

ACM-ICPC Template

tokitsukaze

2021 年 10 月 16 日



目录	
1 字符串	1
1.1 KMP	1
1.1.1 KMP	1
1.1.2 exKMP	1
1.2 Hash	1
1.2.1 hash	1
1.2.2 hash_map	2
1.2.3 BKDRHash	2
1.3 Manacher	2
1.3.1 插字符	2
1.3.2 不插字符	3
1.4 后缀数组	3
1.4.1 倍增 sa	3
1.4.2 SA-IS	4
1.5 自动机	5
1.5.1 AC 自动机	5
1.5.2 大字符集 AC 自动机	5
1.5.3 后缀自动机	6
1.5.4 回文自动机	7
1.5.5 序列自动机	8
1.6 最小表示法	8
1.6.1 最小表示法	8
1.6.2 最大表示法	8
1.7 shift_and	8
2 数据结构	9
2.1 RMQ	9
2.1.1 一维 RMQ	9
2.1.2 二维 RMQ	9
2.2 单调队列	10
2.3 LCA	10
2.3.1 倍增 LCA	10
2.3.2 RMQ 维护欧拉序求 LCA	11
2.4 轻重链剖分	12
2.5 并查集	13
2.5.1 并查集	13
2.5.2 map 实现并查集	14
2.5.3 可撤销并查集	14
2.6 树状数组	14
2.6.1 一维单点 BIT	14
2.6.2 一维区间 BIT	15
2.6.3 二维单点 BIT	15
2.7 线段树	16
2.7.1 线段树	16
2.7.2 动态开点线段树	16
2.7.3 线段树套线段树	17
2.7.4 线段树分裂合并	18
2.8 主席树	19
2.9 李超树	20
2.10 平衡树	21
2.10.1 Treap	21
2.11 字典树	22
2.11.1 trie	22
2.11.2 01trie	23
2.12 kd-tree	23
2.13 虚树	26
2.14 pbds 可并堆	26
2.15 k 叉哈夫曼树	26
2.16 笛卡尔树	27
2.17 析合树	27
3 图论	29
3.1 链式前向星	29
3.2 最短路	29
3.2.1 dijkstra	29
3.2.2 spfa	29
3.2.3 floyd 求最小环	30
3.3 最小生成树	30
3.3.1 kruskal	30
3.3.2 prim	31
3.4 二分图匹配	31
3.4.1 匈牙利算法	31
3.4.2 二分图带权匹配	32
3.5 最大流	33
3.5.1 dinic	33
3.5.2 ISAP	34
3.5.3 high-level-preflow-push	35
3.6 最小费用最大流	36
3.6.1 spfa 费用流	36
3.6.2 dijkstra 费用流	37
3.7 强连通分量	38
3.8 双联通分量	39
3.8.1 边双连通	39
3.9 团	40
3.9.1 最大团	40
3.9.2 极大团计数	40
3.10 拓扑排序	41
3.11 2-sat	41
3.11.1 2-sat 输出任意解	41
3.11.2 2-sat 字典序最小解	42
3.12 支配树	43
4 数论	44
4.1 素数筛	44
4.1.1 埃筛	44
4.1.2 线性筛	44
4.1.3 区间筛	44
4.2 扩展欧几里得	44

4.2.1	exgcd	44	9.4	xor_sum(1,n)	61
4.2.2	ax+by=c	45	9.5	约瑟夫环 kth	61
4.2.3	exgcd 求逆元	45	9.6	判断星期几	62
4.3	中国剩余定理	45	9.7	离散化	62
4.3.1	CRT	45	9.8	网格整数点正方形个数	62
4.3.2	exCRT	45	9.9	模拟退火	62
4.4	组合数	46	9.10	矩形面积并	63
4.4.1	打表	46	9.11	维护不同颜色最值和次值	64
4.4.2	预处理	46			
4.4.3	Lucas 定理	46	10 附录		66
4.4.4	exLucas	46	10.1	NTT 常用模数	66
4.5	欧拉函数	47	10.2	树 hash	66
4.5.1	直接求	47	10.3	线性基求交	67
4.5.2	线性筛	47			
4.6	莫比乌斯函数	48			
4.7	Berlekamp-Massey	48			
4.8	exBSGS	49			
4.9	Miller_Rabin+Pollard_rho	49			
4.10	第二类 Stirling 数	50			
4.11	原根	50			
4.12	二次剩余	51			
5	多项式	51			
5.1	FFT	51			
5.2	NTT	52			
5.3	FWT	53			
5.4	拉格朗日插值	54			
6	矩阵	54			
6.1	矩阵类	54			
6.2	高斯消元	55			
6.2.1	同余方程	55			
6.2.2	同余方程 mod=2	56			
6.3	单纯形	56			
6.4	线性基	57			
7	博弈	58			
7.1	SG 函数	58			
7.1.1	sg 表	58			
7.1.2	记忆化搜索求 sg 函数	58			
7.2	结论	59			
8	dp	59			
8.1	LIS	59			
8.2	LPS	59			
8.3	数位 dp	59			
9	杂项	60			
9.1	FastIO	60			
9.2	O(1) 快速乘	61			
9.3	快速模	61			

