

# Algorithm Library

CRatiQ

South China Normal University

June 1, 2025

# Contents

<b>数学</b>	<b>3</b>
Set Xor-Min . . . . .	3
<b>数据结构</b>	<b>4</b>
半群 deque . . . . .	4
区间众数 . . . . .	5
李超树 . . . . .	6
Splay . . . . .	7
<b>字符串</b>	<b>11</b>
字符串哈希 (随机模数) . . . . .	11
允许 $k$ 次失配的字符串匹配 . . . . .	11
最长公共子串 . . . . .	11
Code . . . . .	11
KMP . . . . .	11
字符串周期 . . . . .	11
统计前缀出现次数 . . . . .	11
求满足一些要求的 Border . . . . .	12
Code . . . . .	12
Z 函数 . . . . .	12
AC 自动机 . . . . .	12
后缀数组 . . . . .	13
(广义) 后缀自动机 . . . . .	14
不同子串个数 . . . . .	14
字典序第 $k$ 大子串 . . . . .	14
最小循环移位 . . . . .	14
出现次数 . . . . .	14
首次出现位置 . . . . .	14
所有出现位置 . . . . .	14
最短未出现字符串 . . . . .	14
最长公共子串 . . . . .	14
Code . . . . .	15
Manacher . . . . .	16
回文自动机 . . . . .	16
本质不同回文子串数 . . . . .	16
回文子串出现次数 . . . . .	16
Code . . . . .	16
含通配符字符串匹配 . . . . .	17
<b>图论</b>	<b>18</b>
拓扑排序 . . . . .	18
树的直径 . . . . .	18
动态树直径 (CF1192B) . . . . .	19
树的重心 . . . . .	20
Dijkstra . . . . .	20
SPFA . . . . .	21
Johnson . . . . .	21
强连通分量 . . . . .	22
边双连通分量 . . . . .	23
轻重链剖分 . . . . .	24
虚树 . . . . .	26
欧拉路径 . . . . .	27
2-SAT . . . . .	27
最大流 . . . . .	28
最小费用最大流 . . . . .	30
二分图最大权匹配 (KM) . . . . .	32

三元环计数 . . . . .	34
树哈希 . . . . .	34
矩阵树定理 . . . . .	35
<b>计算几何</b>	<b>36</b>
EPS . . . . .	36
Point . . . . .	36
Line . . . . .	37
距离 . . . . .	37
点绕中心旋转 . . . . .	37
关于线的对称点 . . . . .	37
位置关系判断 . . . . .	37
线段交点 . . . . .	38
过定点做圆的切线 . . . . .	38
两圆交点 . . . . .	38
多边形面积 . . . . .	39
自适应辛普森法 . . . . .	39
静态凸包 . . . . .	39
旋转卡壳求直径 . . . . .	40
半平面交 . . . . .	40
最小圆覆盖 . . . . .	41

# 数学

## Set Xor-Min

维护一个集合  $S$ , 可以求  $\min_{y \in S} (x \oplus y)$ 。

```
1  struct SetXorMin
2  {
3      static constexpr int L=30;
4      int tot=0;
5      vector<array<int,2>> c;
6      vector<int> s;
7      set<i64> in;
8
9      SetXorMin() {}
10     SetXorMin(int n)
11     {
12         c.resize((n+1)*(L+1));
13         s.resize((n+1)*(L+1));
14     }
15
16     void insert(i64 x)
17     {
18         if (in.count(x))
19             return;
20         in.insert(x);
21         int p=0;
22         for (int i=L;i>=0;i--)
23         {
24             bool o=x>>i&1;
25             if (!c[p][o])
26                 c[p][o]=++tot;
27             s[p=c[p][o]]++;
28         }
29     }
30
31     void erase(i64 x)
32     {
33         if (!in.count(x))
34             return;
35         in.erase(x);
36         int p=0;
37         for (int i=L;i>=0;i--)
38         {
39             bool o=x>>i&1;
40             s[p=c[p][o]]--;
41         }
42     }
43
44     i64 QueryXorMin(i64 x)
45     {
46         int p=0;
47         i64 r=0;
48         for (int i=L;i>=0;i--)
49         {
50             bool o=x>>i&1;
51             if (s[c[p][o]])
52                 p=c[p][o];
53             else
54             {
55                 p=c[p][o^1];
56                 r|=1ll<<i;
57             }
58         }
59         return r;
60     }
61 };
```

