



模板之用了就秃

WUST

So Like Coding? You Baldy

October 10, 2019

Contents

0	Header	1
0.1	pbds	1
0.1.1	head	1
0.2	FastIO	1
0.2.1	FastScanner	1
0.2.2	FastPowAndAdd	1
0.2.3	PythonInput	2
0.2.4	SpecialInput	2
0.3	header	3
0.3.1	comp	4
1	Math	5
1.1	素数	5
1.1.1	Eratosthenes 筛法	5
1.1.2	Euler 筛	5
1.1.3	MillerRabin 素性测试	5
1.1.4	PollardRho 快速因数分解	6
1.1.5	求素因子	6
1.2	约数	6
1.2.1	EulerPhi	6
1.2.2	Sieve	7
1.2.3	gcd	8
1.2.4	解乘法逆元	8
1.3	同余	8
1.3.1	扩展欧几里得算法	8
1.3.2	中国剩余定理	9
1.3.3	扩展中国剩余定理	9
1.3.4	BSGS	10
1.3.5	exBSGS	10
1.3.6	逆元	11
1.3.7	模素数二次同余方程	12
1.4	矩阵与线性方程组	12
1.4.1	矩阵快速幂	12
1.4.2	高斯消元	13
1.4.3	线性基	13
1.5	组合数学	14
1.5.1	Lucas	14
1.5.2	exLucas	15
1.5.3	递推组合数	16
1.5.4	小模数组合数	17
1.5.5	大模数组合数	17
1.6	卷积	18
1.6.1	FFT	18
1.6.2	NTT	19
1.6.3	原根	20
1.6.4	FWT	21
1.7	多项式	22
1.7.1	拉格朗日插值	22
1.7.2	拉格朗日插值 (连续取值)	22
1.8	Others	23
1.8.1	BM	23
1.8.2	exBM	24
1.8.3	杜教筛	29
1.8.4	欧拉降幂	30
1.8.5	公式	32
1.8.6	博弈	34

2	Graph Theory	35
2.1	路径	35
2.1.1	Dijkstra	35
2.1.2	Euler Path	35
2.1.3	K shortest Path(Astar)	36
2.1.4	K shortest Path(可持久化可并堆)	38
2.2	生成树	40
2.2.1	Kruskal	40
2.2.2	Prim	41
2.2.3	最小树形图	41
2.2.4	Matrix Tree	42
2.2.5	Steiner Tree	43
2.3	连通性	44
2.3.1	割点	44
2.3.2	桥	45
2.3.3	强连通分量	45
2.3.4	点双联通分量	46
2.3.5	边双联通分量	47
2.4	二分图匹配	48
2.4.1	Hungary Algorithm	48
2.4.2	Hopcroft-karp Algorithm	48
2.4.3	二分图多重匹配	50
2.4.4	二分图最大权匹配 (KM 算法)	51
2.4.5	一般图匹配带花树	52
2.5	网络流	54
2.5.1	Dinic	54
2.5.2	ISAP	56
2.5.3	MCMF	57
2.5.4	Trick	59
2.5.5	Stoer Wagner	60
2.6	Others	60
2.6.1	拓扑排序	60
2.6.2	2-SAT	61
2.6.3	差分约束系统	62
2.6.4	支配树	63
2.6.5	Stable Matching Problem	65
3	DataStructure	67
3.1	SegmentTreeDS	67
3.1.1	SegmentTree	67
3.1.2	离散化区间	68
3.1.3	动态区间最大子段和	69
3.1.4	动态开点权值线段树	70
3.1.5	扫描线	70
3.2	HLD	75
3.2.1	HLD	75
3.3	RMQ	78
3.3.1	RMQ	78
3.3.2	RMQbyIndex	79
3.3.3	RMQinNM	79
3.4	MO	80
3.4.1	MO	80
3.4.2	MObyModify	81
3.4.3	分块	82
3.4.4	弹飞绵羊	83
3.4.5	树莫队	84
3.5	VirtualTree	86
3.5.1	VirtualTree	86
3.6	PersistentDS	88

3.6.1	主席树区间 k 大	88
3.6.2	可持久化数组	89
3.7	Tree	91
3.7.1	LCA	91
3.7.2	前向星	92
3.7.3	点分治	92
3.8	Others	94
3.8.1	BITinNM	94
3.8.2	静态区间 k 大划分树	95
4	String	97
4.1	KMP	97
4.1.1	KMP	97
4.1.2	exKMP	97
4.2	Trie	98
4.2.1	Trie	98
4.2.2	Persistence Trie	98
4.2.3	01Trie	99
4.3	Manacher	100
4.3.1	Manacher	100
4.4	Aho-Corasick Automation	101
4.4.1	AC Automation	101
4.5	Suffix Array	102
4.5.1	Suffix Array	102
4.6	PalindromicTree	103
4.6.1	PalindromicTree	103
4.7	Hash	105
4.7.1	hash	105
4.7.2	doubleHash	106
4.7.3	二维 hash	109
4.7.4	树 hash 同构	109
4.8	Suffix Automation	110
4.8.1	SAM	110
4.9	Others	112
4.9.1	最小表示法	112
5	dp	113
5.1	BitDP	113
5.1.1	数位 dp 计和	113
5.1.2	两个数数位 dp	113
5.2	Subsequence	114
5.2.1	MaxSum	114
5.2.2	LIS	115
5.2.3	LongestCommonIncrease	115
5.2.4	LCS	115
5.3	Others	117
5.3.1	矩阵快速幂	117
6	Others	119
6.1	mint 类	119
6.2	不重叠区间贪心	119
6.3	BigInt 类	119
6.4	date	125
6.5	Frac 类	125
6.6	模拟退火 (最小圆覆盖)	126
6.7	string 类	127
6.8	前缀异或和	128
6.9	约瑟夫环第 k 个	128
6.10	二分	128

6.11 猛男 IO 挂	129
6.12 贪心结论	130
6.13 builtin	131
6.14 n 以内 k 因子的个数	131

0 Header

0.1 pbds

0.1.1 head

```
1 #include <bits/extc++.h>
2 #pragma comment(linker, "/STACK:102400000,102400000")
3 using namespace __gnu_pbds; // tree, gp_hash_table, trie
4 using namespace __gnu_cxx; // rope
5 tree<TYPE, null_type, less<>, rb_tree_tag, tree_order_statistics_node_update> tr;
6 // 可并堆
7 #include <ext/pb_ds/priority_queue.hpp>
8 using namespace __gnu_pbds;
9 __gnu_pbds::priority_queue<int, greater<int>, pairing_heap_tag> q[maxn];
10 //q[i].join(q[j]) 将j堆并入i
11
```

0.2 FastIO

0.2.1 FastScanner

```
1 // 适用于正负整数
2 template <class T>
3 inline bool scan(T &ret){
4     char c;
5     int sgn;
6     if (c = getchar(), c == EOF) return 0; //EOF
7     while (c != '-' && (c < '0' || c > '9')) c = getchar();
8     sgn = (c == '-') ? -1 : 1;
9     ret = (c == '-') ? 0 : (c - '0');
10    while (c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c - '0');
11    ret *= sgn;
12    return 1;
13 }
14
15 template <class T>
16 inline void out(T x) {
17     if (x > 9) out(x / 10);
18     putchar(x % 10 + '0');
19 }
20
21 inline int read() {
22     int x = 0;
23     char ch = getchar();
24     while (ch > '9' || ch < '0') ch = getchar();
25     while (ch >= '0' && ch <= '9') {
26         x = x * 10 + ch - '0';
27         ch = getchar();
28     }
29     return x;
30 }
```

0.2.2 FastPowAndAdd

```
1 // 精确快速乘
2 ll qpmul(ll a, ll b) {
3     a %= mod; b %= mod;
```

```
4     ll res = 0;
5     while (b > 0) {
6         if (b & 1) {
7             res = (res + a);
8             if (res >= mod) res -= mod;
9         }
10        a = (a + a);
11        if (a >= mod) a -= mod;
12        b >>= 1;
13    }
14    return res;
15 }
16
17 // 0(1)快速乘
18 ll mul2(ll x,ll y,ll p) {
19     ll res=(x*y-(ll)((long double)x/p*y+1.0e-8)*p);
20     return res<0?res+p:res;
21 }
22
23 //int128
24 ll ans = ((__int128) a * b) % p;
25
26 // 10进制快速幂, 直接读入%s,c 预处理字符串len
27 char c[1000005], len;
28 ll qp(ll a) {
29     len--;
30     a %= mod;
31     ll s = a;
32     ll res = 1;
33     while (len >= 0) {
34         ll cur = s;
35         for (int i = 1; i <= c[len] - '0'; ++i) {
36             res = res * s % mod;
37         }
38         for (int i = 1; i < 10; ++i) {
39             cur = cur * s % mod;
40         }
41         s = cur;
42         len--;
43     }
44     return res;
45 }
```

0.2.3 PythonInput

```
1 // python一行读入
2 a,b = map(int, input().split())
3
4 a = []
5 for i in input().split():
6     a.append(int(i))
```

0.2.4 SpecialInput

```
1 // 代替gets
2 scanf("%[^\n]%*c", ss)
3
4 void out2(int x, int flag = 1) {
```

```
5     if (x == 0) {
6         if (flag) putchar('0');
7         return;
8     }
9     out2(x >> 1, 0);
10    putchar('0' + x % 2);
11 }
```

0.3 header

```
1  // Editor -> Live Templates
2  // add template group acm
3  // add template main
4  // C++ Declaration
5
6  #include <bits/stdc++.h>
7  #define ll long long
8  using namespace std;
9
10 template <class T>
11 inline bool scan(T &ret){
12     char c;
13     int sgn;
14     if (c = getchar(), c == EOF) return 0; //EOF
15     while (c != '-' && (c < '0' || c > '9')) c = getchar();
16     sgn = (c == '-') ? -1 : 1;
17     ret = (c == '-') ? 0 : (c - '0');
18     while (c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c - '0');
19     ret *= sgn;
20     return 1;
21 }
22
23 const ll mod = 1e9+7;
24 const int maxn = $MAXN$;
25
26 ll qp(ll x, ll n) {
27     ll res = 1; x %= mod;
28     while (n > 0) {
29         if (n & 1) res = res * x % mod;
30         x = x * x % mod;
31         n >>= 1;
32     }
33     return res;
34 }
35
36 int main(int argc, char* argv[]) {
37     $CONTENT$
38     return 0;
39 }
40
41 // C++ Expression debug
42 freopen("data.in", "r", stdin);
43 freopen("data.out", "w", stdout);
44 clock_t ST = clock();
45 cerr << "time: " << ((clock()-ST)*1000.0 / CLOCKS_PER_SEC) << "ms" << endl;
46
47
48 // C++ Expression tkase
```



```
49 int T;
50 scanf("%d", &T);
51 for (int kase = 1; kase <= T; ++kase) { $CONTENT$;
52 }
```

0.3.1 comp

```
1 // 1 create directory comp
2 // 2 create directory comp/test
3 // 3 create duipai.cpp
4 // add text
5
6 #include<bits/stdc++.h>
7 using namespace std;
8 int main(){
9     int i;
10    for (i=1;;i++){
11        printf("The result of No. %d Case is: ",i);
12        system("python3 rand.py");
13        system("./std < test/data.in > test/std.out");
14        system("./my < test/data.in > test/my.out");
15        if (system("diff test/std.out test/my.out")){
16            printf("Wrong Answer\n");
17            return 0;
18        }
19        else printf("Accepted\n");
20    }
21    return 0;
22 }
23
24
25 // 4 create duipai.sh
26 #!/bin/bash
27 g++ std.cpp -o std
28 g++ my.cpp -o my
29 python3 rand.py
30 ./a.out
31
32 // 5 create rand.py
33 # coding=utf-8
34 from random import randint, choice, shuffle
35 # with open("../cmake-build-debug/data.in", "w") as f:
36 with open("test/data.in", "w") as f:
37     n = randint(1, 10)
38     m = randint(1, 10)
39     f.write(f"{n} {m}")
40
41 // 6 terminal: g++ duipai.cpp
42 // 7 terminal: sudo chmod 777 duipai.sh
43 // 8 add my.cpp and std.cpp
44 // 9 ./duipai.sh
```

1 Math

1.1 素数

1.1.1 Eratosthenes 筛法

```
1 bool vis[(int)1e6+5];
2 int prim[(int)1e5], tot; // tot = 78499
3 void init() {
4     for (int i = 2; i < 1e6+5; ++i) {
5         if (vis[i]) continue;
6         prim[++tot] = i;
7         for (int j = i + i; j < 1e6+5; j += i) vis[j] = 1;
8     }
9 }
```

1.1.2 Euler 筛

```
1 const int maxn = 1e6 + 10;
2 int prime[maxn], v[maxn], n, cnt; //每个合数只会被它的最小质因子p筛一次
3
4 void Euler_Sieve()
5 {
6     for(int i = 2; i <= n; i++)
7     {
8         if(!v[i]) v[i] = i, prime[++cnt] = i;
9         for(int j = 1; j <= cnt && i * prime[j] <= n; j++)
10         {
11             v[i * prime[j]] = prime[j];
12             if(i % prime[j] == 0) break;
13         }
14     }
15 }
```

1.1.3 MillerRabin 素性测试

```
1 typedef long long ll;
2
3 bool check(ll a, ll n)
4 {
5     if(n == 2 || a >= n) return true;
6     if(n == 1 || !(n & 1)) return false;
7     ll d = n - 1;
8     while(!(d & 1)) d >>= 1;
9     ll t = qp(a, d, n);
10    while(d != n - 1 && t != 1 && t != n - 1)
11    {
12        t = mul(t, t, n);
13        d <<= 1;
14    }
15    return t == n - 1 || d & 1;
16 }
17
18 bool Miller_Rabin(ll n)
19 {
20     static vector<ll> t = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
21     if (n <= 1) return false;
22     for (ll k: t) if (!check(k, n)) return false;
```

```

23     return true;
24 }

```

1.1.4 PollardRho 快速因数分解

```

1  mt19937 mt(time(0));
2  ll pollard_rho(ll n, ll c) {
3      ll x = uniform_int_distribution<ll>(1, n - 1)(mt), y = x;
4      auto f = [&](ll v) { ll t = mul(v, v, n) + c; return t < n ? t : t - n; };
5      while (1) {
6          x = f(x); y = f(f(y));
7          if (x == y) return n;
8          ll d = __gcd(abs(x - y), n);
9          if (d != 1) return d;
10     }
11 }
12
13 ll fac[100], fcnt;
14 void get_fac(ll n, ll cc = 19260817) {
15     if (n == 4) { fac[fcnt++] = 2; fac[fcnt++] = 2; return; }
16     if (Miller_Rabin(n)) { fac[fcnt++] = n; return; }
17     ll p = n;
18     while (p == n) p = pollard_rho(n, --cc);
19     get_fac(p); get_fac(n / p);
20 }

```

1.1.5 求素因子

```

1  vector<pair<ll, int>> getFactors(ll x)
2  {
3      vector<pair<ll, int>> fact;
4      for (int i = 0; prime[i] <= x / prime[i]; i++)
5      {
6          if (x % prime[i] == 0)
7          {
8              fact.emplace_back(prime[i], 0);
9              while (x % prime[i] == 0) fact.back().second++, x /= prime[i];
10         }
11     }
12     if (x != 1) fact.emplace_back(x, 1);
13     return fact;
14 }

```

1.2 约数

1.2.1 EulerPhi

```

1  //计算欧拉phi函数, phi(n)且与n互素的正整数个数
2
3  int Euler(int n){
4      int rea=n;
5      for(int i=2; i*i<=n; i++)
6          if(n%i==0)//第一次找到的必为素因子
7          {
8              rea=rea-rea/i;
9              do
10                 n/=i;//把该素因子全部约掉

```

```
11         while(n%i==0);
12     }
13     if(n>1)
14         rea=rea-rea/n;
15     return rea;
16 } //单点欧拉 O(sqrt(n))
17
18 bool boo[50000];
19 int p[20000];
20 void prim(){
21     memset(boo,0,sizeof(boo));
22     boo[0]=boo[1]=1;
23     int k=0;
24     for(int i=2; i<50000; i++)
25     {
26         if(!boo[i])
27             p[k++]=i;
28         for(int j=0; j<k&&i*p[j]<50000; j++)
29         {
30             boo[i*p[j]]=1;
31             if(!(i%p[j]))
32                 break;
33         }
34     }
35 } //筛选法打表
36 int phi(int n)
37 {
38     int rea=n;
39     for(int i=0; p[i]*p[i]<=n; i++) //对于一些不是素数的可不遍历
40         if(n%p[i]==0)
41         {
42             rea=rea-rea/n;
43             do
44                 n/=p[i];
45             while(n%p[i]==0);
46         }
47     if(n>1)
48         rea=rea-rea/n;
49     return rea;
50 } //素数+欧拉
51
52 int euler[maxn];
53 void init() {
54     int i, j;
55     for(i=1; i<maxn; i++)
56         euler[i]=i;
57     for(i=2; i<maxn; i+=2)
58         euler[i]/=2;
59     for(i=3; i<maxn; i+=2)
60         if(euler[i]==i) {
61             for(j=i; j<=maxn; j+=i)
62                 euler[j]=euler[j]/i*(i-1);
63         }
64 } //递推欧拉表
```

1.2.2 Sieve

1 //用类似筛法的方法计算 $\phi(1), \phi(2), \dots, \phi(n)$

```
2 int phi[maxn];
3
4 void phi_table(int n)
5 {
6     for (int i = 2; i <= n; i++) phi[i] = 0;
7     phi[1] = 1;
8     for (int i = 2; i <= n; i++) if (!phi[i])
9         for (int j = i; j <= n; j += i)
10            {
11                if (!phi[j]) phi[j] = j;
12                phi[j] = phi[j] / i * (i - 1);
13            }
14 }
```

1.2.3 gcd

```
1 ll gcd(ll a, ll b) {while(b^=a^=b%=b);return a;}
2
3 void exgcd(ll a, ll b, ll& x, ll& y, ll& c) {
4     if(!b) {y = 0; x = 1; c = a; return;}
5     exgcd(b, a % b, y, x); y -= a / b * x;
6 }
```

1.2.4 解乘法逆元

```
1 void exgcd(ll a, ll b, ll c, ll d, ll &x, ll &y) {
2     ll z = (a + b - 1) / b;
3     if (z <= c / d) {
4         x = z;
5         y = 1;
6         return;
7     }
8     a -= (z - 1) * b; c -= (z - 1) * d;
9     exgcd(d, c, b, a, y, x);
10    x += (z - 1) * y;
11 }
12
13 int main(int argc, char* argv[]) {
14     int T;
15     scanf("%d", &T);
16     ll p, x;
17     for (int kase = 1; kase <= T; ++kase) {
18         scanf("%lld%lld", &p, &x);
19         ll b, y;
20         exgcd(p, x, p, x - 1, b, y);
21         printf("%lld/%lld\n", b * x - p * y, b);
22     }
23     return 0;
24 }
```

1.3 同余

1.3.1 扩展欧几里得算法

```
1 void exgcd(int a, int b, int &x, int &y)
2 {
3     if(b == 0) { x = 1; y = 0; return; }
```

```
4     exgcd(b, a % b, x, y);
5     int t = x; x = y, y = t - a / b * y;
6 }
```

1.3.2 中国剩余定理

```
1  typedef long long ll;
2
3  void exgcd(ll a, ll b, ll &x, ll &y)
4  {
5      if(b == 0) { x = 1; y = 0; return; }
6      exgcd(b, a % b, x, y);
7      ll t = x; x = y, y = t - a / b * y;
8  }
9
10 ll crt(ll *a, ll *m, int n)
11 {
12     ll M = 1, ans = 0;
13     for(int i = 1; i <= n; i++) M *= m[i];
14     for(int i = 1; i <= n; i++)
15     {
16         ll x = 0, y = 0;
17         ll Mi = M / m[i];
18         exgcd(Mi, m[i], x, y);
19         ans = (ans + Mi % M * x % M * a[i] % M + M) % M;
20     }
21     if(ans < 0) ans += M;
22     return ans;
23 }
```

1.3.3 扩展中国剩余定理

```
1  typedef long long ll;
2
3  const int N = 1e5 + 10;
4
5  int n;
6  ll a[N], r[N];
7
8  ll exgcd(ll a, ll b, ll& x, ll& y)
9  {
10     if(b == 0) { x = 1, y = 0; return a; }
11     ll ret = exgcd(b, a % b, y, x); y -= a / b * x;
12     return ret;
13 }
14
15 ll exCRT()
16 {
17     ll M = a[1], R = r[1], x, y, d;
18     for(int i = 2; i <= n; i++)
19     {
20         d = exgcd(M, a[i], x, y);
21         if((R - r[i]) % d) return -1;
22         x = (R - r[i]) / d * x % a[i];
23         R -= M * x;
24         M = M / d * a[i];
25         R %= M;
26     }
```

```
27     return (R % M + M) % M;
28 }
```

1.3.4 BSGS

```
1  int qp(int a, int n, int mod)
2  {
3      long long ans = 1, base = a;
4      while(n)
5      {
6          if(n & 1) (ans *= base) %= mod;
7          (base *= base) %= mod;
8          n >>= 1;
9      }
10     return ans;
11 }
12
13 int BSGS(int a, int b, int p)
14 {
15     map<int, int> hash;
16     b %= p;
17     int t = (int)sqrt(p) + 1;
18     for(int j = 0; j < t; j++)
19     {
20         int val = 1ll * b * qp(a, j, p) % p;
21         hash[val] = j;
22     }
23     a = qp(a, t, p);
24     if(a == 0) return b == 0 ? 1 : -1;
25     for(int i = 0; i <= t; i++)
26     {
27         int val = qp(a, i, p);
28         int j = hash.find(val) == hash.end() ? -1 : hash[val];
29         if(j >= 0 && i * t - j >= 0) return i * t - j;
30     }
31     return -1;
32 }
```

1.3.5 exBSGS

```
1  unordered_map<int, int> Hash;
2
3  int exBSGS(int a, int b, int p)
4  {
5      a %= p, b %= p;
6      if(b == 1) return 0;
7      if(!b && !a) return 1;
8      if(!a) return -1;
9      if(!b)
10     {
11         int ret = 0, d;
12         while((d = __gcd(a, p)) != 1)
13         {
14             ++ret, p /= d;
15             if(p == 1) return ret;
16         }
17         return -1;
18     }
```

```

19     int ret = 0, A = a, B = b, P = p, C = 1, d;
20     while((d = __gcd(A, P)) != 1)
21     {
22         if(B % d) return -1;
23         P /= d, B /= d;
24         C = 111 * C * (A / d) % P;
25         ++ ret;
26         if(C == B) return ret;
27     }
28     Hash.clear();
29     int f = 1, t = sqrt(P) + 1;
30     for(int i = 0; i < t; i++)
31     {
32         Hash[111 * f * B % P] = i;
33         f = 111 * f * A % P;
34     }
35     int tf = f;
36     f = 111 * f * C % P;
37     for(int i = 1; i <= t; i++)
38     {
39         if(Hash.find(f) != Hash.end()) return ret + i * t - Hash[f];
40         f = 111 * f * tf % P;
41     }
42     return -1;
43 }

```

1.3.6 逆元

```

1  /*
2  1. 费马小定理
3  条件: mod为素数
4  */
5  ll inv(ll x){return qp(x,mod-2);}
6
7  /*
8  2. 扩展欧几里得
9  条件: gcd(a,mod)==1
10  如果gcd(a,mod)!=1 返回-1
11  */
12  ll inv(ll a,ll p)
13  {
14      ll g,x,y;
15      g=exgcd(a,p,x,y);
16      return g==1?(x+p)%p:-1;
17  }
18
19  /*
20  3. 公式
21  a/b%mod=c
22  ->a%(b*mod)/b=c
23  */
24
25  /*
26  4. 逆元打表
27  p是模
28  p要求是奇素数
29  */
30  ll inv[MAX];

```



```
31 void getinv(int n,ll p)
32 {
33     ll i;
34     inv[1]=1;
35     for(i=2;i<=n;i++) inv[i]=(p-p/i)*inv[p%i]%p;
36 }
37
38 // log逆元
39 ll dlog(ll g, ll b, ll p) {
40     ll m = sqrt(p - 1);
41     map<ll, ll> powers;
42     for (long j = 0; j < m; j++) powers[qp(g, j, p)] = j;
43     long gm = qp(g, -m + 2 * (p - 1), p);
44     for (int i = 0; i < m; i++) {
45         if (powers[b]) return i * m + powers[b];
46         b = b * gm % p;
47     }
48     return -1;
49 }
```

1.3.7 模素数二次同余方程

```
1 // 要求模为素数, 输入n, mod, 返回  $x^2 \bmod = n$ , 可解任意一次二元方程
2
3 bool Legendre(ll a,ll p) {
4     return qp(a,p-1>>1,p)==1;
5 }
6
7 ll modsqr(ll a,ll p) {
8     ll x;
9     ll i,k,b;
10    if(p==2) x=a%p;
11    else if(p%4==3) x=qp(a,p+1>>2,p);
12    else {
13        for(b=1;Legendre(b,p);++b);
14        i=p-1>>1;
15        k=0;
16        do
17        {
18            i>>=1;
19            k>>=1;
20            if(!((1LL*qp(a,i,p)*qp(b,k,p)+1)%p)) k+=p-1>>1;
21        }while(!(i&1));
22        x=1ll*qp(a,i+1>>1,p)*qp(b,k>>1,p)%p;
23    }
24    return min(x, p - x);
25 //     if(p-x<x) x=p-x;
26 //     if(x==p-x) printf("%d\n",x);
27 //     else printf("%d %d\n",x,p-x);
28 }
```

1.4 矩阵与线性方程组

1.4.1 矩阵快速幂

```
1 const int mod = 1e9 + 7;
2 typedef long long ll;
3
```

```
4 int cur;
5 struct Matrix {ll a[105][105]; };
6
7 Matrix mul(Matrix a, Matrix b)
8 {
9     Matrix res;
10    memset(res.a, 0, sizeof res.a);
11    for(int i = 0; i < cur; i++)
12        for(int j = 0; j < cur; j++)
13            for(int k = 0; k < cur; k++)
14                (res.a[i][j] += a.a[i][k] * b.a[k][j] % mod) %= mod;
15    return res;
16 }
17
18 Matrix pow(Matrix a, ll n)
19 {
20     Matrix ans, base = a;
21     for(int i = 0; i < cur; i++) ans.a[i][i] = 1;
22     while(n)
23     {
24         if(n & 1) ans = mul(ans, base);
25         base = mul(base, base);
26         n >>= 1;
27     }
28     return ans;
29 }
```

1.4.2 高斯消元

```
1 const int N = 20 + 10;
2
3 int n;
4 double b[N], c[N][N];
5 //c: 系数矩阵, b: 常数; 二者一起构成增广矩阵
6
7 void Gaussian_Elimination()
8 {
9     for(int i = 1; i <= n; i++)
10     {
11         //找到x[i]的系数不为0的一个方程
12         for(int j = i; j <= n; j++) if(fabs(c[j][i]) > 1e-8)
13         {
14             for(int k = 1; k <= n; k++) swap(c[i][k], c[j][k]);
15             swap(b[i], b[j]);
16         }
17         //消去其他方程的x[i]的系数
18         for(int j = 1; j <= n; j++)
19         {
20             if(i == j) continue;
21             double rate = c[j][i] / c[i][i];
22             for(int k = i; k <= n; k++) c[j][k] -= c[i][k] * rate;
23             b[j] -= b[i] * rate;
24         }
25     }
26 }
```

1.4.3 线性基

```
1  typedef long long ll;
2
3  struct LinearBasis
4  {
5      ll d[64], tot;
6
7      void ins(ll x)  //插入线性基
8      {
9          for(int i = 63; i >= 0; i --)
10             {
11                 if((x >> i) & 1)
12                     {
13                         if(!d[i]) return void(d[i] = x);
14                         x ^= d[i];
15                     }
16             }
17     }
18
19     ll max_xor()  //在一个序列中取若干个数，使其异或和最大
20     {
21         ll ans = 0;
22         for(int i = 63; i >= 0; i --)
23             if((ans ^ d[i]) > ans) ans ^= d[i];
24         return ans;
25     }
26
27     void init()
28     {
29         for(int i = 0; i < 64; i ++){ if(d[i])
30             for(int j = 0; j < i; j ++){
31                 if(d[i] & (1ll << j)) d[i] ^= d[j];
32             }
33         for(int i = 0; i < 64; i ++){ if(d[i]) d[tot ++] = d[i];
34     }
35
36     ll k_th(ll k)  //取任意个元素进行异或的第k小个数
37     {
38         //考虑能异或出0的情况，tot表示线性基中的元素个数
39         k -= (n != tot);
40         if(k > (1ll << tot)) return -1;
41         ll ans = 0;
42         for(int i = 0; i < tot; i ++){ if(k & (1ll << i)) ans ^= d[i];
43     }
44 };
```

1.5 组合数学

1.5.1 Lucas

```
1  const int maxn = 1e6 + 10;
2
3  ll fac[maxn], inv[maxn], facinv[maxn];
4
5  void init()
6  {
7      fac[0] = inv[0] = facinv[0] = 1;
8      fac[1] = inv[1] = facinv[1] = 1;
9      for(int i = 2; i < maxn; i++)
10     {
```

```
11     fac[i] = fac[i - 1] * i % mod;
12     inv[i] = mod - mod / i * inv[mod % i] % mod;
13     facinv[i] = facinv[i - 1] * inv[i] % mod;
14 }
15 }
16
17 ll C(int n, int k)
18 {
19     if(k > n || k < 0) return 0;
20     return fac[n] * facinv[k] % mod * facinv[n - k] % mod;
21 }
22
23 ll lucas(ll n, ll m)
24 {
25     ll res = 1;
26     while(n && m)
27     {
28         res = res * C(n % mod, m % mod) % mod;
29         n /= mod;
30         m /= mod;
31     }
32     return res;
33 }
```

1.5.2 exLucas

```
1  typedef long long ll;
2
3  ll p, n, m;
4
5  ll exgcd(ll a, ll b, ll &x, ll &y)
6  {
7      if(!b) { x = 1; y = 0; return a; }
8      ll res = exgcd(b, a % b, x, y), t;
9      t = x, x = y, y = t - a / b * y;
10     return res;
11 }
12
13 ll qp(ll a, ll n, ll mod)
14 {
15     ll ans = 1, base = a;
16     for(; n; n >>= 1, (base *= base) %= mod) if(n & 1) (ans *= base) %= mod;
17     return ans;
18 }
19
20 ll fac(ll n, ll a, ll b)
21 {
22     if(!n) return 1;
23     ll res = 1;
24     for(ll i = 2; i <= b; i++)
25         if(i % a) (res *= i) %= b;
26     res = qp(res, n / b, b);
27     for(ll i = 2; i <= n % b; i++)
28         if(i % a) (res *= i) %= b;
29     return res * fac(n / a, a, b) % b;
30 }
31
32 ll inv(ll n, ll mod)
```

```

33 {
34     ll x, y;
35     exgcd(n, mod, x, y);
36     return (x += mod) > mod ? x - mod : x;
37 }
38
39 ll CRT(ll b, ll mod) { return b * inv(p / mod, mod) % p * (p / mod) % p; }
40
41 ll C(ll n, ll m, ll a, ll b)
42 {
43     ll up = fac(n, a, b), d1 = fac(m, a, b), d2 = fac(n - m, a, b);
44     ll k = 0;
45     for(ll i = n; i; i /= a) k += i / a;
46     for(ll i = m; i; i /= a) k -= i / a;
47     for(ll i = n - m; i; i /= a) k -= i / a;
48     return up * inv(d1, b) % b * inv(d2, b) % b * qp(a, k, b) % b;
49 }
50
51 ll exlucas(ll n, ll m)
52 {
53     ll res = 0, tmp = p, b;
54     int lim = sqrt(p) + 5;
55     for(int i = 2; i <= lim; i++) if(tmp % i == 0)
56     {
57         b = 1;
58         while(tmp % i == 0) tmp /= i, b *= i;
59         (res += CRT(C(n, m, i, b), b)) %= p;
60     }
61     if(tmp > 1) (res += CRT(C(n, m, tmp, tmp), tmp)) %= p;
62     return res;
63 }