1 字符串

1.1 KMP

1.1.1 KMP

```

1 //length of min loop len-nex[len]
2 struct KMP
3 {
4     int nex[MAX],len;
5     char s[MAX];
6     void get_next()
7     {
8         int i,j;
9         i=0;
10        j=nex[0]=-1;
11        while(i<len)
12        {
13            if(j==-1||s[i]==s[j]) nex[++i]=++j;
14            else j=nex[j];
15        }
16    }
17    void init(char *_s)
18    {
19        len=strlen(_s);
20        for(int i=0;i<len;i++) s[i]=_s[i];
21        s[len]='\0';
22        get_next();
23    }
24    int match(char *a)//s is a substring of a
25    {
26        int n,i,j;
27        n=strlen(a);
28        for(i=j=0;i<n;i++)
29        {
30            if(j==-1||a[i]==s[j]) j++;
31            else j=nex[j];
32            if(j==len) return 1;
33        }
34        return 0;
35    }
36 }kmp;// kmp.init(s); s[0..len-1]
```

1.1.2 exKMP

```

1 struct exKMP
2 {
3     int next[MAX];
4     void getnext(char *s)
5     {
6         int i,j,pos,len;
7         next[i=0]=len=strlen(s);
8         while(s[i]==s[i+1]&&i+1<len) i++;
9         next[1]=i;
```

```

10        pos=1;
11        for(i=2;i<len;i++)
12        {
13            if(next[i-pos]+i<next[pos]+pos) next[i]=
14                next[i-pos];
15            else
16            {
17                j=max(0,next[pos]+pos-i);
18                while(i+j<len&&s[j]==s[j+i]) j++;
19                next[i]=j;
20                pos=i;
21            }
22        }
23    void work(char *a,char *b,int *ex)// a,b 0~len-1
24    {
25        int i=0,j,pos,lena,lenb;
26        getnext(b);
27        lena=strlen(a);
28        lenb=strlen(b);
29        i=0;
30        while(a[i]==b[i]&&i<lenb&&i<lena) i++;
31        ex[0]=i;
32        pos=0;
33        for(i=1;i<lena;i++)
34        {
35            if(next[i-pos]+i<ex[pos]+pos) ex[i]=next[
36                i-pos];
37            else
38            {
39                j=max(0,ex[pos]+pos-i);
40                while(i+j<lena&&j<lenb&&a[j+i]==b[j])
41                    j++;
42                ex[i]=j;
43                pos=i;
44            }
45        }
46    }exkmp;
```

1.2 Hash

1.2.1 hash

```

1 struct hash_table
2 {
3     ll seed,p;
4     ll Hash[MAX],tmp[MAX];
5     void set(ll _seed,ll _p)
6     {
7         seed=_seed;
8         p=_p;
9     }
10    void work(char *s,int n)
```

```

11 {
12     tmp[0]=1;
13     Hash[0]=0;
14     for(int i=1;i<=n;i++)
15     {
16         tmp[i]=tmp[i-1]*seed%p;
17         Hash[i]=(Hash[i-1]*seed+s[i])%p;//may
            need change
18     }
19 }
20 ll get(int l,int r)
21 {
22     return ((Hash[r]-Hash[l-1]*tmp[r-l+1])%p+p)%p
23     ;
24 }
};

```

1.2.2 hash_map

```

1 struct hash_map
2 {
3     static const int p=999917;
4     ll val[MAX],w[MAX];
5     int tot,head[p],nex[MAX];
6     int top,st[MAX];
7     void clear(){tot=0;while(top) head[st[top
8         --]]=0;}
9     void add(int x,ll y){val[++tot]=y;nex[tot]=head[
10         x];head[x]=tot;w[tot]=0;}
11     bool count(ll y)
12     {
13         int x=y%p;
14         for(int i=head[x];i;i=nex[i])
15         {
16             if(y==val[i]) return 1;
17         }
18         return 0;
19     }
20     ll& operator [] (ll y)
21     {
22         int x=y%p;
23         for(int i=head[x];i;i=nex[i])
24         {
25             if(y==val[i]) return w[i];
26         }
27         add(x,y);
28         st[++top]=x;
29         return w[tot];
30     }
31 }mp;