# 数据结构

## 半群 deque

维护一个半群的 deque，支持前后增删及求和。

```
1  template <class T>
2  struct SWAG
3  {
4      vector<T> l,sl,r,sr;
5
6      void push_front(const T &o)
7      {
8          sl.push_back(sl.empty()?o:o+sl.back());
9          l.push_back(o);
10     }
11
12     void push_back(const T &o)
13     {
14         sr.push_back(sr.empty()?o:sr.back()+o);
15         r.push_back(o);
16     }
17
18     void pop_front()
19     {
20         if (!l.empty())
21         {
22             l.pop_back();
23             sl.pop_back();
24             return;
25         }
26         int n=r.size(),m;
27         if (m=n-1>>1)
28         {
29             l.resize(m);
30             sl.resize(m);
31             for (int i=1;i<=m;i++)
32                 l[m-i]=r[i];
33             sl[0]=l[0];
34             for (int i=1;i<m;i++)
35                 sl[i]=l[i]+sl[i-1];
36         }
37         for (int i=m+1;i<n;i++)
38             r[i-(m+1)]=r[i];
39         m=n-(m+1);
40         r.resize(m);
41         sr.resize(m);
42         if (m)
43         {
44             sr[0]=r[0];
45             for (int i=1;i<m;i++)
46                 sr[i]=sr[i-1]+r[i];
47         }
48     }
49
50     void pop_back()
51     {
52         if (!r.empty())
53         {
54             r.pop_back();
55             sr.pop_back();
56         }
57         else
58         {
59             int n=l.size(),m;
60             if (m=n-1>>1)
61             {
62                 r.resize(m);
63                 sr.resize(m);
64                 for (int i=1;i<=m;i++)
65                     r[m-i]=l[i];
```

```

66         sr[0]=r[0];
67         for (int i=1;i<m;i++)
68             sr[i]=sr[i-1]+r[i];
69     }
70     for (int i=m+1;i<n;i++)
71         l[i-(m+1)]=l[i];
72     m=n-(m+1);
73     l.resize(m);
74     sl.resize(m);
75     if (m)
76     {
77         sl[0]=l[0];
78         for (int i=1;i<m;i++)
79             sl[i]=l[i]+sl[i-1];
80     }
81 }
82 }
83
84 T ask()
85 {
86     assert(l.size()||r.size());
87     if (l.size()&&r.size())
88         return sl.back()+sr.back();
89     return l.size()?sl.back():sr.back();
90 }
91 };
92
93 struct Info
94 {
95     Z k,b;
96
97     Info operator + (const Info &o) const
98     {
99         return {k*o.k,b*o.k+o.b};
100     }
101 };
102
103 Z operator + (const Z &x,const Info &o)
104 {
105     return o.k*x+o.b;
106 }

```

## 区间众数

```

1  template <class T>
2  struct Mode
3  {
4      int n,ksz,m;
5      vector<T> b;
6      vector<vector<int>> pos,f;
7      vector<int> a,blk,id,l;
8
9      Mode(const vector<T> &c):n(c.size()),ksz(max<int>(1,sqrt(n))),
10         m((n+ksz-1)/ksz),b(c),pos(n),f(m,vector<int>(m)),a(n),blk(n),id(n),l(m+1)
11  {
12      sort(b.begin(),b.end());
13      b.erase(unique(b.begin(),b.end()),b.end());
14      for (int i=0;i<n;i++)
15      {
16          a[i]=lower_bound(b.begin(),b.end(),c[i])-b.begin();
17          id[i]=pos[a[i]].size();
18          pos[a[i]].push_back(i);
19      }
20      for (int i=0;i<n;i++)
21          blk[i]=i/ksz;
22      for (int i=0;i<=m;i++)
23          l[i]=min(i*ksz,n);
24
25      vector<int> cnt(b.size());
26      for (int i=0;i<m;i++)
27      {