```

1.5.3 递推组合数

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

```

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

```

$0 \leq m \leq n \leq 10^5$, 模 p 为素数

```

1 const int maxn = 100010;
2 ll f[maxn];
3 ll inv[maxn]; // 阶乘的逆元
4 void CalFact() {
5     f[0] = 1;
6     for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % p;
7     inv[maxn - 1] = qp(f[maxn - 1], p - 2);
8     for (int i = maxn - 2; ~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; }

```

1.5.4 小模数组合数

p 小 n, m 大

```
1
2  const int NICO = 100000+10;
3  const int MOD = 99991;
4  ll f[NICO];
5
6  ll Lucas(ll a, ll k)
7  {
8      ll res = 1;
9      while(a && k)
10     {
11         ll a1 = a % MOD;
12         ll b1 = k % MOD;
13         if(a1 < b1) return 0;
14         res = res*f[a1]*qp(f[b1]*f[a1-b1]%MOD, MOD-2)%MOD;
15         a /= MOD;
16         k /= MOD;
17     }
18     return res;
19 }
20
21 void init()
22 {
23     f[0] = 1;
24     for(int i=1; i<=MOD; i++)
25     {
26         f[i] = f[i-1]*i%MOD;
27     }
28 }
29
30 int main()
31 {
32     init();
33     cout << Lucas(5,2) << endl;
34 }
```

1.5.5 大模数组合数

n, m 小 p 大

```
1  map<int, ll> m;
2
3  const int MOD = 1e9+7;
4  void fun(int n, int k) {
5      for (int i = 2; i <= sqrt(n * 1.0); i++) {
6          while (n % i == 0) {
7              n /= i;
8              m[i] += k;
9          }
10     }
11     if (n > 1) {
12         m[n] += k;
13     }
14 }
15
```

```
16 ll C(ll a, ll b) {
17     if (a < b || a < 0 || b < 0)
18         return 0;
19     m.clear();
20     ll ret = 1;
21     b = min(a - b, b);
22     for (int i = 0; i < b; i++) {
23         fun(a - i, 1);
24     }
25     for (int i = b; i >= 1; i--) {
26         fun(i, -1);
27     }
28     for (__typeof(m.begin()) it = m.begin(); it != m.end(); it++) {
29         if ((*it).second != 0) {
30             ret *= qp((*it).first, (*it).second);
31             ret %= MOD;
32         }
33     }
34     return ret;
35 }
36
37 int main(int argc, char *argv[])
38 {
39     ll a, b;
40     while (scanf("%lld%lld", &a, &b) != EOF) {
41         printf("%lld\n", C(a, b));
42     }
43     return 0;
44 }
```

1.6 卷积

1.6.1 FFT

```
1  const int maxn = 1e7 + 10;
2  const double Pi = acos(-1.0);
3
4  struct complex
5  {
6      double x, y;
7      complex (double xx = 0, double yy = 0) { x = xx, y = yy; }
8  }a[maxn], b[maxn];
9
10 complex operator + (complex a, complex b) { return complex(a.x + b.x, a.y + b.y); }
11 complex operator - (complex a, complex b) { return complex(a.x - b.x, a.y - b.y); }
12 complex operator * (complex a, complex b) { return complex(a.x * b.x - a.y * b.y, a.x * b.y + a.y *
    b.x); }
13
14 int n, m;
15 int l, r[maxn];
16 int limit;
17
18 void FFT(complex *A, int type)
19 {
20     for(int i = 0; i < limit; i++)
21         if(i < r[i]) swap(A[i], A[r[i]]);
22     for(int mid = 1; mid < limit; mid <= 1)
23     {
24         complex Wn(cos(Pi / mid), type * sin(Pi / mid));
```

```
25     for(int R = mid << 1, j = 0; j < limit; j += R)
26     {
27         complex w(1, 0);
28         for(int k = 0; k < mid; k ++, w = w * Wn)
29         {
30             complex x = A[j + k], y = w * A[j + mid + k];
31             A[j + k] = x + y;
32             A[j + mid + k] = x - y;
33         }
34     }
35 }
36 }
37
38 void mul()
39 {
40     l = 0, limit = 1;
41     while(limit <= n + m) limit <=< 1, l ++;
42     for(int i = 0; i < limit; i ++ )
43         r[i] = (r[i >> 1] >> 1) | ( (i & 1) << (l - 1));
44     FFT(a, 1);
45     FFT(b, 1);
46     for(int i = 0; i <= limit; i ++ ) a[i] = a[i] * b[i];
47     FFT(a, -1);
48     for(int i = 0; i <= n + m; i ++ )
49         printf("%d ", (int)(a[i].x / limit + 0.5));
50 }
```

1.6.2 NTT

```
1  const int maxn = 2097152;
2  const int mod = 998244353;
3  const int root = 3;
4  // 998244353 -> 3, 1e9+7 -> 5,
5
6  template<long long mod, long long root>
7  struct NTT {
8      vector<long long> omega;
9
10     NTT() {
11         omega.resize(maxn + 1);
12         long long x = fpow(root, (mod - 1) / maxn);
13         omega[0] = 1ll;
14         for (int i = 1; i <= maxn; ++i)
15             omega[i] = omega[i - 1] * x % mod;
16     }
17
18     long long fpow(long long a, long long n) {
19         (n += mod - 1) %= mod - 1;
20         long long r = 1;
21         for (; n >>= 1) {
22             if (n & 1) (r *= a) %= mod;
23             (a *= a) %= mod;
24         }
25         return r;
26     }
27
28     void bitrev(vector<long long> &v, int n) {
29         int z = __builtin_ctz(n) - 1;
```



```

30     for (int i = 0; i < n; ++i) {
31         int x = 0;
32         for (int j = 0; j <= z; ++j) x ^= (i >> j & 1) << (z - j);
33         if (x > i) swap(v[x], v[i]);
34     }
35 }
36
37 void ntt(vector<long long> &v, int n) {
38     bitrev(v, n);
39     for (int s = 2; s <= n; s <= 1) {
40         int z = s >> 1;
41         for (int i = 0; i < n; i += s) {
42             for (int k = 0; k < z; ++k) {
43                 long long x = v[i + k + z] * omega[maxn / s * k] % mod;
44                 v[i + k + z] = (v[i + k] + mod - x) % mod;
45                 (v[i + k] += x) %= mod;
46             }
47         }
48     }
49 }
50
51 void intt(vector<long long> &v, int n) {
52     ntt(v, n);
53     for (int i = 1; i < n / 2; ++i) swap(v[i], v[n - i]);
54     long long inv = fpow(n, -1);
55     for (int i = 0; i < n; ++i) (v[i] *= inv) %= mod;
56 }
57
58 vector<long long> operator()(vector<long long> a, vector<long long> b) {
59     int sz = 1;
60     while (sz < a.size() + b.size() - 1) sz <= 1;
61     while (a.size() < sz) a.push_back(0);
62     while (b.size() < sz) b.push_back(0);
63     ntt(a, sz), ntt(b, sz);
64     vector<long long> c(sz);
65     for (int i = 0; i < sz; ++i) c[i] = a[i] * b[i] % mod;
66     intt(c, sz);
67     while (c.size() && c.back() == 0) c.pop_back();
68     return c;
69 }
70
71 vector<long long> operator()(vector<long long> a, int n) {
72     int sz = 1;
73     while (sz < n * a.size()) sz <= 1;
74     while (a.size() < sz) a.push_back(0);
75     ntt(a, sz);
76     for (int i = 0; i < sz; ++i) a[i] = fpow(a[i], n);
77     intt(a, sz);
78     while (a.size() && a.back() == 0) a.pop_back();
79     return a;
80 }
81 };
82
83 NTT<mod, root> conv;

```

1.6.3 原根

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

```

2  #define ll long long
3  #define IL inline
4  #define RG register
5  using namespace std;
6
7  ll prm[1000],tot,N,root;
8
9  ll Power(ll bs,ll js,ll MOD){
10     ll S = 1,T = bs;
11     while(js){
12         if(js&1)S = S*T%MOD;
13         T = T*T%MOD;
14         js >>= 1;
15     } return S;
16 }
17
18 IL ll GetRoot(RG ll n){
19     RG ll tmp = n - 1 , tot = 0;
20     for(RG ll i = 2; i <= sqrt(tmp); i ++){
21         if(tmp%i==0){
22             prm[++tot] = i;
23             while(tmp%i==0)tmp /= i;
24         }
25     }
26     if(tmp != 1)prm[++tot] = tmp;           //质因数分解
27     for(RG ll g = 2; g <= n-1; g ++){
28         bool flag = 1;
29         for(RG int i = 1; i <= tot; i ++){   //检测是否符合条件
30             if(Power(g,(n-1)/prm[i],n) == 1)
31                 { flag = 0; break; }
32         }
33         if(flag)return g;
34     }return 0;                             //无解
35 }
36
37 int main(){
38     cin >> N;
39     root = GetRoot(N);
40     cout<<root<<endl;
41     return 0;
42 }

```

1.6.4 FWT

```

1  //C_k=\sum_{i \oplus j=k} A_i B_j$
2  //FWT 完后需要先模一遍
3  template<typename T>
4  void fwt(ll a[], int n, T f) {
5      for (int d = 1; d < n; d *= 2)
6          for (int i = 0, t = d * 2; i < n; i += t)
7              for(int j = 0; j < d; j ++)
8                  f(a[i + j], a[i + j + d]);
9  }
10
11 void AND(ll& a, ll& b) { a += b; }
12 void OR(ll& a, ll& b) { b += a; }
13 void XOR (ll& a, ll& b) {
14     ll x = a, y = b;

```

```
15     a = (x + y) % mod;
16     b = (x - y + mod) % mod;
17 }
18 void rAND(ll& a, ll& b) { a -= b; }
19 void rOR(ll& a, ll& b) { b -= a; }
20 void rXOR(ll& a, ll& b) {
21     static ll INV2 = (mod + 1) / 2;
22     ll x = a, y = b;
23     a = (x + y) * INV2 % mod;
24     b = (x - y + mod) * INV2 % mod;
25 }
26
27 //FWT 子集卷积
28 a[popcount(x)][x] = A[x]
29 b[popcount(x)][x] = B[x]
30 fwt(a[i]) fwt(b[i])
31 c[i + j][x] += a[i][x] * b[j][x]
32 rfw(c[i])
33 ans[x] = c[popcount(x)][x]
```

1.7 多项式

1.7.1 拉格朗日插值

```
1  typedef long long ll;
2
3  const int mod = 998244353;
4  const int maxn = 1e5 + 10;
5
6  int x[maxn], y[maxn];
7
8  int qp(int a, int n)
9  {
10     ll ans = 1, base = a;
11     for(; n; (base *= base) %= mod, n >>= 1) if(n & 1) (ans *= base) %= mod;
12     return ans;
13 }
14
15 int lagrange(int n, int *x, int *y, int xi)
16 {
17     int ans = 0;
18     for(int i = 0; i <= n; i++)
19     {
20         int s1 = 1, s2 = 1;
21         for(int j = 0; j <= n; j++) if(i != j)
22         {
23             s1 = 1ll * s1 * (xi - x[j]) % mod;
24             s2 = 1ll * s2 * (x[i] - x[j]) % mod;
25         }
26         ans = (1ll * ans + 1ll * y[i] * s1 % mod * qp(s2, mod - 2) % mod) % mod;
27     }
28     return (ans + mod) % mod;
29 }
```

1.7.2 拉格朗日插值 (连续取值)

```
1  const int mod = 'edit';
2  const int maxn = 'edit';
```

```

3
4 int x[maxn], y[maxn];
5 int s1[maxn], s2[maxn], ifac[maxn];
6
7 //如果x的取值是连续一段, 可以做到O(n)求解
8 int lagrange(int n, int *x, int *y, int xi)
9 {
10     int ans = 0;
11     s1[0] = (xi - x[0]) % mod, s2[n + 1] = 1;
12     for(int i = 1; i <= n; i++) s1[i] = 1ll * s1[i - 1] * (xi - x[i]) % mod;
13     for(int i = n; i >= 0; i--) s2[i] = 1ll * s2[i + 1] * (xi - x[i]) % mod;
14     ifac[0] = ifac[1] = 1;
15     for(int i = 2; i <= n; i++) ifac[i] = -1ll * mod / i * ifac[mod % i] % mod;
16     for(int i = 2; i <= n; i++) ifac[i] = 1ll * ifac[i] * ifac[i - 1] % mod;
17     for(int i = 0; i <= n; i++)
18         (ans += 1ll * y[i] * (i == 0 ? 1 : s1[i - 1]) % mod * s2[i + 1] % mod * ifac[i] % mod * (((
19             n - i) & 1) ? -1 : 1) * ifac[n - i] % mod) %= mod;
20     return (ans + mod) % mod;
21 }

```

1.8 Others

1.8.1 BM

```

1 //Berlekamp-Massey
2 typedef vector<int> VI;
3 namespace linear_seq
4 {
5     #define rep(i,a,n) for (int i=a;i<n;i++)
6     #define SZ(x) ((int)(x).size())
7     #define pb(x) push_back(x)
8     const ll mod=1e9+7;
9     ll powmod(ll a,ll b){ll res=1;a%=mod; assert(b>=0); for(;b>=>1){if(b&1)res=res*a%mod;a=a%mod;}return res;}
10     const int N=10010;
11     ll res[N],base[N],_c[N],_md[N];
12     vector<int> Md;
13     void mul(ll *a,ll *b,int k)
14     {
15         rep(i,0,k+k) _c[i]=0;
16         rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
17         for (int i=k+k-1;i>=k;i--) if (_c[i])
18             rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
19         rep(i,0,k) a[i]=_c[i];
20     }
21     int solve(ll n,VI a,VI b){
22         ll ans=0,pnt=0;
23         int k=SZ(a);
24         assert(SZ(a)==SZ(b));
25         rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
26         Md.clear();
27         rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
28         rep(i,0,k) res[i]=base[i]=0;
29         res[0]=1;
30         while ((1ll<<pnt)<=n) pnt++;
31         for (int p=pnt;p>=0;p--) {
32             mul(res,res,k);
33             if ((n>>p)&1) {
34                 for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;

```

```

35         rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
36     }
37 }
38 rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
39 if (ans<0) ans+=mod;
40 return ans;
41 }
42 VI BM(VI s){
43     VI C(1,1),B(1,1);
44     int L=0,m=1,b=1;
45     rep(n,0,SZ(s)){
46         ll d=0;
47         rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
48         if(d==0) ++m;
49         else if(2*L<=n){
50             VI T=C;
51             ll c=mod-d*powmod(b,mod-2)%mod;
52             while (SZ(C)<SZ(B)+m) C.pb(0);
53             rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
54             L=n+1-L; B=T; b=d; m=1;
55         } else {
56             ll c=mod-d*powmod(b,mod-2)%mod;
57             while (SZ(C)<SZ(B)+m) C.pb(0);
58             rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
59             ++m;
60         }
61     }
62     return C;
63 }
64 int gao(VI a,ll n)
65 {
66     VI c=BM(a);
67     c.erase(c.begin());
68     rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
69     return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
70 }
71 };//linear_seq::gao(VI{},n-1)

```

1.8.2 exBM

```

1 // given first m items init[0..m-1] and coefficients trans[0..m-1] or
2 // given first 2 *m items init[0..2m-1], it will compute trans[0..m-1]
3 // for you. trans[0..m] should be given as that
4 //     init[m] = sum_{i=0}^{m-1} init[i] * trans[i]
5 struct LinearRecurrence
6 {
7     using int64 = long long;
8     using vec = std::vector<int64>;
9
10    static void extand(vec& a, size_t d, int64 value = 0)
11    {
12        if (d <= a.size()) return;
13        a.resize(d, value);
14    }
15    static vec BerlekampMassey(const vec& s, int64 mod)
16    {
17        std::function<int64(int64)> inverse = [&](int64 a) {
18            return a == 1 ? 1 : (int64)(mod - mod / a) * inverse(mod % a) % mod;

```

```
19     };
20     vec A = {1}, B = {1};
21     int64 b = s[0];
22     for (size_t i = 1, m = 1; i < s.size(); ++i, m++)
23     {
24         int64 d = 0;
25         for (size_t j = 0; j < A.size(); ++j)
26         {
27             d += A[j] * s[i - j] % mod;
28         }
29         if (!(d % mod)) continue;
30         if (2 * (A.size() - 1) <= i)
31         {
32             auto temp = A;
33             extend(A, B.size() + m);
34             int64 coef = d * inverse(b) % mod;
35             for (size_t j = 0; j < B.size(); ++j)
36             {
37                 A[j + m] -= coef * B[j] % mod;
38                 if (A[j + m] < 0) A[j + m] += mod;
39             }
40             B = temp, b = d, m = 0;
41         }
42         else
43         {
44             extend(A, B.size() + m);
45             int64 coef = d * inverse(b) % mod;
46             for (size_t j = 0; j < B.size(); ++j)
47             {
48                 A[j + m] -= coef * B[j] % mod;
49                 if (A[j + m] < 0) A[j + m] += mod;
50             }
51         }
52     }
53     return A;
54 }
55 static void exgcd(int64 a, int64 b, int64& g, int64& x, int64& y)
56 {
57     if (!b)
58         x = 1, y = 0, g = a;
59     else
60     {
61         exgcd(b, a % b, g, y, x);
62         y -= x * (a / b);
63     }
64 }
65 static int64 crt(const vec& c, const vec& m)
66 {
67     int n = c.size();
68     int64 M = 1, ans = 0;
69     for (int i = 0; i < n; ++i) M *= m[i];
70     for (int i = 0; i < n; ++i)
71     {
72         int64 x, y, g, tm = M / m[i];
73         exgcd(tm, m[i], g, x, y);
74         ans = (ans + tm * x * c[i] % M) % M;
75     }
76     return (ans + M) % M;
77 }
```

```

78 static vec ReedsSloane(const vec& s, int64 mod)
79 {
80     auto inverse = [](int64 a, int64 m) {
81         int64 d, x, y;
82         exgcd(a, m, d, x, y);
83         return d == 1 ? (x % m + m) % m : -1;
84     };
85     auto L = [](const vec& a, const vec& b) {
86         int da = (a.size() > 1 || (a.size() == 1 && a[0])) ? a.size() - 1 : -1000;
87         int db = (b.size() > 1 || (b.size() == 1 && b[0])) ? b.size() - 1 : -1000;
88         return std::max(da, db + 1);
89     };
90     auto prime_power = [&](const vec& s, int64 mod, int64 p, int64 e) {
91         // linear feedback shift register mod p^e, p is prime
92         std::vector<vec> a(e), b(e), an(e), bn(e), ao(e), bo(e);
93         vec t(e), u(e), r(e), to(e, 1), uo(e), pw(e + 1);
94         ;
95         pw[0] = 1;
96         for (int i = pw[0] = 1; i <= e; ++i) pw[i] = pw[i - 1] * p;
97         for (int64 i = 0; i < e; ++i)
98         {
99             a[i] = {pw[i]}, an[i] = {pw[i]};
100            b[i] = {0}, bn[i] = {s[0] * pw[i] % mod};
101            t[i] = s[0] * pw[i] % mod;
102            if (t[i] == 0)
103            {
104                t[i] = 1, u[i] = e;
105            }
106            else
107            {
108                for (u[i] = 0; t[i] % p == 0; t[i] /= p, ++u[i])
109                ;
110            }
111        }
112        for (size_t k = 1; k < s.size(); ++k)
113        {
114            for (int g = 0; g < e; ++g)
115            {
116                if (L(an[g], bn[g]) > L(a[g], b[g]))
117                {
118                    ao[g] = a[e - 1 - u[g]];
119                    bo[g] = b[e - 1 - u[g]];
120                    to[g] = t[e - 1 - u[g]];
121                    uo[g] = u[e - 1 - u[g]];
122                    r[g] = k - 1;
123                }
124            }
125            a = an, b = bn;
126            for (int o = 0; o < e; ++o)
127            {
128                int64 d = 0;
129                for (size_t i = 0; i < a[o].size() && i <= k; ++i)
130                {
131                    d = (d + a[o][i] * s[k - i]) % mod;
132                }
133                if (d == 0)
134                {
135                    t[o] = 1, u[o] = e;
136                }

```

```

137         else
138         {
139             for (u[o] = 0, t[o] = d; t[o] % p == 0; t[o] /= p, ++u[o])
140                 ;
141             int g = e - 1 - u[o];
142             if (L(a[g], b[g]) == 0)
143             {
144                 extend(bn[o], k + 1);
145                 bn[o][k] = (bn[o][k] + d) % mod;
146             }
147             else
148             {
149                 int64 coef = t[o] * inverse(to[g], mod) % mod * pw[u[o] - uo[g]] % mod;
150                 int m = k - r[g];
151                 extend(an[o], ao[g].size() + m);
152                 extend(bn[o], bo[g].size() + m);
153                 for (size_t i = 0; i < ao[g].size(); ++i)
154                 {
155                     an[o][i + m] -= coef * ao[g][i] % mod;
156                     if (an[o][i + m] < 0) an[o][i + m] += mod;
157                 }
158                 while (an[o].size() && an[o].back() == 0) an[o].pop_back();
159                 for (size_t i = 0; i < bo[g].size(); ++i)
160                 {
161                     bn[o][i + m] -= coef * bo[g][i] % mod;
162                     if (bn[o][i + m] < 0) bn[o][i + m] -= mod;
163                 }
164                 while (bn[o].size() && bn[o].back() == 0) bn[o].pop_back();
165             }
166         }
167     }
168 }
169 return std::make_pair(an[0], bn[0]);
170 };
171
172 std::vector<std::tuple<int64, int64, int>> fac;
173 for (int64 i = 2; i * i <= mod; ++i)
174 {
175     if (mod % i == 0)
176     {
177         int64 cnt = 0, pw = 1;
178         while (mod % i == 0) mod /= i, ++cnt, pw *= i;
179         fac.emplace_back(pw, i, cnt);
180     }
181 }
182 if (mod > 1) fac.emplace_back(mod, mod, 1);
183 std::vector<vec> as;
184 size_t n = 0;
185 for (auto&& x : fac)
186 {
187     int64 mod, p, e;
188     vec a, b;
189     std::tie(mod, p, e) = x;
190     auto ss = s;
191     for (auto&& x : ss) x %= mod;
192     std::tie(a, b) = prime_power(ss, mod, p, e);
193     as.emplace_back(a);
194     n = std::max(n, a.size());
195 }

```



```
196     vec a(n), c(as.size()), m(as.size());
197     for (size_t i = 0; i < n; ++i)
198     {
199         for (size_t j = 0; j < as.size(); ++j)
200         {
201             m[j] = std::get<0>(fac[j]);
202             c[j] = i < as[j].size() ? as[j][i] : 0;
203         }
204         a[i] = crt(c, m);
205     }
206     return a;
207 }
208
209 LinearRecurrence(const vec& s, const vec& c, int64 mod) : init(s), trans(c), mod(mod), m(s.size
    ()) {}
210 LinearRecurrence(const vec& s, int64 mod, bool is_prime = true) : mod(mod)
211 {
212     vec A;
213     if (is_prime)
214         A = BerlekampMassey(s, mod);
215     else
216         A = ReedsSloane(s, mod);
217     if (A.empty()) A = {0};
218     m = A.size() - 1;
219     trans.resize(m);
220     for (int i = 0; i < m; ++i)
221     {
222         trans[i] = (mod - A[i + 1]) % mod;
223     }
224     std::reverse(trans.begin(), trans.end());
225     init = {s.begin(), s.begin() + m};
226 }
227 int64 calc(int64 n)
228 {
229     if (mod == 1) return 0;
230     if (n < m) return init[n];
231     vec v(m), u(m << 1);
232     int msk = !n;
233     for (int64 m = n; m > 1; m >>= 1) msk <<= 1;
234     v[0] = 1 % mod;
235     for (int x = 0; msk; msk >>= 1, x <<= 1)
236     {
237         std::fill_n(u.begin(), m * 2, 0);
238         x |= !(n & msk);
239         if (x < m)
240             u[x] = 1 % mod;
241         else
242             { // can be optimized by fft/ntt
243                 for (int i = 0; i < m; ++i)
244                 {
245                     for (int j = 0, t = i + (x & 1); j < m; ++j, ++t)
246                     {
247                         u[t] = (u[t] + v[i] * v[j]) % mod;
248                     }
249                 }
250                 for (int i = m * 2 - 1; i >= m; --i)
251                 {
252                     for (int j = 0, t = i - m; j < m; ++j, ++t)
253                     {
```

```
254         u[t] = (u[t] + trans[j] * u[i]) % mod;
255     }
256 }
257 }
258     v = {u.begin(), u.begin() + m};
259 }
260     int64 ret = 0;
261     for (int i = 0; i < m; ++i)
262     {
263         ret = (ret + v[i] * init[i]) % mod;
264     }
265     return ret;
266 }
267
268     vec init, trans;
269     int64 mod;
270     int m;
271 };
```

1.8.3 杜教筛

```
1  #include <bits/stdc++.h>
2  #include <tr1/unordered_map>
3
4  using namespace std;
5  typedef long long ll;
6
7  const int N = 5e6;
8
9  bool vis[N + 1];
10 int mu[N + 1], sumu[N + 1], prim[N + 1], cnt;
11
12 tr1::unordered_map<int, int> Smu;
13
14 void get_mu(int n)
15 {
16     mu[1] = 1;
17     for(int i = 2; i <= n; i++)
18     {
19         if(!vis[i]) { prim[++ cnt] = i; mu[i] = -1; }
20         for(int j = 1; j <= cnt && prim[j] * i <= n; j++)
21         {
22             vis[prim[j] * i] = 1;
23             if(i % prim[j] == 0) break;
24             else mu[i * prim[j]] = -mu[i];
25         }
26     }
27     for(int i = 1; i <= n; i++) sumu[i] = sumu[i - 1] + mu[i];
28 }
29
30 int phi[N + 1]; ll sumphi[N + 1];
31 tr1::unordered_map<ll, ll> Sphi;
32
33 void get(int n)
34 {
35     phi[1] = mu[1] = 1;
36     for(int i = 2; i <= n; i++)
37     {
```

```

38     if(!vis[i])
39     {
40         prim[++ cnt] = i;
41         mu[i] = -1; phi[i] = i - 1;
42     }
43     for(int j = 1; j <= cnt && prim[j] * i <= n; j ++)
44     {
45         vis[i * prim[j]] = 1;
46         if(i % prim[j] == 0)
47         {
48             phi[i * prim[j]] = phi[i] * prim[j];
49             break;
50         }
51         else mu[i * prim[j]] = -mu[i], phi[i * prim[j]] = phi[i] * (prim[j] - 1);
52     }
53
54 }
55 for(int i = 1; i <= n; i ++) sumu[i] = sumu[i - 1] + mu[i], sumphi[i] = sumphi[i - 1] + phi[i];
56 }
57
58 ll getSum_mu(int x)
59 {
60     if(x <= N) return sumu[x];
61     if(Smu[x]) return Smu[x];
62     int ans = 1;
63     for(int l = 2, r; l >= 0 && l <= x && r < 2147483647; l = r + 1)
64     {
65         r = x / (x / l);
66         ans -= (r - l + 1) * getSum_mu(x / l);
67     }
68     return Smu[x] = ans;
69 }
70
71 ll getSum_phi(ll x)
72 {
73     if(x <= N) return sumphi[x];
74     if(Sphi[x]) return Sphi[x];
75     ll ans = x * (x + 1) / 2;
76     for(ll l = 2, r; l <= x; l = r + 1)
77     {
78         r = x / (x / l);
79         ans -= (r - l + 1) * getSum_phi(x / l);
80     }
81     return Sphi[x] = ans;
82 }

```

1.8.4 欧拉降幂

```

1  const int maxn = 1e7+50;
2
3  int prim[maxn], vis[maxn];
4  int tot, phi[maxn];
5  struct node {
6      ll res;
7      bool v;
8  };
9
10 node qpow(ll A, ll B, ll C) {

```

```
11     ll re = 1;
12     bool flag = true;
13     while (B) {
14         if (B & 1) {
15             if ((re * A) >= C) flag = 0;
16             re = re % C;
17         }
18         B = B >> 1;
19         if (B) {
20             if (A >= C) flag = 0;
21             A %= C;
22             if ((A * A) >= C) flag = 0;
23             A %= C;
24         }
25     }
26     return node{re, flag};
27 }
28
29 void init(int n) {
30     phi[1] = 1;
31     for (int i = 2; i <= n; i++) {
32         if (!vis[i]) {
33             prim[++tot] = i;
34             phi[i] = i - 1;
35         }
36         for (int j = 1; j <= tot && prim[j] * i <= n; j++) {
37             vis[i * prim[j]] = 1;
38             if (i % prim[j] == 0) {
39                 phi[i * prim[j]] = phi[i] * prim[j];
40                 break;
41             } else phi[i * prim[j]] = phi[i] * (prim[j] - 1);
42         }
43     }
44 }
45 }
46
47
48 inline ll Euler(ll x) {
49     if (x <= maxn) return phi[x];
50     return 0;
51 }
52
53 node f(ll a, ll k, ll p) {
54     if (p == 1) return node{0, 0};
55     if (k == 0) return node{a % p, a < p};
56     ll ep = Euler(p);
57     node tmp = f(a, k - 1, ep);
58     if (__gcd(a, p) == 1) return qpow(a, tmp.res, p);
59     if (!tmp.v) {
60         tmp.res += ep;
61     }
62     return qpow(a, tmp.res, p);
63 }
64
65 int main() {
66     ll a, k, p;
67     init(1e7+2);
68     int T;
69     scanf("%d", &T);
```

```

70     for (int kase = 1; kase <= T; ++kase) {
71         // k次a次方模p的值
72         scanf("%lld%lld%lld", &a, &k, &p);
73         if (k == 0) printf("%lld\n", 1 % p);
74         else printf("%lld\n", f(a, k - 1, p).res);
75     }
76     return 0;
77 }