```

1.2.3 BKDRHash

```

1 struct BKDRHash
2 {
3     static const ull seed=1313131;//
4         31,131,1313,13131,131313
5     static const int p=2000007;
6     ull Hash[MAX],tmp[MAX];
7     ull val[MAX];
8     int last[p+10],nex[MAX],cnt;
9     void init();//clear hash table
10    {
11        mem(last,0);
12        cnt=0;
13    }
14    bool insert(ull x)
15    {
16        int u=x%p;
17        for(int i=last[u];i;i=nex[i])
18        {
19            if(val[i]==x) return 1;
20        }
21        nex[++cnt]=last[u];
22        last[u]=cnt;
23        val[cnt]=x;
24        return 0;
25    }
26    void work(char *s,int n)
27    {
28        tmp[0]=1;
29        Hash[0]=0;
30        for(int i=1;i<=n;i++)
31        {
32            tmp[i]=tmp[i-1]*seed;
33            Hash[i]=Hash[i-1]*seed+s[i];
34        }
35        ull get(int l,int r)
36        {
37            return Hash[r]-Hash[l-1]*tmp[r-l+1];
38        }
39    }bkdr; //bkdr.init();

```

1.3 Manacher

1.3.1 插字符

```

1 struct Manacher
2 {
3     int p[MAX<<1];
4     char s[MAX<<1];
5     int work(char *a)
6     {
7         int len,i,mid,r,res=0;
8         len=strlen(a+1);

```

```

9     for(i=1;i<=len;i++)
10     {
11         p[i]=0;
12         s[2*i-1]='%';
13         s[2*i]=a[i];
14     }
15     s[len=len*2+1]='%';
16     mid=r=0;
17     for(i=1;i<=len;i++)
18     {
19         if(i<r) p[i]=min(p[2*mid-i],r-i);
20         else p[i]=1;
21         while(i-p[i]>=1&&i+p[i]<=len&&s[i-p[i]]==
22             s[i+p[i]]) p[i]++;
23         if(i+p[i]>r)
24         {
25             r=i+p[i];
26             mid=i;
27         }
28         res=max(res,p[i]-1);
29     }
30     return res;
31 }la;

```

```

25         res=max(res,p[i]*2+1);
26     }
27     //even
28     r=mid=0;
29     mem(p,0);
30     for(i=2;i<=len;i++)
31     {
32         if(r>i) p[i]=min(p[2*mid-i],r-i+1);
33         while(i+p[i]<=len&&s[i+p[i]]==s[i-p[i]
34             ]-1))
35         {
36             //palindrome substring s[i-p[i]-1,i+p[
37                 i]]
38             p[i]++;
39         }
40         if(i+p[i]-1>r)
41         {
42             r=i+p[i]-1;
43             mid=i;
44         }
45         res=max(res,p[i]*2);
46     }
47     return res;
48 }la;

```

1.3.2 不插字符

```

1 struct Manacher
2 {
3     int p[MAX];
4     int work(char *s)//return max length of
5         palindrome
6     {
7         int r,mid,i,len,res=0;
8         len=strlen(s+1);
9         //odd
10        r=mid=0;
11        mem(p,0);
12        for(i=1;i<=len;i++)
13        {
14            //palindrome substring s[i,i]
15            if(r>i) p[i]=min(p[2*mid-i],r-i);
16            while(i+p[i]+1<=len&&s[i+p[i]+1]==s[i-p[i]
17                ]-1))
18            {
19                //palindrome substring s[i-p[i]-1,i+p[
20                    i]+1]
21                p[i]++;
22            }
23            if(i+p[i]>r)
24            {
25                r=i+p[i];
26                mid=i;
27            }
28        }
29    }
30 }

```

1.4 后缀数组

1.4.1 倍增 sa

```

1 struct SA
2 {
3     int s[MAX],n,SZ;
4     int c[MAX],rk[MAX],tp[MAX];
5     void init(char *ss) //s[1..n]
6     {
7         SZ=0;
8         n=strlen(ss+1);
9         for(int i=1;i<=n;i++)
10         {
11             s[i]=ss[i];
12             SZ=max(SZ,s[i]);
13         }
14     }
15     void get_sa(int *sa,int *height)
16     {
17         int m,i,j,k,now;
18         m=SZ;
19         for(i=1;i<=m;i++) c[i]=0;
20         for(i=1;i<=n;i++) c[rk[i]=s[i]]++;
21         for(i=2;i<=m;i++) c[i]+=c[i-1];
22         for(i=n;i;i--) sa[c[rk[i]]--]=i;
23         for(k=1;k<=n;k<=1)
24         {

```

```

25     now=0;
26     for(i=n-k+1;i<=n;i++) tp[+now]=i;
27     for(i=1;i<=n;i++)
28     {
29         if(sa[i]>k) tp[+now]=sa[i]-k;
30     }
31     for(i=1;i<=m;i++) c[i]=0;
32     for(i=1;i<=n;i++) c[rk[i]]++;
33     for(i=2;i<=m;i++) c[i]+=c[i-1];
34     for(i=n;i;i--)
35     {
36         sa[c[rk[tp[i]]]--]=tp[i];
37         tp[i]=0;
38     }
39     swap(rk,tp);
40     rk[sa[1]]=1;
41     now=1;
42     for(i=2;i<=n;i++)
43     {
44         if(!(tp[sa[i]]==tp[sa[i-1]]&&tp[sa[i]+
45             k]==tp[sa[i-1]+k])) now++;
46         rk[sa[i]]=now;
47     }
48     if(now==n) break;
49     m=now;
50     height[1]=0;
51     k=0;
52     for(i=1;i<=n;i++)
53     {
54         if(rk[i]==1) continue;
55         if(k) k--;
56         j=sa[rk[i]-1];
57         while(j+k<=n&&i+k<=n&&s[i+k]==s[j+k]) k
58             ++;
59         height[rk[i]]=k;
60     }
61 }da;