```

```

28         cnt.assign(b.size(),0);
29         pair<int,int> cur={0,0};
30         for (int j=i;j<m;j++)
31         {
32             for (int k=l[j];k<l[j+1];k++)
33                 cur=max(cur,{++cnt[a[k]],a[k]});
34             f[i][j]=cur.second;
35         }
36     }
37 }
38
39 pair<T,int> ask(int L,int R)
40 {
41     int val=blk[L]==blk[R-1]?0:f[blk[L]+1][blk[R-1]-1],i;
42     int cnt=lower_bound(pos[val].begin(),pos[val].end(),R)-
43         lower_bound(pos[val].begin(),pos[val].end(),L);
44     for (int i=min(R,l[blk[L]+1])-1;i>=L;i--)
45     {
46         auto &v=pos[a[i]];
47         while (id[i]+cnt<v.size()&&v[id[i]+cnt]<R)
48             cnt++,val=a[i];
49         if (a[i]>val&&id[i]+cnt-1<v.size()&&v[id[i]+cnt-1]<R)
50             val=a[i];
51     }
52     for (int i=max(L,l[blk[R-1]]);i<R;i++)
53     {
54         auto &v=pos[a[i]];
55         while (id[i]>=cnt&&v[id[i]-cnt]>=L)
56             cnt++,val=a[i];
57         if (a[i]>val&&id[i]>=cnt-1&&v[id[i]-cnt+1]>=L)
58             val=a[i];
59     }
60     return {b[val],cnt};
61 }
62 };

```

## 李超树

```

1  constexpr i64 inf=9e18;
2
3  template <class Info>
4  struct SGT
5  {
6      int cnt=0;
7      vector<Info> a;
8      vector<int> ls,rs;
9      i64 z,y,L,R;
10
11      SGT(int n,i64 l,i64 r)
12      {
13          int N=(n+7)*64;
14          a.resize(N);
15          ls.resize(N);
16          rs.resize(N);
17          L=l,R=r,cnt=1;
18          a[1]={0,inf};
19      }
20
21  private:
22      void insert(int &p,i64 l,i64 r,Info v)
23      {
24          if (!p)
25          {
26              p=++cnt;
27              a[p]={0,inf};
28          }
29          i64 m=(l+r)>>1;
30          if (z<=l&&r<=y)
31          {
32              if (a[p].y(m)>v.y(m)) swap(a[p],v);
33              if (a[p].y(l)>v.y(l)) insert(ls[p],l,m,v);

```

```

34         else if (a[p].y(r)>v.y(r)) insert(rs[p],m+1,r,v);
35         return;
36     }
37     if (z<=m) insert(ls[p],l,m,v);
38     if (y>m) insert(rs[p],m+1,r,v);
39 }
40 public:
41     void insert(i64 l,i64 r,const Info &v)
42     {
43         z=l,y=r;
44         int p=1;
45         insert(p,L,R,v);
46     }
47
48     i64 QueryMin(i64 p)
49     {
50         i64 res=a[1].y(p),l=L,r=R,x=1;
51         while (l<r)
52         {
53             i64 m=(l+r)>>1;
54             if (p<=m)
55                 x=ls[x],r=m;
56             else
57                 x=rs[x],l=m+1;
58             if (!x) return res;
59             res=min(res,a[x].y(p));
60         }
61         return res;
62     }
63 };
64
65 struct Info
66 {
67     i64 k,b;
68
69     i64 y(const i64 &x) const { return k*x+b; }
70 };

```

## Splay

```

1  template <class Info,class Tag>
2  struct Splay
3  {
4      #define _rev
5      struct Node
6      {
7          Node *c[2],*f;
8          int siz;
9          Info s,v;
10         Tag t;
11
12         Node():c{},f(0),siz(1),s(),v(),t() {}
13         Node(Info x):c{},f(0),siz(1),s(x),v(x),t() {}
14
15         void operator += (const Tag &o)
16         {
17             s+=o,v+=o,t+=o;
18         }
19         #ifdef _rev
20         void swap() { swap(c[0],c[1]); }
21         #endif
22
23         void pushup()
24         {
25             if (c[0])
26                 s=c[0]->s+v,siz=c[0]->siz+1;
27             else s=v,siz=1;
28             if (c[1])
29                 s=s+c[1]->s,siz+=c[1]->siz;
30         }
31     }