```

1.8.5 公式

- 约数定理：若 $n = \prod_{i=1}^k p_i^{a_i}$ ，则
 - 约数个数 $f(n) = \prod_{i=1}^k (a_i + 1)$
 - 约数和 $g(n) = \prod_{i=1}^k (\sum_{j=0}^{a_i} p_i^j)$
- 小于 n 且互素的数之和为 $n\varphi(n)/2$
- 若 $\gcd(n, i) = 1$ ，则 $\gcd(n, n - i) = 1 (1 \leq i \leq n)$
- 错排公式： $D(n) = (n - 1)(D(n - 2) + D(n - 1)) = \sum_{i=2}^n \frac{(-1)^i n!}{i!} = \lfloor \frac{n!}{e} + 0.5 \rfloor$
- 部分错排公式： $n + m$ 个数中 m 个数必须错排求排列数
 - $1 \text{ dp}[i] = n * \text{dp}[i - 1] + (i - 1) * (\text{dp}[i - 1] + \text{dp}[i - 2]);$
 - $2 \text{ dp}[0] = n!;$
 - $3 \text{ dp}[1] = n * n!;$
 - $\text{dp}[m]$ 为所求解
- 海伦公式： $S = \sqrt{p(p - a)(p - b)(p - c)}$ ，其中 $p = \frac{a + b + c}{2}$
- 求 $C(n, k)$ 中素因子 P 的个数：把 n 转化为 P 进制，并记它每个位上的和为 $S1$ 把 $n - k$ ， k 做同样的处理，得到 $S2$ ， $S3$ 则答案为： $\frac{S2 + S3 - S1}{P - 1}$
- 威尔逊定理： $p \text{ is prime} \Rightarrow (p - 1)! \equiv -1 \pmod{p}$
- 欧拉定理： $\gcd(a, n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1 \pmod{n}$
- 欧拉定理推广： $\gcd(n, p) = 1 \Rightarrow a^n \equiv a^{n \% \varphi(p)} \pmod{p}$
- 模的幂公式： $a^n \pmod{m} = \begin{cases} a^n \pmod{m} & n < \varphi(m) \\ a^{n \% \varphi(m) + \varphi(m)} \pmod{m} & n \geq \varphi(m) \end{cases}$
- 素数定理：对于不大于 n 的素数个数 $\pi(n)$ ， $\lim_{n \rightarrow \infty} \pi(n) = \frac{n}{\ln n}$
- 位数公式：正整数 x 的位数 $N = \log_{10}(n) + 1$
- 斯特灵公式 $n! \approx \sqrt{2\pi n} (\frac{n}{e})^n$
- 设 $a > 1, m, n > 0$ ，则 $\gcd(a^m - 1, a^n - 1) = a^{\gcd(m, n)} - 1$
- 设 $a > b, \gcd(a, b) = 1$ ，则 $\gcd(a^m - b^m, a^n - b^n) = a^{\gcd(m, n)} - b^{\gcd(m, n)}$

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

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

- 求和公式：

- $\sum k = \frac{n(n+1)}{2}$
- $\sum 2k - 1 = n^2$
- $\sum k^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum (2k - 1)^2 = \frac{n(4n^2 - 1)}{3}$

- (e) $\sum k^3 = \left(\frac{n(n+1)}{2}\right)^2$
 (f) $\sum (2k-1)^3 = n^2(2n^2-1)$
 (g) $\sum k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
 (h) $\sum k^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$
 (i) $\sum k(k+1) = \frac{n(n+1)(n+2)}{3}$
 (j) $\sum k(k+1)(k+2) = \frac{n(n+1)(n+2)(n+3)}{4}$
 (k) $\sum k(k+1)(k+2)(k+3) = \frac{n(n+1)(n+2)(n+3)(n+4)}{5}$

18. 若 $\gcd(m, n) = 1$, 则:

- (a) 最大不能组合的数为 $m * n - m - n$
 (b) 不能组合数个数 $N = \frac{(m-1)(n-1)}{2}$

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

20. 若 p 为素数, 则 $(x+y+\dots+w)^p \equiv x^p + y^p + \dots + w^p \pmod{p}$

21. 卡特兰数: 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012

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

22. 伯努利数: $B_n = -\frac{1}{n+1} \sum_{i=0}^{n-1} C_{n+1}^i B_i$

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

23. 二项式反演:

$$f_n = \sum_{i=0}^n (-1)^i \binom{n}{i} g_i \Leftrightarrow g_n = \sum_{i=0}^n (-1)^i \binom{n}{i} f_i$$

$$f_n = \sum_{i=0}^n \binom{n}{i} g_i \Leftrightarrow g_n = \sum_{i=0}^n (-1)^{n-i} \binom{n}{i} f_i$$

24. 莫比乌斯反演:

- (a) 令 $f(d) = \sum_{i=1}^n \sum_{j=1}^m [\gcd(i, j) = d]$
 (b) $F(n) = \sum_{n|d} f(d) = \lfloor \frac{N}{n} \rfloor \lfloor \frac{M}{n} \rfloor$
 (c) 有 $f(n) = \sum_{n|d} \mu(\lfloor \frac{d}{n} \rfloor) F(d)$
 (d) $\phi(n) = \sum_{d|n} d * \mu(n/d)$

25. 2 的 n 次方, 在 pow 时可以精确输出最大 2^{1023} , pow(2,1023)

26. FFT 常用素数

$r \cdot 2^k + 1$	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	119	23	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
194555039024054273	27	56	5
4179340454199820289	29	57	3

1.8.6 博弈

- 1 // `bash` 博弈, n 个物品, 轮流取 $[1, m]$ 个物品, 无法取则失败
- 2 // 当且仅当 $n = (m + 1) * r$ 时先手败
- 3 // `Nim` 博弈: 每轮从若干堆石子中的一堆取走若干颗。先手必胜条件为石子数量异或非零

2 Graph Theory

2.1 路径

2.1.1 Dijkstra

```
1  const int maxn = 1e5 + 10;
2  const int inf = 0x3f3f3f3f;
3
4  int head[maxn], dis[maxn], cnt, n;
5
6  struct Edge { int nex,to,w; }edge[20*maxn];
7
8  void add(int u,int v,int w)
9  {
10     edge[++cnt].nex=head[u];
11     edge[cnt].w=w;
12     edge[cnt].to=v;
13     head[u]=cnt;
14 }
15
16 void dijkstra(int s)
17 {
18     priority_queue<pair<int, int>, vector<pair<int, int> >, greater<pair<int, int> > > que;
19     memset(dis, 0x3f, sizeof dis);
20     que.push({0, s}); dis[s] = 0;
21     while(!que.empty())
22     {
23         auto f = que.top(); que.pop();
24         int u = f.second, d = f.first;
25         if(d != dis[u]) continue;
26         for(int i = head[u]; ~i; i = edge[i].nex)
27         {
28             int v = edge[i].to, w = edge[i].w;
29             if(dis[u] + w < dis[v])
30             {
31                 dis[v] = dis[u] + w;
32                 que.push({dis[v], v});
33             }
34         }
35     }
36 }
```

2.1.2 Euler Path

```
1  int S[N << 1], top;
2  Edge edges[N << 1];
3  set<int> G[N];
4
5  void DFS(int u) {
6      S[top++] = u;
7      for (int eid: G[u])
8      {
9          int v = edges[eid].get_other(u);
10         G[u].erase(eid);
11         G[v].erase(eid);
12         DFS(v);
13         return;
14     }
```



```
15 }
16
17 void fleury(int start)
18 {
19     int u = start;
20     top = 0; path.clear();
21     S[top++] = u;
22     while (top)
23     {
24         u = S[--top];
25         if (!G[u].empty())
26             DFS(u);
27         else path.push_back(u);
28     }
29 }
```

2.1.3 K shortest Path(Astar)

```
1  const int inf = 0x3f3f3f3f;
2  const int maxn = 1000 + 10;
3  const int maxm = 100000 + 10;
4
5  int n, k, cnt, head[maxn], revhead[maxn], dis[maxn];
6  bool vis[maxn];
7
8  struct node { int v, w, nex; } edge[maxm], revedge[maxm];
9
10 void init()
11 {
12     cnt = 0;
13     memset(head, 0xff, sizeof head);
14     memset(revhead, 0xff, sizeof revhead);
15 }
16
17 void add(int u, int v, int w)
18 {
19     edge[cnt].v = v, revedge[cnt].v = u;
20     edge[cnt].w = revedge[cnt].w = w;
21     edge[cnt].nex = head[u];
22     revedge[cnt].nex = revhead[v];
23     head[u] = revhead[v] = cnt;
24     cnt++;
25 }
26
27 void spfa(int src)    //建立反向图，求图中所有点到终点的最短路径
28 {
29     for (int i = 1; i <= n; i++) dis[i] = inf;
30     memset(vis, false, sizeof vis);
31     vis[src] = 0;
32     queue<int> que;
33     que.push(src);
34     dis[src] = 0;
35     while (!que.empty())
36     {
37         int u = que.front();
38         que.pop();
39         vis[u] = false;
40         for (int i = revhead[u]; ~i; i = revedge[i].nex)
```

```

41     {
42         int v = revedge[i].v, w = revedge[i].w;
43         if (dis[v] > dis[u] + w)
44         {
45             dis[v] = dis[u] + w;
46             if (!vis[v])
47             {
48                 que.push(v);
49                 vis[v] = true;
50             }
51         }
52     }
53 }
54 }
55
56 struct A
57 {
58     int f, g, h;    //f(n),g(n),h(n)函数
59     int id;         //当前点的编号
60     bool operator<(const A a) const
61     {
62         //定义比较函数
63         if (a.f == f) return a.g < g;
64         return a.f < f;
65     };
66
67     int Astar(int src, int des)
68     {
69         int cnt = 0;
70         priority_queue<A> Q;
71         if (src == des) k++;    //如果起点即为终点
72         if (dis[src] == inf) return -1;    //如果起点不能到达终点
73         A st, now, tmp;
74         st.id = src, st.g = 0, st.f = st.g + dis[src];    //定义起始节点
75         Q.push(st);
76         while (!Q.empty())
77         {
78             now = Q.top();
79             Q.pop();
80             if (now.id == des)    //如果当前节点为终点
81             {
82                 cnt++;
83                 if (cnt == k) return now.g;    //找到第k短路
84             }
85             for (int i = head[now.id]; ~i; i = edge[i].nex)
86             {
87                 tmp.id = edge[i].v;
88                 tmp.g = now.g + edge[i].w;    //到该点的实际花费
89                 tmp.f = tmp.g + dis[tmp.id];    //到最终状态的估计花费
90                 Q.push(tmp);
91             }
92         }
93         return -1;    //路径总数小于k
94     }
95
96     int main()
97     {
98         int m, s, t, u, v, w;
99         while (scanf("%d%d", &n, &m) != EOF)

```

```

100     {
101         init();
102         while (m--)
103         {
104             scanf("%d%d%d", &u, &v, &w);
105             add(u, v, w);
106         }
107         scanf("%d%d%d", &s, &t, &k);
108         spfa(t);    //求所有点到终点的最短路
109         printf("%d\n", Astar(s, t));
110     }
111     return 0;
112 }

```

2.1.4 K shortest Path(可持久化可并堆)

```

1  #include <bits/stdc++.h>
2  #include<ext/pb_ds/priority_queue.hpp>
3
4  using namespace std;
5
6  const int N = '';
7  const int M = '';
8  const int logM = 20;
9  const int inf = 0x3f3f3f3f;
10
11 int n, m, k, S, T;
12
13 struct Edge{ int nex, to, w; };
14
15 struct Graph
16 {
17     int head[N], cnt;
18     Edge edge[M];
19     void init(int n) { for(int i = 0; i <= n; i++) head[i] = 0; cnt = 0; }
20     void addedge(int u, int v, int val) { edge[++ cnt].nex = head[u], edge[cnt].to = v, edge[cnt].w
        = val, head[u] = cnt; }
21 }g, rg;
22
23 int dis[N];
24
25 void dijkstra()
26 {
27     priority_queue<pair<int, int>, vector<pair<int, int> >, greater<pair<int, int> > > que;
28     memset(dis, inf, sizeof dis);
29     que.push({0, T}); dis[T] = 0;
30     const int *head = rg.head; const Edge *edge = rg.edge;
31     while(!que.empty())
32     {
33         auto f = que.top(); que.pop();
34         int u = f.second, d = f.first;
35         if(d != dis[u]) continue;
36         for(int i = head[u]; i; i = edge[i].nex)
37         {
38             int v = edge[i].to, w = edge[i].w;
39             if(dis[u] + w < dis[v]) { dis[v] = dis[u] + w; que.push({dis[v], v}); }
40         }
41     }

```

```
42 }
43
44 bool tree_edge[M], vis[N];
45 int fa[N], st[N], top;
46
47 void dfs(int u)
48 {
49     vis[u] = true;
50     st[++ top] = u;
51     for(int i = rg.head[u]; i; i = rg.edge[i].nex)
52     {
53         int v = rg.edge[i].to;
54         if(!vis[v] && dis[v] == dis[u] + rg.edge[i].w)
55         {
56             fa[v] = u;
57             tree_edge[i] = true;
58             dfs(v);
59         }
60     }
61 }
62
63 namespace LT
64 {
65     int son[M * logM][2];
66     int ht[M * logM], val[M * logM], id[M * logM];
67     int tot;
68
69     int newnode(int _val, int _id, int _dis = 0)
70     {
71         int now = ++ tot;
72         val[now] = _val, id[now] = _id;
73         ht[now] = _dis, son[now][0] = son[now][1] = 0;
74         return now;
75     }
76
77     int _copy(int ori)
78     {
79         int now = ++tot;
80         val[now] = val[ori], id[now] = id[ori];
81         ht[now] = ht[ori], son[now][0] = son[ori][0], son[now][1] = son[ori][1];
82         return now;
83     }
84
85     int merge(int a, int b)
86     {
87         if(!a || !b) return a | b;
88         if(val[a] > val[b]) swap(a, b);
89         int now = _copy(a);
90         son[now][1] = merge(son[now][1], b);
91         if(ht[son[now][0]] < ht[son[now][1]]) swap(son[now][0], son[now][1]);
92         ht[now] = ht[son[now][1]] + 1;
93         return now;
94     }
95
96     void insert(int &rt, int val, int id) { rt = merge(newnode(val, id), rt); }
97 }
98
99 int rt[M];
100
```

```
101 void build_heap()
102 {
103     for(int i = 1; i <= top; i++)
104     {
105         int u = st[i];
106         rt[u] = rt[fa[u]];
107         for(int i = g.head[u]; i; i = g.edge[i].nex)
108         {
109             int v = g.edge[i].to;
110             if(!tree_edge[i] && dis[v] != inf) LT::insert(rt[u], dis[v] - dis[u] + g.edge[i].w, v);
111         }
112     }
113 }
114
115 int solve(int k)
116 {
117     if(k == 1) return dis[S];
118     __gnu_pbds::priority_queue<pair<int, int>, greater<pair<int, int>>> que;
119     que.push({dis[S] + LT::val[rt[S]], rt[S]});
120     while(!que.empty())
121     {
122         pair<int, int> f = que.top(); que.pop();
123         if(--k == 1) return f.first;
124         int v = f.first, u = f.second;
125         int lc = LT::son[u][0], rc = LT::son[u][1], o = LT::id[u];
126         if(rt[o]) que.push({v + LT::val[rt[o]], rt[o]});
127         if(lc) que.push({v + LT::val[lc] - LT::val[u], lc});
128         if(rc) que.push({v + LT::val[rc] - LT::val[u], rc});
129     }
130     return -1;
131 }
132
133 void init()
134 {
135     g.init(n), rg.init(n);
136     memset(rt, 0, sizeof rt);
137     memset(tree_edge, 0, sizeof tree_edge);
138     top = LT::tot = 0;
139 }
140
141 void getans()
142 {
143     //input S-T
144     init();
145     dijkstra();
146     dfs(T);
147     build_heap();
148     cout << solve(k);
149 }
```

2.2 生成树

2.2.1 Kruskal

```
1 const int maxn = 1e5 + 10;
2
3 int n, m, pre[maxn];
4 struct edge {int u, v, w; } es[maxn];
5 int Find(int x) { return x == pre[x] ? x : pre[x] = Find(pre[x]); }
```

```
6 bool cmp(const edge &x, const edge &y) { return x.cost < y.cost; }
7
8 int kruskal()
9 {
10     sort(es, es + m, cmp);
11     int res = 0;
12     for(int i = 0; i < m; i ++)
13     {
14         int fx = Find(es[i].u), fy = Find(es[i].v);
15         if(fx != fy) pre[fx] = fy, res += es[i].cost;
16     }
17     return res;
18 }
```

2.2.2 Prim

```
1 const int maxn = 1000 + 10;
2 const int inf = 0x3f3f3f3f;
3
4 int n, mp[maxn][maxn], cost[maxn];
5 bool vis[maxn];
6
7 int prim()
8 {
9     for(int i = 0; i < n; i ++) cost[i] = inf, vis[i] = false;
10    int res = 0; cost[0] = 0;
11    for(;;)
12    {
13        int v = -1;
14        for(int u = 0; u < n; u ++)
15            if(!vis[u] && (v == -1 || cost[u] < cost[v])) v = u;
16        if(v == -1) break;
17        res += cost[v];
18        vis[v] = true;
19        for(int u = 0; u < n; u ++) cost[u] = min(cost[u], mp[v][u]);
20    }
21    return res;
22 }
```

2.2.3 最小树形图

```
1 const int INF = 0x3f3f3f3f;
2 const int maxn = 10000;
3 const int maxm = 10000;
4
5 struct Edge{int u,v,cost; } edge[maxm];
6
7 int pre[maxn], id[maxn], vis[maxn], in[maxn];
8
9 int zhuliu(int root, int n, int m)
10 {
11     int res=0, u, v;
12     for(;;)
13     {
14         for(int i=0; i<n; i++) in[i] = INF;
15         for(int i=0; i<m; i++) if(edge[i].u != edge[i].v && edge[i].cost < in[edge[i].v])
16         {
17             pre[edge[i].v] = edge[i].u;
```

```
18         in[edge[i].v] = edge[i].cost;
19     }
20     for(int i=0; i<n; i++) if(i != root && in[i] ==INF) return -1;
21     int tn=0;
22     memset(id, 0xff, sizeof id);
23     memset(vis, 0xff, sizeof vis);
24     in[root] = 0;
25     for(int i=0; i<n;i++)
26     {
27         res += in[i];
28         v = i;
29         while( vis[v] != i && id[v] == -1 && v!= root) vis[v] = i, v = pre[v];
30         if(v != root && id[v] == -1)
31         {
32             for(int u = pre[v]; u != v; u = pre[u]) id[u] = tn;
33             id[v] = tn++;
34         }
35     }
36     if(tn == 0) break;
37     for(int i=0; i<n; i++) if(id[i] == -1) id[i] = tn++;
38     for(int i=0; i<m; )
39     {
40         v = edge[i].v;
41         edge[i].u = id[edge[i].u];
42         edge[i].v = id[edge[i].v];
43         if(edge[i].u != edge[i].v) edge[i++].cost -= in[v];
44         else swap(edge[i], edge[--m]);
45     }
46     n = tn;
47     root = id[root];
48 }
49 return res;
50 }
```

2.2.4 Matrix Tree

```
1  const int N = 305;
2  const int mod = 1e9 + 7;
3
4  int n, m, a[N][N];
5
6  int Gauss(int n) {
7      int ans = 1;
8      for (int i = 1; i <= n; i++) {
9          for (int k = i + 1; k <= n; k++) {
10             while (a[k][i]) {
11                 int d = a[i][i] / a[k][i];
12                 for (int j = i; j <= n; j++) {
13                     a[i][j] = (a[i][j] - 1LL * d * a[k][j] % mod + mod) % mod;
14                 }
15                 std::swap(a[i], a[k]);
16                 ans = - ans;
17             }
18         }
19         ans = 1LL * ans * a[i][i] % mod;
20     }
21     return (ans % mod + mod) % mod;
22 }
```

```

23 int main() {
24     scanf("%d%d", &n, &m);
25     for (int i = 1; i <= m; i++) {
26         int u, v;
27         scanf("%d%d", &u, &v);
28         a[u][v]--, a[v][u]--;
29         a[u][u]++, a[v][v]++;
30     }
31     printf("%d\n", Gauss(n - 1));
32     return 0;
33 }

```

2.2.5 Steiner Tree

```

1  /*BZOJ:4774
2  无向图G从1-n进行编号，选择一些边，使对于 $1 \leq i \leq d$ , i号点和n-i+1号点连通，最小化选出的所有边权值和。
3  1. 枚举子树形态  $dp[S][i] = \min(dp[s] + dp[S \setminus s])$ 
4  2. 按照边进行松弛  $dp[S][i] = \min(dp[S][j] + w[j][i])$ 
5  其中 $S$ 为选取的子集， $s$ 和 $S \setminus s$ 为 $S$ 的状态划分。第二类转移方程可以通过跑一次最短路进行松弛。
6  本题需要再做一次子集dp，因为不成对的点可能不连通。
7  */
8  #include <bits/stdc++.h>
9
10 using namespace std;
11
12 const int maxn = 1e4 + 10;
13 const int inf = 0x3f3f3f3f;
14
15 int head[maxn], cnt;
16 struct Edge {int nex, to, w; }edge[maxn<<1];
17
18 void add(int u, int v, int w)
19 {
20     edge[cnt].nex = head[u];
21     edge[cnt].to = v;
22     edge[cnt].w = w;
23     head[u] = cnt++;
24 }
25
26 int f[1<<10][maxn], ans[20];
27 bool in[maxn];
28
29 queue<int> que;
30
31 void spfa(int S)
32 {
33     while(!que.empty())
34     {
35         int u = que.front(); que.pop();
36         in[u] = false;
37         for(int i = head[u]; ~i; i = edge[i].nex)
38         {
39             int v = edge[i].to;
40             if(f[S][v] > f[S][u] + edge[i].w)
41             {
42                 f[S][v] = f[S][u] + edge[i].w;
43                 if(!in[v]) que.push(v), in[v] = true;
44             }

```



```
45     }
46 }
47 }
48
49 int Steiner_Tree(int n, int d)
50 {
51     memset(f, 0x3f, sizeof f);
52     for(int i = 1; i <= d; i++)
53         f[1 << (i - 1)][i] = f[1 << (d + i - 1)][n - i + 1] = 0;
54     int lim = 1 << (d < 1);
55     for(int S = 1; S < lim; S++)
56     {
57         for(int i = 1; i <= n; i++)
58         {
59             for(int s = (S - 1) & S; s; s = (s - 1) & S)
60                 f[S][i] = min(f[S][i], f[s][i] + f[S ^ s][i]);
61             if(f[S][i] != inf) que.push(i), in[i] = true;
62         }
63         spfa(S);
64     }
65     lim = 1 << d;
66     memset(ans, 0x3f, sizeof ans);
67     for(int S = 1; S < lim; S++)
68         for(int i = 1; i <= n; i++)
69             ans[S] = min(ans[S], f[S ^ (S << d)][i]);
70     for(int S = 1; S < lim; S++)
71         for(int s = (S - 1) & S; s; s = (s - 1) & S)
72             ans[S] = min(ans[S], ans[s] + ans[S ^ s]);
73     return ans[lim - 1] == inf ? -1 : ans[lim - 1];
74 }
75
76 int main()
77 {
78     int n, m, d, u, v, w;
79     scanf("%d%d%d", &n, &m, &d);
80     memset(head, 0xff, sizeof head);
81     while(m--)
82     {
83         scanf("%d%d%d", &u, &v, &w);
84         add(u, v, w);
85         add(v, u, w);
86     }
87     printf("%d\n", Steiner_Tree(n, d));
88     return 0;
89 }
```

2.3 连通性

2.3.1 割点

```
1  const int maxn = 1e4 + 10;
2
3  vector<int> edge[maxn];
4  int n, dfn[maxn], low[maxn], cnt = 0;
5  bool vis[maxn], cut[maxn];
6
7  void Tarjan(int u, int fa)
8  {
9      dfn[u] = low[u] = ++cnt;
```

```

10     vis[u] = true;
11     int children = 0;
12     for (int i = 0; i < edge[u].size(); i++)
13     {
14         int v = edge[u][i];
15         if (v != fa && vis[v])
16             low[u] = min(low[u], dfn[v]);
17         else if (!vis[v])
18         {
19             Tarjan(v, u);
20             children++;
21             low[u] = min(low[u], low[v]);
22             if (fa == -1 && children > 1) //若u是根节点且子节点数大于1
23                 cut[u] = true;        //u是割点
24             else if (fa != -1 && low[v] >= dfn[u]) //若u不是根节点且v不能访问到u的父节点
25                 cut[u] = true;        //u是割点
26         }
27     }
28 }

```

2.3.2 桥

```

1  const int maxn = 1e4 + 10;
2
3  vector<int> edge[maxn];
4  int n, dfn[maxn], low[maxn], father[maxn], cnt = 0;
5  bool bridge[maxn][maxn];
6
7  void Tarjan(int u, int fa)
8  {
9      dfn[u] = low[u] = ++cnt;
10     for (int i = 0; i < edge[u].size(); i++)
11     {
12         int v = edge[u][i];
13         if (!dfn[v]) //未访问节点v
14         {
15             Tarjan(v, u);
16             low[u] = min(low[u], low[v]);
17             if (low[v] > dfn[u]) //节点v到达祖先必须经过(u,v)
18                 bridge[u][v] = bridge[v][u] = true; // (u,v)是桥
19         }
20         else if (fa != v) //u的父节点不是v, (u,v)不存在重边
21             low[u] = min(low[u], dfn[v]);
22     }
23 }

```

2.3.3 强连通分量

```

1  const int maxn=1000+10;
2
3  vector<int> edge[maxn];
4
5  int dfn[maxn], low[maxn];
6  int stack[maxn], index, tot;
7  int belong[maxn], inde[maxn], outde[maxn], scc;
8  bool vis[maxn];
9
10 void add(int u, int v)

```

```
11 {
12     edge[u].push_back(v);
13     edge[v].push_back(u);
14 }
15
16 void Tarjan(int u)
17 {
18     dfn[u] = low[u] = ++tot;
19     stack[++index] = u;
20     vis[u] = true;
21     int v;
22     for(int i = 0; i < edge[u].size(); i++)
23     {
24         v = edge[u][i];
25         if(!dfn[v])
26         {
27             Tarjan(v);
28             low[u] = min(low[v], low[u]);
29         }
30         else if(vis[v]) low[u] = min(low[v], dfn[u]);
31     }
32     if(dfn[u] == low[u])
33     {
34         scc++;
35         do
36         {
37             v = stack[index--];
38             vis[v] = false;
39             belong[v] = scc;
40         } while(v != u);
41     }
42 }
```

2.3.4 点双联通分量

```
1  const int maxn = 10000 + 10;
2
3  struct Edge{ int u, v; };
4  vector<int> G[maxn], bcc[maxn];
5
6  int dfn[maxn], low[maxn], bccno[maxn], idx, bcc_cnt, bridge;
7  bool iscut[maxn];
8
9  stack<Edge> st;
10
11 void dfs(int u, int pre)
12 {
13     dfn[u] = low[u] = ++idx;
14     int child = 0;
15     for(auto v : G[u])
16     {
17         if(v == pre) continue;
18         if(!dfn[v])
19         {
20             child++;
21             st.push({u, v});
22             dfs(v, u);
23             low[u] = min(low[u], low[v]);
```

```

24         if(low[v] >= dfn[u])
25         {
26             iscut[u] = true;
27             bcc[++bcc_cnt].clear();
28             Edge x;
29             do
30             {
31                 x = st.top(); st.pop();
32                 if(bccno[x.u] != bcc_cnt) { bcc[bcc_cnt].push_back(x.u); bccno[x.u] = bcc_cnt;
33             }
34                 if(bccno[x.v] != bcc_cnt) { bcc[bcc_cnt].push_back(x.v); bccno[x.v] = bcc_cnt;
35             }
36             } while(x.u != u || x.v != v);
37         }
38         else if(dfn[v] < dfn[u])
39         {
40             st.push({u, v});
41             low[u] = min(low[u], dfn[v]);
42         }
43     }
44     if(pre < 0 && child == 1) iscut[u] = 0;
45 }
46
47 void find_bcc(int n)
48 {
49     memset(dfn, 0, sizeof dfn);
50     memset(iscut, 0, sizeof iscut);
51     memset(bccno, 0, sizeof bccno);
52     for(int i = 1; i <= bcc_cnt; i++) bcc[i].clear();
53     idx = bcc_cnt = bridge = 0;
54     for(int i = 0; i < n; i++) if(!dfn[i]) dfs(i, -1);
55 }

```

2.3.5 边双联通分量

```

1  const int maxn = 10000 + 10;
2
3  int low[maxn], dfn[maxn], head[maxn], cnt, idx;
4  int cutEdge[maxn << 2];
5  struct Edge { int nex, v; } edge[maxn << 2];
6
7  void add(int u, int v) { edge[cnt].nex = head[u], edge[cnt].v = v, head[u] = cnt++; }
8
9  void dfs(int u, int pre)
10 {
11     low[u] = dfn[u] = ++idx;
12     for(int i = head[u]; ~i; i = edge[i].nex)
13     {
14         int v = edge[i].v;
15         if(v == pre) continue;
16         if(!dfn[v])
17         {
18             dfs(v, u);
19             low[u] = min(low[u], low[v]);
20             if(low[v] > dfn[u]) cutEdge[i] = cutEdge[i ^ 1] = 1;
21         }

```

```

22     else if(dfn[v] < dfn[u]) low[u] = min(low[u], dfn[v]);
23 }
24 }

```

2.4 二分图匹配

1. 二分图中的最大匹配数 = 最小点覆盖数
2. 最小路径覆盖 = 最小路径覆盖 = $|G|$ - 最大匹配数
3. 二分图最大独立集 = 顶点数 - 最小点覆盖
4. 二分图的最大团 = 补图的最大独立集

2.4.1 Hungary Algorithm

```

1  const int maxn = 150;
2
3  int n;
4  int edge[maxn][maxn];
5  int linker[maxn];
6  bool vis[maxn];
7
8  bool path(int u)
9  {
10     for (int v = 1; v <= n; v++)
11     {
12         if (edge[u][v] && !vis[v])
13         {
14             vis[v] = true;
15             if (linker[v] == -1 || path(linker[v]))
16             {
17                 linker[v] = u;
18                 return true;
19             }
20         }
21     }
22     return false;
23 }
24
25 int hungary()
26 {
27     int res = 0;
28     memset(linker, 0xff, sizeof(linker));
29     for (int i = 1; i <= n; i++)
30     {
31         memset(vis, false, sizeof(vis));
32         res += path(i);
33     }
34     return res;
35 }

```

2.4.2 Hopcroft-karp Algorithm

```

1  //复杂度 $O(n^{0.5} * m)$ , 注意这个板子的下标是从0开始的
2
3  const int MAXN = 3010; //左边节点数量、右边节点数量
4  const int MAXM = 3010 * 3010; //边的数量
5  const int INF = 0x3f3f3f3f;
6