```

1.4.2 SA-IS

```

1 struct SA
2 {
3     char S[MAX]; int n,m;
4     int s[MAX<<1],t[MAX<<1],H[MAX],sa[MAX],r[MAX],p[
5         MAX],c[MAX],w[MAX];
6     inline int trans(int n,const char* S){
7         int m=*max_element(S+1,S+1+n);
8         for(int i=1;i<=n;++i) r[S[i]]=1;
9         for(int i=1;i<=m;++i) r[i]+=r[i-1];
10        for(int i=1;i<=n;++i) s[i]=r[S[i]];
11        return r[m];

```

```

12 #define ps(x) sa[w[s[x]]--]=x
13 #define pl(x) sa[w[s[x]]++]=x
14 inline void radix(int* v,int* s,int* t,int n,int
15     m,int n1){
16     memset(sa,0,n+1<<2); memset(c,0,m+1<<2);
17     for(int i=1;i<=n;++i) ++c[s[i]];
18     for(int i=1;i<=m;++i) w[i]=c[i]+c[i-1];
19     for(int i=n1;i;--i) ps(v[i]);
20     for(int i=1;i<=m;++i) w[i]=c[i-1]+1;
21     for(int i=1;i<=n;++i) if(sa[i]>1 && t[sa[i
22         ]-1]) pl(sa[i]-1);
23     for(int i=1;i<=m;++i) w[i]=c[i];
24     for(int i=n;i;--i) if(sa[i]>1 && !t[sa[i]-1])
25         ps(sa[i]-1);
26 }
27 inline void SAIS(int n,int m,int* s,int* t,int*
28     p){
29     int n1=0,ch=r[1]=0,*s1=s+n; t[n]=0;
30     for(int i=n-1;i;--i) t[i]=s[i]==s[i+1]?t[i
31         +1]:s[i]>s[i+1];
32     for(int i=2;i<=n;++i) r[i]=t[i-1]&&!t[i]?(p
33         [++n1]=i,n1):0;
34     radix(p,s,t,n,m,n1);
35     for(int i=1,x,y;i<=n;++i) if(x=r[sa[i]]){
36         if(ch<=1 || p[x+1]-p[x]!=p[y+1]-p[y]) ++
37             ch;
38         else for(int j=p[x],k=p[y];j<=p[x+1];++j
39             ,++k)
40             if((s[j]<<1|t[j])^(s[k]<<1|t[k])){ ++
41                 ch; break; }
42         s1[y=x]=ch;
43     }
44     if(ch<n1) SAIS(n1,ch,s1,t+n,p+n1);
45     else for(int i=1;i<=n1;++i) sa[s1[i]]=i;
46     for(int i=1;i<=n1;++i) s1[i]=p[sa[i]];
47     radix(s1,s,t,n,m,n1);
48 }
49 inline void get_sa(int n,const char* S,int *ssa,
50     int *h){
51     int m=trans(++n,S); SAIS(n,m,s,t,p);
52     for(int i=1;i<=n;++i) r[sa[i]]=sa[i+1]=i;
53     for(int i=1,j,k=0;i<=n;++i) if(r[i]>1){
54         for(j=sa[r[i]-1];S[i+k]==S[j+k];++k);
55         if(H[r[i]]=k) --k;
56     }
57     for(int i=1;i<=n;i++)
58     {
59         ssa[i]=sa[i];
60         h[i]=H[i];
61     }
62 }
63 }sa;

```

1.5 自动机

1.5.1 AC 自动机

```

1 struct AC_Automaton
2 {
3     static const int K=26;//may need change
4     int nex[MAX][K],fail[MAX],cnt[MAX],last[MAX];
5     int root,tot;
6     inline int getid(char c){return c-'a';};//may
7         need change
8     int newnode()
9     {
10         mem(nex[tot],0);
11         fail[tot]=0;
12         cnt[tot]=0;
13         return tot++;
14     }
15     void init()
16     {
17         tot=0;
18         root=newnode();
19     }
20     void insert(char *s)
21     {
22         int len,now,i;
23         len=strlen(s);
24         now=root;
25         for(i=0;i<len;i++)
26         {
27             int t=getid(s[i]);
28             if(!nex[now][t]) nex[now][t]=newnode();
29             now=nex[now][t];
30         }
31         cnt[now]++;
32     }
33     void work()
34     {
35         int i,now;
36         queue<int>q;
37         for(i=0;i<K;i++)
38         {
39             if(nex[root][i]) q.push(nex[root][i]);
40         }
41         while(!q.empty())
42         {
43             now=q.front();
44             q.pop();
45             //suffix link
46             if(cnt[fail[now]]) last[now]=fail[now];
47             else last[now]=last[fail[now]];
48             /*
49             may need add something here:
50             cnt[now]+=cnt[fail[now]];
51             */