```



```

32     void pushdown()
33     {
34         for (auto x:c)
35             if (x)
36                 *x+=t;
37         t=Tag();
38     }
39
40     void zigzag()
41     {
42         Node *y=f,*z=y->f;
43         bool isl=y->c[0]==this;
44         if (z) z->c[z->c[1]==y]=this;
45         f=z,y->f=this;
46         y->c[isl^1]=c[isl];
47         if (c[isl]) c[isl]->f=y;
48         c[isl]=y;
49         y->pushup();
50     }
51
52     //only used for makeroor
53     void splay(Node *tg)
54     {
55         for (Node *y=f;y!=tg;zigzag(),y=f)
56             if (Node *z=y->f;z!=tg)
57                 (z->c[1]==y^y->c[1]==this?this:y)->zigzag();
58         pushup();
59     }
60
61     void clear()
62     {
63         for (Node *x:c)
64             if (x)
65                 x->clear();
66         delete this;
67     }
68 };
69
70 Node *rt;
71 int shift;
72
73 Splay()
74 {
75     rt=new Node;
76     rt->c[1]=new Node;
77     rt->c[1]->f=rt;
78     rt->siz=2;
79 }
80
81 Splay(vector<Info> &a,int l,int r)
82 {
83     shift=l-1;
84     rt=new Node;
85     rt->c[1]=new Node;
86     rt->c[1]->f=rt;
87     if (l<r)
88     {
89         rt->c[1]->c[0]=build(a,l,r);
90         rt->c[1]->c[0]->f=rt->c[1];
91     }
92     rt->c[1]->pushup();
93     rt->pushup();
94 }
95
96 Node *build(vector<Info> &a,int l,int r)
97 {
98     if (l==r) return 0;
99     int m=(l+r)>>1;
100     Node *x=new Node(a[m]);
101     x->c[0]=build(a,l,m);
102     x->c[1]=build(a,m+1,r);

```

```

103     for (Node *y:x->c)
104         if (y) y->f=x;
105     x->pushup();
106     return x;
107 }
108
109 void makeroot(Node *u,Node *tg)
110 {
111     if (!tg) rt=u;
112     u->splay();
113 }
114
115 void findKth(int k,Node *tg)
116 {
117     Node *x=rt;
118     while (1)
119     {
120         x->pushdown();
121         int res=x->c[0]?x->c[0]->siz:0;
122         if (res+1==k)
123         {
124             x->splay(tg);
125             if (!tg) rt=x;
126             return;
127         }
128         if (res>=k) x=x->c[0];
129         else x=x->c[1],k-=res+1;
130     }
131 }
132
133 void split(int l,int r)
134 {
135     findKth(l,0);
136     findKth(r+2,rt);
137 }
138
139 #ifdef _rev
140 void reverse(int l,int r)
141 {
142     l-=shift;
143     r-=shift+1;
144     if (l>r) return;
145     split(l,r);
146     *(rt->c[1]->c[0])+=Tag(1);
147 }
148 #endif
149
150 //insert before pos
151 void insert(int pos,Info x)
152 {
153     pos-=shift;
154     split(pos,pos-1);
155     rt->c[1]->c[0]=new Node(x);
156     rt->c[1]->c[0]->f=rt->c[1];
157     rt->c[1]->pushup();
158     rt->pushup();
159 }
160
161 void insert(int pos,vector<Info> &a,int l,int r)
162 {
163     pos-=shift;
164     split(pos,pos-1);
165     rt->c[1]->c[0]=build(a,l,r);
166     rt->c[1]->c[0]->f=rt->c[1];
167     rt->c[1]->pushup();
168     rt->pushup();
169 }
170
171 void erase(int pos)
172 {
173     pos-=shift;

