

# Algorithm Library

CRatiQ

South China Normal University

February 4, 2025

# Contents

<b>常用文件</b>	<b>3</b>
DEBUG 头	3
__int128 输出流	3
常用数学函数	3
纳秒级随机种子	4
Linux 对拍	4
<b>数学</b>	<b>4</b>
欧拉筛	4
取模类 (MInt)	5
组合数	7
多项式	8
原根表	11
线性基	12
min-plus 卷积	13
模意义分数还原	13
Exgcd	14
一次函数下取整区间和	14
二元一次不定方程	15
行列式求值	15
高斯消元法	16
枚举二进制下有 $k$ 个 1 的数	16
<b>数据结构</b>	<b>17</b>
并查集 (启发式合并 + 带撤销)	17
状压 RMQ	17
ST 表	18
树状数组	19
线段树	19
可并堆 (pb_ds)	22
成员函数	22
示例	22
平衡树 (pb_ds)	22
成员函数	22
示例	23
<b>字符串</b>	<b>23</b>
字符串哈希 (随机模数)	23
允许 $k$ 次失配的字符串匹配	23
最长公共子串	24
Code	24
KMP	24
字符串周期	24
统计前缀出现次数	24
求满足一些要求的 Border	24
Code	24
Z 函数	25
AC 自动机	25
后缀数组	26
(广义) 后缀自动机	27
不同子串个数	27
字典序第 $k$ 大子串	27
最小循环移位	27
出现次数	27
首次出现位置	27
所有出现位置	27

最短未出现字符串 . . . . .	27
最长公共子串 . . . . .	27
Code . . . . .	27
Manacher . . . . .	28
回文自动机 . . . . .	29
本质不同回文子串数 . . . . .	29
回文子串出现次数 . . . . .	29
Code . . . . .	29
含通配符字符串匹配 . . . . .	30
<b>图论</b>	<b>31</b>
拓扑排序 . . . . .	31
树的直径 . . . . .	31
树的重心 . . . . .	31
Dijkstra . . . . .	32
SPFA . . . . .	32
Johnson . . . . .	33
强连通分量 . . . . .	33
边双连通分量 . . . . .	35
轻重链剖分 . . . . .	36
欧拉路径 . . . . .	37
2-SAT . . . . .	38
最大流 . . . . .	39
最小费用最大流 . . . . .	40
二分图最大权匹配 (KM) . . . . .	42
三元环计数 . . . . .	44
树哈希 . . . . .	45
矩阵树定理 . . . . .	45
<b>计算几何</b>	<b>46</b>
EPS . . . . .	46
Point . . . . .	46
Line . . . . .	47
距离 . . . . .	47
点绕中心旋转 . . . . .	47
关于线的对称点 . . . . .	47
位置关系判断 . . . . .	48
线段交点 . . . . .	48
过定点做圆的切线 . . . . .	49
两圆交点 . . . . .	49
多边形面积 . . . . .	49
自适应辛普森法 . . . . .	49
静态凸包 . . . . .	50
旋转卡壳求直径 . . . . .	50
半平面交 . . . . .	50

## 常用文件

### DEBUG 头

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  using i64=long long;
4  using i128=__int128;
5
6  namespace DBG
7  {
8      template <class T>
9      void _dbg(const char *f,T t) { cerr<<f<<'\n'; }
10
11     template <class A,class... B>
12     void _dbg(const char *f,A a,B... b)
13     {
14         while (*f!=',') cerr<<*f++;
15         cerr<<'\n';
16         _dbg(f+1,b...);
17     }
18
19     template <class T>
20     ostream& operator << (ostream& os,const vector<T> &v)
21     {
22         os<<"[ ";
23         for (const auto &x:v) os<<x<<" ";
24         os<<"]";
25         return os;
26     }
27
28     #define dbg(...) _dbg(#__VA_ARGS__, __VA_ARGS__)
29 }
30
31 using namespace DBG;
```

### \_\_int128 输出流

```
1  ostream &operator << (ostream &os,i128 n)
2  {
3      string s;
4      bool neg=n<0;
5      if (neg) n=-n;
6      while (n)
7      {
8          s+='0'+n%10;
9          n/=10;
10     }
11     if (neg) s+='-';
12     reverse(s.begin(),s.end());
13     if (s.empty()) s+='0';
14     return os<<s;
15 }
```

### 常用数学函数

```
1  i64 ceilDiv(i64 n,i64 m)
2  {
3      if (n>=0) return (n+m-1)/m;
4      else return n/m;
5  }
6
7  i64 floorDiv(i64 n,i64 m)
8  {
9      if (n>=0) return n/m;
10     else return (n-m+1)/m;
11 }
12
13 i128 gcd(i128 a,i128 b)
14 {
```

```

15     return b?gcd(b,a%b):a;
16 }

```

## 纳秒级随机种子

```

1 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

```

## Linux 对拍

记得先 `chmod 777 check.sh`.

```

1 for ((i=0;i<100;i++))
2 do
3     ./A_Generator > A.in
4     ./A < A.in > A.out
5     ./A_Good < A.in > A.ans
6
7     if diff A.out A.ans;
8     then
9         echo "AC"
10    else
11        echo "WA"
12        exit 1
13    fi
14 done

```

## 数学

### 欧拉筛

时间复杂度为  $\mathcal{O}(n)$ 。

$\phi$  为欧拉函数  $\varphi(n)$ ,  $\mu$  为莫比乌斯函数  $\mu(n)$ ,  $d$  为约数个数  $\sigma_0(n)$ ,  $f$  为约数和  $\sigma_1(n)$ 。

假如一个积性函数  $f$  满足: 对于任意质数  $p$  和正整数  $k$ , 可以在  $\mathcal{O}(1)$  时间内计算  $f(p^k)$ , 那么可以在  $\mathcal{O}(n)$  时间内筛出  $f(1), f(2), \dots, f(n)$  的值。

设合数  $n$  的质因子分解是  $\prod_{i=1}^k p_i^{\alpha_i}$ , 其中  $p_1 < p_2 < \dots < p_k$  为质数, 我们在线性筛中记录  $g_n = p_1^{\alpha_1}$ , 假如  $n$  被  $x \cdot p$  筛掉 ( $p$  是质数), 那么  $g$  满足如下递推式:

$$g_n = \begin{cases} g_x \cdot p & x \bmod p = 0 \\ p & \text{otherwise} \end{cases}$$

假如  $n = g_n$ , 说明  $n$  就是某个质数的次幂, 可以  $\mathcal{O}(1)$  计算  $f(n)$ ; 否则,  $f(n) = f(\frac{n}{g_n}) \cdot f(g_n)$ 。

```

1 vector<int> minp,primes;
2 // vector<int> phi;
3 // vector<int> mu;
4 // vector<int> d,num;
5 // vector<int> f,g;
6
7 void sieve(int n)
8 {
9     minp.assign(n+1,0);
10    primes.clear();
11    // phi.assign(n+1,0);
12    // mu.assign(n+1,0);
13    // d.assign(n+1,0);
14    // num.assign(n+1,0);
15    // f.assign(n+1,0);
16    // g.assign(n+1,0);
17    // phi[1]=1;
18    // mu[1]=1;
19    // d[1]=1;

```

```

20 // f[1]=g[1]=1;
21 for (int i=2;i<=n;i++)
22 {
23     if (!minp[i])
24     {
25         minp[i]=i;
26         primes.push_back(i);
27         // phi[i]=i-1;
28         // mu[i]=-1;
29         // d[i]=2;
30         // num[i]=1;
31         // f[i]=g[i]=i+1;
32     }
33     for (auto p:primes)
34     {
35         if (i*p>n) break;
36         minp[i*p]=p;
37         if (p==minp[i])
38         {
39             // phi[i*p]=phi[i]*p;
40             // mu[i*p]=0;
41             // num[i*p]=num[i]+1;
42             // d[i*p]=d[i]/num[i*p]*(num[i*p]+1);
43             // g[i*p]=g[i]*p+1;
44             // f[i*p]=f[i]/g[i]*g[i*p];
45             break;
46         }
47         // phi[i*p]=phi[i]*phi[p];
48         // mu[i*p]=-mu[i];
49         // num[i*p]=1;
50         // d[i*p]=d[i]<<1;
51         // f[i*p]=f[i]*f[p];
52         // g[i*p]=p+1;
53     }
54 }
55 }

```

## 取模类 (MInt)

对 MInt<0> 修改 Mod 可以起到动态模数的效果，但常数较大。

```

1  template <class T>
2  constexpr T power(T a,i64 b)
3  {
4      T res=1;
5      for (;b>=1,a*=a)
6          if (b&1) res*=a;
7      return res;
8  }
9
10 template <int P>
11 struct MInt
12 {
13     int x;
14     constexpr MInt():x{} {}
15     constexpr MInt(i64 x):x{norm(x%getMod())} {}
16
17     static int Mod;
18     constexpr static int getMod()
19     {
20         if (P>0) return P;
21         else return Mod;
22     }
23
24     constexpr static void setMod(int Mod_) { Mod=Mod_; }
25
26     constexpr int norm(int x) const
27     {
28         if (x<0) x+=getMod();
29         if (x>=getMod()) x-=getMod();
30         return x;

```

```

31     }
32
33     constexpr int val() const { return x; }
34
35     explicit constexpr operator int () const { return x; }
36
37     constexpr MInt operator - () const
38     {
39         MInt res;
40         res.x=norm(getMod()-x);
41         return res;
42     }
43
44     constexpr MInt inv() const
45     {
46         assert(x!=0);
47         return power(*this,getMod()-2);
48     }
49
50     constexpr MInt &operator *= (MInt rhs) &
51     {
52         x=1ll*x*rhs.x%getMod();
53         return *this;
54     }
55
56     constexpr MInt &operator += (MInt rhs) &
57     {
58         x=norm(x+rhs.x);
59         return *this;
60     }
61
62     constexpr MInt &operator -= (MInt rhs) &
63     {
64         x=norm(x-rhs.x);
65         return *this;
66     }
67
68     constexpr MInt &operator /= (MInt rhs) &
69     {
70         return *this*=rhs.inv();
71     }
72
73     friend constexpr MInt operator * (MInt lhs,MInt rhs)
74     {
75         MInt res=lhs;
76         res*=rhs;
77         return res;
78     }
79
80     friend constexpr MInt operator + (MInt lhs,MInt rhs)
81     {
82         MInt res=lhs;
83         res+=rhs;
84         return res;
85     }
86
87     friend constexpr MInt operator - (MInt lhs,MInt rhs)
88     {
89         MInt res=lhs;
90         res-=rhs;
91         return res;
92     }
93
94     friend constexpr MInt operator / (MInt lhs,MInt rhs)
95     {
96         MInt res=lhs;
97         res/=rhs;
98         return res;
99     }
100
101     friend constexpr istream &operator >> (istream &is,MInt &a)

```

```

102     {
103         i64 v;
104         is>>v;
105         a=MInt(v);
106         return is;
107     }
108
109     friend constexpr ostream &operator << (ostream &os,const MInt &a) { return os<<a.val(); }
110
111     friend constexpr bool operator == (MInt lhs,MInt rhs) { return lhs.val()==rhs.val(); }
112
113     friend constexpr bool operator != (MInt lhs,MInt rhs) { return lhs.val() != rhs.val(); }
114 };
115
116 template<>
117 int MInt<0>::Mod=1;
118
119 template<int V,int P>
120 constexpr MInt<P> CInv=MInt<P>(V).inv();

```

## 组合数

```

1 struct Comb
2 {
3     int n;
4     vector<Z> _fac,_inv,_finv;
5
6     Comb():n{0},_fac{1},_inv{0},_finv{1}{}
7     Comb(int n):Comb() { init(n); }
8
9     void init(int m)
10    {
11        m=min(m,Z::getMod()-1);
12        if (m<=n) return;
13        _fac.resize(m+1);
14        _inv.resize(m+1);
15        _finv.resize(m+1);
16
17        for (int i=n+1;i<=m;i++)
18            _fac[i]=_fac[i-1]*i;
19        _finv[m]=_fac[m].inv();
20        for (int i=m;i>n;i--)
21        {
22            _finv[i-1]=_finv[i]*i;
23            _inv[i]=_finv[i]*_fac[i-1];
24        }
25        n=m;
26    }
27
28    Z fac(int m)
29    {
30        if (m>n) init(m<<1);
31        return _fac[m];
32    }
33
34    Z finv(int m)
35    {
36        if (m>n) init(m<<1);
37        return _finv[m];
38    }
39
40    Z inv(int m)
41    {
42        if (m>n) init(m<<1);
43        return _inv[m];
44    }
45
46    Z binom(int n,int m)
47    {
48        if (n<m||m<0) return 0;
49        return fac(n)*finv(m)*finv(n-m);

```