```

```

51         for(i=0;i<K;i++)
52         {
53             if(nex[now][i])
54             {
55                 fail[nex[now][i]]=nex[fail[now]][i];
56                 q.push(nex[now][i]);
57             }
58             else nex[now][i]=nex[fail[now]][i];
59         }
60     }
61 }
62 int query(char *s)
63 {
64     int len,now,i,res;
65     len=strlen(s);
66     now=root;
67     res=0;
68     for(i=0;i<len;i++)
69     {
70         int t=getid(s[i]);
71         now=nex[now][t];
72         int tmp=now;
73         while(tmp&&cnt[tmp]!=-1)
74         {
75             res+=cnt[tmp];
76             cnt[tmp]=-1;
77             tmp=last[tmp];
78         }
79     }
80     return res;
81 }
82 //build fail tree
83 vector<int> mp[MAX];
84 void build_tree()
85 {
86     for(int i=0;i<=tot;i++) mp[i].clear();
87     for(int i=1;i<tot;i++) mp[fail[i]].pb(i);
88 }
89 }ac;

```

1.5.2 大字符集 AC 自动机

```

1 struct AC_Automaton
2 {
3     map<int,int> nex[MAX];
4     VI toplist;
5     int fail[MAX],last[MAX],cnt[MAX];
6     int root,tot;
7     int newnode()
8     {
9         tot++;
10        nex[tot].clear();
11        return tot;

```



```

12     }
13     void init()
14     {
15         toplist.clear();
16         tot=0;
17         root=newnode();
18     }
19     void insert(VI &s)
20     {
21         int len,now,i;
22         len=sz(s);
23         now=root;
24         for(i=0;i<len;i++)
25         {
26             int t=s[i];
27             if(!nex[now].count(t)) nex[now][t]=
28                 newnode();
29             now=nex[now][t];
30         }
31         cnt[now]=1;
32     }
33     void work()
34     {
35         int i,now;
36         queue<int>q;
37         for(auto it:nex[root])
38         {
39             fail[it.se]=root;
40             q.push(it.se);
41         }
42         fail[root]=1;
43         while(!q.empty())
44         {
45             now=q.front();
46             q.pop();
47             toplist.pb(now);
48             //suffix link
49             /* if(cnt[fail[now]]) last[now]=fail[now];
50             else last[now]=last[fail[now]];*/
51             cnt[now]+=cnt[fail[now]];
52             for(auto it:nex[now])
53             {
54                 int fail_now=fail[now];
55                 while(fail_now>1&&!nex[fail_now].count
56                     (it.fi)) fail_now=fail[fail_now];
57                 if(nex[fail_now].count(it.fi)) fail[it
58                     .se]=nex[fail_now][it.fi];
59                 else fail[it.se]=root;
60                 q.push(it.se);
61             }
62         }
63     }
64     int query(VI& s,int x)
65     {

```

```

63         int len,now,i,res;
64         len=sz(s);
65         now=root;
66         res=0;
67         for(i=0;i<len;i++)
68         {
69             int t=s[i];
70             while(now>1&&!nex[now].count(t)) now=fail
71                 [now];
72             if(nex[now].count(t)) now=nex[now][t];
73             else now=root;
74             //do something
75         }
76         return res;
77     }
78     void toptans()
79     {
80         for(int i=sz(toplist)-1;~i;i--)*do something
81         */;
82     }
83 }ac;

```

1.5.3 后缀自动机

```

1 struct Suffix_Automaton
2 {
3     static const int N=MAX<<1;
4     static const int K=26;// char size: [0,25]
5     int tot,last,nex[N][K],fa[N],len[N],cnt[N],
6         maxlen;
7     int newnode()
8     {
9         tot++;
10        fa[tot]=len[tot]=cnt[tot]=0;
11        mem(nex[tot],0);
12        return tot;
13    }
14    void init()
15    {
16        fa[0]=len[0]=cnt[0]=0;
17        mem(nex[0],0);
18        tot=0;
19        maxlen=0;
20        last=newnode();
21    }
22    void add(int x)
23    {
24        int p,q,np,nq;
25        p=last;
26        np=last=newnode();
27        len[np]=len[p]+1;
28        maxlen=max(maxlen,len[np]);
29        cnt[last]=1;
30        while(p&&!nex[p][x])