```

```
7 struct Edge
8 {
9     int v;
10    int next;
11 } edge[MAXM];
12
13 int nx, ny;
14 int cnt;
15 int dis;
16
17 int first[MAXN];
18 int xlink[MAXN], ylink[MAXN];
19 /*xlink[i]表示左集合顶点所匹配的右集合顶点序号, ylink[i]表示右集合i顶点匹配到的左集合顶点序号。*/
20 int dx[MAXN], dy[MAXN];
21 /*dx[i]表示左集合i顶点的距离编号, dy[i]表示右集合i顶点的距离编号*/
22 int vis[MAXN]; //寻找增广路的标记数组
23
24 void init()
25 {
26     cnt = 0;
27     memset(first, -1, sizeof(first));
28     memset(xlink, -1, sizeof(xlink));
29     memset(ylink, -1, sizeof(ylink));
30 }
31
32 void read_graph(int u, int v)
33 {
34     edge[cnt].v = v;
35     edge[cnt].next = first[u], first[u] = cnt++;
36 }
37
38 int bfs()
39 {
40     queue<int> q;
41     dis = INF;
42     memset(dx, -1, sizeof(dx));
43     memset(dy, -1, sizeof(dy));
44     for (int i = 0; i < nx; i++)
45     {
46         if (xlink[i] == -1)
47         {
48             q.push(i);
49             dx[i] = 0;
50         }
51     }
52     while (!q.empty())
53     {
54         int u = q.front();
55         q.pop();
56         if (dx[u] > dis) break;
57         for (int e = first[u]; e != -1; e = edge[e].next)
58         {
59             int v = edge[e].v;
60             if (dy[v] == -1)
61             {
62                 dy[v] = dx[u] + 1;
63                 if (ylink[v] == -1) dis = dy[v];
64             }
65             else
```

```

66             dx[ylink[v]] = dy[v] + 1;
67             q.push(ylink[v]);
68         }
69     }
70 }
71 }
72 return dis != INF;
73 }
74
75 int find(int u)
76 {
77     for (int e = first[u]; e != -1; e = edge[e].next)
78     {
79         int v = edge[e].v;
80         if (!vis[v] && dy[v] == dx[u] + 1)
81         {
82             vis[v] = 1;
83             if (ylink[v] != -1 && dy[v] == dis) continue;
84             if (ylink[v] == -1 || find(ylink[v]))
85             {
86                 xlink[u] = v, xlink[v] = u;
87                 return 1;
88             }
89         }
90     }
91     return 0;
92 }
93
94 int MaxMatch()
95 {
96     int ans = 0;
97     while (bfs())
98     {
99         memset(vis, 0, sizeof(vis));
100         for (int i = 0; i < nx; i++)
101             if (xlink[i] == -1)
102                 ans += find(i);
103     }
104     return ans;
105 }

```

2.4.3 二分图多重匹配

```

1  const int maxn = 1e2 + 5; // 左边最大点数
2  const int maxm = 1e2 + 5; // 右边最大点数
3  int graph[maxn][maxm], vis[maxm]; // 图G和增广路访问标记
4  int match[maxm][maxn]; // 左边元素与右边元素第n次匹配
5  int nx, ny, m; // 左边点数, 右边点数, 边数
6  int vol[maxm]; // 右边点多重匹配可容纳值
7  int cnt[maxm]; // 右边点已匹配值
8
9  bool find_path(int u) // 找增广路
10 {
11     for (int i = 0; i < ny; i++) // 注意, 这里节点是从0开始编号, 题目有时是从1开始编号
12     {
13         if (graph[u][i] && !vis[i]) // 不在增广路
14         {
15             vis[i] = 1; // 放进增广路

```

```

16         if (cnt[i] < vol[i])//如果当前已匹配数量小于可容纳量，则直接匹配
17         {
18             match[i][cnt[i]++] = u;
19             return true;
20         }
21         for (int j = 0; j < cnt[i]; j++)
22         {
23             if (find_path(match[i][j]))//如果先前已匹配右边的点能另外找到增广路，则此点仍可匹配
24             {
25                 match[i][j] = u;
26                 return true;
27             }
28         }
29     }
30 }
31 return false;
32 }
33
34 int max_match()//计算多重匹配的最大匹配数
35 {
36     int res = 0;
37     memset(match, -1, sizeof(match));
38     memset(cnt, 0, sizeof(cnt));
39     for (int i = 0; i < nx; i++)
40     {
41         memset(vis, 0, sizeof(vis));
42         if (find_path(i)) res++;
43     }
44     return res;
45 }
46
47 bool all_match()//判断左边的点是否都与右边的点匹配了
48 {
49     memset(cnt, 0, sizeof(cnt));
50     for (int i = 0; i < nx; i++)
51     {
52         memset(vis, 0, sizeof(vis));
53         if (!find_path(i)) return false;
54     }
55     return true;
56 }

```

2.4.4 二分图最大权匹配 (KM 算法)

```

1  const int maxn=1000+10;
2  const int inf=0x3f3f3f3f;
3
4  int n;
5  int lx[maxn],ly[maxn],edge[maxn][maxn];
6  int match[maxn],delta;
7  bool vx[maxn],vy[maxn];
8
9  bool dfs(int x) //DFS增广，寻找相等子图的完备匹配
10 {
11     vx[x]=true;
12     for(int y=1;y<=n;y++)
13     {
14         if(!vy[y])

```



```

15     {
16         int tmp=lx[x]+ly[y]-edge[x][y];
17         if(!tmp) //edge(x,y)为可行边
18         {
19             vy[y]=true;
20             if(!match[y]||dfs(match[y]))
21             {
22                 match[y]=x;
23                 return true;
24             }
25         }
26         else delta=min(delta,tmp);
27     }
28 }
29 return false;
30 }
31
32 void KM()
33 {
34     for(int i=1;i<=n;i++) //初始化可行顶标的值
35     {
36         lx[i]=-inf;
37         ly[i]=0;
38         for(int j=1;j<=n;j++)
39             lx[i]=max(lx[i],edge[i][j]);
40     }
41     memset(match,0,sizeof(match));
42     for(int x=1;x<=n;x++)
43     {
44         for(;;)
45         {
46             delta=inf;
47             memset(vx,0,sizeof(vx));
48             memset(vy,0,sizeof(vy));
49             if(dfs(x)) break;
50             for(int i=1;i<=n;i++) //修改顶标
51             {
52                 if(vx[i]) lx[i]-=delta;
53                 if(vy[i]) ly[i]+=delta;
54             }
55         }
56     }
57 }

```

2.4.5 一般图匹配带花树

```

1 //一般图匹配，带花树算法
2 const int maxn = 1000 + 10;
3
4 vector<int> edge[maxn];
5 queue<int> que;
6
7 int n, pre[maxn], type[maxn], link[maxn], nex[maxn], vis[maxn];
8
9 void add(int u, int v)
10 {
11     edge[u].push_back(v);
12     edge[v].push_back(u);

```

```
13 }
14
15 int Find(int x)
16 {
17     return x == pre[x] ? x : pre[x] = Find(pre[x]);
18 }
19
20 void combine(int x, int lca)    //如果找到奇环, 对当前点x和找到的
21 {
22     while (x != lca)
23     {
24         int u = link[x], v = nex[u];
25         if (Find(v) != lca) nex[v] = u;
26         if (type[u] == 1) type[u] = 2, que.push(u);
27         pre[Find(x)] = Find(u);
28         pre[Find(u)] = Find(v);
29         x = v;
30     }
31 }
32
33 void contrack(int x, int y)
34 {
35     int lca = x;
36     memset(vis, 0, sizeof(vis));
37     for (int i = x; i; i = nex[link[i]])
38     {
39         i = Find(i);
40         vis[i] = 1;
41     }
42     for (int i = y; i; i = nex[link[i]])
43     {
44         i = Find(i);
45         if (vis[i])
46         {
47             lca = i;
48             break;
49         }
50     }
51     if (lca != Find(x)) nex[x] = y;
52     if (lca != Find(y)) nex[y] = x;
53     combine(x, lca);
54     combine(y, lca);
55 }
56
57 void bfs(int s)
58 {
59     memset(type, 0, sizeof(type));
60     memset(nex, 0, sizeof(nex));
61     for (int i = 1; i <= n; i++) pre[i] = i;
62     while (!que.empty()) que.pop();
63     que.push(s);
64     type[s] = 2;
65     while (!que.empty())
66     {
67         int x = que.front();
68         que.pop();
69         for (int i = 0; i < edge[x].size(); i++)
70         {
71             int y = edge[x][i];
```

```

72         if (Find(x) == Find(y) || link[x] == y || type[y] == 1) continue;
73         if (type[y] == 2) contrack(x, y);
74         else if (link[y])
75         {
76             nex[y] = x;
77             type[y] = 1;
78             type[link[y]] = 2;
79             que.push(link[y]);
80         } else
81         {
82             nex[y] = x;
83             int pos = y, u = nex[pos], v = link[u];
84             while (pos)
85             {
86                 link[pos] = u;
87                 link[u] = pos;
88                 pos = v;
89                 u = nex[pos];
90                 v = link[u];
91             }
92             return;
93         }
94     }
95 }
96 }
97
98 int maxmatch()
99 {
100     for (int i = 1; i <= n; i++) if (!link[i]) bfs(i);
101     int ans = 0;
102     for (int i = 1; i <= n; i++) if (link[i]) ans++;
103     return ans / 2;
104 }
105
106 void init()
107 {
108     for (int i = 1; i <= n; i++) edge[i].clear();
109     memset(link, 0, sizeof(link));
110 }

```

2.5 网络流

2.5.1 Dinic

```

1  const int MAX_V = 1000 + 10;
2  const int INF = 0x3f3f3f3f;
3
4  //用于表示边的结构体 (终点, 流量, 反向边)
5  struct edge{int to, cap, rev;};
6
7  vector<edge> G[MAX_V]; //图的邻接表表示
8  int level[MAX_V]; //顶点到源点的距离标号
9  int iter[MAX_V]; //当前弧
10
11 void add(int from, int to, int cap)
12 {
13     G[from].push_back((edge){to, cap, (int)G[to].size()});
14     G[to].push_back((edge){from, 0, (int)G[from].size() - 1});
15 }

```

```
16
17 //计算从源点出发的距离标号
18 void bfs(int s)
19 {
20     memset(level, -1, sizeof(level));
21     queue<int> que;
22     level[s] = 0;
23     que.push(s);
24     while(!que.empty())
25     {
26         int v = que.front(); que.pop();
27         for(int i = 0; i < G[v].size(); i++)
28         {
29             edge &e = G[v][i];
30             if(e.cap > 0 && level[e.to] < 0)
31             {
32                 level[e.to] = level[v] + 1;
33                 que.push(e.to);
34             }
35         }
36     }
37 }
38
39 //通过DFS寻找增广路
40 int dfs(int v, int t, int f)
41 {
42     if(v == t) return f;
43     for(int &i = iter[v]; i < G[v].size(); i++)
44     {
45         edge &e = G[v][i];
46         if(e.cap > 0 && level[v] < level[e.to])
47         {
48             int d = dfs(e.to, t, min(f, e.cap));
49             if(d > 0)
50             {
51                 e.cap -= d;
52                 G[e.to][e.rev].cap += d;
53                 return d;
54             }
55         }
56     }
57     return 0;
58 }
59
60 //求解从s到t的最大流
61 int max_flow(int s, int t)
62 {
63     int flow = 0;
64     for(;;)
65     {
66         bfs(s);
67         if(level[t] < 0) return flow;
68         memset(iter, 0, sizeof(iter));
69         int f;
70         while((f = dfs(s,t,INF)) > 0) flow += f;
71     }
72 }
```

2.5.2 ISAP

```
1 struct Edge {
2     int from, to, cap, flow;
3     Edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f) {}
4 };
5
6 bool operator<(const Edge& a, const Edge& b) {
7     return a.from < b.from || (a.from == b.from && a.to < b.to);
8 }
9
10 struct ISAP {
11     int n, m, s, t;
12     vector<Edge> edges;
13     vector<int> G[maxn];
14     bool vis[maxn];
15     int d[maxn];
16     int cur[maxn];
17     int p[maxn];
18     int num[maxn];
19
20     void AddEdge(int from, int to, int cap) {
21         edges.push_back(Edge(from, to, cap, 0));
22         edges.push_back(Edge(to, from, 0, 0));
23         m = edges.size();
24         G[from].push_back(m - 2);
25         G[to].push_back(m - 1);
26     }
27
28     bool BFS() {
29         memset(vis, 0, sizeof(vis));
30         queue<int> Q;
31         Q.push(t);
32         vis[t] = 1;
33         d[t] = 0;
34         while (!Q.empty()) {
35             int x = Q.front();
36             Q.pop();
37             for (int i = 0; i < G[x].size(); i++) {
38                 Edge& e = edges[G[x][i] ^ 1];
39                 if (!vis[e.from] && e.cap > e.flow) {
40                     vis[e.from] = 1;
41                     d[e.from] = d[x] + 1;
42                     Q.push(e.from);
43                 }
44             }
45         }
46         return vis[s];
47     }
48
49     void init(int n) {
50         this->n = n;
51         for (int i = 0; i < n; i++) G[i].clear();
52         edges.clear();
53     }
54
55     int Augment() {
56         int x = t, a = INF;
57         while (x != s) {
```

```
58     Edge& e = edges[p[x]];
59     a = min(a, e.cap - e.flow);
60     x = edges[p[x]].from;
61 }
62 x = t;
63 while (x != s) {
64     edges[p[x]].flow += a;
65     edges[p[x] ^ 1].flow -= a;
66     x = edges[p[x]].from;
67 }
68 return a;
69 }
70
71 int Maxflow(int s, int t) {
72     this->s = s;
73     this->t = t;
74     int flow = 0;
75     BFS();
76     memset(num, 0, sizeof(num));
77     for (int i = 0; i < n; i++) num[d[i]]++;
78     int x = s;
79     memset(cur, 0, sizeof(cur));
80     while (d[s] < n) {
81         if (x == t) {
82             flow += Augment();
83             x = s;
84         }
85         int ok = 0;
86         for (int i = cur[x]; i < G[x].size(); i++) {
87             Edge& e = edges[G[x][i]];
88             if (e.cap > e.flow && d[x] == d[e.to] + 1) {
89                 ok = 1;
90                 p[e.to] = G[x][i];
91                 cur[x] = i;
92                 x = e.to;
93                 break;
94             }
95         }
96         if (!ok) {
97             int m = n - 1;
98             for (int i = 0; i < G[x].size(); i++) {
99                 Edge& e = edges[G[x][i]];
100                 if (e.cap > e.flow) m = min(m, d[e.to]);
101             }
102             if (--num[d[x]] == 0) break;
103             num[d[x] = m + 1]++;
104             cur[x] = 0;
105             if (x != s) x = edges[p[x]].from;
106         }
107     }
108     return flow;
109 }
110 };
```

2.5.3 MCMF

```
1 const int maxn = 10000 + 10;
2 const int inf = 0x3f3f3f3f;
```

```

3
4 struct Edge { int from, to, cap, flow, cost; };
5
6 struct MCMF
7 {
8     int n, m;
9     vector<Edge> edges;
10    vector<int> G[maxn];
11    bool inq[maxn];
12    int dis[maxn], path[maxn], a[maxn];
13
14    void init(int n)
15    {
16        this->n = n;
17        for(int i = 0; i <= n; i++)
18            G[i].clear();
19        edges.clear();
20    }
21
22    void addEdge(int from, int to, int cap, int cost)
23    {
24        edges.push_back(Edge{from, to, cap, 0, cost});
25        edges.push_back(Edge{to, from, 0, 0, -cost});
26        m = edges.size();
27        G[from].push_back(m - 2);
28        G[to].push_back(m - 1);
29    }
30
31    bool Bellman_Ford(int s, int t, int& flow, int& cost)
32    {
33        for(int i = 0; i <= n; i++) dis[i] = inf;
34        memset(inq, 0, sizeof inq);
35        dis[s]=0, inq[s]=true, path[s]=0, a[s]=inf;
36        queue<int> Q;
37        Q.push(s);
38        while(!Q.empty())
39        {
40            int u = Q.front(); Q.pop();
41            inq[u] = false;
42            for(int i = 0; i < G[u].size(); i++)
43            {
44                Edge& e = edges[G[u][i]];
45                if(e.cap > e.flow && dis[e.to] > dis[u] + e.cost)
46                {
47                    dis[e.to] = dis[u] + e.cost;
48                    path[e.to] = G[u][i];
49                    a[e.to] = min(a[u], e.cap - e.flow);
50                    if(!inq[e.to])
51                    {
52                        Q.push(e.to);
53                        inq[e.to] = true;
54                    }
55                }
56            }
57        }
58        if(dis[t] == inf) return false;    //求最小费用最大流
59        //if(1ll * dis[t] * a[t] > 0) return false; 求可行流最小费用，因此当费用增量大于0时不继续增加
流量
60        flow += a[t];

```

```

61     cost += dis[t] * a[t];
62     for(int u = t; u != s; u = edges[path[u]].from)
63     {
64         edges[path[u]].flow += a[t];
65         edges[path[u] ^ 1].flow -= a[t];
66     }
67     return true;
68 }
69
70 int mincostMaxFlow(int s, int t)
71 {
72     int flow = 0, cost = 0;
73     while(Bellman_Ford(s, t, flow, cost));
74     return cost;
75 }
76 };

```

2.5.4 Trick

建模技巧

二分图带权最大独立集。给出一个二分图，每个结点上有一个正权值。要求选出一些点，使得这些点之间没有边相连，且权值和最大。

解：在二分图的基础上添加源点 S 和汇点 T ，然后从 S 向所有 X 集合中的点连一条边，所有 Y 集合中的点向 T 连一条边，容量均为该点的权值。 X 结点与 Y 结点之间的边的容量均为无穷大。这样，对于图中的任意一个割，将割中的边对应的结点删掉就是一个符合要求的解，权值和为所有权减去割的容量。因此，只要求出最小割，就能求出最大权和。

公平分配问题。把 m 个任务分配给 n 个处理器。其中每个任务有两个候选处理器，可以任选一个分配。要求所有处理器中，任务数最多的那个处理器所分配的任务数尽量少。不同任务的候选处理器集 $\{p_1, p_2\}$ 保证不同。

解：本题有一个比较明显的二分图模型，即 X 结点是任务， Y 结点是处理器。二分答案 x ，然后构图，首先从源点 S 出发向所有的任务结点引一条边，容量等于 1，然后从每个任务结点出发引两条边，分别到达它所能分配到的两个处理器结点，容量为 1，最后从每个处理器结点出发引一条边到汇点 T ，容量为 x ，表示选择该处理器的任务不能超过 x 。这样网络中的每个单位流量都是从 S 流到一个任务结点，再到处理器结点，最后到汇点 T 。只有当网络中的总流量等于 m 时才意味着所有任务都选择了一个处理器。这样，我们通过 $O(\log m)$ 次最大流便算出了答案。

区间 k 覆盖问题。数轴上有一些带权值的左闭右开区间。选出权和尽量大的一些区间，使得任意一个数最多被 k 个区间覆盖。

解：本题可以用最小费用流解决，构图方法是把每个数作为一个结点，然后对于权值为 w 的区间 $[u, v)$ 加边 $u \rightarrow v$ ，容量为 1，费用为 $-w$ 。再对所有相邻的点加边 $i \rightarrow i+1$ ，容量为 k ，费用为 0。最后，求最左点到最右点的最小费用最大流即可，其中每个流量对应一组互不相交的区间。如果数值范围太大，可以先进行离散化。

最大闭合子图。给定带权图 G （权值可正可负），求一个权和最大的点集，使得起点在该点集中的任意弧，终点也在该点集中。

解：新增附加源 s 和附加汇 t ，从 s 向所有正权点引一条边，容量为权值；从所有负权点向汇点引一条边，容量为权值的相反数。求出最小割以后， $S - \{s\}$ 就是最大闭合子图。

最大密度子图。给出一个无向图，找一个点集，使得这些点之间的边数除以点数的值（称为子图的密度）最大。

解：如果两个端点都选了，就必然要选边，这就是一种推导。如果把每个点和每条边都看成新图中的结点，可以把问题转化为最大闭合子图。

无源汇有上下界可行流：附加源 S 和汇 T ；对于边 (u, v, min, max) ，记 $d[u]- = min, d[v]+ = max$ ，并添加弧 $(u, v, max - min)$ ；对于流量不平衡的点 u ，设多余流量为 W ，如果 $W > 0$ ，添加弧 $S \rightarrow u : W$ ，否则若 $W < 0$ ，添加弧 $u \rightarrow T : -W$ ，求改造后的网络 $S - T$ 最大流即可，当且仅当所有附加弧满载时原图有可行流。

有源汇有上下界可行流：建 $t \rightarrow s$ ，容量为 inf ，然后和无源汇相同。

有源汇有上下界最大/最小流：与上面相同，跑完可行流 $S \rightarrow T$ 后去掉边 $t \rightarrow s$ ，最大流为加 $s \rightarrow t$ ，最小流为 $G[s][t].\text{cap} - \text{maxflow}(t, s)$ 。

2.5.5 Stoer Wagner

```

1  #define INF 100000000
2  bool vis[maxn], com[maxn];
3  int mp[maxn][maxn], w[maxn], s, t;
4
5  int maxadj(int n, int v) {
6      int CUT = 0;
7      memset(vis, 0, sizeof vis);
8      memset(w, 0, sizeof w);
9      for (int i = 0; i < n; ++i) {
10         int num = 0, mx = -INF;
11         for (int j = 0; j < v; ++j) {
12             if (!com[j] && !vis[j] && w[j] > mx) {
13                 mx = w[j];
14                 num = j;
15             }
16         }
17         vis[num] = 1;
18         s = t;
19         t = num;
20         CUT = w[t];
21         for (int j = 0; j < v; ++j) {
22             if (!com[j] && !vis[j]) w[j] += mp[num][j];
23         }
24     }
25     return CUT;
26 }
27
28 int stoer(int v) {
29     int mincut = INF;
30     int n = v;
31     memset(com, 0, sizeof com);
32     for (int i = 0; i < v - 1; ++i) {
33         int cut;
34         s = 0, t = 0;
35         cut = maxadj(n, v);
36         n--;
37         if (cut < mincut) mincut = cut;
38         com[t] = 1;
39         for (int j = 0; j < v; ++j) {
40             if (!com[j]) {
41                 mp[j][s] += mp[j][t];
42                 mp[s][j] += mp[t][j];
43             }
44         }
45     }
46     return mincut;
47 }

```

2.6 Others

2.6.1 拓扑排序

```

1  const int maxn = 1e5 + 10;
2
3  vector<int> edge[maxn];
4  int indegree[maxn];
5
6  void add(int u, int v)
7  {
8      edge[u].push_back(v);
9      indegree[v]++;
10 }
11
12 void Toposort(int n)
13 {
14     queue<int> que;
15     for (int i = 1; i <= n; i++)
16         if (!indegree[i]) que.push(i);    //将图中没有前驱，即入度为0的点加入队列
17     while (!que.empty())
18     {
19         int u = que.front();
20         que.pop();
21         indegree[u] = -1;    //从图中删去此顶点
22         for (int i = 0; i < edge[u].size(); i++)
23         {
24             int v = edge[u][i];
25             indegree[v]--;    //删去图中以u为尾的弧
26             if (!indegree[v]) que.push(v);    //将新增的当前入度为0的点压入队列中
27         }
28     }
29 }

```

2.6.2 2-SAT

```

1  /*2-SAT连边含义：选A必选B
2      点$x_i$表示选，$x_i'$表示不选
3      1. 必选$x_i$，等价于$x_i=1$：$x_i' \rightarrow x_i$
4      2. 必不选$x_i$，等价于$x_i=0$，$x_i \rightarrow x_i'$
5      3. $x_i$与$x_j$中至少选择一个，等价于$x_i \vee x_j=1$，连边$x_i' \rightarrow x_j$，$x_j' \rightarrow x_i$
6      4. $x_i$与$x_j$不都选，等价于$x_i \wedge x_j=0$，连边$x_i \rightarrow x_j'$，$x_j \rightarrow x_i'$
7      5. $x_i$与$x_j$情况相同，等价于$x_i \oplus x_j=0$，连边$x_i \rightarrow x_j$，$x_i' \rightarrow x_j'$，$x_j \rightarrow x_i$，$x_j' \rightarrow x_i'$
8      6. $x_i$与$x_j$情况相反，等价于$x_i \oplus x_j=1$，连边$x_i \rightarrow x_j'$，$x_i' \rightarrow x_j$，$x_j \rightarrow x_i'$，$x_j' \rightarrow x_i$
9  */
10
11 const int maxn = 2e6 + 10;
12
13 int n, m, a, va, b, vb;
14 int low[maxn], dfn[maxn], color[maxn], cnt, scc_cnt;
15 bool instack[maxn];
16
17 vector<int> g[maxn];
18 stack<int> st;
19
20 void Tarjan(int u)
21 {
22     low[u] = dfn[u] = ++cnt;
23     st.push(u);
24     instack[u] = true;
25     for(const auto &v : g[u])
26     {

```

```

27         if(!dfn[v]) Tarjan(v), low[u] = min(low[u], low[v]);
28         else if(instack[v]) low[u] = min(low[u], dfn[v]);
29     }
30     if(low[u] == dfn[u])
31     {
32         ++scc_cnt;
33         do {
34             color[u] = scc_cnt;
35             u = st.top(); st.pop();
36             instack[u] = false;
37         } while(low[u] != dfn[u]);
38     }
39 }
40
41 inline void add(int a, int b) { g[a].push_back(b); }
42
43 inline void AND(int a, int b, int c)
44 {
45     if(c == 1) add(a, a + n), add(b, b + n);
46     else add(a + n, b), add(b + n, a);
47 }
48
49 inline void OR(int a, int b, int c)
50 {
51     if(c == 0) add(a + n, a), add(b + n, b);
52     else add(a, b + n), add(b, a + n);
53 }
54
55 inline void XOR(int a, int b, int c)
56 {
57     if(c == 0) add(a, b), add(a + n, b + n), add(b, a), add(b + n, a + n);
58     else add(a, b + n), add(a + n, b), add(b, a + n), add(b + n, a);
59 }
60
61 bool TWO_SAT()
62 {
63     input();
64     for(int i = 1; i <= (n << 1); i++) if(!dfn[i]) Tarjan(i);
65     for(int i = 1; i <= n; i++)
66         if(color[i] == color[i + n]) return false;
67     for(int i = 1; i <= n; i++)
68         printf("%d ", color[i] > color[i + n]);
69     return true;
70 }

```

2.6.3 差分约束系统

```

1  //以 $x_i - x_j$ 为约束条件，建图求最短路后得到的是最大解。所有的解都不大于且尽可能逼近 $dis[x0]$ 
2  //最短路对应最大解，最长路对应最小解
3
4  const int maxn = 1000 + 10;
5  const int inf = 0x3f3f3f3f;
6
7  struct Edge
8  {
9      int nex, to, w;
10 } edge[10 * maxn];
11

```

```
12 int head[maxn], cnt, dis[maxn], n;
13 bool vis[maxn];
14
15 void init()
16 {
17     cnt = 0;
18     memset(head, 0xff, sizeof head);
19 }
20
21 void add(int u, int v, int w)
22 {
23     edge[cnt].nex = head[u];
24     edge[cnt].to = v;
25     edge[cnt].w = w;
26     head[u] = ++cnt;
27 }
28
29 void spfa(int u)
30 {
31     int u, v, w;
32     for (int i = 1; i <= n; i++) dis[i] = inf, vis[i] = false;
33     dis[u] = 0;
34     queue<int> que;
35     que.push(u);
36     vis[u] = true;
37     while (!que.empty())
38     {
39         u = que.front();
40         que.pop();
41         vis[u] = false;
42         for (int i = head[u]; ~i; i = edge[i].nex)
43         {
44             v = edge[i].v, w = edge[i].w;
45             if (dis[u] + w < dis[v])
46             {
47                 dis[v] = dis[u] + w;
48                 if (!vis[v])
49                 {
50                     que.push(v);
51                     vis[v] = true;
52                 }
53             }
54         }
55     }
56 }
```

2.6.4 支配树

```
1 const int N = 2e5 + 10;
2
3 int n, m;
4
5 struct G
6 {
7     vector<int> edge[N];
8     inline void add(int u, int v) { edge[u].push_back(v); }
9 }a, b, c, d;
10
```

```
11 int dfn[N], id[N], fa[N], cnt;
12
13 void dfs(int u)
14 {
15     dfn[u] = ++ cnt; id[cnt] = u;
16     int len = a.edge[u].size();
17     for(auto v : a.edge[u]) if(!dfn[v]) { fa[v] = u; dfs(v); }
18 }
19
20 int semi[N], idom[N], belong[N], val[N];
21
22 int find(int x)
23 {
24     if(x == belong[x]) return x;
25     int tmp = find(belong[x]);
26     if(dfn[semi[val[belong[x]]]] < dfn[semi[val[x]]]) val[x] = val[belong[x]];
27     return belong[x] = tmp;
28 }
29
30 void tarjan()
31 {
32     for(int i = cnt; i > 1; i --)
33     {
34         int u = id[i];
35         for(auto v : b.edge[u])
36         {
37             if(!dfn[v]) continue;
38             find(v);
39             if(dfn[semi[val[v]]] < dfn[semi[u]]) semi[u] = semi[val[v]];
40         }
41         c.add(semi[u], u);
42         belong[u] = fa[u];
43         u = fa[u];
44         for(auto v : c.edge[u])
45         {
46             find(v);
47             if(semi[val[v]] == u) idom[v] = u;
48             else idom[v] = val[v];
49         }
50     }
51     for(int i = 2; i <= cnt; i ++ )
52     {
53         int u = id[i];
54         if(idom[u] != semi[u]) idom[u] = idom[idom[u]];
55     }
56 }
57
58 int ans[N];
59
60 void dfs_ans(int u)
61 {
62     ans[u] = 1;
63     for(auto v : d.edge[u]) dfs_ans(v), ans[u] += ans[v];
64 }
65
66 void solve()
67 {
68     int u, v;
69     scanf("%d%d", &n, &m);
```

```
70     while(m --)
71     {
72         scanf("%d%d", &u, &v);
73         a.add(u, v);
74         b.add(v, u);
75     }
76     for(int i = 1; i <= n; i ++) semi[i] = belong[i] = val[i] = i;
77     dfs(1);
78     tarjan();
79     for(int i = 2; i <= n; i ++) d.add(idom[i], i);
80     dfs_ans(1);
81     for(int i = 1; i <= n; i ++) printf("%d ", ans[i]);
82 }
```

2.6.5 Stable Matching Problem

```
1  const int maxn = 1000 + 10;
2
3  int pre[maxn][maxn], order[maxn][maxn], nex[maxn];
4  int hus[maxn], wife[maxn];
5  queue<int> que;
6
7  void engage(int man, int woman)
8  {
9      int m = hus[woman];
10     if(m) wife[m] = 0, q.push(m);
11     wife[man] = woman;
12     hus[woman] = man;
13 }
14
15 int solve()
16 {
17     for(int i = 1; i <= n; i ++)
18     {
19         for(int j = 1; j <= n; j ++)
20             scanf("%d", &pre[i][j]);
21         nex[i] = 1;
22         wife[i] = 0;
23         que.push(i);
24     }
25     for(int i = 1; i <= n; i ++)
26     {
27         for(int j = 1; j <= n; j ++)
28         {
29             int x;
30             scanf("%d", &x);
31             order[i][x] = j;
32         }
33         hus[i] = 0;
34     }
35
36     while(!que.empty())
37     {
38         int man = que.front(); que.pop();
39         int woman = pre[man][nex[man] ++];
40         if(!hus[woman]) engage(man, woman);
41         else if(order[woman][man] < order[woman][hus[woman]]) engage(man, woman);
42         else que.push(man);
43     }
```

```
43     }  
44 }
```