```

50     }
51 } comb;

```

## 多项式

```

1  vector<int> rev;
2  vector<Z> roots{0,1};
3
4  void dft(vector<Z> &a)
5  {
6      int n=a.size();
7      if (int(rev.size())!=n)
8      {
9          int k=__builtin_ctz(n)-1;
10         rev.resize(n);
11         for (int i=0;i<n;i++)
12             rev[i]=rev[i>>1]>>1|(i&1)<<k;
13     }
14     for (int i=0;i<n;i++)
15         if (rev[i]<i)
16             swap(a[i],a[rev[i]]);
17     if (int(roots.size())<n)
18     {
19         int k=__builtin_ctz(roots.size());
20         roots.resize(n);
21         while ((1<<k)<n)
22         {
23             Z e=power(Z(3),(P-1)>>(k+1));
24             for (int i=1<<(k-1);i<(1<<k);i++)
25             {
26                 roots[i<<1]=roots[i];
27                 roots[i<<1|1]=roots[i]*e;
28             }
29             k++;
30         }
31     }
32     for (int k=1;k<n;k<<=1)
33         for (int i=0;i<n;i+=k*2)
34             for (int j=0;j<k;j++)
35             {
36                 Z u=a[i+j],v=a[i+j+k]*roots[j+k];
37                 a[i+j]=u+v;
38                 a[i+j+k]=u-v;
39             }
40 }
41
42 void idft(vector<Z> &a)
43 {
44     int n=a.size();
45     reverse(a.begin()+1,a.end());
46     dft(a);
47     Z inv=(1-P)/n;
48     for (int i=0;i<n;i++) a[i]*=inv;
49 }
50
51 struct Poly
52 {
53     vector<Z> a;
54
55     Poly(){}
56     explicit Poly(int size,function<Z(int)>f=[](int) { return 0; }):a(size)
57     { for (int i=0;i<size;i++) a[i]=f(i); }
58     Poly(const vector<Z> &a):a(a){}
59     Poly(const initializer_list<Z> &a):a(a){}
60
61     int size() const { return a.size(); }
62
63     void resize(int n) { a.resize(n); }
64
65     Z operator [] (int idx) const
66     {

```

```

67         if (idx<size()) return a[idx];
68         else return 0;
69     }
70
71     Z &operator [] (int idx) { return a[idx]; }
72
73     Poly mulxk(int k) const
74     {
75         auto b=a;
76         b.insert(b.begin(),k,0);
77         return Poly(b);
78     }
79
80     Poly modxk(int k) const
81     {
82         k=min(k,size());
83         return Poly(vector<Z>(a.begin(),a.begin()+k));
84     }
85
86     Poly divxk(int k) const
87     {
88         if (size()<=k) return Poly();
89         return Poly(vector<Z>(a.begin()+k,a.end()));
90     }
91
92     friend Poly operator + (const Poly &a,const Poly &b)
93     {
94         vector<Z> res(max(a.size(),b.size()));
95         for (int i=0;i<int(res.size());i++)
96             res[i]=a[i]+b[i];
97         return Poly(res);
98     }
99
100    friend Poly operator - (const Poly &a,const Poly &b)
101    {
102        vector<Z> res(max(a.size(),b.size()));
103        for (int i=0;i<int(res.size());i++)
104            res[i]=a[i]-b[i];
105        return Poly(res);
106    }
107
108    friend Poly operator - (const Poly &a)
109    {
110        vector<Z> res(a.size());
111        for (int i=0;i<int(res.size());i++)
112            res[i]=-a[i];
113        return Poly(res);
114    }
115
116    friend Poly operator * (Poly a,Poly b)
117    {
118        if (!a.size()||!b.size()) return Poly();
119        if (a.size()<b.size()) swap(a,b);
120        if (b.size()<128)
121        {
122            Poly c(a.size()+b.size()-1);
123            for (int i=0;i<a.size();i++)
124                for (int j=0;j<b.size();j++)
125                    c[i+j]+=a[i]*b[j];
126            return c;
127        }
128        int sz=1,tot=a.size()+b.size()-1;
129        while (sz<tot) sz<<=1;
130        a.a.resize(sz);
131        b.a.resize(sz);
132        dft(a.a);
133        dft(b.a);
134        for (int i=0;i<sz;i++)
135            a.a[i]=a[i]*b[i];
136        idft(a.a);
137        a.resize(tot);

```

```

138     return a;
139 }
140
141 friend Poly operator * (Z a, Poly b)
142 {
143     for (int i=0; i<int(b.size()); i++) b[i]*=a;
144     return b;
145 }
146
147 friend Poly operator * (Poly a, Z b)
148 {
149     for (int i=0; i<int(a.size()); i++) a[i]*=b;
150     return a;
151 }
152
153 Poly &operator += (Poly b) { return (*this)=(*this)+b; }
154 Poly &operator -= (Poly b) { return (*this)=(*this)-b; }
155 Poly &operator *= (Poly b) { return (*this)=(*this)*b; }
156 Poly &operator *= (Z b) { return (*this)=(*this)*b; }
157
158 Poly deriv() const
159 {
160     if (a.empty()) return Poly();
161     vector<Z> res(size()-1);
162     for (int i=0; i<size()-1; i++)
163         res[i]=(i+1)*a[i+1];
164     return Poly(res);
165 }
166
167 Poly integr() const
168 {
169     vector<Z> res(size()+1);
170     for (int i=0; i<size(); i++)
171         res[i+1]=a[i]/(i+1);
172     return Poly(res);
173 }
174
175 Poly inv(int m) const
176 {
177     Poly x{a[0].inv()};
178     int k=1;
179     while (k<m)
180     {
181         k<=1;
182         x=(x*(Poly{2}-modxk(k)*x)).modxk(k);
183     }
184     return x.modxk(m);
185 }
186
187 Poly ln(int m) const { return (deriv()*inv(m)).integr().modxk(m); }
188
189 Poly exp(int m) const
190 {
191     Poly x{1};
192     int k=1;
193     while (k<m)
194     {
195         k<=1;
196         x=(x*(Poly{1}-x.ln(k)+modxk(k))).modxk(k);
197     }
198     return x.modxk(m);
199 }
200
201 Poly pow(int k, int m) const
202 {
203     int i=0;
204     while (i<size()&& a[i].val()==0) i++;
205     if (i==size() || 1ll*i*k>=m) return Poly(vector<Z>(m));
206     Z v=a[i];
207     auto f=divxk(i)*v.inv();
208     return (f.ln(m-i*k)*k).exp(m-i*k).mulxk(i*k)*power(v, k);

```

```

209     }
210
211     Poly sqrt(int m) const
212     {
213         Poly x{1};
214         int k=1;
215         while (k<m)
216         {
217             k<<=1;
218             x=(x+(modxk(k)*x.inv(k)).modxk(k))*((P+1)/2);
219         }
220         return x.modxk(m);
221     }
222     Poly mult(Poly b) const
223     {
224         if (b.size()==0) return Poly();
225         int n=b.size();
226         reverse(b.a.begin(),b.a.end());
227         return ((*this)*b).divxk(n-1);
228     }
229
230     vector<Z> eval(vector<Z> x) const
231     {
232         if (size()==0) return vector<Z>(x.size(),0);
233         const int n=max(int(x.size()),size());
234         vector<Poly> q(n<<2);
235         vector<Z> ans(x.size());
236         x.resize(n);
237         function<void(int,int,int)> build=[&](int p,int l,int r)
238         {
239             if (r-l==1) q[p]=Poly{1,-x[l]};
240             else
241             {
242                 int m=(l+r)>>1;
243                 build(p<<1,l,m);
244                 build(p<<1|1,m,r);
245                 q[p]=q[p<<1]*q[p<<1|1];
246             }
247         };
248         function<void(int,int,int,const Poly&)> work=[&](int p,int l,int r,const Poly &num)
249         {
250             if (r-l==1)
251             {
252                 if (l<int(ans.size())) ans[l]=num[0];
253             }
254             else
255             {
256                 int m=(l+r)>>1;
257                 work(p<<1,l,m,num.mult(q[p<<1|1]).modxk(m-l));
258                 work(p<<1|1,m,r,num.mult(q[p<<1]).modxk(r-m));
259             }
260         };
261         build(1,0,n);
262         work(1,0,n,mult(q[1].inv(n)));
263         return ans;
264     }
265 };

```

## 原根表

	prime	r	k	g
1	3	1	1	2
2	5	1	2	2
3	17	1	4	3
4	97	3	5	5
5	193	3	6	5
6	257	1	8	3
7	7681	15	9	17
8	12289	3	12	11
9	40961	5	13	3
10	65537	1	16	3

```

12 786433          3  18 10
13 5767169        11 19 3
14 7340033        7  20 3
15 23068673       11 21 3
16 104857601      25 22 3
17 167772161      5  25 3
18 469762049      7  26 3
19 1004535809     479 21 3
20 2013265921     15 27 31
21 2281701377     17 27 3
22 3221225473     3  30 5
23 75161927681    35 31 3
24 77309411329    9  33 7
25 206158430209   3  36 22
26 2061584302081  15 37 7
27 2748779069441  5  39 3
28 6597069766657  3  41 5
29 39582418599937 9  42 5
30 79164837199873 9  43 5
31 263882790666241 15 44 7
32 1231453023109121 35 45 3
33 1337006139375617 19 46 3
34 3799912185593857 27 47 5
35 4222124650659841 15 48 19
36 7881299347898369 7  50 6
37 31525197391593473 7  52 3
38 180143985094819841 5  55 6
39 1945555039024054273 27 56 5
40 4179340454199820289 29 57 3

```

## 线性基

```

1  struct LB
2  {
3      static constexpr int L=60;
4      array<i64,L+1> a{};
5
6      LB(){}
7
8      LB(const vector<i64> &v) { init(v); }
9
10     bool insert(i64 t)
11     {
12         for (int i=L;i>=0;i--)
13             if (t&(1ll<<i))
14             {
15                 if (!a[i])
16                 {
17                     a[i]=t;
18                     return 1;
19                 }
20                 else t^=a[i];
21             }
22         return 0;
23     }
24
25     void init(const vector<i64> &v) { for (auto x:v) insert(x); }
26
27     bool check(i64 t)
28     {
29         for (int i=L;i>=0;i--)
30             if (t&(1ll<<i))
31                 if (!a[i]) return 0;
32                 else t^=a[i];
33         return 1;
34     }
35
36     i64 QueryMax()
37     {
38         i64 res=0;
39         for (int i=L;i>=0;i--)

```

```

40         res=max(res,res^a[i]);
41     return res;
42 }
43
44 i64 QueryMin()
45 {
46     for (int i=0;i<=L;i++)
47         if (a[i]) return a[i];
48     return 0;
49 }
50
51 i64 QueryKth(int k)
52 {
53     i64 res=0;
54     int cnt=0;
55     array<i64,L+1> tmp{};
56     for (int i=0;i<=L;i++)
57     {
58         for (int j=i-1;j>=0;j--)
59             if (a[i]&(1ll<<j)) a[i]^=a[j];
60         if (a[i]) tmp[cnt++]=a[i];
61     }
62     if (k>=(1ll<<cnt)) return -1;
63     for (int i=0;i<cnt;i++)
64         if (k&(1ll<<i)) res^=tmp[i];
65     return res;
66 }
67 };

```

## min-plus 卷积

$\mathcal{O}(n \log n)$ , 但要求  $b$  是凸的。

```

1  template <class T>
2  vector<T> min_plus_convolution(const vector<T> &a,const vector<T> &b)
3  {
4      int n=a.size(),m=b.size();
5      vector<T> c(n+m-1);
6
7      function<void(int,int,int,int)> solve=[&](int l,int r,int ql,int qr)
8      {
9          if (l>r) return;
10         int mid=(l+r)>>1;
11         while (ql+m<=l) ++ql;
12         while (qr>r) --qr;
13         int qmid=-1;
14         c[mid]=inf;
15         for (int i=ql;i<=qr;i++)
16         {
17             if (a[i]+b[mid-i]-i<c[mid])
18             {
19                 c[mid]=a[i]+b[mid-i];
20                 qmid=i;
21             }
22             else if (mid-i>=0&&mid-i<m) qmid=i;
23         }
24         solve(l,mid-1,ql,mid);
25         solve(mid+1,r,qmid,qr);
26     };
27
28     solve(0,n+m-2,0,n-1);
29     return c;
30 }

```

## 模意义分数还原

分别是求：分子不大于  $A$  时分子最大的分数；分子分母最大值最小的分数。