```

```

30     {
31         nex[p][x]=np;
32         p=fa[p];
33     }
34     if(p==0) fa[np]=1;
35     else
36     {
37         q=nex[p][x];
38         if(len[q]==len[p]+1) fa[np]=q;
39         else
40         {
41             nq=newnode();
42             memcpy(nex[nq],nex[q],sizeof(nex[q]));
43             len[nq]=len[p]+1;
44             maxlen=max(maxlen,len[nq]);
45             fa[nq]=fa[q];
46             fa[q]=fa[np]=nq;
47             while(p&&nex[p][x]==q)
48             {
49                 nex[p][x]=nq;
50                 p=fa[p];
51             }
52         }
53     }
54 }
55 int sum[N],tp[N];
56 void topsort()
57 {
58     int i;
59     for(i=1;i<=maxlen;i++) sum[i]=0;
60     for(i=1;i<=tot;i++) sum[len[i]]++;
61     for(i=1;i<=maxlen;i++) sum[i]+=sum[i-1];
62     for(i=1;i<=tot;i++) tp[sum[len[i]]--]=i;
63     for(i=tot;i;i--) cnt[fa[tp[i]]]+=cnt[tp[i]];
64 }
65 void build_tree(VI mp[])
66 {
67     for(int i=1;i<=tot;i++) mp[i].clear();
68     for(int i=1;i<=tot;i++) mp[fa[i]].pb(i);
69 }
70 int pos[N],id[N];
71 void init_pos(char *s,int n)//s[1..n]
72 {
73     int now=1;
74     for(int i=1;i<=tot;i++) id[i]=-1;
75     for(int i=1;i<=n;i++)
76     {
77         now=nex[now][s[i]-'a'];
78         pos[i]=now;
79         id[now]=i;
80     }
81 }
82 int st[N][21];
83 void init_ST()

```

```

84     {
85         int i,j,x;
86         for(i=1;i<=tot;i++)
87         {
88             x=tp[i];
89             st[x][0]=fa[x];
90             for(j=1;j<20;j++)
91             {
92                 st[x][j]=st[st[x][j-1]][j-1];
93             }
94         }
95     }
96     int get_substr(int l,int r)//init_pos init_ST
97     {
98         int now,tmp,i;
99         now=pos[r];
100         for(i=19;~i;i--)
101         {
102             tmp=st[now][i];
103             if(tmp&&len[tmp]>=r-l+1) now=tmp;
104         }
105         return now;
106     }
107 }sam;// sam.init();

```

1.5.4 回文自动机

```

1 struct Palindrome_Tree
2 {
3     int len[MAX],nex[MAX][26],fail[MAX],last,s[MAX],
4     tot,n;
5     int cnt[MAX],deep[MAX];
6     int newnode(int l)
7     {
8         mem(nex[tot],0);
9         fail[tot]=0;
10        deep[tot]=cnt[tot]=0;
11        len[tot]=l;
12        return tot++;
13    }
14    void init()
15    {
16        tot=n=last=0;
17        newnode(0);
18        newnode(-1);
19        s[0]=-1;
20        fail[0]=1;
21    }
22    int get_fail(int x)
23    {
24        while(s[n-len[x]-1]!=s[n]) x=fail[x];
25        return x;
26    }
27    void add(int t)//attention the type of t is int

```

```

27 {
28     int id,now;
29     s[++n]=t;
30     now=get_fail(last);
31     if(!nex[now][t])
32     {
33         id=newnode(len[now]+2);
34         fail[id]=nex[get_fail(fail[now])][t];
35         deep[id]=deep[fail[id]]+1;
36         nex[now][t]=id;
37     }
38     last=nex[now][t];
39     cnt[last]++;
40 }
41 void count()
42 {
43     for(int i=tot-1;i;i--) cnt[fail[i]]+=cnt[i];
44 }
45 void build_tree(VI mp[])// root is 0
46 {
47     for(int i=0;i<=tot+1;i++) mp[i].clear();
48     for(int i=1;i<tot;i++) mp[fail[i]].pb(i);
49 }
50 }pam; //pam.init();

```

1.5.5 序列自动机

```

1 int nex[MAX][26];
2 void work(char *s,int len)
3 {
4     mem(nex[len],0);
5     for(int i=len;i;i--)
6     {
7         for(int j=0;j<26;j++)
8         {
9             nex[i-1][j]=nex[i][j];
10        }
11        nex[i-1][s[i]-'a']=i;
12    }
13 }

```

1.6 最小表示法

1.6.1 最小表示法

```

1 int min_representation(char *s)
2 {
3     int i=0,j=1,k=0;
4     int len=strlen(s);
5     while(i<len&&j<len&&k<len)
6     {
7         if(s[(i+k)%len]==s[(j+k)%len])k++;
8         else

```

```

9         {
10             if(s[(i+k)%len]>s[(j+k)%len])i=i+k+1;
11             else j=j+k+1;
12             if(i==j)j++;
13             k=0;
14         }
15     }
16     return i<j?i:j;
17 }

```

1.6.2 最大表示法

```

1 int max_representation(char *s)
2 {
3     int i,j,k,len,t;
4     len=strlen(s);
5     i=k=0;
6     j=1;
7     while(i<len&&j<len&&k<len)
8     {
9         t=s[(i+k)%len]-s[(j+k)%len];
10        if(!t) k++;
11        else
12        {
13            if(t>0)
14            {
15                if(j+k+1>i) j=j+k+1;
16                else j=i+1;
17            }
18            else if(i+k+1>j) i=i+k+1;
19            else i=j+1;
20            k=0;
21        }
22    }
23    return i<j?i:j;
24 }