3 DataStructure

3.1 SegmentTreeDS

3.1.1 SegmentTree

```
1  const int maxn = 2e5+5;
2  // 序列
3  int a[maxn];
4
5  struct SegmentTree {
6  #define TYPE int
7  #define USELAZY 0
8      TYPE val[maxn << 2];
9      int sz;
10 //    check this type
11     vector<int> lazy;
12
13     inline TYPE comb(const TYPE& a, const TYPE& b) {
14         TYPE res;
15         res = a + b;
16         return res;
17     }
18
19     int le, re, k;
20
21     inline void build(int rt, int l, int r) {
22         if (USELAZY) lazy[rt] = 0;
23         if (l == r) {
24             val[rt] = a[l];
25             return;
26         }
27         int mid = l + r >> 1;
28         build(rt << 1, l, mid);
29         build(rt << 1 | 1, mid + 1, r);
30         pushup(rt);
31     }
32     inline void build() {build(1, 1, sz);}
33
34     inline void init(int sz_) {
35         sz = sz_;
36         lazy.resize(sz_ << 2);
37         build();
38     }
39     inline void pushup(int rt) {val[rt] = comb(val[rt << 1], val[rt << 1 | 1]);}
40     inline void deal(int rt, int kt) {
41 //        todo:
42         val[rt] = comb(val[rt], kt);
43     }
44     inline void pushdown(int rt, int len) {
45         if (lazy[rt]) {
46             // check the lazy change
47             lazy[rt << 1] += lazy[rt];
48             lazy[rt << 1 | 1] += lazy[rt];
49             deal(rt << 1, lazy[rt]);
50             deal(rt << 1 | 1, lazy[rt]);
51             lazy[rt] = 0;
52         }
53     }
```



```

54
55     inline void update(int rt, int l, int r) {
56         if (le <= l && r <= re) {
57             deal(rt, k);
58             return;
59         }
60         if (USELAZY) pushdown(rt, r - l + 1);
61         int mid = l + r >> 1;
62         if (le <= mid) update(rt << 1, l, mid);
63         if (re > mid) update(rt << 1 | 1, mid + 1, r);
64         pushup(rt);
65     }
66
67     inline TYPE query(int rt, int l, int r) {
68         if (le <= l && r <= re) {
69             return val[rt];
70         }
71         if (USELAZY) pushdown(rt, r - l + 1);
72         // check the zero type
73         TYPE res;
74         int mid = l + r >> 1;
75         if (le <= mid) res = comb(res, query(rt << 1, l, mid));
76         if (re > mid) res = comb(res, query(rt << 1 | 1, mid + 1, r));
77         return res;
78     }
79
80     // check return type
81     inline int query(int l, int r) {
82         le = l, re = r;
83         return query(1, 1, sz);
84     }
85     inline void modify(int l, int r, int kt) {
86         le = l, re = r, k = kt;
87         update(1, 1, sz);
88     }
89
90     // inline void pt(int rt, int l, int r) {
91     //     if (l == r) {
92     //         printf("%d ", val[l]);
93     //         return;
94     //     }
95     //     pushdown(rt, r - l + 1);
96     //     int mid = l + r >> 1;
97     //     if (le <= mid) pt(rt << 1, l, mid);
98     //     if (re > mid) pt(rt << 1 | 1, mid + 1, r);
99     // }
100
101 #undef TYPE
102 };

```

3.1.2 离散化区间

```

1 // 原题1e5个区间有2e5个端点，离散化出来4e5个区间
2 // 然后线段树需要4e5*4=16e5的大小
3 // 注意三个数组要开离散化数量的四倍，如果不需要sz可以不用这个数组。
4 int val[maxn << 4];
5 int lpos[maxn << 2], rpos[maxn << 2], tot, sz[maxn << 2];
6 vector<int> xpos;

```

```

7  sort(xpos.begin(), xpos.end());
8  xpos.erase(unique(xpos.begin(), xpos.end()), xpos.end());
9  tot = 1;
10 lpos[1] = rpos[1] = xpos[0];
11 sz[1] = 1;
12 for (int i = 1; i < xpos.size(); ++i) {
13     if (xpos[i] - xpos[i - 1] != 1) {
14         lpos[++tot] = xpos[i - 1] + 1;
15         rpos[tot] = xpos[i] - 1;
16         sz[tot] = rpos[tot] - lpos[tot] + 1;
17     }
18     ++tot;
19     lpos[tot] = rpos[tot] = xpos[i];
20     sz[tot] = 1;
21 }
22 le = lower_bound(lpos + 1, lpos + 1 + tot, p[i].x) - lpos;
23 re = upper_bound(rpos + 1, rpos + 1 + tot, p[i].y) - rpos - 1;

```

3.1.3 动态区间最大子段和

```

1  namespace ST {
2      struct node{
3          ll ans,ls,rs,sum;
4      }xx[maxn << 2];
5      inline void pushdown(int x){
6          xx[x].sum=xx[x<<1].sum+xx[x<<1|1].sum;
7          xx[x].ls=max(xx[x<<1].ls,xx[x<<1].sum+xx[x<<1|1].ls);
8          xx[x].rs=max(xx[x<<1|1].rs,xx[x<<1|1].sum+xx[x<<1].rs);
9          xx[x].ans=max(xx[x<<1].ans,max(xx[x<<1|1].ans,xx[x<<1].rs+xx[x<<1|1].ls));
10         return;
11     }
12     inline void build(int k,int l,int r){
13         if(l==r){
14             xx[k].ls=xx[k].rs=xx[k].ans=xx[k].sum=0;
15             return;
16         }
17         int mid=l+r>>1;
18         build(k<<1,l,mid),build(k<<1|1,mid+1,r);
19         pushdown(k);
20         return;
21     }
22     inline void change(int k,int l,int r,int x,int y,int w){ // 1, 1, n
23         if(x<=l&&r<=y){
24             xx[k].ls += w;
25             xx[k].rs += w;
26             xx[k].ans += w;
27             xx[k].sum += w;
28         } // xx[k].ls=xx[k].rs=xx[k].ans=xx[k].sum=w;
29         return;
30     }
31     int mid=l+r>>1;
32     if(x<=mid) change(k<<1,l,mid,x,y,w);
33     if(mid<y) change(k<<1|1,mid+1,r,x,y,w);
34     pushdown(k);
35     return;
36 }
37 inline node query(int k,int l,int r,int x,int y){
38     if(x<=l&&r<=y) {

```

```

39         return xx[k];
40     }
41     int mid=l+r>>1;
42     if(x<=mid&&!(mid<y)) return query(k<<1,l,mid,x,y);
43     else if(!(x<=mid)&&mid<y) return query(k<<1|1,mid+1,r,x,y);
44     else{
45         node st,t1=query(k<<1,l,mid,x,y),t2=query(k<<1|1,mid+1,r,x,y);
46         st.sum=t1.sum+t2.sum;
47         st.ls=max(t1.ls,t1.sum+t2.ls);
48         st.rs=max(t2.rs,t2.sum+t1.rs);
49         st.ans=max(t1.ans,max(t2.ans,t1.rs+t2.ls));
50         return st;
51     }
52 }
53 }

```

3.1.4 动态开点权值线段树

```

1  int root[100005];
2  int ls[1800000], rs[1800000], sum[1800000];
3  int sz = 0;
4
5  void insert(int &k, int l, int r, int val){
6      if (!k) k = ++sz;
7      if (l == r) {
8          sum[k] = 1;
9          return;
10     }
11     int mid = (l + r) >> 1;
12     if (val <= mid) insert(ls[k], l, mid, val);
13     else insert(rs[k], mid + 1, r, val);
14     sum[k] = sum[ls[k]] + sum[rs[k]];
15 }
16
17 int query(int k, int l, int r, int rank) {
18     if (l == r) return l;
19     int mid = (l + r) >> 1;
20     if (sum[ls[k]] >= rank) return query(ls[k], l, mid, rank);
21     else return query(rs[k], mid + 1, r, rank - sum[ls[k]]);
22 }
23 int merge(int x, int y)
24 {
25     if (!x) return y;
26     if (!y) return x;
27     ls[x] = merge(ls[x], ls[y]);
28     rs[x] = merge(rs[x], rs[y]);
29     sum[x] = sum[ls[x]] + sum[rs[x]];
30     return x;
31 }
32 insert(root[i], 1, n, a[i]);
33 query(root[p], 1, n, x);

```

3.1.5 扫描线

```

1  // 范用型扫描线, del储存上界+1, add储存下界, 先del后add即可
2  struct node {
3      int lpos, rpos, linepos;
4      bool operator < (const node& oth) const {

```

```
5         return linepos < oth.linepos;
6     }
7 };
8 vector<node> add, del;
9 int delpos = 0;
10 int res = 0;
11 for (int addpos = 0; addpos < add.size(); ++addpos) {
12     while (delpos < del.size() && del[delpos].linepos <= add[addpos].linepos) {
13         up(del[delpos].lpos, del[delpos].rpos, -1);
14         delpos ++;
15     }
16     up(add[addpos].lpos, add[addpos].rpos, 1);
17     res = max(res, val[1]);
18 }
19
20 // 求面积并
21 #define maxn 222
22 #define tmp (st<<1)
23 #define mid ((l+r)>>1)
24 #define lson l,mid,tmp
25 #define rson mid+1,r,tmp|1
26 using namespace std;
27 int cnt[maxn<<2];
28 double sum[maxn<<2];
29 double x[maxn];
30 struct Seg{
31     double h,l,r;
32     int s;
33     Seg(){}
34     Seg(double a,double b,double c,int d):l(a),r(b),h(c),s(d){}
35     bool operator<(const Seg &cmp)const{
36         return h<cmp.h;
37     }
38 }ss[maxn];
39 void push_up(int st,int l,int r){
40     if(cnt[st])sum[st]=x[r+1]-x[l];
41     else if(l==r)sum[st]=0;
42     else sum[st]=sum[tmp]+sum[tmp|1];
43 }
44 void update(int L,int R,int c,int l,int r,int st){
45     if(L<=l&&r<=R){
46         cnt[st]+=c;
47         push_up(st,l,r);
48         return ;
49     }
50     if(L<=mid)update(L,R,c,lson);
51     if(R>mid)update(L,R,c,rson);
52     push_up(st,l,r);
53 }
54 int main(){
55     int n,tot=1,m;
56     while(scanf("%d",&n)&&n){
57         double a,b,c,d;
58         m=0;
59         while(n--){
60             scanf("%lf%lf%lf%lf",&a,&b,&c,&d);
61             x[m]=a;
62             ss[m++]=Seg(a,c,b,1);
63             x[m]=c;
```

```

64         ss[m++]=Seg(a,c,d,-1);
65     }
66     sort(x,x+m);
67     sort(ss,ss+m);
68     double ans=0;
69     for(int i=0;i<m;++i){
70         int l=lower_bound(x,x+m,ss[i].l)-x;
71         int r=lower_bound(x,x+m,ss[i].r)-x-1;
72         update(l,r,ss[i].s,0,m-1,1);
73         ans+=sum[1]*(ss[i+1].h-ss[i].h);
74     }
75     printf("Test case #%dnTotal explored area: %.2lfnn",tot++,ans);
76 }
77 return 0;
78 }
79
80 // 面积交
81 #include<bits/stdc++.h>
82 #define maxn 100005
83 #define lson l,mid,rt<<1
84 #define rson mid+1,r,rt<<1|1
85 #define pb push_back
86 using namespace std;
87
88 double tree[maxn<<2],tree2[maxn<<2];
89 int lazy[maxn<<2];
90 vector<double>ve;
91
92 struct seg{
93     double l,r,h;
94     int flag;
95     seg(){}
96     seg(double _l,double _r,double _h,int _flag){l=_l,r=_r,h=_h,flag=_flag;}
97     bool operator<(const seg &b)const{return h<b.h;}
98 }s[maxn];
99
100 void push_up(int l,int r,int rt){
101     if(lazy[rt]) tree[rt]=ve[r]-ve[l-1];
102     else if(l==r) tree[rt]=0;
103     else tree[rt]=tree[rt<<1]+tree[rt<<1|1];
104 }
105
106 void push_up2(int l,int r,int rt){
107     if(lazy[rt]>1) tree2[rt]=ve[r]-ve[l-1];
108     else if(l==r) tree2[rt]=0;
109     else if(lazy[rt]==1)tree2[rt]=tree[rt<<1]+tree[rt<<1|1];
110     else tree2[rt]=tree2[rt<<1]+tree2[rt<<1|1];
111 }
112
113 void build(int l,int r,int rt){
114     tree[rt]=0,lazy[rt]=0;
115     if(l==r) return;
116     int mid=l+r>>1;
117     build(lson);
118     build(rson);
119 }
120
121 void add(int L,int R,int v,int l,int r,int rt){
122     if(L<=l&&R>=r){

```

```

123         lazy[rt]+=v;
124         push_up(1,r,rt);
125         push_up2(1,r,rt);
126         return;
127     }
128     int mid=l+r>>1;
129     if(L<=mid) add(L,R,v,lson);
130     if(R>mid) add(L,R,v,rson);
131     push_up(1,r,rt);
132     push_up2(1,r,rt);
133 }
134
135 int getid(double x){ return lower_bound(ve.begin(),ve.end(),x)-ve.begin()+1;}
136
137 int main(){
138     int n;
139     int Case=1;
140     int T;
141     scanf("%d",&T);
142     while(T--){
143         scanf("%d",&n);
144         ve.clear();
145         int tot=0;
146         double x1,y1,x2,y2;
147         for(int i=1;i<=n;i++){
148             scanf("%lf %lf %lf %lf",&x1,&y1,&x2,&y2);
149             ve.pb(x1),ve.pb(x2);
150             s[++tot]=seg(x1,x2,y1,1);
151             s[++tot]=seg(x1,x2,y2,-1);
152         }
153         sort(ve.begin(),ve.end());
154         ve.erase(unique(ve.begin(),ve.end()),ve.end());
155         sort(s+1,s+tot+1);
156         int N=ve.size();
157         build(1,N,1);
158         double ans=0;
159         for(int i=1;i<tot;i++){
160             int L=getid(s[i].l);
161             int R=getid(s[i].r)-1;
162             add(L,R,s[i].flag,1,N,1);
163             ans+=tree2[1]*(s[i+1].h-s[i].h);
164         }
165         printf("%.2f\n",ans);
166     }
167 }
168
169 // 求周长并
170 #include<bits/stdc++.h>
171 #define maxn 100005
172 #define lson l,mid,rt<<1
173 #define rson mid+1,r,rt<<1|1
174 #define pb push_back
175 using namespace std;
176
177 int tree[maxn<<2];
178 int lazy[maxn<<2];
179 vector<int>ve[2];
180 int k;
181

```

```
182 struct seg{
183     int l,r,h;
184     int flag;
185     seg(){ }
186     seg(int _l,int _r,int _h,int _flag){l=_l,r=_r,h=_h,flag=_flag;}
187     bool operator< (const seg &b) const {return h<b.h;}
188 }s[maxn];
189
190 void push_up(int l,int r,int rt){
191     if(lazy[rt]) tree[rt]=ve[k][r]-ve[k][l-1];
192     else if(l==r) tree[rt]=0;
193     else tree[rt]=tree[rt<<1]+tree[rt<<1|1];
194 }
195
196 void build(int l,int r,int rt){
197     tree[rt]=0,lazy[rt]=0;
198     if(l==r) return;
199     int mid=l+r>>1;
200     build(lson);
201     build(rson);
202 }
203
204 void add(int L,int R,int v,int l,int r,int rt){
205     if(L<=l&&R>=r){
206         lazy[rt]+=v;
207         push_up(l,r,rt);
208         return;
209     }
210     int mid=l+r>>1;
211     if(L<=mid) add(L,R,v,lson);
212     if(R>mid) add(L,R,v,rson);
213     push_up(l,r,rt);
214 }
215
216 int getid(int x){return lower_bound(ve[k].begin(),ve[k].end(),x)-ve[k].begin()+1;}
217
218 int main(){
219     int n;
220     while(~scanf("%d",&n)){
221         ve[0].clear();
222         ve[1].clear();
223         int x1,y1,x2,y2;
224         for(int i=1;i<=n;i++){
225             scanf("%d %d %d %d",&x1,&y1,&x2,&y2);
226             ve[0].pb(x1),ve[0].pb(x2);
227             ve[1].pb(y1),ve[1].pb(y2);
228             s[i]=seg(x1,x2,y1,1);
229             s[i+n]=seg(x1,x2,y2,-1);
230             s[i+n+n]=seg(y1,y2,x1,1);
231             s[i+n+n+n]=seg(y1,y2,x2,-1);
232         }
233         int ans=0;
234         int pos=1;
235         for(k=0;k<2;k++){
236             sort(ve[k].begin(),ve[k].end());
237             ve[k].erase(unique(ve[k].begin(),ve[k].end()),ve[k].end());
238             sort(s+pos,s+pos+n*n);
239             int N=ve[k].size();
240             build(1,N,1);
```

```

241         int pre=0;
242         for(int i=pos;i<pos+n+n;i++){
243             int L=getid(s[i].l);
244             int R=getid(s[i].r)-1;
245             add(L,R,s[i].flag,1,N,1);
246             ans+=abs(tree[1]-pre);
247             pre=tree[1];
248         }
249         pos+=n+n;
250     }
251     printf("%d\n",ans);
252 }
253 }

```

3.2 HLD

3.2.1 HLD

```

1  #include <bits/stdc++.h>
2  #define ll long long
3  using namespace std;
4  /*
5   node 计算点权, path 下放后计算边权, edge 根据边的编号计算边权
6   work 中没有build需手动写
7   sz[] 数组, 以x为根的子树节点个数
8   top[] 数组, 当前节点的所在链的顶端节点
9   son[] 数组, 重儿子
10  deep[] 数组, 当前节点的深度
11  fa[] 数组, 当前节点的父亲
12  idx[] 数组, 树中每个节点剖分后的新编号
13  rnk[] 数组, idx的逆, 表示线段上中当前位置表示哪个节点
14  */
15
16  const int maxn = 1e5+5;
17
18  int sz[maxn], top[maxn], son[maxn], deep[maxn], fa[maxn], idx[maxn], rnk[maxn];
19  int tot;
20  int n, le, re;
21  ll k;
22
23  struct HLD {
24  #define type int
25
26      struct edge {
27          int a, b;
28          type v;
29
30          edge(int _a, int _b, type _v = 0) : a(_a), b(_b), v(_v) {}
31      };
32
33      struct node {
34          int to;
35          type w;
36
37          node() {}
38
39          node(int _to, type _w) : to(_to), w(_w) {}
40      };
41

```



```
42     vector<int> mp[maxn];
43     vector<edge> e;
44
45     void init(int _n) {
46         n = _n;
47         for (int i = 0; i <= n; i++) mp[i].clear();
48         e.clear();
49         e.push_back(edge(0, 0));
50     }
51
52     void add_edge(int a, int b, type v = 0) {
53 //         e.push_back(edge(a,b,v));
54         mp[a].push_back(b);
55         mp[b].push_back(a);
56     }
57
58     void dfs1(int x, int pre, int h) {
59         int i, to;
60         deep[x] = h;
61         fa[x] = pre;
62         sz[x] = 1;
63         for (i = 0; i < (int) (mp[x].size()); i++) {
64             to = mp[x][i];
65             if (to == pre) continue;
66             dfs1(to, x, h + 1);
67             sz[x] += sz[to];
68             if (son[x] == -1 || sz[to] > sz[son[x]]) son[x] = to;
69         }
70     }
71
72     void dfs2(int x, int tp) {
73         int i, to;
74         top[x] = tp;
75         idx[x] = ++tot;
76         rnk[idx[x]] = x;
77         if (son[x] == -1) return;
78         dfs2(son[x], tp);
79         for (i = 0; i < (int) (mp[x].size()); i++) {
80             to = mp[x][i];
81             if (to != son[x] && to != fa[x]) dfs2(to, to);
82         }
83     }
84
85     void work(int _rt = 1) {
86         memset(son, -1, sizeof son);
87         tot = 0;
88         dfs1(_rt, 0, 0);
89         dfs2(_rt, _rt);
90     }
91
92     int LCA(int x, int y) {
93         while (top[x] != top[y]) {
94             if (deep[top[x]] < deep[top[y]]) swap(x, y);
95             x = fa[top[x]];
96         }
97         if (deep[x] > deep[y]) swap(x, y);
98         return x;
99     }
100
```

```
101 void modify_node(int x, int y, type val) {
102     while (top[x] != top[y]) {
103         if (deep[top[x]] < deep[top[y]]) swap(x, y);
104         le = idx[top[x]], re = idx[x];
105         k = val;
106         update(1, 1, n);
107         x = fa[top[x]];
108     }
109     if (deep[x] > deep[y]) swap(x, y);
110     le = idx[x], re = idx[y];
111     k = val;
112     update(1, 1, n);
113 }
114
115 type query_node(int x, int y) {
116     type res = 0;
117     while (top[x] != top[y]) {
118         if (deep[top[x]] < deep[top[y]]) swap(x, y);
119         le = idx[top[x]], re = idx[x];
120         res += query(1, 1, n);
121         x = fa[top[x]];
122     }
123     if (deep[x] > deep[y]) swap(x, y);
124     le = idx[x], re = idx[y];
125     res += query(1, 1, n);
126     return res;
127 }
128
129 //path
130 // void init_path()
131 // {
132 //     v[idx[rt]]=0;
133 //     for(int i=1;i<n;i++)
134 //     {
135 //         if(deep[e[i].a]<deep[e[i].b]) swap(e[i].a,e[i].b);
136 //         a[idx[e[i].a]]=e[i].v;
137 //     }
138 //     build(n);
139 // }
140 void modify_edge(int id, type val) {
141     if (deep[e[id].a] > deep[e[id].b]) {
142         le = idx[e[id].a], re = idx[e[id].a];
143         k = val;
144         update(1, 1, n);
145     } else {
146         le = idx[e[id].b], re = idx[e[id].b];
147         k = val;
148         update(1, 1, n);
149     }
150 }
151
152 void modify_path(int x, int y, type val) {
153     while (top[x] != top[y]) {
154         if (deep[top[x]] < deep[top[y]]) swap(x, y);
155         le = idx[top[x]], re = idx[x];
156         k = val;
157         update(1, 1, n);
158         x = fa[top[x]];
159     }
```

```
160         if (deep[x] > deep[y]) swap(x, y);
161         if (x != y) {
162             le = idx[x] + 1, re = idx[y];
163             k = val;
164             update(1, 1, n);
165         }
166     }
167
168     type query_path(int x, int y) {
169         type res = 0;
170         while (top[x] != top[y]) {
171             if (deep[top[x]] < deep[top[y]]) swap(x, y);
172             le = idx[top[x]], re = idx[x];
173             res += query(1, 1, n);
174             x = fa[top[x]];
175         }
176         if (deep[x] > deep[y]) swap(x, y);
177         if (x != y) {
178             le = idx[x] + 1, re = idx[y];
179             res += query(1, 1, n);
180         }
181         return res;
182     }
183
184 #undef type
185 } hld;
```

3.3 RMQ

3.3.1 RMQ

```
1  int A[maxn];
2  int maxx[maxn][22];
3  void RMQ(int n) {
4      for (int i = 1; i <= n; i++)
5          maxx[i][0] = A[i];
6      for (int j = 1; (1 << j) <= n; j++) {
7          for (int i = 1; i + (1 << j) - 1 <= n; i++) {
8              maxx[i][j] = max(maxx[i][j - 1], maxx[i + (1 << (j - 1))][j - 1]);
9          }
10     }
11 }
12 int query(int l, int r) {
13     int k = 0;
14     while ((1 << (k + 1)) <= r - l + 1) k++;
15     return max(maxx[l][k], maxx[r - (1 << k) + 1][k]);
16 }
17
18 template <typename T, class F = function<T(const T&, const T&)>>
19 class SparseTable {
20 public:
21     int n;
22     vector<vector<T>> mat;
23     F func;
24
25     SparseTable(const vector<T>& a, const F& f) : func(f) {
26         n = static_cast<int>(a.size());
27         int max_log = 32 - __builtin_clz(n);
28         mat.resize(max_log);
```

```

29     mat[0] = a;
30     for (int j = 1; j < max_log; j++) {
31         mat[j].resize(n - (1 << j) + 1);
32         for (int i = 0; i <= n - (1 << j); i++) {
33             mat[j][i] = func(mat[j - 1][i], mat[j - 1][i + (1 << (j - 1))]);
34         }
35     }
36 }
37
38 T get(int from, int to) const {
39     assert(0 <= from && from <= to && to <= n - 1);
40     int lg = 32 - __builtin_clz(to - from + 1) - 1;
41     return func(mat[lg][from], mat[lg][to - (1 << lg) + 1]);
42 }
43 };
44 //静态区间最大值
45 SparseTable<int> solve(v, [&](int i, int j) {return max(i, j);});

```

3.3.2 RMQbyIndex

```

1 //下标RMQ
2 int v[MAX],maxx[MAX][22],minn[MAX][22];
3 int pmax(int a,int b){return v[a]>v[b]?a:b;}
4 int pmin(int a,int b){return v[a]<v[b]?a:b;}
5 void RMQ(int n) {
6     int i,j;
7     for(i=1;i<=n;i++) {
8         maxx[i][0]=minn[i][0]=i;
9     }
10    for(j=1;1<<(j-1)<=n;j++) {
11        for(i=1;i+(1<<(j-1))-1<=n;i++) {
12            int t=1<<(j-1);
13            maxx[i][j]=pmax(maxx[i][j-1],maxx[i+t][j-1]);
14            minn[i][j]=pmin(minn[i][j-1],minn[i+t][j-1]);
15        }
16    }
17 }
18 int query(int l,int r) {
19     int j=(int)(log10(r-l+1)/log10(2))+1;
20     int i=r-(1<<(j-1))+1;
21     return pmax(maxx[l][j-1],maxx[i][j-1]);
22 // return pmin(minn[l][j-1],minn[i][j-1]);
23 }

```

3.3.3 RMQinNM

```

1 //二维RMQ
2 int v[302][302];
3 int maxx[302][302][9][9],minn[302][302][9][9];
4 void RMQ(int n,int m)
5 {
6     int i,j,ii,jj;
7     for(i=1;i<=n;i++)
8     {
9         for(j=1;j<=m;j++)
10        {
11            maxx[i][j][0][0]=minn[i][j][0][0]=v[i][j];
12        }
13    }
14 }

```

```

13     }
14     for(ii=0;(1<<ii)<=n;ii++)
15     {
16         for(jj=0;(1<<jj)<=m;jj++)
17         {
18             if(ii+jj)
19             {
20                 for(i=1;i+(1<<ii)-1<=n;i++)
21                 {
22                     for(j=1;j+(1<<jj)-1<=m;j++)
23                     {
24                         if(ii)
25                         {
26                             minn[i][j][ii][jj]=min(minn[i][j][ii-1][jj],minn[i+(1<<(ii-1))][j][ii
-1][jj]);
27                             maxx[i][j][ii][jj]=max(maxx[i][j][ii-1][jj],maxx[i+(1<<(ii-1))][j][ii
-1][jj]);
28                         }
29                         else
30                         {
31                             minn[i][j][ii][jj]=min(minn[i][j][ii][jj-1],minn[i][j+(1<<(jj-1))][ii][
jj-1]);
32                             maxx[i][j][ii][jj]=max(maxx[i][j][ii][jj-1],maxx[i][j+(1<<(jj-1))][ii][
jj-1]);
33                         }
34                     }
35                 }
36             }
37         }
38     }
39 }
40 int query(int x1,int y1,int x2,int y2)
41 {
42     int k1=0;
43     while((1<<(k1+1))<=x2-x1+1) k1++;
44     int k2=0;
45     while((1<<(k2+1))<=y2-y1+1) k2++;
46     x2=x2-(1<<k1)+1;
47     y2=y2-(1<<k2)+1;
48     return max(max(maxx[x1][y1][k1][k2],maxx[x1][y2][k1][k2]),max(maxx[x2][y1][k1][k2],maxx[x2][y2
][k1][k2]));
49 // return min(min(minn[x1][y1][k1][k2],minn[x1][y2][k1][k2]),min(minn[x2][y1][k1][k2],minn[x2][y2
][k1][k2]));
50 }

```

3.4 MO

3.4.1 MO

```

1 // const int maxn = 50005;
2
3 struct MO {
4     int l, r, id;
5 }q[maxn];
6
7 int n, m, col[maxn], block, belong[maxn];
8 int vis[maxn * 10];
9 ll res[maxn], ans;
10 bool cmp(const MO& a, const MO& b) { return belong[a.l] == belong[b.l] ? a.r < b.r : a.l < b.l; }

```

```
11 void add(int x) {
12     vis[x] ++;
13     ans += 1ll * x * (vis[x] * vis[x] - (vis[x] - 1) * (vis[x] - 1));
14 }
15
16 void del(int x) {
17     vis[x] --;
18     ans -= 1ll * x * ((vis[x] + 1) * (vis[x] + 1) - vis[x] * vis[x]);
19 }
20
21 int main() {
22     scanf("%d%d", &n, &m);
23     block = sqrt(n);
24     for (int i = 1; i <= n; ++i) {
25         scanf("%d", &col[i]);
26         belong[i] = i / block + 1;
27     }
28     for (int i = 1; i <= m; ++i) {
29         scanf("%d%d", &q[i].l, &q[i].r);
30         q[i].id = i;
31     }
32     sort(q + 1, q + 1 + m, cmp);
33     int l = 1, r = 0;
34     for (int i = 1; i <= m; ++i) {
35         while(r < q[i].r) add(col[++r]);
36         while(r > q[i].r) del(col[r--]);
37         while(l < q[i].l) del(col[l++]);
38         while(l > q[i].l) add(col[--l]);
39         res[q[i].id] = ans;
40     }
41     for (int i = 1; i <= m; ++i) printf("%lld\n", res[i]);
42     return 0;
43 }
```

3.4.2 MObyModify

```
1  #include <bits/stdc++.h>
2  #define ll long long
3  using namespace std;
4  const int maxn = 50005;
5
6  struct MO {
7      int l, r, id, oppre;
8  }q[maxn];
9
10 int n, m, col[maxn], block, belong[maxn], colpre[maxn];
11 int changepos[maxn], changepre[maxn], changenow[maxn];
12 int vis[maxn * 20];
13 int ans;
14 int res[maxn];
15 bool cmp(const MO& a, const MO& b) {
16     if (belong[a.l] != belong[b.l]) return a.l < b.l;
17     if (belong[a.r] != belong[b.r]) return a.r < b.r;
18     return a.oppre < b.oppre;
19 }
20 void add(int x) {}
21
22 void del(int x) {}
```

```
23
24 void unmodify(int pos, int now) {
25     if (q[pos].l <= changepos[now] && changepos[now] <= q[pos].r) {
26         del(changenow[now]);
27         add(changepre[now]);
28     }
29     col[changepos[now]] = changepre[now];
30 }
31
32 void modify(int pos, int now) {
33     if (q[pos].l <= changepos[now] && changepos[now] <= q[pos].r) {
34         del(changepre[now]);
35         add(changenow[now]);
36     }
37     col[changepos[now]] = changenow[now];
38 }
39
40 int main() {
41     scanf("%d%d", &n, &m);
42     block = pow(n, 0.66666);
43     for (int i = 1; i <= n; ++i) {
44         scanf("%d", &col[i]);
45         colpre[i] = col[i];
46         belong[i] = i / block + 1;
47     }
48     char s[2];
49     int t = 0, t2 = 0;
50     for (int i = 1; i <= m; ++i) {
51         scanf("%s", s);
52         if (s[0] == 'Q') {
53             ++t;
54             scanf("%d%d", &q[t].l, &q[t].r);
55             q[t].oppre = t2;
56             q[t].id = t;
57         } else {
58             ++t2;
59             scanf("%d%d", &changepos[t2], &changenow[t2]);
60             changepre[t2] = colpre[changepos[t2]];
61             colpre[changepos[t2]] = changenow[t2];
62         }
63     }
64     sort(q + 1, q + 1 + t, cmp);
65     int l = 1, r = 0, now = 0;
66     for (int i = 1; i <= t; ++i) {
67         while(r < q[i].r) add(col[++r]);
68         while(r > q[i].r) del(col[r--]);
69         while(l < q[i].l) del(col[l++]);
70         while(l > q[i].l) add(col[--l]);
71         while (now < q[i].oppre) modify(i, ++now);
72         while (now > q[i].oppre) unmodify(i, now--);
73         res[q[i].id] = ans;
74     }
75     for (int i = 1; i <= t; ++i) printf("%d\n", res[i]);
76     return 0;
77 }
```