```

1  pair<int,int> restore(int q,int A)
2  {

```

```

3     int x=q,y=P,a=1,b=0;
4     while (x>A)
5     {
6         swap(x,y);
7         swap(a,b);
8         a-=x/y*b;
9         x%=y;
10    }
11    return make_pair(x,a);
12 }
13
14 pair<int,int> restore(int x)
15 {
16     vector<int> a;
17     int p=P;
18     Z inv=Z(x).inv();
19     while (x)
20     {
21         a.push_back(x);
22         swap(x,p);
23         x%=p;
24     }
25     pair<int,int> res{P,P};
26     for (auto ca:a)
27     {
28         int cb=(Z(ca)*inv).x;
29         ca=min(ca,P-ca);
30         cb=min(cb,P-cb);
31         if (max(res.first,res.second)>max(ca,cb))
32             res={ca,cb};
33     }
34     return res;
35 }

```

## Exgcd

可以证明  $|x| \leq b, |y| \leq a$ 。

```

1 void exgcd(i64 a,i64 b,i64 &x,i64 &y)
2 {
3     if (!b)
4     {
5         x=1; y=0;
6         return;
7     }
8     exgcd(b,a%b,x,y);
9     swap(x,y);
10    y-=a/b*x;
11    return;
12 }

```

## 一次函数下取整区间和

求  $\sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$ 。

其实是类欧，时间复杂度为  $\mathcal{O}(\log n)$ 。

```

1 //求 sum_{i=0}^{n} floor(a*i+b/c).
2 i64 floorSum(i64 a,i64 b,i64 c,i64 n)
3 {
4     i64 res=0;
5     if (a>=c)
6     {
7         res+=n*(n+1)*(a/c)/2;
8         a%=c;
9     }
10    if (b>=c)
11    {
12        res+=(n+1)*(b/c);
13        b%=c;

```

```

14     }
15     i64 m=(a*n+b)/c;
16     if (m==0) return res;
17     res+=n*m-floorSum(c,c-b-1,a,m-1);
18     return res;
19 }

```

## 二元一次不定方程

给定不定方程  $ax + by = c$ 。

若该方程无整数解，输出  $-1$ 。

若该方程有整数解，且有正整数解，则输出其正整数解的数量，所有正整数解中  $x$  的最小值，所有正整数解中  $y$  的最小值，所有正整数解中  $x$  的最大值，以及所有正整数解中  $y$  的最大值。

若方程有整数解，但没有正整数解，输出所有整数解中  $x$  的最小正整数值， $y$  的最小正整数值。

```

1 void exgcd(i64 a,i64 b,i64 &x,i64 &y)
2 {
3     if (!b)
4     {
5         x=1; y=0;
6         return;
7     }
8     exgcd(b,a%b,x,y);
9     swap(x,y);
10    y-=a/b*x;
11    return;
12 }
13
14 i64 ceilDiv(i64 n,i64 m)
15 {
16     if (n>=0) return (n+m-1)/m;
17     else return n/m;
18 }
19
20 i64 floorDiv(i64 n,i64 m)
21 {
22     if (n>=0) return n/m;
23     else return (n-m+1)/m;
24 }
25
26 void R()
27 {
28     i64 a,b,c,x,y,t;
29     cin>>a>>b>>c;
30     t=__gcd(a,b);
31     if (c%t)
32     {
33         cout<<"-1\n";
34         return;
35     }
36     exgcd(a,b,x,y);
37     x*=c/t,y*=c/t;
38     i64 l=ceilDiv(1ll-x,b/t),r=floorDiv(y-1ll,a/t);
39     if (l>r) cout<<x+l*b/t<<' '<<y-r*a/t<<'\n';
40     else cout<<r-l+1ll<<' '<<x+l*b/t<<' '<<y-r*a/t<<' '<<x+r*b/t<<' '<<y-l*a/t<<'\n';
41     return;
42 }

```

## 行列式求值

时间复杂度为  $\mathcal{O}(n^3)$ 。

```

1 Z det(vector<vector<Z>> a)
2 {
3     int n=a.size(),fl=1;
4     Z res=1;
5     for (int i=0;i<n;i++)

```



```

6     {
7         for (int j=i+1;j<n;j++)
8         {
9             while (a[i][i].x)
10            {
11                int d=a[j][i].x/a[i][i].x;
12                for (int k=i;k<n;k++)
13                    a[j][k]-=a[i][k]*d;
14                swap(a[i],a[j]);
15                fl=-fl;
16            }
17            swap(a[i],a[j]);
18            fl=-fl;
19        }
20    }
21    for (int i=0;i<n;i++) res*=a[i][i];
22    res*=fl;
23    return res;
24 }

```

## 高斯消元法

返回 -1 代表无解, 其余情况返回自由元数。

```

1  using Real=long double;
2  constexpr Real eps=1e-8;
3
4  int Gauss(vector<vector<Real>> a,vector<Real> &x)
5  {
6      int n=a.size(),i=0,j=0;
7      for (;i<n&&j<n;i++,j++)
8      {
9          int mx=i;
10         for (int k=i+1;k<n;k++)
11             if (abs(a[k][j])>abs(a[mx][j]))
12                 mx=k;
13         if (mx!=i) swap(a[mx],a[i]);
14         if (fabs(a[i][j])<eps)
15         {
16             i--;
17             continue;
18         }
19         for (int k=i+1;k<n;k++)
20             if (fabs(a[k][j])>eps)
21             {
22                 Real t=a[k][j]/a[i][j];
23                 for (int l=j;l<n;l++)
24                     a[k][l]-=a[i][l]*t;
25                 a[k][j]=0;
26             }
27     }
28     for (int k=i;k<n;k++)
29         if (fabs(a[k][j])>eps)
30             return -1;//No solution
31     if (i<n) return n-i;//number of free elements
32     for (int k=n-1;k>=0;k--)
33     {
34         for (int l=k+1;l<n;l++)
35             a[k][n]-=a[k][l]*x[l];
36         x[k]=a[k][n]/a[k][k];
37     }
38     return 0;//Only one solution
39 }

```

## 枚举二进制下有 $k$ 个 1 的数

```

1  for (int s=(1<<k)-1,t;s<1<<n;t=s+(s&-s),s=(s&-t)>>__lg(s&-s)+1|t)

```

## 数据结构

### 并查集（启发式合并 + 带撤销）

```
1 struct DSU
2 {
3     int n=0;
4     vector<int> fa,siz;
5     stack<int> s;
6
7     DSU(int n) { init(n); }
8
9     void init(int n)
10    {
11        fa.resize(n);
12        iota(fa.begin(),fa.end(),0);
13        siz.assign(n,1);
14        while (!s.empty()) s.pop();
15    }
16
17    int get(int x) { return fa[x]==x?x:get(fa[x]); }
18
19    void merge(int x,int y)
20    {
21        x=get(x),y=get(y);
22        if (x==y) return;
23        if (siz[x]<siz[y]) swap(x,y);
24        s.push(y),fa[y]=x,siz[x]+=siz[y];
25    }
26
27    void undo()
28    {
29        if (s.empty()) return;
30        int y=s.top();
31        s.pop();
32        siz[fa[y]]-=siz[y];
33        fa[y]=y;
34    }
35
36    void back(int t=0) { while (s.size()) undo(); }
37 };
```

### 状压 RMQ

```
1 template <class T,class Cmp=less<T>>
2 struct RMQ
3 {
4     const Cmp cmp=Cmp();
5     static constexpr unsigned B=64;
6     using u64=unsigned long long;
7     int n;
8     vector<vector<T>> a;
9     vector<T> pre,suf,ini;
10    vector<u64> stk;
11
12    RMQ() {}
13    RMQ(const vector<T> &v) { init(v); }
14
15    void init(const vector<T> &v)
16    {
17        n=v.size();
18        pre=suf=ini=v;
19        stk.resize(n);
20        if (!n) return;
21        const int M=(n-1)/B+1;
22        const int lg=__lg(M);
23        a.assign(lg+1,vector<T>(M));
24        for (int i=0;i<M;i++)
25        {
26            a[0][i]=v[i*B];
```

```

27         for (int j=1;j<B&& i*B+j<n;j++)
28             a[0][i]=min(a[0][i],v[i*B+j],cmp);
29     }
30     for (int i=1;i<n;i++)
31         if (i%B) pre[i]=min(pre[i],pre[i-1],cmp);
32     for (int i=n-2;i>=0;i--)
33         if (i%B!=B-1) suf[i]=min(suf[i],suf[i+1],cmp);
34     for (int j=0;j<lg;j++)
35         for (int i=0;i+(2<<j)<=M;i++)
36             a[j+1][i]=min(a[j][i],a[j][i+(1<<j)],cmp);
37     for (int i=0;i<M;i++)
38     {
39         const int l=i*B;
40         const int r=min(1U*n,l+B);
41         u64 s=0;
42         for (int j=l;j<r;j++)
43         {
44             while (s&&cmp(v[j],v[__lg(s)+l])) s^=1ULL<<__lg(s);
45             s|=1ULL<<(j-l);
46             stk[j]=s;
47         }
48     }
49 }
50
51 //查询区间 [l,r) 的 RMQ
52 T operator()(int l,int r)
53 {
54     if (l/B!=(r-1)/B)
55     {
56         T ans=min(suf[l],pre[r-1],cmp);
57         l=l/B+1,r=r/B;
58         if (l<r)
59         {
60             int k=__lg(r-l);
61             ans=min({ans,a[k][l],a[k][r-(1<<k)]},cmp);
62         }
63         return ans;
64     }
65     else
66     {
67         int x=B*(l/B);
68         return ini[___builtin_ctzll(stk[r-1]>>(l-x))+l];
69     }
70 }
71 };

```

## ST 表

```

1  template <class T>
2  struct ST
3  {
4      int n;
5      vector<vector<T>> a;
6
7      ST() {}
8      ST(const vector<T> &v) { init(v); }
9
10     void init(const vector<T> &v)
11     {
12         n=v.size();
13         if (!n) return;
14         const int lg=__lg(n);
15         a.assign(lg+1,vector<T>(n));
16         a[0]=v;
17         for (int j=0;j<lg;j++)
18             for (int i=0;i+(2<<j)<=n;i++)
19                 a[j+1][i]=__gcd(a[j][i],a[j][i+(1<<j)]);
20     }
21
22     T operator()(int l,int r)
23     {

```

```

24         int k=__lg(r-l);
25         return __gcd(a[k][l],a[k][r-(1<<k)]);
26     }
27 };

```

## 树状数组

```

1  template <class T>
2  struct BIT
3  {
4      int n;
5      vector<T> a;
6
7      BIT(int n_=0) { init(n_); }
8
9      void init(int n_)
10     {
11         n=n_;
12         a.assign(n,T{});
13     }
14
15     void add(int x,const T &v)
16     {
17         for (int i=x+1;i<=n;i+=i&-i)
18             a[i-1]=a[i-1]+v;
19     }
20
21     //查询区间 [0,x)
22     T sum(int x)
23     {
24         T ans{};
25         for (int i=x;i>0;i-=i&-i)
26             ans=ans+a[i-1];
27         return ans;
28     }
29
30     //查询区间 [l,r)
31     T rangeSum(int l,int r) { return sum(r)-sum(l); }
32
33     int select(const T &k)
34     {
35         int x=0;
36         T cur{};
37         for (int i=1<<__lg(n);i>=1)
38         {
39             if (x+i<=n&&cur+a[x+i-1]<=k)
40             {
41                 x+=i;
42                 cur=cur+a[x-1];
43             }
44         }
45         return x;
46     }
47 };

```

## 线段树

```

1  template <class Info,class Tag>
2  struct SGT
3  {
4      int n;
5      vector<Info> info;
6      vector<Tag> tag;
7
8      SGT():n(0) {}
9      SGT(int n_,Info v_=Info()) { init(n_,v_); }
10
11     template <class T>
12     SGT(vector<T> init_) { init(init_); }
13

```

```

14 void init(int n_,Info v_=Info()) { init(vector(n_,v_)); }
15
16 template <class T>
17 void init(vector<T> init_)
18 {
19     n=init_.size();
20     info.assign(4<<__lg(n),Info());
21     tag.assign(4<<__lg(n),Tag());
22     function<void(int,int,int)> build=[&](int p,int l,int r)
23     {
24         if (r-l==1)
25         {
26             info[p]=init_[l];
27             return;
28         }
29         int m=(l+r)>>1;
30         build(p<<1,l,m);
31         build(p<<1|1,m,r);
32         pushup(p);
33     };
34     build(1,0,n);
35 }
36
37 void pushup(int p) { info[p]=info[p<<1]+info[p<<1|1]; }
38
39 void apply(int p,const Tag &v)
40 {
41     info[p].apply(v);
42     tag[p].apply(v);
43 }
44
45 void pushdown(int p)
46 {
47     apply(p<<1,tag[p]);
48     apply(p<<1|1,tag[p]);
49     tag[p]=Tag();
50 }
51
52 void modify(int p,int l,int r,int x,const Info &v)
53 {
54     if (r-l==1)
55     {
56         info[p]=v;
57         return;
58     }
59     int m=(l+r)>>1;
60     pushdown(p);
61     if (x<m) modify(p<<1,l,m,x,v);
62     else modify(p<<1|1,m,r,x,v);
63     pushup(p);
64 }
65
66 //O(log n) 单点修改
67 void modify(int p,const Info &v) { modify(1,0,n,p,v); }
68
69 Info rangeQuery(int p,int l,int r,int x,int y)
70 {
71     if (l>=y||r<=x) return Info();
72     if (l>=x&&r<=y) return info[p];
73     int m=(l+r)>>1;
74     pushdown(p);
75     return rangeQuery(p<<1,l,m,x,y)+rangeQuery(p<<1|1,m,r,x,y);
76 }
77
78 //O(log n) 区间查询 [l,r)
79 Info rangeQuery(int l,int r) { rangeQuery(1,0,n,l,r); }
80
81 void rangeApply(int p,int l,int r,int x,int y,const Tag &v)
82 {
83     if (l>=y||r<=x) return;
84     if (l>=x&&r<=y)

```