```

1.7 shift_and

```

1 void shift_and(char *s,char *t)//s[1..n],t[1..m] (n
2     <=m)
3 {
4     static const int SZ=26;
5     int n,m,i;
6     bitset<MAX> b[SZ],d;
7     for(i=0;i<SZ;i++) b[i].reset();
8     d.reset();
9     n=strlen(s+1);
10    m=strlen(t+1);
11    for(i=1;i<=n;i++)
12    {
13        b[s[i]-'a'].set(i-1);//change
14        //other matching character sets

```

```

14     }
15     for(i=1;i<=m;i++)
16     {
17         d<<=1;
18         d.set(0);
19         d&=(b[t[i]-'a']);//change
20         if(d[n-1]==1)//successful match
21         {
22
23         }
24     }
25 }

```

2 数据结构

2.1 RMQ

2.1.1 一维 RMQ

```

1 struct RMQ
2 {
3     #define type int
4     type v[MAX];
5     int pmax(int a,int b){return v[a]>v[b]?a:b;}
6     int pmin(int a,int b){return v[a]<v[b]?a:b;}
7     int lg[MAX],bin[22];
8     int pmx[MAX][22],pmn[MAX][22];
9     type mx[MAX][22],mn[MAX][22];
10    void work(int n,type *a)
11    {
12        int i,j;
13        for(i=bin[0]=1;1<<(i-1)<=n;i++) bin[i]=(bin[i-1]<<1);
14        for(i=2,lg[1]=0;i<=n;i++) lg[i]=lg[i>>1]+1;
15        for(i=1;i<=n;i++)
16        {
17            v[i]=a[i];
18            mx[i][0]=mn[i][0]=v[i];
19            pmx[i][0]=pmn[i][0]=i;
20        }
21        for(j=1;1<<(j-1)<=n;j++)
22        {
23            for(i=1;i+bin[j]-1<=n;i++)
24            {
25                mx[i][j]=max(mx[i][j-1],mx[i+bin[j]-1][j-1]);
26                mn[i][j]=min(mn[i][j-1],mn[i+bin[j]-1][j-1]);
27                pmx[i][j]=pmax(pmx[i][j-1],pmx[i+bin[j]-1][j-1]);
28                pmn[i][j]=pmin(pmn[i][j-1],pmn[i+bin[j]-1][j-1]);
29            }

```

```

30        }
31    }
32    type ask_max(int l,int r)
33    {
34        int t=lg[r-l+1];
35        return max(mx[l][t],mx[r-bin[t]+1][t]);
36    }
37    type ask_min(int l,int r)
38    {
39        int t=lg[r-l+1];
40        return min(mn[l][t],mn[r-bin[t]+1][t]);
41    }
42    int ask_pmax(int l,int r)
43    {
44        int t=lg[r-l+1];
45        return pmax(pmx[l][t],pmx[r-bin[t]+1][t]);
46    }
47    int ask_pmin(int l,int r)
48    {
49        int t=lg[r-l+1];
50        return pmin(pmn[l][t],pmn[r-bin[t]+1][t]);
51    }
52    #undef type
53 }rmq;

```

2.1.2 二维 RMQ

```

1 int v[302][302];
2 int maxx[302][302][9][9],minn[302][302][9][9];
3 void RMQ(int n,int m)
4 {
5     int i,j,ii,jj;
6     for(i=1;i<=n;i++)
7     {
8         for(j=1;j<=m;j++)
9         {
10            maxx[i][j][0][0]=minn[i][j][0][0]=v[i][j];
11        }
12    }
13    for(ii=0;(1<<ii)<=n;ii++)
14    {
15        for(jj=0;(1<<jj)<=m;jj++)
16        {
17            if(!(ii+jj)) continue;
18            for(i=1;i+(1<<ii)-1<=n;i++)
19            {
20                for(j=1;j+(1<<jj)-1<=m;j++)
21                {
22                    if(ii)
23                    {
24                        minn[i][j][ii][jj]=min(minn[i][j][ii-1][jj],minn[i+(1<<(ii-1))][j][ii-1][jj]);