3.4.3 分块

1 // 非预处理数组版

```
2 inline int belong(int x) { return (x - 1) / block + 1; }
3 inline int lpos(int x) { return 1 + (x - 1) * block; }
4 inline int rpos(int x) { return min(n, x * block); }
5 int sz = (n - 1) / block + 1;
6
7 // 预处理版, maxn大于1e6已经不可能处理了
8 const int maxb = 1005;
9 int n, m;
10 int belong[maxn], lpos[maxb], rpos[maxb];
11 int val[maxn], lazy[maxb];
12 int block;
13
14 scanf("%d", &n);
15 block = sqrt(n);
16 for (int i = 1; i <= n; ++i) {
17     scanf("%d", &val[i]);
18     belong[i] = (i - 1) / block + 1;
19 }
20 int sz = (n - 1) / block + 1;
21 for (int i = 1; i <= sz; ++i) {
22     lpos[i] = 1 + (i - 1) * block;
23     rpos[i] = i * block;
24 }
25 rpos[sz] = n;
```

3.4.4 弹飞绵羊

```
1 int n, m;
2 int belong[maxn], lpos[maxn], rpos[maxn];
3 int val[maxn], nxt[maxn], k[maxn], lst[maxn];
4 int block;
5
6 void update(int pos) {
7     int llim = lpos[belong[pos]], rlim = rpos[belong[pos]];
8     for (int i = pos; i >= llim; --i) {
9         if (val[i] + i > rlim) {
10             k[i] = 1;
11             nxt[i] = val[i] + i;
12             if (val[i] + i > n) lst[i] = i;
13             else lst[i] = lst[nxt[i]];
14         } else {
15             k[i] = 1 + k[val[i] + i];
16             nxt[i] = nxt[val[i] + i];
17             lst[i] = lst[val[i] + i];
18         }
19     }
20 }
21
22 void init() {
23     for (int i = n; i >= 1; --i) {
24         int rlim = rpos[belong[i]];
25         if (val[i] + i > rlim) {
26             k[i] = 1;
27             nxt[i] = val[i] + i;
28             if (val[i] + i > n) lst[i] = i;
29             else lst[i] = lst[nxt[i]];
30         } else {
31             k[i] = 1 + k[val[i] + i];
```



```
32         nxt[i] = nxt[val[i] + i];
33         lst[i] = lst[val[i] + i];
34     }
35 }
36 }
37
38 int query(int pos) {
39     int res = 0;
40     while (pos <= n) {
41         res += k[pos];
42         if (nxt[pos] > n) printf("%d ", lst[pos]);
43         pos = nxt[pos];
44     }
45     return res;
46 }
47
48 int main(int argc, char* argv[]) {
49     scanf("%d%d", &n, &m);
50     block = sqrt(n) * 1.6 + 1;
51     for (int i = 1; i <= n; ++i) {
52         scanf("%d", &val[i]);
53         belong[i] = (i - 1) / block + 1;
54     }
55     int sz = (n - 1) / block + 1;
56     for (int i = 1; i <= sz; ++i) {
57         lpos[i] = 1 + (i - 1) * block;
58         rpos[i] = i * block;
59     }
60     rpos[sz] = n;
61     init();
62     while (m--) {
63         int op;
64         scanf("%d", &op);
65         if (op == 1) {
66             int pos;
67             scanf("%d", &pos);
68             printf("%d\n", query(pos));
69         } else {
70             int pos, kl;
71             scanf("%d%d", &pos, &kl);
72             val[pos] = kl;
73             update(pos);
74         }
75     }
76     return 0;
77 }
```

3.4.5 树莫队

```
1 // rnk保存欧拉序
2 int sz[maxn], top[maxn], son[maxn], deep[maxn], fa[maxn], idx[maxn], ed[maxn], rnk[maxn*2];
3 int tot, n, m;
4 vector<int> edge[maxn];
5 int val[maxn];
6 vector<int> xpos;
7
8 inline void dfs1(int u, int pre, int h) {
9     deep[u] = h;
```

```
10     fa[u] = pre;
11     sz[u] = 1;
12     for (auto to : edge[u]) {
13         if (to == pre) continue;
14         dfs1(to, u, h + 1);
15         sz[u] += sz[to];
16         if (son[u] == 0 || sz[to] > sz[son[u]]) son[u] = to;
17     }
18 }
19
20 inline void dfs2(int u, int tp) {
21     top[u] = tp;
22     idx[u] = ++tot, rnk[tot] = u;
23     if (son[u] == 0) {
24         ed[u] = ++tot, rnk[tot] = u;
25         return;
26     }
27     dfs2(son[u], tp);
28     for (auto to : edge[u]) {
29         if (to != son[u] && to != fa[u]) dfs2(to, to);
30     }
31     ed[u] = ++tot, rnk[tot] = u;
32 }
33
34 inline int LCA(int x, int y) {
35     while (top[x] != top[y]) {
36         if (deep[top[x]] < deep[top[y]]) swap(x, y);
37         x = fa[top[x]];
38     }
39     if (deep[x] > deep[y]) swap(x, y);
40     return x;
41 }
42
43 int belong[maxn*2], block;
44 int res[maxn], ans;
45 // 每个点是否访问 (欧拉序去重)
46 int vis[maxn];
47 // 标记数组
48 int pre[maxn];
49 struct MO {
50     int l, r, id, lca;
51     bool operator < (const MO& oth) const {
52         return belong[l] == belong[oth.l] ? r < oth.r : belong[l] < belong[oth.l];
53     }
54 }q[maxn];
55
56 inline void add(int x) {
57     pre[x] ++;
58     if (pre[x] == 1) ans ++;
59 }
60
61 inline void del(int x) {
62     pre[x] --;
63     if (pre[x] == 0) ans --;
64 }
65
66 inline void deal(int x) {
67     vis[x] ? del(val[x]) : add(val[x]);
68     vis[x] = !vis[x];
```

```

69 }
70
71 int main(int argc, char* argv[]) {
72     scanf("%d%d", &n, &m);
73     block = sqrt(n);
74     xpos.resize(n + 1);
75     for (int i = 1; i <= n; ++i) {
76         scanf("%d", &val[i]);
77         xpos[i] = val[i];
78     }
79     sort(xpos.begin(), xpos.end());
80     xpos.erase(unique(xpos.begin(), xpos.end()), xpos.end());
81     for (int i = 1; i <= n; ++i) val[i] = lower_bound(xpos.begin(), xpos.end(), val[i]) - xpos.
        begin();
82     // 欧拉序长度为n两倍所以分块要分两倍大小
83     for (int i = 1; i <= n * 2; ++i) {
84         belong[i] = (i - 1) / block + 1;
85     }
86     for (int i = 1, u, v; i < n; ++i) {
87         scanf("%d%d", &u, &v);
88         edge[u].push_back(v);
89         edge[v].push_back(u);
90     }
91     // 树剖预处理lca
92     dfs1(1, 0, 0);
93     dfs2(1, 1);
94     for (int i = 1, x, y; i <= m; ++i) {
95         scanf("%d%d", &x, &y);
96         if (idx[x] > idx[y]) swap(x, y);
97         int _lca = LCA(x, y);
98         q[i].id = i;
99         if (_lca == x) q[i].l = idx[x], q[i].r = idx[y], q[i].lca = 0;
100        else q[i].l = ed[x], q[i].r = idx[y], q[i].lca = _lca;
101        // cerr << q[i].l << " " << q[i].r << " " << q[i].id << " " << q[i].lca << endl;
102    }
103    sort(q + 1, q + 1 + m);
104    int l = 1, r = 0;
105    for (int i = 1; i <= m; ++i) {
106        while(r < q[i].r) deal(rnk[++r]);
107        while(r > q[i].r) deal(rnk[r--]);
108        while(l < q[i].l) deal(rnk[l++]);
109        while(l > q[i].l) deal(rnk[--l]);
110        if (q[i].lca) deal(q[i].lca);
111        res[q[i].id] = ans;
112        if (q[i].lca) deal(q[i].lca);
113    }
114    for (int i = 1; i <= m; ++i) {
115        printf("%d\n", res[i]);
116    }
117    return 0;
118 }

```

3.5 VirtualTree

3.5.1 VirtualTree

```

1 const int pow2 = 19;
2 const int maxn = 1 << pow2;
3 vector<int> adj0[maxn], adj1[maxn];

```

```

4  int st[maxn << 1][pow2 + 1], dep[maxn], euler[maxn], euler_clock;
5  // fa0 是原树的父节点
6  // fa1 是虚树的父节点
7  // len 是虚树每个节点的权重, 每个节点代表原树的几个节点, 也是虚树到它父节点的链的长度
8  int stk[maxn], fa0[maxn], fa1[maxn], len[maxn];
9  ll val[maxn];
10
11 void link0(int u, int v) { adj0[u].emplace_back(v); adj0[v].emplace_back(u); }
12 void link1(int u, int v) { adj1[u].emplace_back(v); adj1[v].emplace_back(u); }
13 void dfs0(int u, int p) {
14     fa0[u] = p;
15     dep[u] = dep[p] + 1;
16     st[++euler_clock][0] = u;
17     euler[u] = euler_clock;
18     for (const auto& v : adj0[u]) if (v != p) {
19         dfs0(v, u);
20         st[++euler_clock][0] = u;
21     }
22 }
23 inline bool cmp(int u, int v) {return dep[u] < dep[v];}
24 inline int upper(int u, int v) {return cmp(u, v) ? u : v;}
25 void lca_init() {
26     for (int i = 0; i != 31 - __builtin_clz(euler_clock); ++i)
27         for (int j = 1; j + (1 << (i + 1)) <= euler_clock; ++j)
28             st[j][i + 1] = upper(st[j][i], st[j + (1 << i)][i]);
29 }
30 inline int lca(int u, int v) {
31     if (u == v) return u;
32     u = euler[u];
33     v = euler[v];
34     if (u > v) swap(u, v);
35     int temp = 31 - __builtin_clz(++v - u);
36     return upper(st[u][temp], st[v - (1 << temp)][temp]);
37 }
38 void build(vector<int>& key) {
39     sort(key.begin(), key.end(), [&] (int u, int v) { return euler[u] < euler[v]; });
40     key.resize(unique(key.begin(), key.end()) - key.begin());
41     int top = 0;
42     for (const auto& u : key) {
43         if (!top) {
44             stk[++top] = u;
45             continue;
46         }
47         int p = lca(u, stk[top]);
48         while (euler[p] < euler[stk[top]]) {
49             if (euler[p] >= euler[stk[top - 1]]) {
50                 link1(p, stk[top]);
51                 if (stk[--top] != p) stk[++top] = p;
52                 break;
53             }
54             link1(stk[top - 1], stk[top]);
55             --top;
56         }
57         stk[++top] = u;
58     }
59     while (top > 1) {
60         link1(stk[top - 1], stk[top]);
61         --top;
62     }

```

```

63 }
64
65 void dfs1(int u, int p) {
66     fa1[u] = p;
67     val[u] = 0;
68     len[u] = dep[u] - dep[p];
69     for (const auto& v : adj1[u]) if (v != p) dfs1(v, u);
70 }
71
72 int main() {
73     // 多组清空操作
74     for (int i = 1; i <= n; ++i) {
75         adj0[i].clear();
76         adj1[i].clear();
77     }
78     euler_clock = 0;
79
80     // 读入原树 link0 加边
81     // 读入处理关键节点存入vector key, 包含1和链的端点和他们的lca的父节点 (lca如果为1就不加)。
82     dfs0(1, 0);
83     lca_init();
84
85     vector<int> key(1, 1);
86     for (auto& q : query) {
87         cin >> q.u >> q.v;
88         key.emplace_back(q.u);
89         key.emplace_back(q.v);
90         int p = lca(q.u, q.v);
91         if (p != 1) key.emplace_back(fa0[p]);
92     }
93
94     build(key);
95     dfs1(1, 0);
96     return 0;
97 }

```

3.6 PersistentDS

3.6.1 主席树区间 k 大

```

1  // const int maxn = 100005;
2  int n, m;
3  int a[maxn];
4  int root[maxn];
5  int cnt = 0;
6  vector<int> b;
7  struct node {
8      int l, r, val;
9  }p[maxn * 40];
10
11 void update(int l, int r, int pre, int &now, int pos) {
12     now = ++cnt;
13     p[now] = p[pre];
14     p[now].val++;
15     if (l == r) {
16         return;
17     }
18     int mid = l + r >> 1;
19     if (pos <= mid) update(l, mid, p[pre].l, p[now].l, pos);

```

```
20     else update(mid + 1, r, p[pre].r, p[now].r, pos);
21 }
22
23 int query(int l, int r, int x, int y, int k) {
24     if (l == r) return b[l - 1];
25     int mid = l + r >> 1;
26     int temp = p[p[y].l].val - p[p[x].l].val;
27     if (k <= temp) return query(l, mid, p[x].l, p[y].l, k);
28     return query(mid + 1, r, p[x].r, p[y].r, k - temp);
29 }
30
31 int main(int argc, char *argv[])
32 {
33     while (scanf("%d%d", &n, &m) != EOF) {
34         b.clear();
35         cnt = 0;
36         for (int i = 1; i <= n; ++i) scanf("%d", &a[i]), b.push_back(a[i]);
37         sort(b.begin(), b.end());
38         b.erase(unique(b.begin(), b.end()), b.end());
39         for (int i = 1; i <= n; ++i) {
40             update(1, b.size(), root[i - 1], root[i], lower_bound(b.begin(), b.end(), a[i]) - b.
begin() + 1);
41         }
42         int L, R, k;
43         while (m--) {
44             scanf("%d%d%d", &L, &R, &k);
45             printf("%d\n", query(1, b.size(), root[L - 1], root[R], k));
46         }
47     }
48     return 0;
49 }
```

3.6.2 可持久化数组

```
1  /*1、操作将u, v合并 2、操作回退 */
2  const int maxn = 2e5+5;
3  int n, m, sz;
4  int root[maxn], ls[maxn*40], rs[maxn*40], v[maxn*40], deep[maxn*40];
5  int has[maxn];
6
7  void build(int &k, int l, int r) {
8      if (!k) k = ++sz;
9      if (l == r) {
10         v[k] = l;
11         return;
12     }
13     int mid = (l + r) >> 1;
14     build(ls[k], l, mid);
15     build(rs[k], mid + 1, r);
16 }
17
18 void modify(int l, int r, int x, int &y, int pos, int val) {
19     y = ++sz;
20     if (l == r) {
21         v[y] = val;
22         deep[y] = deep[x];
23         return;
24     }
```

```
25     ls[y] = ls[x];
26     rs[y] = rs[x];
27     int mid = (l + r) >> 1;
28     if (pos <= mid)
29         modify(l, mid, ls[x], ls[y], pos, val);
30     else modify(mid + 1, r, rs[x], rs[y], pos, val);
31 }
32
33 int query(int k, int l, int r, int pos) {
34     if (l == r) return k;
35     int mid = (l + r) >> 1;
36     if (pos <= mid) return query(ls[k], l, mid, pos);
37     else return query(rs[k], mid + 1, r, pos);
38 }
39
40 void add(int k, int l, int r, int pos) {
41     if (l == r) {
42         deep[k]++;
43         return;
44     }
45     int mid = (l + r) >> 1;
46     if (pos <= mid) add(ls[k], l, mid, pos);
47     else add(rs[k], mid + 1, r, pos);
48 }
49
50 int find(int k, int x) {
51     int p = query(k, 1, n, x);
52     if (x == v[p]) return p;
53     return find(k, v[p]);
54 }
55
56 int main() {
57     int T = read();
58     while (T--) {
59         sz = 0;
60         memset(root, 0, sizeof root);
61         memset(ls, 0, sizeof ls);
62         memset(rs, 0, sizeof rs);
63         n = read();
64         has[0] = n;
65         m = read();
66         build(root[0], 1, n);
67         int f, k, a, b;
68         for (int i = 1; i <= m; i++) {
69             f = read();
70             if (f == 1) {
71                 root[i] = root[i - 1];
72                 has[i] = has[i - 1];
73                 a = read();
74                 b = read();
75                 int p = find(root[i], a), q = find(root[i], b);
76                 if (v[p] == v[q]) continue;
77                 has[i]--;
78                 if (deep[p] > deep[q]) swap(p, q);
79                 modify(1, n, root[i - 1], root[i], v[p], v[q]);
80                 if (deep[p] == deep[q]) add(root[i], 1, n, v[q]);
81             } else if (f == 2) {
82                 k = read();
83                 root[i] = root[k];
```

```
84         has[i] = has[k];
85     }
86     printf("%d\n", has[i]);
87 }
88 }
89 return 0;
90 }
```

3.7 Tree

3.7.1 LCA

```
1  // const int maxn = 1e5 + 10;
2
3  // 普通倍增lca
4  int n, dep[maxn], fa[maxn][30];
5  vector<int> edge[maxn];
6
7  void dfs(int u, int pre) {
8      dep[u] = dep[pre] + 1, fa[u][0] = pre;
9      for(int i = 1; (1 << i) <= n; i++)
10         fa[u][i] = fa[fa[u][i - 1]][i - 1];
11     for(auto v : edge[u]) if(v != pre) dfs(v, u);
12 }
13
14 int LCA(int u, int v) {
15     if(dep[u] < dep[v]) swap(u, v);
16     int d = dep[u] - dep[v];
17     for(int i = 0; (1 << i) <= d; i++)
18         if((1 << i) & d) u = fa[u][i];
19     if(u == v) return u;
20     for(int i = 20; i >= 0; i--)
21         if(fa[u][i] != fa[v][i])
22             u = fa[u][i], v = fa[v][i];
23     return fa[u][0];
24 }
25
26
27 // 欧拉序lca
28 // pow2 = 19
29 // maxn = 1 << pow2
30 int st[maxn << 1][pow2 + 1], dep[maxn], euler[maxn], euler_clock, fa[maxn];
31 void dfs(int u, int p) {
32     fa[u] = p;
33     dep[u] = dep[p] + 1;
34     st[++euler_clock][0] = u;
35     euler[u] = euler_clock;
36     for (const auto& v : adj0[u]) if (v != p) {
37         dfs(v, u);
38         st[++euler_clock][0] = u;
39     }
40 }
41 void lca_init() {
42     for (int i = 0; i != 31 - __builtin_clz(euler_clock); ++i)
43         for (int j = 1; j + (1 << (i + 1)) <= euler_clock; ++j)
44             st[j][i + 1] = upper(st[j][i], st[j + (1 << i)][i]);
45 }
46 inline int lca(int u, int v) {
47     if (u == v) return u;
```



```
48     u = euler[u];
49     v = euler[v];
50     if (u > v) swap(u, v);
51     int temp = 31 - __builtin_clz(++v - u);
52     return upper(st[u][temp], st[v - (1 << temp)][temp]);
53 }
54
55 // dfs(1, 0);
56 // lca_init();
57
58 // 另有树剖lca详见hld模板
```

3.7.2 前向星

```
1 // 清零 head 和 tot
2 const int maxm = 4e5+5;
3 int ver[maxm], Next[maxm], head[maxn], edge[maxm];
4 void addEdge(int u, int v, int w){
5     ver[++tot]=v;
6     Next[tot]=head[u];
7     head[u]=tot;
8     edge[tot]=w;
9 }
10
11 for(int i = head[u]; i; i=Next[i])
```

3.7.3 点分治

```
1 int n, k;
2
3 // 清零 head 和 tot
4 const int maxm = maxn * 2;
5 int ver[maxm], Next[maxm], head[maxn], edge[maxm];
6 int tot;
7 void addEdge(int u, int v, int w){
8     ver[++tot]=v;
9     Next[tot]=head[u];
10    head[u]=tot;
11    edge[tot]=w;
12 }
13
14 int sz[maxn], vis[maxn];
15 int rt, mxsz, has;
16
17 void getrt(int u, int pre) {
18     sz[u] = 1;
19     int mxnow = 0;
20     for (int i = head[u]; i; i = Next[i]) {
21         int v = ver[i];
22         if (v == pre || vis[v]) continue;
23         getrt(v, u);
24         sz[u] += sz[v];
25         mxnow = max(mxnow, sz[v]);
26     }
27     mxnow = max(mxnow, has - sz[u]);
28     if (mxnow < mxsz) {
29         mxsz = mxnow, rt = u;
30     }
```

```
31 }
32
33 int dl[maxn], r;
34 int val[maxn];
35
36 void getdis(int u, int pre) {
37     dl[r++] = val[u];
38     for (int i = head[u]; i; i = Next[i]) {
39         int v = ver[i];
40         if (v == pre || vis[v]) continue;
41         val[v] = val[u] + edge[i];
42         getdis(v, u);
43     }
44 }
45
46 ll cal(int u, int pre) {
47     r = 0;
48     val[u] = pre;
49     getdis(u, 0);
50     ll sum = 0;
51     sort(dl, dl + r);
52     r --;
53     int l = 0;
54     while (l < r) {
55         if (dl[l] + dl[r] > k) r --;
56         else sum += r - l, l ++;
57     }
58     return sum;
59 }
60
61 ll res = 0;
62 void dfs(int u) {
63     res += cal(u, 0);
64     vis[u] = 1;
65     for (int i = head[u]; i; i = Next[i]) {
66         int v = ver[i];
67         if (vis[v]) continue;
68         res -= cal(v, edge[i]);
69         has = sz[v];
70         mxsz = 0x3f3f3f3f;
71         getrt(v, 0);
72         dfs(rt);
73     }
74 }
75
76 int main(int argc, char* argv[]) {
77     while (scanf("%d%d", &n, &k) != EOF && (n || k)) {
78         tot = 0; memset(head, 0, sizeof head);
79         memset(vis, 0, sizeof vis);
80         res = 0;
81         for (int i = 1, u, v, w; i < n; ++i) {
82             scanf("%d%d%d", &u, &v, &w);
83             addEdge(u, v, w);
84             addEdge(v, u, w);
85         }
86         mxsz = 0x3f3f3f3f;
87         has = n;
88         getrt(1, 0);
89         dfs(rt);
```

```
90     printf("%lld\n", res);
91 }
92 return 0;
93 }
```

3.8 Others

3.8.1 BITinNM

```
1 struct Fenwick_Tree {
2 #define type int
3     type bit[maxn][maxn];
4     int n, m;
5     void init(int _n, int _m) {
6         n = _n;
7         m = _m;
8         mem(bit, 0);
9     }
10    int lowbit(int x) { return x & (-x); }
11    void update(int x, int y, type v) {
12        int i, j;
13        for (i = x; i <= n; i += lowbit(i)) {
14            for (j = y; j <= m; j += lowbit(j)) {
15                bit[i][j] += v;
16            }
17        }
18    }
19    type get(int x, int y) {
20        type i, j, res = 0;
21        for (i = x; i > 0; i -= lowbit(i)) {
22            for (j = y; j > 0; j -= lowbit(j)) {
23                res += bit[i][j];
24            }
25        }
26        return res;
27    }
28    type query(int x1, int x2, int y1, int y2) {
29        x1--;
30        y1--;
31        return get(x2, y2) - get(x1, y2) - get(x2, y1) + get(x1, y1);
32    }
33 #undef type
34 } tr;
35
36 // 二维区间前缀和写法 (非树状数组)
37 inline void range_add(int xa, int ya, int xb, int yb) { add(xa, ya, 1), add(xa, yb + 1, -1), add(xb
    + 1, ya, -1), add(xb + 1, yb + 1, 1); }
38 inline ll range_ask(int xa, int ya, int xb, int yb){ return ask(xb, yb) - ask(xb, ya - 1) - ask(xa
    - 1, yb) + ask(xa - 1, ya - 1); }
39 inline void build() {
40     for (int i = 1; i < n + 5; ++i) {
41         for (int j = 1; j < m + 5; ++j) {
42             if (st[i][j] > 1) st[i][j] = 1;
43             st[i][j] += st[i - 1][j] + st[i][j - 1] - st[i - 1][j - 1];
44         }
45     }
46 }
47
48 // 二维树状数组区间加与求和
```

```

49 ll t1[maxn][maxn], t2[maxn][maxn], t3[maxn][maxn], t4[maxn][maxn];
50 void add(ll x, ll y, ll z){
51     for(int X = x; X <= n; X += X & -X)
52         for(int Y = y; Y <= m; Y += Y & -Y){
53             t1[X][Y] += z;
54             t2[X][Y] += z * x;
55             t3[X][Y] += z * y;
56             t4[X][Y] += z * x * y;
57         }
58 }
59 ll ask(ll x, ll y){
60     ll res = 0;
61     for(int i = x; i; i -= i & -i)
62         for(int j = y; j; j -= j & -j)
63             res += (x + 1) * (y + 1) * t1[i][j]
64                 - (y + 1) * t2[i][j]
65                 - (x + 1) * t3[i][j]
66                 + t4[i][j];
67     return res;
68 }
69
70 // 区间加, 询问单点: 直接维护前缀差分数组, 求单点=普通求前缀和

```

3.8.2 静态区间 k 大划分树

```

1 // const int maxn = 100010;
2 int tree[20][maxn];
3 // 读入sorted并排序, 赋值给tree的第0层
4 int sorted[maxn];
5 int toleft[20][maxn];
6 // 保存左子树的和
7 // ll sum[20][maxn];
8
9 // 1, n, 0
10 void build(int l, int r, int dep) {
11     if (l == r) return;
12     // sum[dep][0] = 0;
13     toleft[dep][0] = 0;
14     int mid = l + r >> 1;
15     int same = mid - l + 1;
16     for (int i = l; i <= r; ++i) {
17         if (tree[dep][i] < sorted[mid]) same--;
18     }
19     int lpos = l, rpos = mid + 1;
20     for (int i = l; i <= r; ++i) {
21         // sum[dep][i] = sum[dep][i - 1];
22         if (tree[dep][i] < sorted[mid]) {
23             // sum[dep][i] += tree[dep][i];
24             tree[dep + 1][lpos++] = tree[dep][i];
25         }
26         else if (tree[dep][i] == sorted[mid] && same > 0) {
27             // sum[dep][i] += tree[dep][i];
28             tree[dep + 1][lpos++] = tree[dep][i];
29             same--;
30         } else tree[dep + 1][rpos++] = tree[dep][i];
31         toleft[dep][i] = toleft[dep][l - 1] + lpos - 1;
32     }
33     build(l, mid, dep + 1);

```

```
34     build(mid + 1, r, dep + 1);
35 }
36
37 // (1~k-1)的数的和, 注意每次查询前初始化
38 // ll res = 0;
39
40 // L = 1, R = n, dep = 0, l, r是查询区间
41 int query(int L, int R, int l, int r, int dep, int k) {
42     if (l == r) return tree[dep][l];
43     int mid = (L + R) >> 1;
44     int cnt = toleft[dep][r] - toleft[dep][l - 1];
45     if (cnt >= k) {
46         int newl = L + toleft[dep][l - 1] - toleft[dep][L - 1];
47         int newr = newl + cnt - 1;
48         return query(L, mid, newl, newr, dep + 1, k);
49     } else {
50         int newr = r + toleft[dep][R] - toleft[dep][r];
51         int newl = newr - (r - l - cnt);
52         // res += sum[dep][r] - sum[dep][l - 1];
53         return query(mid + 1, R, newl, newr, dep + 1, k - cnt);
54     }
55 }
56
57
58 scan(n), scan(m);
59 for (int i = 1; i <= n; ++i) {
60     scan(sorted[i]);
61     tree[0][i] = sorted[i];
62 }
63 sort(sorted + 1, sorted + 1 + n);
64 build(1, n, 0);
65 int l, r, k;
66 while (m--) {
67     scan(l), scan(r), scan(k);
68     printf("%d\n", query(1, n, l, r, 0, k));
69 }
```

4 String

4.1 KMP

4.1.1 KMP

```

1 // nxt[0]表示失配到完全不匹配
2 int nxt[maxm];
3
4 void getNext(char *s, int len) {
5     int i = 0, j = -1;
6     nxt[i] = j;
7     while (i < len) {
8         if (j == -1 || s[i] == s[j]) nxt[++i] = ++j;
9         else j = nxt[j];
10    }
11 }
12
13 // a为原串, b为模式串, 下标从0开始, 找第一个出现模式串的位置 (起点为1), 找不到返回-1
14 int KMP(char *a, char *b, int n, int m) {
15     getNext(b, m);
16     int i = 0, j = 0;
17     while (i < n && j < m) {
18         if (j == -1 || a[i] == b[j]) ++i, ++j;
19         else j = nxt[j];
20    }
21    return j == m ? i - m + 1 : -1;
22 }

```

4.1.2 exKMP

```

1 const int maxn = 1e5 + 10;
2 int nex[maxn], extend[maxn];
3
4 //预处理计算Next数组
5 void getNext(char *str)
6 {
7     int i = 0, j, po, len = strlen(str);
8     nex[0] = len; //初始化nex[0]
9     while (str[i] == str[i + 1] && i + 1 < len) i++; //计算nex[1]
10    nex[1] = i;
11    po = 1; //初始化po的位置
12    for (int i = 2; i < len; i++)
13    {
14        if (nex[i - po] + i < nex[po] + po) //第一种情况, 可以直接得到nex[i]的值
15            nex[i] = nex[i - po];
16        else //第二种情况, 要继续匹配才能得到nex[i]的值
17        {
18            j = nex[po] + po - i;
19            if (j < 0) j = 0; //如果i>po+nex[po],则要从头开始匹配
20            while (i + j < len && str[j] == str[j + i]) j++;
21            nex[i] = j;
22            po = i; //更新po的位置
23        }
24    }
25 }
26
27 void EXKMP(char *s1, char *s2)
28 {

```

```
29     int i = 0, j, po, len = strlen(s1), l2 = strlen(s2);
30     getNext(s2);
31     while (s1[i] == s2[i] && i < l2 && i < len) i++;
32     extend[0] = i;
33     po = 0;
34     for (int i = 1; i < len; i++)
35     {
36         if (nex[i - po] + i < extend[po] + po)
37             extend[i] = nex[i - po];
38         else
39         {
40             j = extend[po] + po - i;
41             if (j < 0) j = 0;
42             while (i + j < len && j < l2 && s1[j + i] == s2[j]) j++;
43             extend[i] = j;
44             po = i;
45         }
46     }
47 }
```

4.2 Trie

4.2.1 Trie

```
1  const int maxn = 2e6 + 10;
2
3  int trie[maxn][30], tot;
4  bool flag[maxn];
5
6  void insert_ch(char *str)
7  {
8      int len = strlen(str);
9      int root = 0;
10     for (int i = 0; i < len; i++)
11     {
12         int id = str[i] - 'a';
13         if (!trie[root][id]) trie[root][id] = ++tot;
14         root = trie[root][id];
15     }
16     flag[root] = true;
17 }
18
19 bool find_ch(char *str)
20 {
21     int len = strlen(str);
22     int root = 0;
23     for (int i = 0; i < len; i++)
24     {
25         int id = str[i] - 'a';
26         if (!trie[root][id]) return false;
27         root = trie[root][id];
28     }
29     return true;
30 }
```