```

85     {
86         apply(p,v);
87         return;
88     }
89     int m=(l+r)>>1;
90     pushdown(p);
91     rangeApply(p<<1,l,m,x,y,v);
92     rangeApply(p<<1|1,m,r,x,y,v);
93     pushup(p);
94 }
95
96 //O(log n) 区间操作 [l,r)
97 void rangeApply(int l,int r,const Tag &v) { rangeApply(1,0,n,l,r,v); }
98
99 //O(log n) 区间 [l,r) 内查找第一个合法位置
100 template <class F>
101 int findFirst(int p,int l,int r,int x,int y,F pred)
102 {
103     if (l>=y||r<=x||!pred(info[p])) return -1;
104     if (r-l==1) return l;
105     int m=(l+r)>>1;
106     pushdown(p);
107     int res=findFirst(p<<1,l,m,x,y,pred);
108     if (res==-1) res=findFirst(p<<1|1,m,r,x,y,pred);
109     return res;
110 }
111
112 template <class F>
113 int findFirst(int l,int r,F pred) { return findFirst(1,0,n,l,r,pred); }
114
115 template <class F>
116 int findLast(int p,int l,int r,int x,int y,F pred)
117 {
118     if (l>=y||r<=x||!pred(info[p])) return -1;
119     if (r-l==1) return l;
120     int m=(l+r)>>1;
121     pushdown(p);
122     int res=findLast(p<<1|1,m,r,x,y,pred);
123     if (res==-1) res=findLast(p<<1,l,m,x,y,pred);
124     return res;
125 }
126
127 template <class F>
128 int findLast(int l,int r,F pred) { return findLast(1,0,n,l,r,pred); }
129 };
130
131 //这里默认乘法优先 (x*a+b)*c+d=x*(a*c)+(b*c+d)
132 struct Tag
133 {
134     i64 a=1,b=0;
135     void apply(Tag t)
136     {
137         a*=t.a;
138         b=b*t.a+t.b;
139     }
140 };
141
142 struct Info
143 {
144     i64 x=0,l=0,r=0;
145     void apply(Tag t)
146     {
147         int len=r-l+1;
148         x=x*t.a+len*t.b;
149     }
150 };
151
152 Info operator + (Info a,Info b)
153 {
154     return {a.x+b.x,min(a.l,b.l),max(a.r,b.r)};
155 }

```

## 可并堆 (pb\_ds)

### 成员函数

- push(): 向堆中压入一个元素, 返回该元素位置的迭代器。
- pop(): 将堆顶元素弹出。
- top(): 返回堆顶元素。
- size() 返回元素个数。
- empty() 返回是否非空。
- modify(point\_iterator, const key): 把迭代器位置的 key 修改为传入的 key, 并对底层储存结构进行排序。
- erase(point\_iterator): 把迭代器位置的键值从堆中擦除。
- join(\_\_gnu\_pbds::priority\_queue &other): 把 other 合并到 \*this 并把 other 清空。

### 示例

```
1  #include <algorithm>
2  #include <cstdio>
3  #include <ext/pb_ds/priority_queue.hpp>
4  #include <iostream>
5  using namespace __gnu_pbds;
6  // 由于面向 OIer, 本文以常用堆 : pairing_heap_tag 作为范例
7  // 为了更好的阅读体验, 定义宏如下 :
8  using pair_heap = __gnu_pbds::priority_queue<int>;
9  pair_heap q1; // 大根堆, 配对堆
10 pair_heap q2;
11 pair_heap::point_iterator id; // 一个迭代器
12
13 int main() {
14     id = q1.push(1);
15     // 堆中元素 : [1];
16     for (int i = 2; i <= 5; i++) q1.push(i);
17     // 堆中元素 : [1, 2, 3, 4, 5];
18     std::cout << q1.top() << std::endl;
19     // 输出结果 : 5;
20     q1.pop();
21     // 堆中元素 : [1, 2, 3, 4];
22     id = q1.push(10);
23     // 堆中元素 : [1, 2, 3, 4, 10];
24     q1.modify(id, 1);
25     // 堆中元素 : [1, 1, 2, 3, 4];
26     std::cout << q1.top() << std::endl;
27     // 输出结果 : 4;
28     q1.pop();
29     // 堆中元素 : [1, 1, 2, 3];
30     id = q1.push(7);
31     // 堆中元素 : [1, 1, 2, 3, 7];
32     q1.erase(id);
33     // 堆中元素 : [1, 1, 2, 3];
34     q2.push(1), q2.push(3), q2.push(5);
35     // q1 中元素 : [1, 1, 2, 3], q2 中元素 : [1, 3, 5];
36     q2.join(q1);
37     // q1 中无元素, q2 中元素 : [1, 1, 1, 2, 3, 3, 5];
38 }
```

## 平衡树 (pb\_ds)

### 成员函数

- insert(x): 向树中插入一个元素 x, 返回 std::pair<point\_iterator, bool>。
- erase(x): 从树中删除一个元素/迭代器 x, 返回一个 bool 表明是否删除成功。
- order\_of\_key(x): 返回 x 以 Cmp\_Fn 比较的排名。
- find\_by\_order(x): 返回 Cmp\_Fn 比较的排名所对应元素的迭代器。
- lower\_bound(x): 以 Cmp\_Fn 比较做 lower\_bound, 返回迭代器。
- upper\_bound(x): 以 Cmp\_Fn 比较做 upper\_bound, 返回迭代器。
- join(x): 将 x 树并入当前树, 前提是两棵树的类型一样, x 树被删除。
- split(x,b): 以 Cmp\_Fn 比较, 小于等于 x 的属于当前树, 其余的属于 b 树。

- `empty()`: 返回是否为空。
- `size()`: 返回大小。

注意 `join(x)` 函数需要保证并入树的键的值域与被并入树的键的值域 **不相交**（也就是说并入树内所有值必须全部大于/小于当前树内的所有值），否则会抛出 `join_error` 异常。

如果要合并两棵值域有交集的树，需要将一棵树的元素一一插入到另一棵树中。

## 示例

```
1 // Common Header Simple over C++11
2 #include <iostream>
3 using namespace std;
4 using ll = long long;
5 using ull = unsigned long long;
6 using ld = long double;
7 using pii = pair<int, int>;
8 #include <ext/pb_ds/assoc_container.hpp>
9 #include <ext/pb_ds/tree_policy.hpp>
10 __gnu_pbds::tree<pair<int, int>, __gnu_pbds::null_type, less<pair<int, int>>,
11                 __gnu_pbds::rb_tree_tag,
12                 __gnu_pbds::tree_order_statistics_node_update>
13     trr;
14
15 int main() {
16     int cnt = 0;
17     trr.insert(make_pair(1, cnt++));
18     trr.insert(make_pair(5, cnt++));
19     trr.insert(make_pair(4, cnt++));
20     trr.insert(make_pair(3, cnt++));
21     trr.insert(make_pair(2, cnt++));
22     // 树上元素 {{1,0},{2,4},{3,3},{4,2},{5,1}}
23     auto it = trr.lower_bound(make_pair(2, 0));
24     trr.erase(it);
25     // 树上元素 {{1,0},{3,3},{4,2},{5,1}}
26     auto it2 = trr.find_by_order(1);
27     cout << (*it2).first << endl;
28     // 输出排名 0 1 2 3 中的排名 1 的元素的 first:1
29     int pos = trr.order_of_key(*it2);
30     cout << pos << endl;
31     // 输出排名
32     decltype(trr) newtr;
33     trr.split(*it2, newtr);
34     for (auto i = newtr.begin(); i != newtr.end(); ++i) {
35         cout << (*i).first << ' ';
36     }
37     cout << endl;
38     // {4,2},{5,1} 被放入新树
39     trr.join(newtr);
40     for (auto i = trr.begin(); i != trr.end(); ++i) {
41         cout << (*i).first << ' ';
42     }
43     cout << endl;
44     cout << newtr.size() << endl;
45     // 将 newtr 树并入 trr 树, newtr 树被删除。
46     return 0;
47 }
```

## 字符串

### 字符串哈希（随机模数）

#### 允许 $k$ 次失配的字符串匹配

枚举原串起点，二分出第一个失配位置，直到找不到失配位置或失配次数超过  $k$ ，时间复杂度  $\mathcal{O}(m + kn \log m)$ 。



## 最长公共子串

二分答案，把对应长度串的哈希值丢进 `map/unordered_map` 里判就好，时间复杂度  $\mathcal{O}(m + n \log^2 n)$ 。

### Code

```
1 bool isPrime(int n)
2 {
3     if (n<=1) return 0;
4     for (int i=2;i*i<=n;i++)
5         if (n%i==0) return 0;
6     return 1;
7 }
8
9 int findPrime(int n)
10 {
11     while (!isPrime(n)) n++;
12     return n;
13 }
14
15 mt19937 rng(time(0));
16 const int P=findPrime(rng()%9000000000+1000000000);
17 struct StrHash
18 {
19     int n;
20     vector<int> h,p;
21
22     StrHash(const string &s){ init(s); }
23
24     void init(const string &s)
25     {
26         n=s.size();
27         h.resize(n+1);
28         p.resize(n+1);
29         p[0]=1;
30         for (int i=0;i<n;i++) h[i+1]=(10ll*h[i]+s[i]-'a'+1)%P;
31         for (int i=0;i<n;i++) p[i+1]=10ll*p[i]%P;
32     }
33
34     //查询 [l,r) 的区间哈希
35     int get(int l,int r) { return (h[r]+1ll*(P-h[l])*p[r-l])%P; }
36 };
```

## KMP

### 字符串周期

最小正周期是 `n-pre.back()`，反复跳 `pre` 可以得到串的所有周期。

### 统计前缀出现次数

```
1 vector<int> ans(n+1);
2 for (int i=0;i<n;i++) ans[pre[i]]++;
3 for (int i=n-1;i>0;i--) ans[pre[i-1]] += ans[i];
4 for (int i=0;i<n;i++) ans[i]++;
```

### 求满足一些要求的 Border

比如有出现次数要求、两个前缀的最长公共 Border 什么的。

根据 `pre` 指针建出 Border 树，用类似 SAM 的 parent 树的处理方法就好。

### Code

```
1 vector<int> KMP(const string &s)
2 {
3     int now=0;
4     vector<int> pre(s.size(),0);
5     for (int i=1;i<s.size();i++)
```

```

6     {
7         while (now&& s[i]!=s[now]) now=pre[now-1];
8         if (s[i]==s[now]) now++;
9         pre[i]=now;
10    }
11    return pre;
12 }

```

## Z 函数

```

1 vector<int> zFunction(string s)
2 {
3     int n=s.size();
4     vector<int> z(n);
5     z[0]=n;
6     for (int i=1,j=1;i<n;i++)
7     {
8         z[i]=max(0,min(j+z[j]-i,z[i-j]));
9         while (i+z[i]<n&&s[z[i]]==s[i+z[i]]) z[i]++;
10        if (i+z[i]>j+z[j]) j=i;
11    }
12    return z;
13 }

```

## AC 自动机

每个节点代表一个前缀，指针指向最大 Border。

```

1 struct ACAM
2 {
3     static constexpr int ALPHABET=26;
4     struct Node
5     {
6         int len;
7         int link;
8         array<int,ALPHABET> next;
9         Node():len{0},link{0},next{}{}
10    };
11
12    vector<Node> t;
13
14    ACAM() { init(); }
15
16    void init()
17    {
18        t.assign(2,Node());
19        t[0].next.fill(1);
20        t[0].len=-1;
21    }
22
23    int newNode()
24    {
25        t.emplace_back();
26        return t.size()-1;
27    }
28
29    int add(const string &a)
30    {
31        int p=1;
32        for (auto c:a)
33        {
34            int x=c-'a';
35            if (t[p].next[x]==0)
36            {
37                t[p].next[x]=newNode();
38                t[t[p].next[x]].len=t[p].len+1;
39            }
40            p=t[p].next[x];
41        }
42        return p;

```

