6
             for (int k = 0; k < B.size(); k++)</pre>
7
                 if (A[i][k]) //
8
                      for (int j = 0; j < B[0].size(); j++)
    C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % mod;</pre>
9
10
        return C;
11
12 }
13 mat Pow(mat A, ll n)
   {
14
        mat B(A.size(), vec(A.size()));
15
        for (int i = 0; i < A.size(); i++) B[i][i] = 1;</pre>
16
17
        for (; n; n >>= 1, A = mul(A, A))
             if (n \& 1) B = mul(B, A);
18
19
        return B;
20 }
    7.1.2 Gauss Elimination
   void gauss()
1
2
```

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

7.2 Tricks

7.2.1 Stack-Overflow

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

7.2.2 Fast-Scanner

```
template <class T>
1
   inline bool scan_d(T &ret){
2
       char c;
3
4
       int sqn;
       if (c = getchar(), c == EOF) return 0; //EOF
5
       while (c'!='-' \&\& (c < '0' || c > '9')) c = getchar();
6
       sqn = (c == '-') ? -1 : 1;
7
       ret = (c == '-') ? 0 : (c - '0');
8
       while (c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c - '0');</pre>
9
       ret *= sgn;
10
       return 1;
11
   }
12
   inline void out(int x){
13
14
       if(x<0)
15
           putchar('-');
16
           X=-X;
17
       if (x > 9) out(x / 10);
18
19
       putchar(x % 10 + '0');
20 }
   7.2.3 Strok-Sscanf
1 // get some integers in a line
gets(buf);
3 int v;
4 char *p = strtok(buf, " ");
5 while (p){
       sscanf(p, "%d", &v);
6
       p = strtok(NULL," ");
7
   }
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;</pre>
6 struct node{
       int l,r,id;
7
  }Q[MAXN];
8
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;</pre>
       else return x.l/block<y.l/block;</pre>
17
   }
18
   void add(int x){
19
       ANS+=cnt[pre[x]^k];
20
       cnt[pre[x]]++;
21
```

```
}
22
    void del(int x){
23
         cnt[pre[x]]--:
24
         ANS-=cnt[pre[x]^k];
25
    }
26
    int main(){
27
         scanf("%d%d%d",&n,&m,&k);
28
29
         block=(int)sqrt(n);
         pre[0]=0;
30
         for(int i=1;i<=n;i++){</pre>
31
32
              scanf("%d",&a[i]);
33
              pre[i]=a[i]^pre[i-1];
34
         for(int i=1;i<=m;i++){</pre>
35
              scanf("%d%d",&Q[i].1,&Q[i].r);
36
              Q[i].id=i;
37
38
39
         sort(Q+1,Q+1+m,cmp);
40
         ANS=0;
         memset(cnt,0,sizeof(cnt));
41
         cnt[0]=1;
42
         int L=1, R=0;
43
         for(int i=1;i<=m;i++){</pre>
44
              while(L>Q[i].1){L--;add(L-1);};
45
46
              while(L<Q[i].l){del(L-1);L++;}</pre>
              while(R<Q[i].r){R++;add(R);};</pre>
47
              while(R>Q[i].r){del(R);R--;};
48
              ans[Q[i].id]=ANS;
49
50
         for(int i=1;i<=m;i++){</pre>
51
              printf("%lld\n",ans[i]);
52
53
         return 0;
54
   }
55
    7.4 BigNum
    7.4.1 High-precision
1 java
    7.5 VIM
   syntax on
 2
    set nu
    set tabstop=4
3
   set shiftwidth=4
 4
5
   set cin
   set mouse=a
6
 7
   map<F2> :call SetTitle()<CR>
8
   func SetTitle()
9
10
   let l = 0
  let l = l + 1 | call setline(l,'#include <algorithm>')
   let l = l + 1 | call setline(l, '#include <iostream>')
  let l = l + 1 | call setline(l, "#include
let l = l + 1 | call setline(l, "#include
let l = l + 1 | call setline(l, "#include
let l = l + 1 | call setline(l, "#include
                                                        <cstring>')
<string>')
                                                          <cstdio>')
```

```
let l = l + 1 | call setline(l, '#include
                                                           <vector>')
let l = l + 1 \mid call \ setline(l, '#define \ rep(i,a,b) \ for(int \ i=a;i<=b;i++)')
let l = l + 1 \mid call \ setline(l, '#define \ per(i, a, b) \ for(int i=a; i>=b; i--)')
   let l = l + 1 \mid call \ setline(l, '#define \ clr(a,x) \ memset(a,x,sizeof(a))')
let l = l + 1 \mid call setline(l, '#define pb push_back')
39 let l = l + 1 | call setline(l,'int work(){')
40 let l = l + 1 | call setline(l,'')
41 let l = l + 1 \mid 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, '#endif')

46 let l = l + 1 | call setline(l, '#endif')

47 let l = l + 1 | call setline(l, '#ovk();')

48 let l = l + 1 | call setline(l, 'return 0;')
                                                 freopen("input.txt","rt",stdin);')
48 let l = l + 1 \mid call setline(l,
                                                 return 0;')
    let l = l + 1 \mid call setline(l,')
49
50
    endfunc
```

7.6 BASH

7.6.1 a.sh