4.2.2 Persistence Trie

```
1  const int maxn = 1e5 + 10;
```

```
2
3 int a[maxn], rt[maxn], n;
4
5 struct Trie
6 {
7     int tot;
8     int child[maxn * 32][2], sum[maxn * 32];
9     int insert(int x, int val)
10    {
11        int tmp, y;
12        tmp = y = ++tot;
13        for(int i = 30; i >= 0; --i)
14        {
15            child[y][0] = child[x][0];
16            child[y][1] = child[x][1];
17            sum[y] = sum[x] + 1;
18            int t = val >> i & 1;
19            x = child[x][t];
20            child[y][t] = ++tot;
21            y = child[y][t];
22        }
23        sum[y] = sum[x] + 1;
24        return tmp;
25    }
26    int query(int l, int r, int val)
27    {
28        int tmp = 0;
29        for(int i = 30; i >= 0; --i)
30        {
31            int t = val >> i & 1;
32            if(sum[child[r][t^1]] - sum[child[l][t^1]]) tmp += (1<<i), r = child[r][t^1], l = child
[l][t^1];
33            else r = child[r][t], l = child[l][t];
34        }
35        return tmp;
36    }
37 }trie;
```

4.2.3 01Trie

```
1 struct Trie {
2     int tree[maxn*20][2], tot;
3     int flag[maxn*20];
4
5     void insert_ch(int x) {
6         int root = 0;
7         flag[0]++;
8         for (int i = 30; i >= 0; --i) {
9             int id = (x >> i) & 1;
10            if (!tree[root][id]) {
11                tree[root][id] = ++tot;
12                tree[tree[root][id]][0] = tree[tree[root][id]][1] = 0;
13                flag[tree[root][id]] = flag[tree[tree[root][id]][0]] = flag[tree[tree[root][id]]
]] [1]] = 0;
14            }
15            root = tree[root][id];
16            flag[root]++;
17        }
18    }
```



```
18     }
19
20     void del(int x) {
21         int root = 0;
22         flag[0]--;
23         for (int i = 30; i >= 0; --i) {
24             int id = (x >> i) & 1;
25             assert(tree[root][id]);
26             if (flag[tree[root][id]] == 1) {
27                 flag[tree[root][id]] = 0;
28                 tree[root][id] = 0;
29                 return;
30             }
31             root = tree[root][id];
32             flag[root]--;
33         }
34     }
35
36     int find_ch(int x, int flag = 0) { // flag 0 最小异或值, 1 最大异或值
37         int root = 0;
38         int res = 0;
39         for (int i = 30; i >= 0; --i) {
40             int id = ((x >> i) & 1);
41             if (flag) id = !id;
42             if (tree[root][id]) {
43                 root = tree[root][id];
44                 res = res << 1 | id;
45             } else {
46                 root = tree[root][!id];
47                 res = res << 1 | (!id);
48             }
49         }
50         return res;
51     }
52
53     void init() {
54         tree[0][0] = tree[0][1] = 0;
55         tot = 0;
56     }
57 };
```

4.3 Manacher

4.3.1 Manacher

```
1  const int maxn = 1e5 + 10;
2
3  char s[maxn];
4
5  char tmp[maxn << 1];
6  int Len[maxn << 1];
7
8  int init(char *str)
9  {
10     int len = strlen(str);
11     tmp[0] = '@';
12     for (int i = 1; i <= 2 * len; i += 2)
13     {
14         tmp[i] = '#';
```

```
15     tmp[i + 1] = str[i / 2];
16 }
17 tmp[2 * len + 1] = '#';
18 tmp[2 * len + 2] = '$';
19 tmp[2 * len + 3] = 0;
20 return 2 * len + 1;
21 }
22
23 int manacher(char *str)
24 {
25     int mx = 0, ans = 0, pos = 0;
26     int len = init(str);
27     for (int i = 1; i <= len; i++)
28     {
29         if (mx > i) Len[i] = min(mx - i, Len[2 * pos - i]);
30         else Len[i] = 1;
31         while (tmp[i - Len[i]] == tmp[i + Len[i]]) Len[i]++;
32         if (Len[i] + i > mx) mx = Len[i] + i, pos = i;
33     }
34 }
```

4.4 Aho-Corasick Automation

4.4.1 AC Automation

```
1 class AC_automation
2 {
3 public:
4     int trie[maxn][26], cnt;
5     int tag[maxn];
6     int fail[maxn], num[maxn], res[maxn], in[maxn], Map[maxn];
7
8     void init()
9     {
10         memset(trie, 0, sizeof trie);
11         memset(tag, 0, sizeof tag);
12         memset(fail, 0, sizeof fail);
13         cnt = 0;
14     }
15
16     void insert(char *str, int id)
17     {
18         int root = 0;
19         for (int i = 0; str[i]; i++)
20         {
21             int id = str[i] - 'a';
22             if (!trie[root][id]) trie[root][id] = ++cnt;
23             root = trie[root][id];
24         }
25         if (!tag[root]) tag[root] = id;
26         Map[id] = tag[root];
27     }
28
29     void build()
30     {
31         queue<int> que;
32         for (int i = 0; i < 26; i++) if (trie[0][i]) que.push(trie[0][i]);
33         while (!que.empty())
34         {
```

```
35     int k = que.front();
36     que.pop();
37     for (int i = 0; i < 26; i++)
38     {
39         if (trie[k][i])
40         {
41             fail[trie[k][i]] = trie[fail[k]][i];
42             que.push(trie[k][i]);
43             in[fail[trie[k][i]]] ++;
44         } else trie[k][i] = trie[fail[k]][i];
45     }
46 }
47 }
48
49 void toposort()
50 {
51     queue<int> que;
52     for(int i = 1; i <= cnt; i ++) if(in[i] == 0) que.push(i);
53     while(!que.empty())
54     {
55         int u = que.front(); que.pop();
56         res[tag[u]] = num[u];
57         int v = fail[u]; in[v] --;
58         num[v] += num[u];
59         if(in[v] == 0) que.push(v);
60     }
61 }
62
63 void query(char *str, int n)
64 {
65     int u = 0, len = strlen(s);
66     for(int i = 0; i < len; i ++)
67         u = trie[u][str[i] - 'a'], num[u] ++;
68     toposort();
69     for(int i = 1; i <= n; i ++) printf("%d\n", res[Map[i]]);
70 }
71 } AC;
```

4.5 Suffix Array

4.5.1 Suffix Array

```
1 char s[maxn];
2 int sa[maxn], t[maxn], t2[maxn], c[maxn], n;
3
4 //build_sa(n + 1, 130), sa, height下标从1开始, rk下标从0开始
5 void build_sa(int n, int m)
6 {
7     int *x = t, *y = t2;
8     for(int i = 0; i < m; i++) c[i] = 0;
9     for(int i = 0; i < n; i++) c[x[i]] = s[i]++;
10    for(int i = 1; i < m; i++) c[i] += c[i - 1];
11    for(int i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
12    for(int k = 1; k <= n; k <= 1)
13    {
14        int p = 0;
15        for(int i = n - k; i < n; i++) y[p++] = i;
16        for(int i = 0; i < n; i++) if(sa[i] >= k) y[p++] = sa[i] - k;
17        for(int i = 0; i < m; i++) c[i] = 0;
```

```
18     for(int i = 0; i < n; i++) c[x[y[i]]]++;
19     for(int i = 0; i < m; i++) c[i] += c[i - 1];
20     for(int i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
21     swap(x, y);
22     p = 1; x[sa[0]] = 0;
23     for(int i = 1; i < n; i++)
24         x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k] ? p - 1 : p++;
25     if(p >= n) break;
26     m = p;
27 }
28 }
29
30 int rk[maxn], height[maxn];
31
32 void getHeight()
33 {
34     for(int i = 1; i <= n; i++) rk[sa[i]] = i;
35     for(int i = 0, k = 0; i < n; i++)
36     {
37         if(k) k--;
38         int j = sa[rk[i] - 1];
39         while(s[i + k] == s[j + k]) k++;
40         height[rk[i]] = k;
41     }
42 }
43
44 int dp[maxn][20];
45
46 void RMQ()
47 {
48     for(int i = 1; i <= n; i++) dp[i][0] = height[i];
49     for(int j = 1; (1 << j) < maxn; j++)
50         for(int i = 1; i + (1 << j) - 1 <= n; i++)
51             dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
52 }
53
54 int query(int l, int r)
55 {
56     int k = 0;
57     while((1 << (k + 1)) <= r - l + 1) k++;
58     return min(dp[l][k], dp[r - (1 << k) + 1][k]);
59 }
60
61 int lcp(int x, int y)
62 {
63     x = rk[x], y = rk[y];
64     if(x > y) swap(x, y);
65     return query(x + 1, y);
66 }
```

4.6 PalindromicTree

4.6.1 PalindromicTree

```
1 const int maxn = 2e6+6;
2 const int N = 26;
3 const int mod = 51123987;
4
5 struct Palindromic_Tree {
```

```

6  //    vector<pair<int, int> > next[maxn];
7  int next[maxn][N]; //next指针, next指针和字典树类似, 指向的串为当前串两端加上同一个字符构成
8  int fail[maxn]; //fail指针, 失配后跳转到fail指针指向的节点
9  int cnt[maxn]; //表示节点i表示的本质不同的串的个数 (建树时求出的不是完全的, 最后count()函数跑一遍
    以后才是正确的)
10 int num[maxn]; //表示以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数
11 int len[maxn]; //len[i]表示节点i表示的回文串的长度 (一个节点表示一个回文串)
12 int S[maxn]; //存放添加的字符
13 int last; //指向新添加一个字母后所形成的最长回文串表示的节点。
14 int n; //表示添加的字符个数。
15 int p; //表示添加的节点个数。
16 //0向前加, 1向后加字符
17 //int last[2];
18 //int lpos, rpos;
19
20 int newnode(int l) { //新建节点
21 //    next[p].clear();
22     for (int i = 0; i < N; ++i) next[p][i] = 0;
23     cnt[p] = 0;
24     num[p] = 0;
25     len[p] = 1;
26     return p++;
27 }
28
29 void init() { //初始化
30     n = last = p = 0;
31     newnode(0);
32     newnode(-1);
33     S[n] = -1; //开头放一个字符集中没有的字符, 减少特判
34     fail[0] = 1;
35     // lpos 为字符串最大长度
36     // last[0] = last[1] = 0;
37     // lpos = 100000, rpos = lpos - 1;
38     // S[lpos - 1] = S[rpos + 1] = -1;
39 }
40
41 int get_fail(int x) { //和KMP一样, 失配后找一个尽量最长的
42     // op 0 向前, 1 向后
43     // if (op == 0) while (S[lpos + len[x] + 1] != S[lpos]) x = fail[x];
44     // else while (S[rpos - len[x] - 1] != S[rpos]) x = fail[x];
45     while (S[n - len[x] - 1] != S[n]) x = fail[x];
46     return x;
47 }
48
49 //    int find(int u, int c) {
50 //        vector<pair<int, int> > & x = next[u];
51 //        int sz = x.size();
52 //        for(int i = 0; i < sz; ++i) {
53 //            if(x[i].first == c) return x[i].second;
54 //        }
55 //        return 0;
56 //    }
57
58 int add(int c) {
59     // 注意清空左右字符
60     // if (op == 0) S[--lpos] = c, S[lpos - 1] = -1;
61     // else S[++rpos] = c, S[rpos + 1] = -1;
62     S[++n] = c;
63     int cur = get_fail(last); //通过上一个回文串找这个回文串的匹配位置

```

```

64 //     int x = find(cur, c);
65 //     if (!x) {
66 //         if (!next[cur][c]) { // 如果这个回文串没有出现过, 说明出现了一个新的本质不同的回文串
67 //             int now = newnode(len[cur] + 2); // 新建节点
68 //             x = now;
69 //             fail[now] = find(get_fail(fail[cur]), c);
70 //             next[cur].emplace_back(make_pair(c, now));
71 //             fail[now] = next[get_fail(fail[cur])][c]; // 和AC自动机一样建立fail指针, 以便失配后跳转
72 //             next[cur][c] = now;
73 //             num[now] = num[fail[now]] + 1;
74 //         }
75 //         last = x;
76 //         // 修改最终长度
77 //         // if (len[last[op]] == rpos - lpos + 1) last[op ^ 1] = last[op];
78 //         last = next[cur][c];
79 //         cnt[last]++;
80 //         return num[last];
81 //     }
82
83 void count() {
84     for (int i = p - 1; i >= 0; --i) cnt[fail[i]] += cnt[i];
85     // 父亲累加儿子的cnt, 因为如果fail[v]=u, 则u一定是v的子回文串!
86 }
87 } solve;
88
89 char s[maxn];
90
91 // 求相交回文串数量
92 ll a[maxn], b[maxn];
93 int main() {
94     solve.init();
95     int n;
96     scanf("%d", &n);
97     scanf("%s", s);
98     for (int i = 0; i < n; ++i) a[i] = solve.add(s[i] - 'a');
99     solve.init();
100    for (int i = n - 1; i >= 0; --i) b[i] = (b[i + 1] + solve.add(s[i] - 'a')) % mod;
101    ll res = (b[0] * (b[0] - 1) / 2) % mod;
102    for (int i = 0; i < n; ++i) res = ((res - (a[i] * b[i + 1]) + mod) % mod) % mod;
103    printf("%lld\n", res);
104    return 0;
105 }

```

4.7 Hash

4.7.1 hash

```

1 // hash常用素数
2 // 61, 83, 113, 151, 211
3 // 91815541, 38734667, 68861641
4 // 917120411, 687840301, 386910137, 515880193
5 // 1222827239, 1610612741
6
7 typedef unsigned long long ull;
8 struct mhash {
9     // 自然溢出无模数 805306457
10    ull base[maxn];
11    ull hash_index[maxn];
12    ull seed; // 31, 131

```

```

13 void inithash(ull seedt = 31) {
14     base[0] = 1;
15     seed = seedt;
16     for (int i = 1; i < maxn; ++i) base[i] = base[i - 1] * seed;
17 }
18 void H(char *p, int n) { // from 1 to n
19     hash_index[0] = 0;
20     for (int i = 1; i <= n; ++i) hash_index[i] = hash_index[i - 1] * seed + p[i] - 'a';
21 }
22 ull gethash(int s, int e) {
23     return hash_index[e] - hash_index[s - 1] * base[e - s + 1];
24 }
25 };
26
27 // 26个素数, 解决加法hash
28 int prime[] = {34183,13513,152993,13591,19687,350869,111187,766091,769297,
29               633469,752273,298651,617191,880421,136067,
30               1408397,726899,458921,2133701,2599847,2730947,4696343,10267237,
31               18941059,34078909,69208409};

```

4.7.2 doubleHash

```

1 namespace Hash{
2
3     template<class __A,class __B>
4     class Hash{
5     private:
6         static const int size=2000000;
7         __B *hash; __A *0; int sz;
8     public:
9         Hash(int hash_size=size){ sz=hash_size;
10             hash=(__B *)malloc(sizeof(__B)*sz);
11             0=(__A *)malloc(sizeof(__A)*sz);
12             memset(0,0xff,sizeof(__A)*sz);
13         }~Hash(){free(0);free(hash);}
14         __B &operator [] (const __A &_0){
15             int loc=_0%sz;
16             while(~0[loc]&&0[loc]!=_0){
17                 ++loc;
18                 if(loc>sz)loc=0;
19             }if(!~0[loc])0[loc]=_0;
20             return hash[loc];
21         }
22         void clear(){memset(0,0xff,sizeof(__A)*sz);}
23     };
24
25     struct StringDoubleHashResult{
26         int32_t *H1,*H2,c_len,len;
27         StringDoubleHashResult(int32_t sz=0){
28             len=sz; c_len=0; //cur_len;
29             if(len<=0){
30                 H1=H2=0;
31                 return;
32             }
33             H1=(int32_t *)malloc(sizeof(int32_t)*sz);
34             H2=(int32_t *)malloc(sizeof(int32_t)*sz);
35         }
36         ~StringDoubleHashResult(){}

```

```

37     void clear(){free(H1);free(H2);len=0;H1=H2=0;}
38     void resize(int new_len){
39         int32_t *T1=(int32_t *)malloc(sizeof(int32_t)*new_len);
40         int32_t *T2=(int32_t *)malloc(sizeof(int32_t)*new_len);
41         for(int i=0;i<c_len;++i)T1[i]=H1[i],T2[i]=H2[i];
42         free(H1);free(H2); H1=T1; H2=T2; len=new_len;
43     }
44     void erase(int ers_len){//erase suffix
45         c_len-=ers_len;
46         if(c_len<0)c_len=0;
47     }
48     //erase prefix not better than recalc
49 };
50
51 namespace hash_random{
52     const int mod_tot=5;
53     const int mod[]={1000000009,1000000007,998244353,917120411,515880193};
54 };
55
56 class StringDoubleHash{
57 private:
58     static const int enable_random=1;
59     int32_t sz,HA1,HA2;
60     long long B,C;
61     int32_t *H1,*H2;
62 public:
63     StringDoubleHash(int32_t SZ=2e6+5,int32_t ha1=-1,int32_t ha2=-1,int32_t b=-1,int32_t c=-1){
64         sz=SZ;
65         if(enable_random){
66             std::mt19937 rnd(time(0)+19990630);
67             int z1= rnd() % hash_random::mod_tot;
68             int z2= (z1 +rnd()%(hash_random::mod_tot - 1) + 1) % hash_random::mod_tot;
69             if(ha1<0)ha1=hash_random::mod[z1];
70             if(ha2<0)ha2=hash_random::mod[z2];
71             if(b<0)b=rnd()%114514+23333;
72             if(c<0)c=rnd()%1919810+23333;
73         } else {
74             if(ha1<0)ha1=1e9+7;
75             if(ha2<0)ha2=1e9+9;
76             if(b<0)b=114514;
77             if(c<0)c=1919810;
78         }
79         HA1=ha1; HA2=ha2; B=b; C=c;
80         //cerr<<HA1<<" "<<HA2<<" "<<B<<" "<<C<<endl;
81         H1=(int32_t *)malloc(sizeof(int32_t)*sz);
82         H2=(int32_t *)malloc(sizeof(int32_t)*sz);
83         init_hash_val();
84     }
85     ~StringDoubleHash(){free(H1);free(H2);}
86     void init_hash_val(){
87         H1[0]=H2[0]=1;
88         for(int32_t i=1;i<sz;++i){
89             H1[i]=(H1[i-1]*B)%HA1;
90             H2[i]=(H2[i-1]*B)%HA2;
91         }
92     }
93     template <class _Tp>
94     StringDoubleHashResult culc_hash(const _Tp &s,int32_t len,int32_t tot_len=-1){
95         if(tot_len<0)tot_len=len;

```



```

96     StringDoubleHashResult R(tot_len);
97     if(len<=0)return R;
98     R.H1[0]=(s[0]+C)%HA1;
99     R.H2[0]=(s[0]+C)%HA2;
100    for(int32_t i=1;i<len;++i){
101        R.H1[i]=(R.H1[i-1]*B+s[i]+C)%HA1;
102        R.H2[i]=(R.H2[i-1]*B+s[i]+C)%HA2;
103    }
104    R.c_len=len;
105    return R;
106 }
107 // s is the char* first, len is the append length
108 template <class _Tp>
109 void append(StringDoubleHashResult &R,const _Tp &s,int32_t len){
110     if(len<=0)return;
111     int t_len=R.len;
112     while(R.c_len+len>t_len)t_len<=1;
113     if(t_len>R.len)R.resize(t_len);
114     for(int32_t i=R.c_len;i<R.c_len+len;++i){
115         if(i==0){
116             R.H1[i]=(s[i-R.c_len]+C)%HA1;
117             R.H2[i]=(s[i-R.c_len]+C)%HA2;
118         } else {
119             R.H1[i]=(R.H1[i-1]*B+s[i-R.c_len]+C)%HA1;
120             R.H2[i]=(R.H2[i-1]*B+s[i-R.c_len]+C)%HA2;
121         }
122     }
123     R.c_len+=len;
124 }
125 void append(StringDoubleHashResult &R, char s){
126     int t_len=R.len;
127     while(R.c_len+1>t_len)t_len<=1;
128     if(t_len>R.len)R.resize(t_len);
129     for(int32_t i=R.c_len;i<R.c_len+1;++i){
130         if(i==0){
131             R.H1[i]=(s+C)%HA1;
132             R.H2[i]=(s+C)%HA2;
133         } else {
134             R.H1[i]=(R.H1[i-1]*B+s+C)%HA1;
135             R.H2[i]=(R.H2[i-1]*B+s+C)%HA2;
136         }
137     }
138     R.c_len+=1;
139 }
140 //return hash [l,r)
141 ll gethash(const StringDoubleHashResult &R, int32_t l,int32_t r){
142     if(l>r||l<0||r-->R.c_len)return -1;//fail
143     ll v1=l>0?R.H1[l-1]*(long long)H1[r-l+1]%HA1:0;
144     ll v2=l>0?R.H2[l-1]*(long long)H2[r-l+1]%HA2:0;
145     v1=R.H1[r]-v1; v2=R.H2[r]-v2;
146     if(v1<0)v1+=HA1; if(v2<0)v2+=HA2;
147     return v1<<32|v2;
148 }
149 //merge two hashes as one(s1+s2), but need s2's length
150 ll merge_hash(const long long &hs1,const long long &hs2,int lenr){
151     int32_t m1=hs1>>32,m2=hs1&0xffffffffLL;
152     int32_t m3=hs2>>32,m4=hs2&0xffffffffLL;
153     m1=m1*(long long)H1[lenr]%HA1+m3;
154     if(m1>=HA1)m1-=HA1;

```

```

155         m2=m2*(long long)H2[lenr]%HA2+m4;
156         if(m2>=HA2)m2-=HA2;
157         return (long long)m1<<32|m2;
158     }
159 };
160 };

```

4.7.3 二维 hash

```

1  #define ull unsigned long long
2  const int maxn = 1005;
3  ull hs[maxn][maxn];
4  char a[maxn][maxn];
5  int n, m;
6  ull base1 = 131, base2 = 13331;
7  ull pwb1[maxn] = {1}, pwb2[maxn] = {1};
8
9  void init() {
10     for (int i = 1; i < maxn; ++i) {
11         pwb1[i] = pwb1[i - 1] * base1;
12         pwb2[i] = pwb2[i - 1] * base2;
13     }
14 }
15
16 void Hash() {
17     for(int i=1;i<=n;i++)
18         for(int j=1;j<=m;j++)
19             hs[i][j]=hs[i][j-1]*base1+a[i][j] - 'a';
20     for(int i=1;i<=n;i++)
21         for(int j=1;j<=m;j++)
22             hs[i][j]+=hs[i-1][j]*base2;
23 }
24
25 // 右下角(i,j), 行列长度n,m
26 ull getHs(int i, int j, int lenn, int lenm) {
27     return hs[i][j] - hs[i - lenn][j] * pwb2[lenn] -
28           hs[i][j - lenm] * pwb1[lenm] +
29           hs[i - lenn][j - lenm] * pwb2[lenn] * pwb1[lenm];
30 }

```

4.7.4 树 hash 同构

```

1  // n=1e5的话base开2e6+9, 可以输出看到top不比n小即可
2  const int base = 2e6+9;
3  // vis大小要开到素数大小, turn表示当前树的编号, p是预处理数组
4  int vis[base + 1], top, turn, p[base + 1];
5  // 程序开头调用一次
6  void init() {
7      top = 0;
8      for (int i = 2; i <= base; ++i) {
9          if (!vis[i]) {
10             p[++top] = i;
11         }
12         for (int j = 1; j <= top && i * p[j] <= base; ++j) {
13             vis[i * p[j]] = 1;
14             if (i % p[j] == 0) break;
15         }
16     }

```

```

17     assert(top >= maxn);
18 }
19
20 vector<int> edge[maxn];
21 // h[x]表示x这棵子树的hash值, g[x]表示以x为根的hash值
22 int h[maxn], g[maxn], sz[maxn];
23
24 struct TreeHash {
25     int n;
26     // 如果树比较多, 在类内部开edge可能会炸内存, 可以改到外面做前向星
27     // 除了hs是答案其他都可以改到外部, 只有edge需要清零
28     // vector<int> edge[maxn];
29     // int h[maxn], g[maxn], sz[maxn];
30     vector<int> hs;
31
32     void init(int n_ = 0) {
33         n = n_;
34         hs.clear();
35     }
36
37     void dfs1(int u, int pre) {
38         sz[u] = 1;
39         h[u] = 1;
40         for (auto v : edge[u]) {
41             if (v == pre) continue;
42             dfs1(v, u);
43             h[u] = (h[u] + 111 * h[v] * p[sz[v]] % mod) % mod;
44             sz[u] += sz[v];
45         }
46     }
47
48     void dfs2(int u, int pre, int V, int needres = 1) {
49         g[u] = (h[u] + 111 * V * p[n - sz[u]] % mod) % mod;
50         if (needres) hs.push_back(g[u]);
51         for (auto v : edge[u]) {
52             if (v == pre) continue;
53             dfs2(v, u, (g[u] - 111 * h[v] * p[sz[v]] % mod + mod) % mod);
54         }
55     }
56
57     void work(int needres = 1) {
58         // 无根树选一个不存在的点当pre即可, 当多棵无根树判重时需要sort
59         dfs1(1, 0);
60         dfs2(1, 0, 0, needres);
61         sort(hs.begin(), hs.end());
62     }
63 };
64
65 // 获取删掉某叶子节点后以与该叶子节点相邻点开头的hash值
66 // int res = (hs[edge[i][0]] - 2 + mod) % mod;

```

4.8 Suffix Automation

4.8.1 SAM

```

1 const int maxn = 2e4 + 10;
2
3 struct SuffixAutomation
4 {

```

```

5     int last, cnt;
6     int ch[maxn << 1][26], fa[maxn << 1], len[maxn << 1], pos[maxn << 1];
7     int sz[maxn << 1], a[maxn << 1], c[maxn << 1];
8
9     void init()
10    {
11        last = cnt = 1;
12        memset(ch[1], 0, sizeof ch[1]);
13        fa[1] = len[1] = 0;
14    }
15
16    int inline newnode(int idx)
17    {
18        ++cnt;
19        memset(ch[cnt], 0, sizeof ch[cnt]);
20        fa[cnt] = len[cnt] = 0;
21        pos[cnt] = idx;
22        return cnt;
23    }
24
25    void ins(int c)
26    {
27        int p = last , np = newnode(pos[last] + 1);
28        last = np, len[np] = len[p] + 1;
29        for(; p && !ch[p][c]; p = fa[p]) ch[p][c] = np;
30        if(!p) fa[np] = 1;
31        else
32        {
33            int q = ch[p][c];
34            if(len[p] + 1 == len[q]) fa[np] = q;
35            else
36            {
37                int nq = newnode(pos[p] + 1);
38                len[nq] = len[p] + 1;
39                memcpy(ch[nq], ch[q], sizeof ch[q]);
40                fa[nq] = fa[q], fa[q] = fa[np] = nq;
41                for(; ch[p][c] == q; p = fa[p]) ch[p][c] = nq;
42            }
43        }
44        sz[np] = 1;
45    }
46
47    int solve(int n)
48    {
49        /*求两个串的LCS:
50         对一个字符串建立SAM, 记录一个当前匹配的长度Len和当前节点v, 枚举另一个字符串的每个字符;
51         如果p有字符v的转移边出边, 则使Len加一, 并使p转移到出边指向的节点上;
52         否则不断向父节点上跳, 直到当前节点有字符p的转移出边, 或者跳到根节点;
53        */
54        int p = 1, ans = 0, now_len = 0;
55        for(int i = 0; s2[i]; i++)
56        {
57            if(ch[p][s2[i] - 'a']) p = ch[p][s2[i] - 'a'], now_len++;
58            else
59            {
60                for(;p && !ch[p][s2[i] - 'a'] ; p = fa[p]) ;
61                if(p == 0) now_len = 0, p = 1;
62                else now_len = len[p] + 1, p = ch[p][s2[i] - 'a'];
63            }

```

```
64         ans = max(now_len, ans);
65     }
66 }
67
68 void Toposort()
69 {
70     long long ans = 0;
71     for(int i = 1; i <= cnt; i++) c[len[i]] ++;
72     for(int i = 1; i <= cnt; i++) c[i] += c[i - 1];
73     for(int i = 1; i <= cnt; i++) a[c[len[i]] --] = i;
74     for(int i = cnt; i; i--) sz[fa[a[i]]] += sz[a[i]];
75 }
76 }sam;
```

4.9 Others

4.9.1 最小表示法

```
1  // 0起始
2  int Gao(char a[], int len) {
3      int i = 0, j = 1, k = 0;
4      while (i < len && j < len && k < len) {
5          int cmp = a[(j + k) % len] - a[(i + k) % len];
6          if (cmp == 0) k++;
7          else {
8              if (cmp > 0) j += k + 1;
9              else i += k + 1;
10             if (i == j) j++;
11             k = 0;
12         }
13     }
14     return min(i, j);
15 }
```

5 dp

5.1 BitDP

5.1.1 数位 dp 计和

```

1  #include <bits/stdc++.h>
2  #define ll long long
3  using namespace std;
4  const int mod = 998244353;
5  pair<ll, ll> dp[20][1<<10];
6  bool vis[20][1<<10];
7  int k;
8  int t[20];
9  ll base[20];
10
11 pair<ll, ll> dfs(int pos, int state, bool limit, bool lead) {
12     if (pos == -1) return __builtin_popcount(state) <= k ? make_pair(1, 0) : make_pair(0, 0);
13     if (!limit && !lead && vis[pos][state]) return dp[pos][state];
14     int up = limit ? t[pos] : 9;
15     pair<ll, ll> res = {0, 0};
16     for (int i = 0; i <= up; ++i) {
17         int n_s = state;
18         if (lead && i == 0) n_s = 0;
19         else n_s = state | (1 << i);
20         auto tmp = dfs(pos - 1, n_s, limit && i == t[pos], lead && i == 0);
21         ll pre = 1ll * i * base[pos] % mod;
22         (res.first += tmp.first) %= mod;
23         (res.second += tmp.second + pre * tmp.first) %= mod;
24     }
25     if (!limit && !lead) dp[pos][state] = res, vis[pos][state] = 1;
26     return res;
27 }
28
29 ll solve(ll x) {
30     int pos = 0;
31     do {
32         t[pos++] = x % 10;
33     } while (x /= 10);
34     return dfs(pos - 1, 0, true, true).second;
35 }
36
37 int main(int argc, char *argv[])
38 {
39     base[0] = 1;
40     for (int i = 1; i < 20; ++i) base[i] = base[i - 1] * 10;
41     ll l, r;
42     scanf("%lld%lld%d", &l, &r, &k);
43     printf("%lld\n", (solve(r) - solve(l - 1) + mod) % mod);
44     return 0;
45 }

```

5.1.2 两个数数位 dp

```

1  // 二进制数位dp, 求a $\\in$ $1\sim x$ 和 b $\\in$ $1\sim y$, 满足 $a \& b > c$ || $a \wedge b < c$的对数
2  ll dp[maxn][2][2][2][2];
3  int a[maxn], b[maxn], c[maxn];
4
5