```

43     }
44
45     void work()
46     {
47         queue<int> q;
48         q.push(1);
49         while (!q.empty())
50         {
51             int x=q.front();
52             q.pop();
53             for (int i=0;i<ALPHABET;i++)
54             {
55                 if (t[x].next[i]==0) t[x].next[i]=t[t[x].link].next[i];
56                 else
57                 {
58                     t[t[x].next[i]].link=t[t[x].link].next[i];
59                     q.push(t[x].next[i]);
60                 }
61             }
62         }
63     }
64
65     int next(int p,int x) { return t[p].next[x]; }
66
67     int link(int p) { return t[p].link; }
68
69     int size() { return t.size(); }
70 };

```

## 后缀数组

```

1  struct SA
2  {
3      int n;
4      vector<int> sa,rk,lc;
5      SA(const string &s)
6      {
7          n=s.length();
8          sa.resize(n);
9          rk.resize(n);
10         lc.resize(n-1);
11         iota(sa.begin(),sa.end(),0);
12         sort(sa.begin(),sa.end(),[&](int a,int b){ return s[a]<s[b]; });
13         rk[sa[0]]=0;
14         for (int i=1;i<n;i++) rk[sa[i]]=rk[sa[i-1]]+(s[sa[i]]!=s[sa[i-1]]);
15         int k=1;
16         vector<int> tmp,cnt(n);
17         tmp.reserve(n);
18         while (rk[sa[n-1]]<n-1)
19         {
20             tmp.clear();
21             for (int i=0;i<k;i++) tmp.push_back(n-k+i);
22             for (auto i:sa)
23                 if (i>=k) tmp.push_back(i-k);
24             fill(cnt.begin(),cnt.end(),0);
25             for (int i=0;i<n;i++) cnt[rk[i]]++;
26             for (int i=1;i<n;i++) cnt[i]+=cnt[i-1];
27             for (int i=n-1;i>=0;i--) sa[--cnt[rk[tmp[i]]]]=tmp[i];
28             swap(rk,tmp);
29             rk[sa[0]]=0;
30             for (int i=1;i<n;i++)
31                 rk[sa[i]]=rk[sa[i-1]]+(tmp[sa[i-1]]<tmp[sa[i]] || sa[i-1]+k==n || tmp[sa[i-1]+k]<tmp[sa[i]+k]);
32             k<<=1;
33         }
34         for (int i=0,j=0;i<n;i++)
35         {
36             if (rk[i]==0) j=0;
37             else
38             {
39                 for (j--;j>0;i+j<n&&sa[rk[i]-1]+j<n&&s[i+j]==s[sa[rk[i]-1]+j]); j++;
40                 lc[rk[i]-1]=j;

```

```

41         } //lc[i]:lcp(sa[i],sa[i+1]),lcp(sa[i],sa[j])=min{lc[i...j-1]}
42     }
43 }
44 };

```

## (广义) 后缀自动机

每个节点代表的是一个 `endpos` 集合，指针指向最小超集。

### 不同子串个数

考虑节点  $i$  代表的子串数是  $\text{len}(i) - \text{len}(\text{link}(i))$ ，求和即可。

### 字典序第 $k$ 大子串

等价自动机上第  $k$  大路径，预处理每个状态后续路径数后 `dfs` 即可。

### 最小循环移位

对  $S + S$  建自动机，字典序最小的  $|S|$  长路径就是答案。

### 出现次数

每次插入字符后对终点做个标记，答案就是查询串在自动机上对应节点在 `parent` 树上的子树内标记和。

### 首次出现位置

维护每个节点对应首次出现位置 `firstpos`。

具体来说，插入点时  $\text{firstpos}(\text{cur}) = \text{len}(\text{cur}) + 1$ ，克隆点时  $\text{firstpos}(\text{clone}) = \text{firstpos}(q)$ 。

答案即为  $\text{firstpos}(t) - |T| + 1$ 。

### 所有出现位置

每次插入字符后对终点做个标记，查询时遍历 `parent` 树上的子树内标记并输出。

### 最短未出现字符串

自动机上 `dp` 即可，如果没有转移 `dp` 值就是 1，否则是各转移最小 `dp` 值加一，答案是根的 `dp` 值。

### 最长公共子串

把串都丢到自动机里，每次记录节点被哪些串占用，被所有串占用节点中 `len` 最大的就是答案。

## Code

```

1  struct SAM
2  {
3      static constexpr int ALPHABET=26;
4      struct Node
5      {
6          int len;
7          int link;
8          array<int,ALPHABET> next;
9          Node():len{},link{},next{} {}
10     };
11
12     vector<Node> t;
13
14     SAM() { init(); }
15
16     void init()
17     {
18         t.assign(2,Node());

```

```

19         t[0].next.fill(1);
20         t[0].len=-1;
21     }
22
23     int newNode()
24     {
25         t.emplace_back();
26         return t.size()-1;
27     }
28
29     int extend(int lst,int c)
30     {
31         if (t[lst].next[c]&& t[t[lst].next[c]].len==t[lst].len+1)
32             return t[lst].next[c];
33         int p=lst,np=newNode(),flag=0;
34         t[np].len=t[p].len+1;
35         while (!t[p].next[c])
36         {
37             t[p].next[c]=np;
38             p=t[p].link;
39         }
40         if (!p)
41         {
42             t[np].link=1;
43             return np;
44         }
45         int q=t[p].next[c];
46         if (t[q].len==t[p].len+1)
47         {
48             t[np].link=q;
49             return np;
50         }
51         if (p==lst) flag=1,np=0,t.pop_back();
52         int nq=newNode();
53         t[nq].link=t[q].link;
54         t[nq].next=t[q].next;
55         t[nq].len=t[p].len+1;
56         t[q].link=t[np].link=nq;
57         while (p&& t[p].next[c]==q)
58         {
59             t[p].next[c]=nq;
60             p=t[p].link;
61         }
62         return flag?nq:np;
63     }
64
65     int add(const string &a)
66     {
67         int p=1;
68         for (auto c:a) p=extend(p,c-'a');
69         return p;
70     }
71
72     int next(int p,int x) { return t[p].next[x]; }
73
74     int link(int p) { return t[p].link; }
75
76     int len(int p) { return t[p].len; }
77
78     int size() { return t.size(); }
79 };

```

## Manacher

```

1 vector<int> manacher(vector<int> s)
2 {
3     vector<int> t{0};
4     for (auto c:s)
5     {
6         t.push_back(c);
7         t.push_back(0);

```

```

8     }
9     int n=t.size();
10    vector<int> r(n);
11    for (int i=0,j=0;i<n;i++)
12    {
13        if (j*2-i>=0&&j+r[j]>i) r[i]=min(r[j*2-i],j+r[j]-i);
14        while (i-r[i]>=0&&i+r[i]<n&&t[i-r[i]]==t[i+r[i]]) r[i]++;
15        if (i+r[i]>j+r[j]) j=i;
16    }
17    return r;
18 }

```

## 回文自动机

每个节点代表的是一个回文子串，指针指向最长回文后缀。

### 本质不同回文子串数

即自动机点数，记得减去奇偶根。

### 回文子串出现次数

即 fail 树子树内终点标记和。

## Code

```

1  struct PAM
2  {
3      static constexpr int ALPHABET_SIZE=28;
4      struct Node
5      {
6          int len,link,cnt;
7          array<int,ALPHABET_SIZE> next;
8          Node():len{},link{},cnt{},next{}{}
9      };
10     vector<Node> t;
11     int suff;
12     string s;
13
14     PAM() { init(); }
15
16     void init()
17     {
18         t.assign(2,Node());
19         t[0].len=-1;
20         suff=1;
21         s.clear();
22     }
23
24     int newNode()
25     {
26         t.emplace_back();
27         return t.size()-1;
28     }
29
30     bool add(char c,char offset='a')
31     {
32         int pos=s.size();
33         s+=c;
34         int let=c-offset;
35         int cur=suff,curlen=0;
36         while (1)
37         {
38             curlen=t[cur].len;
39             if (pos-curlen-1>=0&&s[pos-curlen-1]==s[pos]) break;
40             cur=t[cur].link;
41         }
42         if (t[cur].next[let])
43         {

```

```

44         suff=t[cur].next[let];
45         return 0;
46     }
47     int num=newNode();
48     suff=num;
49     t[num].len=t[cur].len+2;
50     t[cur].next[let]=num;
51     if (t[num].len==1)
52     {
53         t[num].link=t[num].cnt=1;
54         return 1;
55     }
56     while (1)
57     {
58         cur=t[cur].link;
59         curlen=t[cur].len;
60         if (pos-curlen-1>=0&&s[pos-curlen-1]==s[pos])
61         {
62             t[num].link=t[cur].next[let];
63             break;
64         }
65     }
66     t[num].cnt=t[t[num].link].cnt+1;
67     return 1;
68 }
69 };

```

## 含通配符字符串匹配

返回匹配的位置集合。

```

1  vector<int> match(string &s,string &t)
2  {
3      static mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
4      static array<Z,256> c;
5      static bool initd=0;
6      if (!initd)
7      {
8          initd=1;
9          for (Z &x:c) x=rng();
10         c['*']=0;//wildcard
11     }
12     int n=s.size(),m=t.size();
13     if (n<m) return {};
14     vector<int> res;
15     Poly f(n),ff(n),fff(n),g(m),gg(m),ggg(m);
16     for (int i=0;i<n;i++)
17     {
18         f[i]=c[s[i]];
19         ff[i]=f[i]*f[i];
20         fff[i]=ff[i]*f[i];
21     }
22     for (int i=0;i<m;i++)
23     {
24         g[i]=c[t[m-i-1]];
25         gg[i]=g[i]*g[i];
26         ggg[i]=gg[i]*g[i];
27     }
28     Poly fffg=fff*g,ffgg=ff*gg,fggg=f*ggg;
29     for (int i=0;i<=n-m;i++)
30         if ((fffg[m-1+i]+fggg[m-1+i]-ffgg[m-1+i]*2)==0)
31             res.push_back(i);
32     return res;
33 }
34 /*
35 a***b***c*****
36 bcd
37
38 match(s,t)=[ 1, 4, 5, 7, 9, 10, 11, 12, ]
39 */

```

## 图论

### 拓扑排序

```
1 vector<int> topo(vector<vector<int>> &adj)
2 {
3     int n=adj.size();
4     vector<int> res,in(n);
5     queue<int> q;
6     for (int u=0;u<n;u++)
7         for (int v:adj[u])
8             in[v]++;
9     for (int u=0;u<n;u++)
10        if (!in[u])
11            q.push(u);
12    while (!q.empty())
13    {
14        int u=q.front();
15        q.pop();
16        res.push_back(u);
17        for (int v:adj[u])
18        {
19            in[v]--;
20            if (!in[v]) q.push(v);
21        }
22    }
23    return res;
24 }
```

### 树的直径

```
1 int diameter(vector<vector<int>> &adj)
2 {
3     int n=adj.size(),d=0;
4     vector<int> dp(n);
5
6     auto dfs=[&](auto &self,int u,int f)->void
7     {
8         for (int v:adj[u])
9         {
10             if (v==f) continue;
11             self(self,v,u);
12             d=max(d,dp[u]+dp[v]+1);//w(u,v)=1
13             dp[u]=max(dp[u],dp[v]+1);//w(u,v)=1
14         }
15     };
16
17     dfs(dfs,0,0);
18     return d;
19 }
```

### 树的重心

```
1 vector<int> centroid(vector<vector<int>> &adj,int rt)
2 {
3     int n=adj.size();
4     vector<int> siz(n),res(n),w(n),fa(n);
5
6     auto dfs=[&](auto &self,int u,int f)->void
7     {
8         siz[u]=1,res[u]=u,fa[u]=f;
9         for (int v:adj[u])
10        {
11            if (v==f) continue;
12            self(self,v,u);
13            siz[u]+=siz[v];
14            w[u]=max(w[u],siz[v]);
15        }
16        for (int v:adj[u])
17        {
```



```

18         if (v==f) continue;
19         int p=res[v];
20         while (p!=u)
21         {
22             if (max(w[p],siz[u]-siz[p])<=siz[u]/2)
23             {
24                 res[u]=p;
25                 break;
26             }
27             else p=fa[p];
28         }
29     }
30 };
31
32 dfs(dfs,rt,rt);
33 return res;
34 }

```

## Dijkstra

注意设定合适的 inf.

```

1 vector<i64> dijk(const vector<vector<pair<int,i64>>> &adj,int s)
2 {
3     int n=adj.size();
4     using pa=pair<i64,int>;
5     vector<i64> d(n,inf);
6     vector<int> ed(n);
7     priority_queue<pa,vector<pa>,greater<pa>> q;
8     q.push({0,s}); d[s]=0;
9     while (!q.empty())
10    {
11        int u=q.top().second;
12        q.pop();
13        ed[u]=1;
14        for (auto [v,w]:adj[u])
15            if (d[u]+w<d[v])
16            {
17                d[v]=d[u]+w;
18                q.push({d[v],v});
19            }
20        while (!q.empty()&&ed[q.top().second]) q.pop();
21    }
22    return d;
23 }

```

## SPFA

注意设定合适的 inf.