```

```

174     split(pos,pos);
175     delete rt->c[1]->c[0];
176     rt->c[1]->c[0]=0;
177     rt->c[1]->pushup();
178     rt->pushup();
179 }
180
181 void erase(int l,int r)
182 {
183     l-=shift,r-=shift+1;
184     if (l>r) return;
185     split(l,r);
186     rt->c[1]->c[0]->clear();
187     rt->c[1]->c[0]=0;
188     rt->c[1]->pushup();
189     rt->pushup();
190 }
191
192 void modify(int pos,Info x)
193 {
194     pos-=shift;
195     findKth(pos+1,0);
196     rt->v=x;
197     rt->pushup();
198 }
199
200 void rangeApply(int l,int r,Tag w)
201 {
202     l-=shift,r-=shift+1;
203     if (l>r) return;
204     split(l,r);
205     Node *x=rt->c[1]->c[0];
206     *x+=w;
207     rt->c[1]->pushup();
208     rt->pushup();
209 }
210
211 Info rangeQuery(int l,int r)
212 {
213     l-=shift,r-=shift+1;
214     split(l,r);
215     return rt->c[1]->c[0]->s;
216 }
217
218 ~Splay() { rt->clear(); }
219 #undef _rev
220 };
221
222 struct Tag
223 {
224     bool rev=0;
225
226     Tag() {}
227     Tag(bool c):rev(c) {}
228
229     void operator += (const Tag &o)
230     {
231         rev^=o.rev;
232     }
233 };
234
235 struct Info
236 {
237     i64 x=0;
238
239     void operator += (const Tag &o) const
240     {
241
242     }
243
244     Info operator + (const Info &o) const

```

```

245     {
246         return {x+o.x};
247     }
248 };

```

## 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 }

```

## 动态树直径 (CF1192B)

指支持动态修改树边的权值, 复杂度为  $\mathcal{O}(\log n)$ 。

代码 d,e->D,E 那段是题目强制在线的解密。

```

1  struct Tag
2  {
3      i64 dt=0;
4      void apply(Tag t)
5      {
6          dt+=t.dt;
7      }
8  };
9
10 struct Info
11 {
12     i64 ans=0,mx=0,mn=1e18,lm=0,rm=0;
13     void apply(Tag t)
14     {
15         mx+=t.dt;
16         mn+=t.dt;
17         lm-=t.dt;
18         rm-=t.dt;
19     }
20 };
21
22 Info operator + (Info a,Info b)
23 {
24     Info c;
25     c.ans=max({a.ans,b.ans,a.rm+b.mx,a.mx+b.lm});
26     c.mx=max(a.mx,b.mx);
27     c.mn=min(a.mn,b.mn);
28     c.lm=max({a.lm,b.lm,b.mx-2*a.mn});
29     c.rm=max({a.rm,b.rm,a.mx-2*b.mn});
30     return c;
31 }
32
33 void R()
34 {
35     i64 n,q,w;
36     cin>>n>>q>>w;
37     vector<int> in(n),out(n),ord;
38     vector<i64> dep(n,-1);
39     vector<array<i64,3>> edges(n-1);
40     vector<vector<array<i64,2>>> adj(n);
41     for (int i=1;i<n;i++)
42     {
43         i64 a,b,c;
44         cin>>a>>b>>c;
45         a--,b--;
46         edges[i-1]={a,b,c};
47         adj[a].push_back({b,c});
48         adj[b].push_back({a,c});
49     }
50
51     auto dfs=[&](auto &self,int u)->void
52     {
53         in[u]=out[u]=ord.size();
54         ord.push_back(u);
55         for (auto [v,w]:adj[u])
56         {
57             if (dep[v]!=-1) continue;
58             dep[v]=dep[u]+w;
59             self(self,v);
60             out[u]=ord.size();
61             ord.push_back(u);
62         }

```

```

63     };
64
65     dep[0]=0;
66     dfs(dfs,0);
67
68     SGT<Info,Tag> sgt(ord.size());
69     for (int i=0;i<ord.size();i++)
70         sgt.modify(i,{0ll,dep[ord[i]],dep[ord[i]],-dep[ord[i]],-dep[ord[i]]});
71
72     i64 las=0;
73     for (int i=0;i<q;i++)
74     {
75         i64 d,e,D,E;
76         cin>>d>>e;
77         D=(d+las)%(n-1);
78         E=(e+las)%w;
79         auto &[x,y,w]=edges[D];
80         if (in[x]>in[y]) swap(x,y);
81         sgt.rangeApply(in[y],out[y]+1,{E-w});
82         w=E;
83         cout<<(las=sgt.rangeQuery(0,ord.size()).ans)<<"\n";
84     }
85     return;
86 }

```

## 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,cnt;
61 };
62
63 Graph compress()
64 {
65     Graph G;
66     G.n=cnt;
67     G.siz.resize(cnt);
68     G.cnt.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.cnt.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.cnt[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 };