```

```
6 void cal(int *xt, ll x) {
7     int has = 0;
8     while (x) {
9         xt[has++] = x % 2;
10        x /= 2;
11    }
12 }
13
14 ll dfs(int pos, int o1, int o2, int lim1, int lim2) {
15     if (pos < 0) return 1;
16     ll &t = dp[pos][o1][o2][lim1][lim2];
17     if (t != -1) return t;
18     int up1 = o1 ? a[pos] : 1;
19     int up2 = o2 ? b[pos] : 1;
20     ll res = 0;
21     for (int i = 0; i <= up1; ++i) {
22         for (int j = 0; j <= up2; ++j) {
23             int t1 = i & j;
24             int t2 = i ^ j;
25             if (lim1 && t1 > c[pos]) continue;
26             if (lim2 && t2 < c[pos]) continue;
27             res += dfs(pos - 1, o1 && i == up1, o2 && j == up2, lim1 && t1 == c[pos], lim2 && t2 ==
                c[pos]);
28         }
29     }
30     return t = res;
31 }
32
33 ll solve(ll x, ll y, ll z) {
34     memset(dp, -1ll, sizeof dp);
35     for (int i = 0; i < 33; ++i) a[i] = b[i] = c[i] = 0;
36     cal(a, x);
37     cal(b, y);
38     cal(c, z);
39     return dfs(32, 1, 1, 1, 1);
40 }
41
42 int main(int argc, char *argv[]) {
43     int T;
44     scanf("%d", &T);
45     ll x, y, z;
46     for (int kase = 1; kase <= T; ++kase) {
47         scanf("%lld%lld%lld", &x, &y, &z);
48         ll res = solve(x, y, z);
49         res -= max(0ll, y - z + 1);
50         res -= max(0ll, x - z + 1);
51         printf("%lld\n", x * y - res);
52     }
53     return 0;
54 }
```

5.2 Subsequence

5.2.1 MaxSum

```
1 // 传入序列a和长度n, 返回最大子序列和
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 }
```

5.2.2 LIS

```
1 // 简单写法(下标从0开始,只返回长度)
2 int dp[N];
3 int LIS(int a[], int n)
4 {
5     memset(dp, 0x3f, sizeof(dp));
6     for (int i = 0; i < n; i++) *lower_bound(dp, dp + n, a[i]) = a[i];
7     return lower_bound(dp, dp + n, INF) - dp;
8 }
9
10 // 小常数nlogn求序列用树状数组维护dp即可
11 // dp[i] = max(dp[j]) + 1 (j < i && a[j] < a[i])
```

5.2.3 LongestCommonIncrease

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

5.2.4 LCS

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define M 30005
5 #define SIZE 128
6 #define WORDMAX 3200
7 #define BIT 32
8
9 char s1[M], s2[M];
10 int nword;
11 unsigned int str[SIZE][WORDMAX];
12 unsigned int tmp1[WORDMAX], tmp2[WORDMAX];
13
14 void pre(int len)
15 {
16     int i, j;
17     memset(str, 0, sizeof(str));
```



```
18     for(i = 0; i < len; i ++)  
19         str[s1[i]][i / BIT] |= 1 << (i % BIT);  
20 }  
21  
22 void cal(unsigned int *a, unsigned int *b, char ch)  
23 {  
24     int i, bottom = 1, top;  
25     unsigned int x, y;  
26     for(i = 0; i < nword; i ++)  
27     {  
28         y = a[i];  
29         x = y | str[ch][i];  
30         top = (y >> (BIT - 1)) & 1;  
31         y = (y << 1) | bottom;  
32         if(x < y) top = 1;  
33         b[i] = x & ((x - y) ^ x);  
34         bottom = top;  
35     }  
36 }  
37  
38 int bitcnt(unsigned int *a)  
39 {  
40     int i, j, res = 0, t;  
41     unsigned int b[5] = {0x55555555, 0x33333333, 0x0f0f0f0f, 0x00ff00ff, 0x0000ffff}, x;  
42     for(i = 0; i < nword; i ++)  
43     {  
44         x = a[i];  
45         t = 1;  
46         for(j = 0; j < 5; j ++, t <= 1)  
47             x = (x & b[j]) + ((x >> t) & b[j]);  
48         res += x;  
49     }  
50     return res;  
51 }  
52  
53 void process()  
54 {  
55     int i, j, len1, len2;  
56     unsigned int *a, *b, *t;  
57     len1 = strlen(s1);  
58     len2 = strlen(s2);  
59     nword = (len1 + BIT - 1) / BIT;  
60     pre(len1);  
61     memset(tmp1, 0, sizeof(tmp1));  
62     a = &tmp1[0];  
63     b = &tmp2[0];  
64     for(i = 0; i < len2; i ++)  
65     {  
66         cal(a, b, s2[i]);  
67         t = a; a = b; b = t;  
68     }  
69     printf("%d\n", bitcnt(a));  
70 }  
71  
72 int main()  
73 {  
74     while(scanf("%s%s", s1, s2) != EOF)  
75         process();  
76     return 0;
```

77 }

5.3 Others

问题 设 $f(i) = \min(y[k] - s[i] \times x[k]), k \in [1, i-1]$, 现在要求出所有 $f(i), i \in [1, n]$
考虑两个决策 j 和 k , 如果 j 比 k 优, 则

$$y[j] - s[i] \times x[j] < y[k] - s[i] \times x[k]$$

化简得:

$$\frac{y_j - y_k}{x_j - x_k} < s_i$$

不等式左边是个斜率, 我们把它设为 $\text{slope}(j, k)$

我们可以维护一个单调递增的队列, 为什么呢?

因为如果 $\text{slope}(q[i-1], q[i]) > \text{slope}(q[i], q[i+1])$, 那么当前者成立时, 后者必定成立。即 $q[i]$ 决策优于 $q[i-1]$ 决策时, $q[i+1]$ 必然优于 $q[i]$, 因此 $q[i]$ 就没有存在的必要了。所以我们要维护递增的队列。

那么每次的决策点 i , 都要满足

$$\begin{cases} \text{slope}(q[i-1], q[i]) < s[i] \\ \text{slope}(q[i], q[i+1]) \geq s[i] \end{cases}$$

一般情况去二分这个 i 即可。

如果 $s[i]$ 是单调不降的, 那么对于决策 j 和 $k (j < k)$ 来说, 如果决策 k 优于决策 j , 那么对于 $i \in [k+1, n]$, 都存在决策 k 优于决策 j , 因此决策 j 就可以舍弃了。这样的话我们可以用单调队列进行优化, 可以少个 \log 。

单调队列滑动窗口最大值

```
1 // k为滑动窗口的大小, 数列下标从1开始, d为序列长度+1
2 deque<int> q;
3 for (int i = 0, j = 0; i + k <= d; i++)
4 {
5     while (j < i + k)
6     {
7         while (!q.empty() && a[q.back()] < a[j]) q.pop_back();
8         q.push_back(j++);
9     }
10    while (q.front() < i) q.pop_front();
11    // a[q.front()]为当前滑动窗口的最大值
12 }
```

5.3.1 矩阵快速幂

```
1 struct Matrix {
2     int sz;
3     // int n, m;
4     ll a[maxn][maxn];
5     Matrix(int sz_ = 0):sz(sz_) {
6         memset(a, 0, sizeof a);
7     }
8     void pr() {
9         printf("*\n");
10        for(int i = 0; i < sz; ++i) {
11            for (int j = 0; j < sz; ++j) {
12                printf("%lld ", a[i][j]);
13            }
14            printf("\n");
15        }
16    }
17    void tr() {
```

```
18         for (int i = 0; i < sz; ++i) {
19             for (int j = i + 1; j < sz; ++j) {
20                 swap(a[i][j], a[j][i]);
21             }
22         }
23     }
24 }res, t1;
25
26 void init() {
27     ;
28 }
29
30 Matrix mul(Matrix a, Matrix b)
31 {
32     Matrix res(a.sz);
33     // if (a.m != b.n) return res;
34     for(int i = 0; i < res.sz; i++) // a.n
35         for(int j = 0; j < res.sz; j++) // b.m
36             for(int k = 0; k < res.sz; k++) // a.m, b.n
37                 (res.a[i][j] +=a.a[i][k] * b.a[k][j] % mod) %= mod;
38     return res;
39 }
40
41 Matrix pow(ll n)
42 {
43     init();
44     //for(int i = 0; i < cur; i++) res.a[i][i] = 1;
45     while(n > 0) {
46         if(n & 1) res = mul(res, t1);
47         t1 = mul(t1, t1);
48         n >>= 1;
49     }
50     return res;
51 }
```

6 Others

6.1 mint 类

```
1  const int mod = 998244353;
2
3  struct mint {
4      int n;
5      mint(int n_ = 0) : n(n_) {}
6  };
7
8  mint operator+(mint a, mint b) { return (a.n += b.n) >= mod ? a.n - mod : a.n; }
9  mint operator-(mint a, mint b) { return (a.n -= b.n) < 0 ? a.n + mod : a.n; }
10 mint operator*(mint a, mint b) { return 1LL * a.n * b.n % mod; }
11 mint &operator+=(mint &a, mint b) { return a = a + b; }
12 mint &operator-=(mint &a, mint b) { return a = a - b; }
13 mint &operator*=(mint &a, mint b) { return a = a * b; }
14 ostream &operator<<(ostream &o, mint a) { return o << a.n; }
```

6.2 不重叠区间贪心

```
1  #include <bits/stdc++.h>
2  #define ll long long
3  using namespace std;
4
5  const int maxn = 5e5+5;
6  pair<int, int> a[maxn];
7  int main() {
8      int n;
9      cin >> n;
10     for (int i = 1; i <= n; ++i) {
11         cin >> a[i].second >> a[i].first;
12     }
13     sort(a + 1, a + 1 + n);
14     int res = 1;
15     int tmp = a[1].first;
16     // printf("%d %d\n", a[1].second, a[1].first);
17     for (int i = 2; i <= n; ++i) {
18         if (a[i].second > tmp) {
19             res ++;
20             // printf("%d %d\n", a[i].second, a[i].first);
21             tmp = a[i].first;
22         }
23     }
24     printf("%d\n", res);
25     return 0;
26 }
```

6.3 BigInt 类

```
1  const double PI = acos(-1.0);
2  struct Complex{
3      double x,y;
4      Complex(double _x = 0.0,double _y = 0.0){
5          x = _x;
6          y = _y;
7      }
```

```
8     Complex operator-(const Complex &b)const{
9         return Complex(x - b.x,y - b.y);
10    }
11    Complex operator+(const Complex &b)const{
12        return Complex(x + b.x,y + b.y);
13    }
14    Complex operator*(const Complex &b)const{
15        return Complex(x*b.x - y*b.y,x*b.y + y*b.x);
16    }
17 };
18 void change(Complex y[],int len){
19     int i,j,k;
20     for(int i = 1,j = len/2;i<len-1;i++){
21         if(i < j)    swap(y[i],y[j]);
22         k = len/2;
23         while(j >= k){
24             j = j - k;
25             k = k/2;
26         }
27         if(j < k)    j+=k;
28     }
29 }
30 void fft(Complex y[],int len,int on){
31     change(y,len);
32     for(int h = 2;h <= len;h<=1){
33         Complex wn(cos(on*2*PI/h),sin(on*2*PI/h));
34         for(int j = 0;j < len;j += h){
35             Complex w(1,0);
36             for(int k = j;k < j + h/2;k++){
37                 Complex u = y[k];
38                 Complex t = w*y[k + h/2];
39                 y[k] = u + t;
40                 y[k + h/2] = u - t;
41                 w = w*wn;
42             }
43         }
44     }
45     if(on == -1){
46         for(int i = 0;i < len;i++){
47             y[i].x /= len;
48         }
49     }
50 }
51 class BigInt
52 {
53     #define Value(x, nega) ((nega) ? -(x) : (x))
54     #define At(vec, index) ((index) < vec.size() ? vec[(index)] : 0)
55     static int absComp(const BigInt &lhs, const BigInt &rhs)
56     {
57         if (lhs.size() != rhs.size())
58             return lhs.size() < rhs.size() ? -1 : 1;
59         for (int i = lhs.size() - 1; i >= 0; --i)
60             if (lhs[i] != rhs[i])
61                 return lhs[i] < rhs[i] ? -1 : 1;
62         return 0;
63     }
64     using Long = long long;
65     const static int Exp = 9;
66     const static Long Mod = 1000000000;
```

```
67     mutable std::vector<Long> val;
68     mutable bool nega = false;
69     void trim() const
70     {
71         while (val.size() && val.back() == 0)
72             val.pop_back();
73         if (val.empty())
74             nega = false;
75     }
76     int size() const { return val.size(); }
77     Long &operator[](int index) const { return val[index]; }
78     Long &back() const { return val.back(); }
79     BigInt(int size, bool nega) : val(size), nega(nega) {}
80     BigInt(const std::vector<Long> &val, bool nega) : val(val), nega(nega) {}
81
82 public:
83     friend std::ostream &operator<<(std::ostream &os, const BigInt &n)
84     {
85         if (n.size())
86         {
87             if (n.nega)
88                 putchar('-');
89             for (int i = n.size() - 1; i >= 0; --i)
90             {
91                 if (i == n.size() - 1)
92                     printf("%lld", n[i]);
93                 else
94                     printf("%0*lld", n.Exp, n[i]);
95             }
96         }
97         else
98             putchar('0');
99         return os;
100     }
101     friend BigInt operator+(const BigInt &lhs, const BigInt &rhs)
102     {
103         BigInt ret(lhs);
104         return ret += rhs;
105     }
106     friend BigInt operator-(const BigInt &lhs, const BigInt &rhs)
107     {
108         BigInt ret(lhs);
109         return ret -= rhs;
110     }
111     BigInt(Long x = 0)
112     {
113         if (x < 0)
114             x = -x, nega = true;
115         while (x >= Mod)
116             val.push_back(x % Mod), x /= Mod;
117         if (x)
118             val.push_back(x);
119     }
120     BigInt(const char *s)
121     {
122         int bound = 0, pos;
123         if (s[0] == '-')
124             nega = true, bound = 1;
125         Long cur = 0, pow = 1;
```

```
126     for (pos = strlen(s) - 1; pos >= Exp + bound - 1; pos -= Exp, val.push_back(cur), cur = 0,
127         pow = 1)
128         for (int i = pos; i > pos - Exp; --i)
129             cur += (s[i] - '0') * pow, pow *= 10;
130     for (cur = 0, pow = 1; pos >= bound; --pos)
131         cur += (s[pos] - '0') * pow, pow *= 10;
132     if (cur)
133         val.push_back(cur);
134     }
135     BigInt &operator=(const char *s){
136         BigInt n(s);
137         *this = n;
138         return n;
139     }
140     BigInt &operator=(const Long x){
141         BigInt n(x);
142         *this = n;
143         return n;
144     }
145     friend std::istream &operator>>(std::istream &is, BigInt &n){
146         string s;
147         is >> s;
148         n=(char*)s.data();
149         return is;
150     }
151     BigInt &operator+=(const BigInt &rhs)
152     {
153         const int cap = std::max(size(), rhs.size()) + 1;
154         val.resize(cap);
155         int carry = 0;
156         for (int i = 0; i < cap - 1; ++i)
157         {
158             val[i] = Value(val[i], nega) + Value(At(rhs, i), rhs.nega) + carry, carry = 0;
159             if (val[i] >= Mod)
160                 val[i] -= Mod, carry = 1;
161             else if (val[i] < 0)
162                 val[i] += Mod, carry = -1;
163         }
164         if ((val.back() = carry) == -1) //assert(val.back() == 1 or 0 or -1)
165         {
166             nega = true, val.pop_back();
167             bool tailZero = true;
168             for (int i = 0; i < cap - 1; ++i)
169             {
170                 if (tailZero && val[i])
171                     val[i] = Mod - val[i], tailZero = false;
172                 else
173                     val[i] = Mod - 1 - val[i];
174             }
175         }
176         trim();
177         return *this;
178     }
179     friend BigInt operator-(const BigInt &rhs)
180     {
181         BigInt ret(rhs);
182         ret.nega ^= 1;
183         return ret;
184     }
185 }
```

```
184 BigInt &operator--(const BigInt &rhs)
185 {
186     rhs.nega ^= 1;
187     *this += rhs;
188     rhs.nega ^= 1;
189     return *this;
190 }
191 friend BigInt operator*(const BigInt &lhs, const BigInt &rhs)
192 {
193     int len=1;
194     BigInt ll=lhs,rr=rhs;
195     ll.nega = lhs.nega ^ rhs.nega;
196     while(len<2*lhs.size()||len<2*rhs.size())len<=1;
197     ll.val.resize(len),rr.val.resize(len);
198     Complex x1[len],x2[len];
199     for(int i=0;i<len;i++){
200         Complex nx(ll[i],0.0),ny(rr[i],0.0);
201         x1[i]=nx;
202         x2[i]=ny;
203     }
204     fft(x1,len,1);
205     fft(x2,len,1);
206     for(int i = 0 ; i < len; i++)
207         x1[i] = x1[i] * x2[i];
208     fft( x1 , len , -1 );
209     for(int i = 0 ; i < len; i++)
210         ll[i] = int( x1[i].x + 0.5 );
211     for(int i = 0 ; i < len; i++){
212         ll[i+1]+=ll[i]/Mod;
213         ll[i]%=Mod;
214     }
215     ll.trim();
216     return ll;
217 }
218 friend BigInt operator*(const BigInt &lhs, const Long &x){
219     BigInt ret=lhs;
220     bool negat = ( x < 0 );
221     Long xx = (negat) ? -x : x;
222     ret.nega ^= negat;
223     ret.val.push_back(0);
224     ret.val.push_back(0);
225     for(int i = 0; i < ret.size(); i++)
226         ret[i]*=xx;
227     for(int i = 0; i < ret.size(); i++){
228         ret[i+1]+=ret[i]/Mod;
229         ret[i] %= Mod;
230     }
231     ret.trim();
232     return ret;
233 }
234 BigInt &operator*=(const BigInt &rhs) { return *this = *this * rhs; }
235 BigInt &operator*=(const Long &x) { return *this = *this * x; }
236 friend BigInt operator/(const BigInt &lhs, const BigInt &rhs)
237 {
238     static std::vector<BigInt> powTwo{BigInt(1)};
239     static std::vector<BigInt> estimate;
240     estimate.clear();
241     if (absComp(lhs, rhs) < 0)
242         return BigInt();
```



```
243     BigInt cur = rhs;
244     int cmp;
245     while ((cmp = absComp(cur, lhs)) <= 0)
246     {
247         estimate.push_back(cur), cur += cur;
248         if (estimate.size() >= powTwo.size())
249             powTwo.push_back(powTwo.back() + powTwo.back());
250     }
251     if (cmp == 0)
252         return BigInt(powTwo.back().val, lhs.nega ^ rhs.nega);
253     BigInt ret = powTwo[estimate.size() - 1];
254     cur = estimate[estimate.size() - 1];
255     for (int i = estimate.size() - 1; i >= 0 && cmp != 0; --i)
256         if ((cmp = absComp(cur + estimate[i], lhs)) <= 0)
257             cur += estimate[i], ret += powTwo[i];
258     ret.nega = lhs.nega ^ rhs.nega;
259     return ret;
260 }
261 friend BigInt operator/(const BigInt &num, const Long &x){
262     bool negat = ( x < 0 );
263     Long xx = (negat) ? -x : x;
264     BigInt ret;
265     Long k = 0;
266     ret.val.resize( num.size() );
267     ret.nega = (num.nega ^ negat);
268     for(int i = num.size() - 1 ; i >= 0; i--){
269         ret[i] = ( k * Mod + num[i]) / xx;
270         k = ( k * Mod + num[i]) % xx;
271     }
272     ret.trim();
273     return ret;
274 }
275 bool operator==(const BigInt &rhs) const
276 {
277     return nega == rhs.nega && val == rhs.val;
278 }
279 bool operator!=(const BigInt &rhs) const { return nega != rhs.nega || val != rhs.val; }
280 bool operator>=(const BigInt &rhs) const { return !(*this < rhs); }
281 bool operator>(const BigInt &rhs) const { return !(*this <= rhs); }
282 bool operator<=(const BigInt &rhs) const
283 {
284     if (nega && !rhs.nega)
285         return true;
286     if (!nega && rhs.nega)
287         return false;
288     int cmp = absComp(*this, rhs);
289     return nega ? cmp >= 0 : cmp <= 0;
290 }
291 bool operator<(const BigInt &rhs) const
292 {
293     if (nega && !rhs.nega)
294         return true;
295     if (!nega && rhs.nega)
296         return false;
297     return (absComp(*this, rhs) < 0) ^ nega;
298 }
299 void swap(const BigInt &rhs) const
300 {
301     std::swap(val, rhs.val);
```

```
302         std::swap(nega, rhs.nega);
303     }
304 };
305 BigInt ba,bb;
306 int main(){
307     cin>>ba>>bb;
308     cout << ba + bb << '\n';//和
309     cout << ba - bb << '\n';//差
310     cout << ba * bb << '\n';//积
311     BigInt d;
312     cout << (d = ba / bb) << '\n';//商
313     cout << ba - d * bb << '\n';//余
314     return 0;
315 }
```

6.4 date

```
1  string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};
2  // converts Gregorian date to integer (Julian day number)
3  int DateToInt (int m, int d, int y){
4      return
5          1461 * (y + 4800 + (m - 14) / 12) / 4 +
6          367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
7          3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
8          d - 32075;
9  }
10
11 // converts integer (Julian day number) to Gregorian date: month/day/year
12 void IntToDate (int jd, int &m, int &d, int &y){
13     int x, n, i, j;
14     x = jd + 68569;
15     n = 4 * x / 146097;
16     x -= (146097 * n + 3) / 4;
17     i = (4000 * (x + 1)) / 1461001;
18     x -= 1461 * i / 4 - 31;
19     j = 80 * x / 2447;
20     d = x - 2447 * j / 80;
21     x = j / 11;
22     m = j + 2 - 12 * x;
23     y = 100 * (n - 49) + i + x;
24 }
25 // converts integer (Julian day number) to day of week
26 string IntToDay (int jd){
27     return dayOfWeek[jd % 7];
28 }
```

6.5 Frac 类

```
1 struct Frac {
2     ll a, b;
3     void getJian() {
4         ll gcd = abs(__gcd(a, b));
5         a /= gcd;
6         b /= gcd;
7         if (b < 0) {
8             a = -a;
9             b = -b;
10        }
```

```
10     }
11 }
12 Frac(1l a_ = 1, 1l b_ = 1) {
13     a = a_;
14     b = b_;
15     getJian();
16 }
17 Frac add(const Frac& oth) {
18     1l bt = b * oth.b;
19     1l at = a * oth.b + oth.a * b;
20     return Frac(at, bt);
21 }
22 Frac multi(const Frac& oth) {
23     a *= oth.a;
24     b *= oth.b;
25     getJian();
26     return *this;
27 }
28 bool operator < (const Frac& oth) const {
29     return a * oth.b < b * oth.a;
30 }
31 bool operator == (const Frac& oth) const {
32     return a * oth.b == b * oth.a;
33 }
34 bool operator <= (const Frac& oth) const {
35     return a * oth.b <= b * oth.a;
36 }
37 };
```

6.6 模拟退火 (最小圆覆盖)

```
1  const int maxn = 1e5 + 10;
2  const double eps = 1e-8;
3  const double delta = 0.98;
4  const double inf = 1e18;
5
6  struct Point { double x, y; } p[maxn];
7
8  double dis(Point A, Point B) { return sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y));
9  };
10
11 double Simulate_Annea(int n)
12 {
13     Point S;
14     S.x = S.y = 0;
15     double t = 1000;
16     double res = inf;
17     while(t > eps)
18     {
19         int k = 0;
20         for(int i = 0; i < n; i++) if(dis(S, p[i]) > dis(S, p[k])) k = i;
21         double d = dis(S, p[k]);
22         res = min(res, d);
23         S.x += (p[k].x - S.x) / d * t;
24         S.y += (p[k].y - S.y) / d * t;
25         t *= delta;
26     }
27     return res;
28 }
```

```
27 }
28
29 int main()
30 {
31     int n;
32     scanf("%d", &n);
33     for(int i = 0; i < n; i++) scanf("%lf%lf", &p[i].x, &p[i].y);
34     printf("%.3f\n", Simulate_Annea(n));
35     return 0;
36 }
```

6.7 string 类

```
1  const int maxn = 1005;
2  struct String{
3      int nex[maxn];
4      char x[maxn];
5      int len;
6      int getLength() {
7          return len;
8      }
9      void getNext() {
10         int n = len, i = 0, j = -1;
11         nex[0] = -1;
12         while (i < n) {
13             if (j == -1 || x[i] == x[j]) nex[++i] = ++j;
14             else j = -1;
15         }
16     }
17     void input() {
18         scanf("%s", x);
19         len = strlen(x);
20     }
21     void inputAndCal() {
22         scanf("%s", x);
23         len = strlen(x);
24         getNext();
25     }
26     void show() {
27         printf("%s\n", x);
28     }
29     bool operator < (const String&oth) const {
30         return strcmp(x, oth.x) < 0;
31     }
32     char operator [] (const int a) const {
33         return x[a];
34     }
35     bool substring(String b) {//b is the substring of a
36         int m = len, n = b.getLength();
37         int i = 0, j = 0;
38         while (i < m && j < n) {
39             if (j == -1 || x[i] == b[j]) ++i, ++j;
40             else j = b.nex[j];
41             if (j == n) return true;
42         }
43         return false;
44     }
45 };
```

6.8 前缀异或和

```

1 ll xor_sum(ll n) {
2     ll t=n&3;
3     if (t&1) return t/2ull^1;
4     return t/2ull^n;
5 }

```

6.9 约瑟夫环第 k 个

```

1 ll kth(ll n, ll m, ll k) { // n个人, m间隔, 第k个出列的人
2     if (m == 1) return k;
3     ll res = (m - 1) % (n - k + 1);
4     for (ll i = n - k + 2, stp = 0; i <= n; i += stp, res += stp * m) {
5         if (res + m >= i) {
6             res = (res + m) % i;
7             i++;
8             stp = 0;
9         } else {
10            stp = (i - res - 2) / (m - 1);
11            if (i + stp > n) {
12                res += (n - (i - 1)) * m;
13                break;
14            }
15        }
16    }
17    return res + 1;
18 }
19
20 ll dieInXturn(int n, int k, int x) { // n个人, m间隔, 第k个人出列时间
21     ll tmp = 0;
22     while (n) {
23         x = (x + n) % n;
24         if (k > n) x += (k - x - 1 + n - 1) / n * n;
25         if ((x + 1) % k == 0) {
26             tmp += (x + 1) / k;
27             break;
28         } else {
29             if (k > n) {
30                 tmp += x / k;
31                 ll ttmp = x;
32                 x = x - (x / n + 1) * (x / k) + (x + n) / n * n - k;
33                 n -= ttmp / k;
34             } else {
35                 tmp += n / k;
36                 x = x - x / k;
37                 x += n - n / k * k;
38                 n -= n / k;
39             }
40         }
41     }
42 }
43 return tmp;
44 }

```

6.10 二分

```
1 // a为二分数组, x为需要查找的数, 返回最左端和最右端
2 pair<int, int> F(vector<int> a, int x) {
3     int l = 0, r = a.size() - 1;
4     int lres = -1;
5     while (l <= r) {
6         int mid = l + r >> 1;
7         int tt = a[mid];
8         if (tt >= x) {
9             r = mid - 1;
10        } else if (tt < x) {
11            l = mid + 1;
12        }
13    }
14    if (l >= a.size() || a[l] != x) return make_pair(-1, -1);
15    lres = l;
16    l = 0, r = a.size() - 1;
17    while (l <= r) {
18        int mid = l + r >> 1;
19        int tt = a[mid];
20        if (tt > x) {
21            r = mid - 1;
22        } else if (tt <= x) {
23            l = mid + 1;
24        }
25    }
26    return make_pair(lres, r);
27 }
```

6.11 猛男 IO 挂

```
1 const int LEN = 100000;
2 struct fastio {
3     int it, len;
4     char s[LEN + 5];
5     fastio() {
6         it = len = 0;
7     }
8     char get() {
9         if (it < len) return s[it++];
10        it = 0, len = fread(s, 1, LEN, stdin);
11        return len ? s[it++] : EOF;
12    }
13    bool notend() {
14        char c;
15        for (c = get(); c == ' ' || c == '\n'; c = get());
16        if (it) it--;
17        return c != EOF;
18    }
19    void put(char c) {
20        if (it == LEN) fwrite(s, 1, LEN, stdout), it = 0;
21        s[it++] = c;
22    }
23    void flush() {
24        fwrite(s, 1, it, stdout);
25    }
26 } buff, bufo;
27 inline int getint() {
28     char c;
```

```

29     int res = 0, sig = 1;
30     for (c = buff.get(); c < '0' || c > '9'; c = buff.get()) if (c == '-') sig = -1;
31     for (; c >= '0' && c <= '9'; c = buff.get()) res = res * 10 + (c - '0');
32     return sig * res;
33 }
34 inline ll getll() {
35     char c;
36     ll res = 0, sig = 1;
37     for (c = buff.get(); c < '0' || c > '9'; c = buff.get()) if (c == '-') sig = -1;
38     for (; c >= '0' && c <= '9'; c = buff.get()) res = res * 10 + (c - '0');
39     return sig * res;
40 }
41 inline void putint(int x, char suf) {
42     if (!x) bufo.put('0');
43     else {
44         if (x < 0) bufo.put('-'), x = -x;
45         int k = 0;
46         char s[15];
47         while (x) {
48             s[++k] = x % 10 + '0';
49             x /= 10;
50         }
51         for (; k; k--) bufo.put(s[k]);
52     }
53     bufo.put(suf);
54 }
55 inline void putll(ll x, char suf) {
56     if (!x) bufo.put('0');
57     else {
58         if (x < 0) bufo.put('-'), x = -x;
59         int k = 0;
60         char s[25];
61         while (x) {
62             s[++k] = x % 10 + '0';
63             x /= 10;
64         }
65         for (; k; k--) bufo.put(s[k]);
66     }
67     bufo.put(suf);
68 }
69 inline char get_char() {
70     char c;
71     for (c = buff.get(); c == ' ' || c == '\n'; c = buff.get());
72     return c;
73 }

```

6.12 贪心结论

```

1 // n个区间，挪到使得某个点被所有区间覆盖需要的最少步数时，选择的点是所有区间端点的中位数 (mid-mid+1答案
  都是一样的)
2
3
4 // 不重叠区间贪心
5 pair<int, int> a[maxn];
6 int main() {
7     int n;
8     cin >> n;
9     for (int i = 1; i <= n; ++i) {

```

```
10     cin >> a[i].second >> a[i].first;
11 }
12 sort(a + 1, a + 1 + n);
13 int res = 1;
14 int tmp = a[1].first;
15 // printf("%d %d\n", a[1].second, a[1].first);
16 for (int i = 2; i <= n; ++i) {
17     if (a[i].second > tmp) {
18         res ++;
19         // printf("%d %d\n", a[i].second, a[i].first);
20         tmp = a[i].first;
21     }
22 }
23 printf("%d\n", res);
24 return 0;
25 }
```

6.13 builtin

```
1 __builtin_popcount(unsigned int n) // 1的个数
2 __builtin_parity(unsigned int n) // 奇数个1返回1, 偶数个返回0
3 __builtin_ctz(unsigned int n) // 判断n的二进制末尾后面0的个数
4 __builtin_clz(unsigned int n) // 返回前导0的个数
```

6.14 n 以内 k 因子的个数

```
1 // 返回1~n中k因子的个数
2 ll dig(ll n, ll k) {
3     ll res = 0;
4     while (n > 0) {
5         res += n / k;
6         n /= k;
7     }
8     return res;
9 }
```