```

1 vector<i64> spfa(const vector<vector<pair<int,i64>>> &adj,int s)
2 {
3     int n=adj.size();
4     assert(n);
5     queue<int> q;
6     vector<int> len(n),ed(n);
7     vector<i64> d(n,inf);
8     q.push(s); d[s]=0;
9     while (!q.empty())
10    {
11        int u=q.front();
12        q.pop();
13        ed[u]=0;
14        for (auto [v,w]:adj[u])
15            if (d[u]+w<d[v])
16            {
17                d[v]=d[u]+w;
18                len[v]=len[u]+1;
19                if (len[v]>n) return {};
20                if (!ed[v]) ed[v]=1,q.push(v);

```

```

21     }
22 }
23 return d;
24 }

```

## Johnson

```

1 vector<vector<i64>> dijk(const vector<vector<pair<int,i64>>> &adj)
2 {
3     vector<vector<i64>> res;
4     for (int i=0;i<adj.size();i++)
5         res.push_back(dijk(adj,i));
6     return res;
7 }
8
9 vector<i64> spfa(const vector<vector<pair<int,i64>>> &adj)
10 {
11     int n=adj.size();
12     assert(n);
13     queue<int> q;
14     vector<int> len(n),ed(n,1);
15     vector<i64> d(n);
16     for (int i=0;i<n;i++) q.push(i);
17     while (!q.empty())
18     {
19         int u=q.front();
20         q.pop();
21         ed[u]=0;
22         for (auto [v,w]:adj[u])
23             if (d[u]+w<d[v])
24             {
25                 d[v]=d[u]+w;
26                 len[v]=len[u]+1;
27                 if (len[v]>n) return {};
28                 if (!ed[v]) ed[v]=1,q.push(v);
29             }
30     }
31     return d;
32 }
33
34 vector<vector<i64>> john(vector<vector<pair<int,i64>>> adj)
35 {
36     int n=adj.size();
37     assert(n);
38     auto h=spfa(adj);
39     if (!h.size()) return {};
40     for (int u=0;u<n;u++)
41         for (auto &[v,w]:adj[u])
42             w+=h[u]-h[v];
43     auto res=dijk(adj);
44     for (int u=0;u<n;u++)
45         for (int v=0;v<n;v++)
46             if (res[u][v]!=inf)
47                 res[u][v]-=h[u]-h[v];
48     return res;
49 }

```

## 强连通分量

```

1 struct SCC
2 {
3     int n,cur,cnt;
4     vector<vector<int>> adj;
5     vector<int> stk,dfn,low,bel;
6
7     SCC() {}
8     SCC(int n) { init(n); }
9
10    void init(int n)
11    {

```

```

12     this->n=n;
13     adj.assign(n,{});
14     stk.clear();
15     dfn.assign(n,-1);
16     low.resize(n);
17     bel.assign(n,-1);
18     cur=cnt=0;
19 }
20
21 void add(int u,int v) { adj[u].push_back(v); }
22
23 void dfs(int x)
24 {
25     dfn[x]=low[x]=cur++;
26     stk.push_back(x);
27     for (auto y:adj[x])
28     {
29         if (dfn[y]==-1)
30         {
31             dfs(y);
32             low[x]=min(low[x],low[y]);
33         }
34         else if (bel[y]==-1) low[x]=min(low[x],dfn[y]);
35     }
36     if (dfn[x]==low[x])
37     {
38         int y;
39         do
40         {
41             y=stk.back();
42             bel[y]=cnt;
43             stk.pop_back();
44         } while (y!=x);
45         cnt++;
46     }
47 }
48
49 vector<int> work()
50 {
51     for (int i=0;i<n;i++)
52         if (dfn[i]==-1) dfs(i);
53     return bel;
54 }
55
56 struct Graph
57 {
58     int n;
59     vector<pair<int,int>> edges;
60     vector<int> siz,cnte;
61 };
62
63 Graph compress()
64 {
65     Graph G;
66     G.n=cnt;
67     G.siz.resize(cnt);
68     G.cnte.resize(cnt);
69     for (int i=0;i<n;i++)
70     {
71         G.siz[bel[i]]++;
72         for (auto j:adj[i])
73             if (bel[i]!=bel[j])
74                 G.edges.emplace_back(bel[j],bel[i]);
75     }
76     return G;
77 };
78 };

```

## 边双连通分量

```
1 struct EBCC
2 {
3     int n;
4     vector<vector<int>> adj;
5     vector<int> stk,dfn,low,bel;
6     int cur,cnt;
7
8     EBCC() {}
9     EBCC(int n) { init(n); }
10
11     void init(int n)
12     {
13         this->n=n;
14         adj.assign(n,{});
15         dfn.assign(n,-1);
16         low.resize(n);
17         bel.assign(n,-1);
18         stk.clear();
19         cur=cnt=0;
20     }
21
22     void add(int u,int v)
23     {
24         adj[u].push_back(v);
25         adj[v].push_back(u);
26     }
27
28     void dfs(int x,int p)
29     {
30         dfn[x]=low[x]=cur++;
31         stk.push_back(x);
32         for (auto y:adj[x])
33         {
34             if (y==p) continue;
35             if (dfn[y]==-1)
36             {
37                 dfs(y,x);
38                 low[x]=min(low[x],low[y]);
39             }
40             else if (bel[y]==-1&&dfn[y]<dfn[x]) low[x]=min(low[x],dfn[y]);
41         }
42         if (dfn[x]==low[x])
43         {
44             int y;
45             do
46             {
47                 y=stk.back();
48                 bel[y]=cnt;
49                 stk.pop_back();
50             } while (y!=x);
51             cnt++;
52         }
53     }
54
55     vector<int> work()
56     {
57         dfs(0,-1);
58         return bel;
59     }
60
61     struct Graph
62     {
63         int n;
64         vector<pair<int,int>> edges;
65         vector<int> siz,cnt;
66     };
67
68     Graph compress()
69     {
```

```

70     Graph G;
71     G.n=cnt;
72     G.siz.resize(cnt);
73     G.cnte.resize(cnt);
74     for (int i=0;i<n;i++)
75     {
76         G.siz[bel[i]]++;
77         for (auto j:adj[i])
78         {
79             if (bel[i]<bel[j]) G.edges.emplace_back(bel[i],bel[j]);
80             else if (i<j) G.cnte[bel[i]]++;
81         }
82     }
83     return G;
84 };
85 };

```

## 轻重链剖分

```

1  struct HLD
2  {
3      int n;
4      vector<int> siz,top,dep,pa,in,out,seq;
5      vector<vector<int>> adj;
6      int cur;
7
8      HLD(){}
9      HLD(int n) { init(n); }
10
11     void init(int n)
12     {
13         this->n=n;
14         siz.resize(n);
15         top.resize(n);
16         dep.resize(n);
17         pa.resize(n);
18         in.resize(n);
19         out.resize(n);
20         seq.resize(n);
21         cur=0;
22         adj.assign(n,{});
23     }
24
25     void addEdge(int u,int v)
26     {
27         adj[u].push_back(v);
28         adj[v].push_back(u);
29     }
30
31     void work(int rt=0)
32     {
33         top[rt]=rt;
34         dep[rt]=0;
35         pa[rt]=-1;
36         dfs1(rt);
37         dfs2(rt);
38     }
39
40     void dfs1(int u)
41     {
42         if (pa[u]==-1) adj[u].erase(find(adj[u].begin(),adj[u].end(),pa[u]));
43         siz[u]=1;
44         for (auto &v:adj[u])
45         {
46             pa[v]=u;
47             dep[v]=dep[u]+1;
48             dfs1(v);
49             siz[u]+=siz[v];
50             if (siz[v]>siz[adj[u][0]])
51                 swap(v,adj[u][0]);
52         }

```

```

53     }
54
55     void dfs2(int u)
56     {
57         in[u]=cur++;
58         seq[in[u]]=u;
59         for (auto v:adj[u])
60         {
61             top[v]=(v==adj[u][0])?top[u]:v;
62             dfs2(v);
63         }
64         out[u]=cur;
65     }
66
67     int lca(int u,int v)
68     {
69         while (top[u]!=top[v])
70         {
71             if (dep[top[u]]>dep[top[v]]) u=pa[top[u]];
72             else v=pa[top[v]];
73         }
74         return dep[u]<dep[v]?u:v;
75     }
76
77     int dist(int u,int v) { return dep[u]+dep[v]-(dep[lca(u,v)]<<1); }
78
79     int jump(int u,int k)
80     {
81         if (dep[u]<k) return -1;
82         int d=dep[u]-k;
83         while (dep[top[u]]>d) u=pa[top[u]];
84         return seq[in[u]-dep[u]+d];
85     }
86
87     bool isAncestor(int u,int v) { return in[u]<=in[v]&&in[v]<out[u]; }
88
89     int rootedParent(int u,int v)//u->root,v->point
90     {
91         if (u==v) return u;
92         if (!isAncestor(v,u)) return pa[v];
93         auto it=upper_bound(adj[v].begin(),adj[v].end(),u,[&](int x,int y){ return in[x]<in[y]; })-1;
94         return *it;
95     }
96
97     int rootedSize(int u,int v)//same as rootedParent
98     {
99         if (u==v) return n;
100         if (!isAncestor(v,u)) return siz[v];
101         return n-siz[rootedParent(u,v)];
102     }
103
104     int rootedLca(int a,int b,int c) { return lca(a,b)^lca(b,c)^lca(c,a); }
105 };

```

## 欧拉路径

```

1 vector<int> euler(vector<vector<int>>> adj)
2 {
3     int n=adj.size(),x=0;
4     vector<int> in(n),out(n);
5     for (int u=0;u<n;u++)
6         for (int v:adj[u])
7             out[u]++,in[v]++;
8     for (int i=0;i<n;i++)
9         if (in[i]!=out[i])
10            {
11                if (abs(in[i]-out[i])>1) return {};
12                x++;
13            }
14     if (x>2) return {};
15     for (int i=0;i<n;i++)

```

```

16         if (out[i]>in[i])
17         {
18             x=i;
19             break;
20         }
21     for (int i=0;i<n;i++)
22         sort(adj[i].begin(),adj[i].end(),greater<int>());
23
24     vector<int> res;
25     auto dfs=[&](auto &self,int u)->void
26     {
27         while (!adj[u].empty())
28         {
29             int v=adj[u].back();
30             adj[u].pop_back();
31             self(self,v);
32             res.push_back(v);
33         }
34     };
35
36     dfs(dfs,x);
37     res.push_back(x);
38     reverse(res.begin(),res.end());
39     return res;
40 }

```

## 2-SAT

```

1 struct TwoSat
2 {
3     int n;
4     vector<vector<int>> e;
5     vector<bool> ans;
6
7     TwoSat(int n):n(n),e(n<<1),ans(n){}
8
9     void addClause(int u,bool f,int v,bool g)
10    {
11        e[u*2+!f].push_back(v*2+g);
12        e[v*2+!g].push_back(u*2+f);
13    }
14
15    bool satisfiable()
16    {
17        vector<int> id(n*2,-1),dfn(n*2,-1),low(n*2,-1),stk;
18        int now=0,cnt=0;
19        function<void(int)> tarjan=[&](int u)
20        {
21            stk.push_back(u);
22            dfn[u]=low[u]=now++;
23            for (auto v:e[u])
24            {
25                if (dfn[v]==-1)
26                {
27                    tarjan(v);
28                    low[u]=min(low[u],low[v]);
29                }
30                else if (id[v]==-1)
31                    low[u]=min(low[u],dfn[v]);
32            }
33            if (dfn[u]==low[u])
34            {
35                int v;
36                do
37                {
38                    v=stk.back();
39                    stk.pop_back();
40                    id[v]=cnt;
41                } while (v!=u);
42                cnt++;
43            }

```

```

44     };
45     for (int i=0;i<n*2;i++)
46         if (dfn[i]==-1)
47             tarjan(i);
48     for (int i=0;i<n;i++)
49     {
50         if (id[i*2]==id[i*2+1]) return 0;
51         ans[i]=id[i*2]>id[i*2+1];
52     }
53     return 1;
54 }
55 vector<bool> answer() { return ans; }
56 };

```

## 最大流