```

## 虚树

```

1  struct VirtualTree
2  {
3      int n,rt;
4      HLD hld;
5      vector<int> a;
6      vector<bool> is;
7      vector<vector<int>> son;
8
9      VirtualTree(){}
10     VirtualTree(int n) { init(n); }
11
12     void init(int n)
13     {
14         this->n=n;
15         hld.init(n);
16         is.assign(n,0);
17         son.assign(n,{});
18     }
19
20     void addEdge(int u,int v)
21     {
22         hld.addEdge(u,v);
23     }
24
25     void work(int rt=0)
26     {
27         this->rt=rt;
28         hld.work(rt);
29     }
30
31     void solve(vector<int> &in)
32     {
33         auto cmp=[&](int x,int y)->bool
34         {
35             return hld.in[x]<hld.in[y];
36         };
37
38         for (int x:a)
39         {
40             is[x]=0;
41             son[x].clear();
42         }
43         a=in;
44         for (int x:a) is[x]=1;
45         a.push_back(rt);

```

```

46         sort(a.begin(),a.end(),cmp);
47
48         int k=a.size();
49         for (int i=1;i<k;i++)
50             a.push_back(hld.lca(a[i-1],a[i]));
51         sort(a.begin(),a.end(),cmp);
52         a.erase(unique(a.begin(),a.end()),a.end());
53         for (int i=1;i<a.size();i++)
54             son[hld.lca(a[i-1],a[i])].push_back(a[i]);
55     };
56
57     bool isKey(int u)
58     {
59         return is[u];
60     }
61
62     vector<int>& operator [] (int u)
63     {
64         return son[u];
65     }
66 };

```

## 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)

时间复杂度为  $O(n^3)$ 。

```

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.a-b.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[j]))<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 }

```

## 最小圆覆盖

期望时间复杂度为  $\mathcal{O}(n)$ 。

```

1  using Real=long double;
2
3  //only for 3*3
4  Real det(vector<vector<Real>> a)
5  {
6      Real res=0;
7      for (int i=0; i<3; i++)
8      {
9          Real tmp=1;
10         for (int j=0; j<3; j++)
11             tmp*=a[j][(i+j)%3];
12         res+=tmp;
13     }
14     for (int i=0; i<3; i++)
15     {
16         Real tmp=1;
17         for (int j=0; j<3; j++)
18             tmp*=a[j][(i+j*2)%3];
19         res-=tmp;
20     }
21     return res;
22 }
23
24 mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());
25
26 tuple<Point<Real>, Real> Coverage(vector<Point<Real>> p)
27 {
28     int n=p.size();
29     shuffle(p.begin(), p.end(), rnd);
30     Point<Real> C=p[0];
31     Real r=0;
32     for (int i=0; i<n; i++)
33         if (dis_PP(C, p[i]) > r)
34         {
35             C=p[i], r=0;
36             for (int j=0; j<i; j++)
37                 if (dis_PP(C, p[j]) > r)
38                 {
39                     C=(p[i]+p[j])*0.5;
40                     r=dis_PP(p[i], p[j])*0.5;
41                     for (int k=0; k<j; k++)
42                         if (dis_PP(C, p[k]) > r)
43                         {
44                             array<Real, 3> x, y;
45                             x[0]=p[i].x, y[0]=p[i].y;
46                             x[1]=p[j].x, y[1]=p[j].y;
47                             x[2]=p[k].x, y[2]=p[k].y;
48                             vector<vector<Real>> a(3, vector<Real>(3)), b(a), c(a);
49                             for (int t=0; t<3; t++)
50                             {
51                                 a[t][0]=b[t][0]=x[t]*x[t]+y[t]*y[t];
52                                 c[t][0]=b[t][1]=x[t];
53                                 a[t][1]=c[t][1]=y[t];
54                                 a[t][2]=b[t][2]=c[t][2]=1;
55                             }
56                             Real px=det(a)/det(c)/2.0, py=-det(b)/det(c)/2.0;
57                             C={px, py};
58                             r=dis_PP(C, p[i]);
59                         }
60 }

```

```
61     }  
62     return {C,r};  
63 }
```