```

1  template <class T>
2  struct MaxFlow
3  {
4      struct _Edge
5      {
6          int to;
7          T cap;
8          _Edge(int to,T cap):to(to),cap(cap){}
9      };
10
11     int n;
12     vector<_Edge> e;
13     vector<vector<int>> g;
14     vector<int> cur,h;
15
16     MaxFlow(){}
17     MaxFlow(int n) { init(n); }
18
19     void init(int n)
20     {
21         this->n=n;
22         e.clear();
23         g.assign(n,{});
24         cur.resize(n);
25         h.resize(n);
26     }
27
28     bool bfs(int s,int t)
29     {
30         h.assign(n,-1);
31         queue<int> que;
32         h[s]=0;
33         que.push(s);
34         while (!que.empty())
35         {
36             const int u=que.front();
37             que.pop();
38             for (int i:g[u])
39             {
40                 auto [v,c]=e[i];
41                 if (c>0&&h[v]==-1)
42                 {
43                     h[v]=h[u]+1;
44                     if (v==t) return 1;
45                     que.push(v);
46                 }
47             }
48         }
49         return 0;
50     }
51
52     T dfs(int u,int t,T f)
53     {
54         if (u==t) return f;
55         auto r=f;

```



```

56     for (int &i=cur[u]; i<int(g[u].size()); i++)
57     {
58         const int j=g[u][i];
59         auto [v,c]=e[j];
60         if (c>0&&h[v]==h[u]+1)
61         {
62             auto a=dfs(v,t,min(r,c));
63             e[j].cap-=a;
64             e[j^1].cap+=a;
65             r-=a;
66             if (r==0) return f;
67         }
68     }
69     return f-r;
70 }
71
72 void addEdge(int u,int v,T c)
73 {
74     g[u].push_back(e.size());
75     e.emplace_back(v,c);
76     g[v].push_back(e.size());
77     e.emplace_back(u,0);
78 }
79
80 T flow(int s,int t)
81 {
82     T ans=0;
83     while (bfs(s,t))
84     {
85         cur.assign(n,0);
86         ans+=dfs(s,t,numeric_limits<T>::max());
87     }
88     return ans;
89 }
90
91 vector<bool> minCut()
92 {
93     vector<bool> c(n);
94     for (int i=0; i<n; i++) c[i]=(h[i]!=-1);
95     return c;
96 }
97
98 struct Edge
99 {
100     int from;
101     int to;
102     T cap;
103     T flow;
104 };
105
106 vector<Edge> edges()
107 {
108     vector<Edge> a;
109     for (int i=0; i<e.size(); i+=2)
110     {
111         Edge x;
112         x.from=e[i+1].to;
113         x.to=e[i].to;
114         x.cap=e[i].cap+e[i+1].cap;
115         x.flow=e[i+1].cap;
116         a.push_back(x);
117     }
118     return a;
119 }
120 };

```

## 最小费用最大流

```

1  template <class T>
2  struct MinCostFlow
3  {

```

```

4 struct _Edge
5 {
6     int to;
7     T cap;
8     T cost;
9     _Edge(int to,T cap,T cost):to(to),cap(cap),cost(cost){}
10
11 };
12
13 int n;
14 vector<_Edge> e;
15 vector<vector<int>> g;
16 vector<T> h,dis;
17 vector<int> pre;
18
19 bool john(int s,int t)
20 {
21     dis.assign(n,numeric_limits<T>::max());
22     pre.assign(n,-1);
23     priority_queue<pair<T,int>,vector<pair<T,int>>,greater<pair<T,int>>> q;
24     dis[s]=0;
25     q.emplace(0,s);
26     while (!q.empty())
27     {
28         T d=q.top().first;
29         int u=q.top().second;
30         q.pop();
31         if (dis[u]!=d) continue;
32         for (int i:g[u])
33         {
34             int v=e[i].to;
35             T cap=e[i].cap;
36             T cost=e[i].cost;
37             if (cap>0&&dis[v]>d+h[u]-h[v]+cost)
38             {
39                 dis[v]=d+h[u]-h[v]+cost;
40                 pre[v]=i;
41                 q.emplace(dis[v],v);
42             }
43         }
44     }
45     return dis[t]!=numeric_limits<T>::max();
46 }
47
48 MinCostFlow(){}
49 MinCostFlow(int n) { init(n); }
50
51 void init(int n_)
52 {
53     n=n_;
54     e.clear();
55     g.assign(n,{});
56 }
57
58 void addEdge(int u,int v,T cap,T cost)
59 {
60     g[u].push_back(e.size());
61     e.emplace_back(v,cap,cost);
62     g[v].push_back(e.size());
63     e.emplace_back(u,0,-cost);
64 }
65
66 pair<T,T> flow(int s,int t)
67 {
68     T flow=0;
69     T cost=0;
70     h.assign(n,0);
71     while (john(s,t))
72     {
73         for (int i=0;i<n;i++) h[i]+=dis[i];
74         T aug=numeric_limits<int>::max();

```

```

75         for (int i=t;i!=s;i=e[pre[i]^1].to)
76             aug=min(aug,e[pre[i]].cap);
77         for (int i=t;i!=s;i=e[pre[i]^1].to)
78         {
79             e[pre[i]].cap-=aug;
80             e[pre[i]^1].cap+=aug;
81         }
82         flow+=aug;
83         cost+=aug*h[t];
84     }
85     return make_pair(flow,cost);
86 }
87
88 struct Edge
89 {
90     int from;
91     int to;
92     T cap;
93     T cost;
94     T flow;
95 };
96
97 vector<Edge> edges()
98 {
99     vector<Edge> a;
100     for (int i=0;i<e.size();i+=2)
101     {
102         Edge x;
103         x.from=e[i+1].to;
104         x.to=e[i].to;
105         x.cap=e[i].cap+e[i+1].cap;
106         x.cost=e[i].cost;
107         x.flow=e[i+1].cap;
108         a.push_back(x);
109     }
110     return a;
111 }
112 };

```

## 二分图最大权匹配 (KM)

```

1 //注意将负权边加上 inf, inf 不要设得过大
2 //xy 是左部点对应右部点
3 //yx 是右部点对应左部点
4 template <class T>
5 struct MaxAssignment
6 {
7     vector<T> lx,ly,s,cst;
8     vector<int> xy,yx,p,sx;
9     vector<bool> visx,visy;
10
11     T solve(int nx,int ny,vector<vector<T>> a)
12     {
13         assert(0<=nx&&nx<=ny);
14         assert(int(a.size())==nx);
15         for (int i=0;i<nx;i++)
16         {
17             assert(int(a[i].size())==ny);
18             for (auto x:a[i])
19                 assert(x>=0);
20         }
21         auto upd=[&](int x)->void
22         {
23             for (int y=0;y<ny;y++)
24             {
25                 if (lx[x]+ly[y]-a[x][y]<s[y])
26                 {
27                     s[y]=lx[x]+ly[y]-a[x][y];
28                     sx[y]=x;
29                 }
30             }

```

```

31     return;
32 };
33 cst.resize(nx+1);
34 cst[0]=0;
35 lx.assign(nx,numeric_limits<T>::max());
36 ly.assign(ny,0);
37 xy.assign(nx,-1);
38 yx.assign(ny,-1);
39 sx.resize(ny);
40 for (int cur=0;cur<nx;cur++)
41 {
42     queue<int> q;
43     visx.assign(nx,0);
44     visy.assign(ny,0);
45     s.assign(ny,numeric_limits<T>::max());
46     p.assign(nx,-1);
47     for (int x=0;x<nx;x++)
48     {
49         if (xy[x]==-1)
50         {
51             q.push(x);
52             visx[x]=1;
53             upd(x);
54         }
55     }
56     int ex,ey;
57     bool fl=0;
58     while (!fl)
59     {
60         while (!q.empty()&&!fl)
61         {
62             auto x=q.front();
63             q.pop();
64             for (int y=0;y<ny;y++)
65             {
66                 if (a[x][y]==lx[x]+ly[y]&&!visy[y])
67                 {
68                     if (yx[y]==-1)
69                     {
70                         ex=x;
71                         ey=y;
72                         fl=1;
73                         break;
74                     }
75                     q.push(yx[y]);
76                     p[yx[y]]=x;
77                     visy[y]=visx[yx[y]]=1;
78                     upd(yx[y]);
79                 }
80             }
81         }
82         if (fl) break;
83         T delta=numeric_limits<T>::max();
84         for (int y=0;y<ny;y++)
85             if (!visy[y])
86                 delta=min(delta,s[y]);
87         for (int x=0;x<nx;x++)
88             if (visx[x])
89                 lx[x]-=delta;
90         for (int y=0;y<ny;y++)
91         {
92             if (visy[y])
93                 ly[y]+=delta;
94             else
95                 s[y]-=delta;
96         }
97         for (int y=0;y<ny;y++)
98         {
99             if (!visy[y]&&s[y]==0)
100             {
101                 if (yx[y]==-1)

```

```

102         {
103             ex=sx[y];
104             ey=y;
105             fl=1;
106             break;
107         }
108         q.push(yx[y]);
109         p[yx[y]]=sx[y];
110         visy[y]=visx[yx[y]]=1;
111         upd(yx[y]);
112     }
113 }
114 }
115 cst[cur+1]=cst[cur];
116 for (int x=ex,y=ey,ty;x!=-1;x=p[x],y=ty)
117 {
118     cst[cur+1]+=a[x][y];
119     if (xy[x]!=-1)
120         cst[cur+1]-=a[x][xy[x]];
121     ty=xy[x];
122     xy[x]=y;
123     yx[y]=x;
124 }
125 }
126 return cst[nx];
127 }
128
129 vector<int> assignment() { return xy; }
130
131 pair<vector<T>,vector<T>> labels()
132 { return make_pair(lx,ly); }
133
134 vector<T> weights() { return cst; }
135 };

```

### 三元环计数

时间复杂度为  $\mathcal{O}(m\sqrt{m})$ 。

```

1  i64 triple(vector<pair<int,int>> &edges)
2  {
3      int n=0;
4      for (auto [u,v]:edges) n=max({n,u,v});
5      n++;
6      vector<int> d(n),id(n),rk(n),cnt(n);
7      vector<vector<int>> adj(n);
8      for (auto [u,v]:edges) d[u]++,d[v]++;
9      iota(id.begin(),id.end(),0);
10     sort(id.begin(),id.end(),[&](int x,int y)
11     {
12         return d[x]<d[y];
13     });
14     for (int i=0;i<n;i++) rk[id[i]]=i;
15     for (auto [u,v]:edges)
16     {
17         if (rk[u]>rk[v]) swap(u,v);
18         adj[u].push_back(v);
19     }
20     i64 res=0;
21     for (int i=0;i<n;i++)
22     {
23         for (int u:adj[i]) cnt[u]=1;
24         for (int u:adj[i])
25             for (int v:adj[u])
26                 res+=cnt[v];
27         for (int u:adj[i]) cnt[u]=0;
28     }
29     return res;
30 };

```

## 树哈希

有根树返回各子树 hash 值，无根树返回一个至多长为 2 的 vector。

```
1 vector<int> tree_hash(vector<vector<int>> &adj, int rt)
2 {
3     int n=adj.size();
4     static map<vector<int>, i64> mp;
5     static int id=0;
6     vector<int> h(n);
7
8     auto dfs=[&](auto &self, int u, int f)->void
9     {
10         vector<int> c;
11         for (int v:adj[u])
12             if (v!=f)
13             {
14                 self(self, v, u);
15                 c.push_back(h[v]);
16             }
17         sort(c.begin(), c.end());
18         if (!mp.count(c)) mp[c]=id++;
19         h[u]=mp[c];
20     };
21
22     dfs(dfs, rt, rt);
23     return h;
24 }
25
26 vector<int> tree_hash(vector<vector<int>> &adj)
27 {
28     int n=adj.size();
29     if (n==0) return {};
30     vector<int> siz(n), mx(n);
31
32     auto dfs=[&](auto &self, int u)->void
33     {
34         siz[u]=1;
35         for (int v:adj[u])
36             if (!siz[v])
37             {
38                 self(self, v);
39                 siz[u]+=siz[v];
40                 mx[u]=max(mx[u], siz[v]);
41             }
42         mx[u]=max(mx[u], n-siz[u]);
43     };
44
45     dfs(dfs, 0);
46     int m=*min_element(mx.begin(), mx.end());
47     vector<int> rt;
48     for (int i=0; i<n; i++)
49         if (mx[i]==m)
50             rt.push_back(i);
51     for (int &u:rt) u=tree_hash(adj, u)[u];
52     sort(rt.begin(), rt.end());
53     return rt;
54 }
```

## 矩阵树定理

记度矩阵为  $D$ ，邻接矩阵为  $A$ 。

对无向图情况： $L(G) = D(G) - A(G)$ 。

对有向图外向树情况： $L(G) = D^{in}(G) - A(G)$ 。

对有向图内向树情况： $L(G) = D^{out}(G) - A(G)$ 。

图  $G$  以  $r$  为根的生成树个数等于  $L(G)$  舍去第  $r$  行第  $r$  列的  $n-1$  阶主子式。

代码中  $t=0$  是无向图情况,  $t=1$  是有向图根为 1 的外向树情况。

```
1 void R()
2 {
3     int n,m,t;
4     cin>>n>>m>>t;
5     vector<vector<Z>> L(n-1,vector<Z>(n-1)),D(n,vector<Z>(n)),A(n,vector<Z>(n));
6     for (int i=1;i<=m;i++)
7     {
8         int u,v,w;
9         cin>>u>>v>>w;
10        if (u==v) continue;
11        u--,v--;
12        D[v][v]+=w;
13        A[u][v]+=w;
14        if (t==0)
15        {
16            D[u][u]+=w;
17            A[v][u]+=w;
18        }
19    }
20    for (int i=1;i<n;i++)
21        for (int j=1;j<n;j++)
22            L[i-1][j-1]=D[i][j]-A[i][j];
23    cout<<det(L);
24    return;
25 }
```

## 计算几何

### EPS

```
1 const double eps=1e-8;
2 int sgn(double x)
3 {
4     if (fabs(x)<eps) return 0;
5     if (x>0) return 1;
6     return -1;
7 }
```

### Point

```
1 template <class T>
2 struct Point
3 {
4     T x,y;
5     Point(T x_=0,T y_=0):x(x_),y(y_) {}
6
7     Point &operator += (Point p) &
8     {
9         x+=p.x;
10        y+=p.y;
11        return *this;
12    }
13
14     Point &operator -= (Point p) &
15     {
16        x-=p.x;
17        y-=p.y;
18        return *this;
19    }
20
21     Point &operator *= (T v) &
22     {
23        x*=v;
24        y*=v;
25        return *this;
26    }
27 }
```

```

28 Point operator - () const { return Point(-x,-y); }
29
30 friend Point operator + (Point a,Point b) { return a+=b; }
31 friend Point operator - (Point a,Point b) { return a-=b; }
32 friend Point operator * (Point a,T b) { return a*=b; }
33 friend Point operator * (T a,Point b) { return b*=a; }
34
35 friend bool operator == (Point a,Point b) { return a.x==b.x&& a.y==b.y; }
36
37 friend istream &operator >> (istream &is,Point &p) { return is>>p.x>>p.y; }
38
39 friend ostream &operator << (ostream &os,Point p) { return os<<'('<<p.x<<','<<p.y<<')'; }
40 };
41
42 template <class T>
43 int sgn(const Point<T> &a) { return a.y>0||(a.y==0&&a.x>0)?1:-1; }
44
45 template <class T>
46 T dot(Point<T> a,Point<T> b) { return a.x*b.x+a.y*b.y; }
47
48 template <class T>
49 T cross(Point<T> a,Point<T> b) { return a.x*b.y-a.y*b.x; }
50
51 template <class T>
52 T square(Point<T> p) { return dot(p,p); }
53
54 template <class T>
55 double length(Point<T> p) { return sqrt(double(square(p))); }
56
57 long double length(Point<long double> p) { return sqrt(square(p)); }

```

## Line

```

1 template <class T>
2 struct Line
3 {
4     Point<T> a,b;
5     Line(Point<T> a_=Point<T>(),Point<T> b_=Point<T>()):a(a_),b(b_) {}
6 };

```

## 距离

```

1 template <class T>
2 double dis_PP(Point<T> a,Point<T> b) { return length(a-b); }
3
4 template <class T>
5 double dis_PL(Point<T> a,Line<T> l) { return fabs(cross(a-l.a,a-l.b))/dis_PP(l.a,l.b); }
6
7 template <class T>
8 double dis_PS(Point<T> a,Line<T> l)
9 {
10     if (dot(a-l.a,l.b-l.a)<0) return dis_PP(a,l.a);
11     if (dot(a-l.b,l.a-l.b)<0) return dis_PP(a,l.b);
12     return dis_PL(a,l);
13 }

```

## 点绕中心旋转

```

1 template <class T>
2 Point<T> rotate(Point<T> a,double alpha)
3 { return Point<T>(a.x*cos(alpha)-a.y*sin(alpha),a.x*sin(alpha)+a.y*cos(alpha)); }

```

## 关于线的对称点

```

1 template <class T>
2 Point<T> lineRoot(Point<T> a,Line<T> l)
3 {
4     Point<T> v=l.b-l.a;
5     return l.a+v*(dot(a-l.a,v)/dot(v,v));

```



```

6 }
7
8 template <class T>
9 Point<T> symmetry_PL(Point<T> a,Line<T> l) { return a+(lineRoot(a,l)-a)*2; }

```

## 位置关系判断

```

1 template <class T>
2 bool pointOnSegment(Point<T> a,Line<T> l)
3 { return (sgn(cross(a-l.a,a-l.b))==0)&&(sgn(dot(a-l.a,a-l.b))<=0); }
4
5 template <class T>
6 bool lineCrossLine(Line<T> a,Line<T> b)
7 {
8     double f1=cross(b.a-a.a,a.b-a.a),f2=cross(b.b-a.a,a.b-a.a);
9     double g1=cross(a.a-b.a,b.b-b.a),g2=cross(a.b-b.a,b.b-b.a);
10    return ((f1<0)^(f2<0))&&((g1<0)^(g2<0));
11 }
12
13 template <class T>
14 bool pointOnLineLeft(Point<T> a,Line<T> l) { return cross(l.b-l.a,a-l.a)>0; }
15
16 //适用任意多边形, O(n)
17 template <class T>
18 bool pointInPolygon(Point<T> a,const vector<Point<T>> &p)
19 {
20     int n=p.size();
21     for (int i=0;i<n;i++)
22         if (pointOnSegment(a,Line<T>(p[i],p[(i+1)%n])))
23             return 1;
24     bool t=0;
25     for (int i=0;i<n;i++)
26     {
27         Point<T> u=p[i],v=p[(i+1)%n];
28         if (u.x<a.x&&v.x>=a.x&&pointOnLineLeft(a,Line<T>(v,u))) t^=1;
29         if (u.x>=a.x&&v.x<a.x&&pointOnLineLeft(a,Line<T>(u,v))) t^=1;
30     }
31     return t;
32 }
33
34 //适用凸多边形, O(log n)
35 template <class T>
36 bool pointInPolygon_(Point<T> a,const vector<Point<T>> &p)
37 {
38     int n=p.size();
39     if (cross(a-p[0],p[1]-p[0])<0||cross(a-p[0],p[n-1]-p[0])>0) return 0;
40     if (pointOnSegment(a,Line<T>(p[0],p[1]))||pointOnSegment(a,Line<T>(p[n-1],p[0]))) return 1;
41     int l=1,r=n-1;
42     while (l+1<r)
43     {
44         int mid=(l+r)>>1;
45         if (cross(a-p[l],p[mid]-p[l])<0) l=mid;
46         else r=mid;
47     }
48     if (cross(a-p[l],p[r]-p[l])>0) return 0;
49     if (pointOnSegment(a,Line<T>(p[l],p[r]))) return 1;
50     return 1;
51 }

```

## 线段交点

```

1 //小 心 平 行
2 template <class T>
3 Point<T> lineIntersection(Line<T> a,Line<T> b)
4 {
5     Point<T> u=a.b-a.a,v=a.b-a.a,w=b.b-b.a;
6     double t=cross(u,w)/cross(w,v);
7     return a.a+t*v;
8 }

```

## 过定点做圆的切线

```
1 template <class T>
2 vector<Line<T>> tan_PC(Point<T> a,Point<T> c,T r)
3 {
4     Point<T> v=c-a;
5     vector<Line<T>> res;
6     int dis=dis_PP(a,c);
7     if (sgn(dis-r)==0) res.push_back(rotate(v,acos(-1)/2));
8     else if (dis>r)
9     {
10         double alpha=asin(r/dis);
11         res.push_back(rotate(v,alpha));
12         res.push_back(rotate(v,-alpha));
13     }
14     return res;
15 }
```

## 两圆交点

```
1 template <class T>
2 vector<Point<T>> circleIntersection(Point<T> c1,T r1,Point<T> c2,T r2)
3 {
4     auto get=[&](Point<T> c,T r,double alpha)->Point<T>
5     { return Point<T>(c.x+cos(alpha)*r,c.y+sin(alpha)*r); };
6
7     auto angle=[&](Point<T> a)->double { return atan2(a.x,a.y); };
8
9     vector<Point<T>> res;
10    double d=dis_PP(c1,c2);
11    if (sgn(d)==0) return res;
12    if (sgn(r1+r2-d)<0) return res;
13    if (sgn(fabs(r1-r2)-d)>0) return res;
14    double alpha=angle(c2-c1);
15    double beta=acos((r1*r1-r2*r2+d*d)/(r1*d*2));
16    Point<T> p1=get(c1,r1,alpha-beta),p2=get(c1,r1,alpha+beta);
17    res.push_back(p1);
18    if (p1!=p2) res.push_back(p2);
19    return res;
20 }
```

## 多边形面积

```
1 template <class T>
2 double polygonArea(const vector<Point<T>> &p)
3 {
4     int n=p.size();
5     double res=0;
6     for (int i=1;i<n-1;i++) res+=cross(p[i]-p[0],p[i+1]-p[0]);
7     return fabs(res/2);
8 }
```

## 自适应辛普森法

```
1 //注意边界函数值不能小于 eps
2 double f(double x) { return pow(x,0.5); }
3 double calc(double l,double r)
4 {
5     double mid=(l+r)/2.0;
6     return (r-l)*(f(l)+f(r)+f(mid)*4.0)/6.0;
7 }
8 double simpson(double l,double r,double lst)
9 {
10    double mid=(l+r)/2.0;
11    double fl=calc(l,mid),fr=calc(mid,r);
12    if (sgn(fl+fr-lst)==0) return fl+fr;
13    else return simpson(l,mid,fl)+simpson(mid,r,fr);
14 }
```

## 静态凸包

```
1  template <class T>
2  vector<Point<T>> getHull(vector<Point<T>> p)
3  {
4      vector<Point<T>> h,l;
5      sort(p.begin(),p.end(),[&](auto a,auto b)
6      {
7          if (a.x!=b.x) return a.x<b.x;
8          else return a.y<b.y;
9      });
10     p.erase(unique(p.begin(),p.end()),p.end());
11     if (p.size()<=1) return p;
12     for (auto a:p)
13     {
14         while (h.size()>1&&sgn(cross(a-h.back(),a-h[h.size()-2]))<=0) h.pop_back();
15         while (l.size()>1&&sgn(cross(a-l.back(),a-l[l.size()-2]))>=0) l.pop_back();
16         l.push_back(a);
17         h.push_back(a);
18     }
19     l.pop_back();
20     reverse(h.begin(),h.end());
21     h.pop_back();
22     l.insert(l.end(),h.begin(),h.end());
23     return l;
24 }
```

## 旋转卡壳求直径

```
1  template <class T>
2  double getDiameter(vector<Point<T>> p)
3  {
4      double res=0;
5      if (p.size()==2) return dis_PP(p[0],p[1]);
6      int n=p.size();
7      p.push_back(p.front());
8      int j=2;
9      for (int i=0;i<n;i++)
10     {
11         while (sgn(cross(p[i+1]-p[i],p[j]-p[i])-cross(p[i+1]-p[i],p[j+1]-p[i]))<0)
12             j=(j+1)%n;
13         res=max({res,dis_PP(p[i],p[j]),dis_PP(p[i+1],p[j])});
14     }
15     return res;
16 }
```

## 半平面交

```
1  template <class T>
2  vector<Point<T>> hp(vector<Line<T>> lines)
3  {
4      sort(lines.begin(),lines.end(),[&](auto l1,auto l2)
5      {
6          auto d1=l1.b-l1.a;
7          auto d2=l2.b-l2.a;
8
9          if (sgn(d1)!=sgn(d2)) return sgn(d1)==1;
10         return cross(d1,d2)>0;
11     });
12
13     deque<Line<T>> ls;
14     deque<Point<T>> ps;
15     for (auto l:lines)
16     {
17         if (ls.empty())
18         {
19             ls.push_back(l);
20             continue;
21         }
22         while (!ps.empty()&&!pointOnLineLeft(ps.back(),l))
```

```

23     {
24         ps.pop_back();
25         ls.pop_back();
26     }
27     while (!ps.empty() && !pointOnLineLeft(ps[0], l))
28     {
29         ps.pop_front();
30         ls.pop_front();
31     }
32     if (cross(l.b-l.a, ls.back().b-ls.back().a)==0)
33     {
34         if (dot(l.b-l.a, ls.back().b-ls.back().a)>0)
35         {
36             if (!pointOnLineLeft(ls.back().a, l))
37             {
38                 assert(ls.size()==1);
39                 ls[0]=l;
40             }
41             continue;
42         }
43         return {};
44     }
45     ps.push_back(lineIntersection(ls.back(), l));
46     ls.push_back(l);
47 }
48 while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0]))
49 {
50     ps.pop_back();
51     ls.pop_back();
52 }
53 if (ls.size()<=2) return {};
54 ps.push_back(lineIntersection(ls[0], ls.back()));
55 return vector(ps.begin(), ps.end());
56 }

```