```

```

25         maxx[i][j][ii][jj]=max(maxx[i][j][ii-1][jj],maxx[i+(1<<(ii-1))][j][ii-1][jj]);
26     }
27     else
28     {
29         minn[i][j][ii][jj]=min(minn[i][j][ii][jj-1],minn[i][j+(1<<(jj-1))][ii][jj-1]);
30         maxx[i][j][ii][jj]=max(maxx[i][j][ii][jj-1],maxx[i][j+(1<<(jj-1))][ii][jj-1]);
31     }
32 }
33 }
34 }
35 }
36 }
37 int ask_max(int x1,int y1,int x2,int y2)
38 {
39     int k1=0;
40     while((1<<(k1+1))<=x2-x1+1) k1++;
41     int k2=0;
42     while((1<<(k2+1))<=y2-y1+1) k2++;
43     x2=x2-(1<<k1)+1;
44     y2=y2-(1<<k2)+1;
45     return max(max(maxx[x1][y1][k1][k2],maxx[x1][y2][k1][k2]),max(maxx[x2][y1][k1][k2],maxx[x2][y2][k1][k2]));
46 }
47 int ask_min(int x1,int y1,int x2,int y2)
48 {
49     int k1=0;
50     while((1<<(k1+1))<=x2-x1+1) k1++;
51     int k2=0;
52     while((1<<(k2+1))<=y2-y1+1) k2++;
53     x2=x2-(1<<k1)+1;
54     y2=y2-(1<<k2)+1;
55     return min(min(minn[x1][y1][k1][k2],minn[x1][y2][k1][k2]),min(minn[x2][y1][k1][k2],minn[x2][y2][k1][k2]));
56 }

```

2.2 单调队列

```

1 struct Monotone_queue
2 {
3     #define type int
4     type v[MAX][2]; // 0 is min, 1 is max
5     int p[MAX][2];
6     int l[2],r[2];
7     void clear()
8     {

```

```

9         l[0]=r[0]=0;
10        l[1]=r[1]=0;
11    }
12    void insert(type x,int pos)
13    {
14        while(r[0]-l[0]&&v[r[0]-1][0]>=x) r[0]--;
15        v[r[0]][0]=x;
16        p[r[0]++][0]=pos;
17        while(r[1]-l[1]&&v[r[1]-1][1]<=x) r[1]--;
18        v[r[1]][1]=x;
19        p[r[1]++][1]=pos;
20    }
21    void erase(int pos)
22    {
23        while(r[0]-l[0]&&p[l[0]][0]<=pos) l[0]++;
24        while(r[1]-l[1]&&p[l[1]][1]<=pos) l[1]++;
25    }
26    type get_min(){return v[l[0]][0];}
27    type get_max(){return v[l[1]][1];}
28    #undef type
29 }dq;

```

2.3 LCA

2.3.1 倍增 LCA

```

1 //O(nlogn)-O(logn)
2 struct LCA
3 {
4     int fa[MAX][22],dep[MAX],n,limt,bin[22];
5     VI mp[MAX];
6     void init(int _n)
7     {
8         n=_n;
9         for(limt=1;1<<(limt-1)<=n;limt++);
10        for(int i=bin[0]=1;1<<(i-1)<=n;i++) bin[i]=(
11            bin[i-1]<<1);
12        for(int i=0;i<=n;i++)
13        {
14            mp[i].clear();
15            mem(fa[i],0);
16        }
17    }
18    void add_edge(int a,int b)
19    {
20        mp[a].pb(b);
21        mp[b].pb(a);
22    }
23    void dfs(int x,int pre)
24    {
25        for(int i=1;bin[i]<=dep[x];i++) fa[x][i]=fa[
26            fa[x][i-1]][i-1];
27        for(int i=0;i<sz(mp[x]);i++)
28        {

```

```

27     int to=mp[x][i];
28     if(to==pre) continue;
29     dep[to]=dep[x]+1;
30     fa[to][0]=x;
31     dfs(to,x);
32 }
33 }
34 void work(int rt)
35 {
36     dep[rt]=0;
37     dfs(rt,0);
38 }
39 int go(int x,int d)
40 {
41     for(int i=0;i<=limt&&i++)
42     {
43         if(bin[i]&d)
44         {
45             d^=bin[i];
46             x=fa[x][i];
47         }
48     }
49     return x;
50 }
51 int lca(int a,int b)
52 {
53     if(dep[a]<dep[b]) swap(a,b);
54     a=go(a,dep[a]-dep[b]);
55     if(a==b) return a;
56     for(int i=limt;~i;i--)
57     {
58         if(fa[a][i]!=fa[b][i])
59         {
60             a=fa[a][i];
61             b=fa[b][i];
62         }
63     }
64     return fa[a][0];
65 }
66 }lca;

```

2.3.2 RMQ 维护欧拉序求 LCA

```

1 // O(nlogn)-O(1)
2 struct LCA
3 {
4     #define type int
5     struct node{int to;type w;node(){}}node(int _to,
6         type _w):to(_to),w(_w){};
7     type dis[MAX];
8     int path[2*MAX],deep[2*MAX],first[MAX],len[MAX],
9         tot,n;
10    int dp[2*MAX][22];
11    vector<node> mp[MAX];

```

```

10 void dfs(int x,int pre,int h)
11 {
12     int i;
13     path[++tot]=x;
14     first[x]=tot;
15     deep[tot]=h;
16     for(i=0;i<mp[x].size();i++)
17     {
18         int to=mp[x][i].to;
19         if(to==pre) continue;
20         dis[to]=dis[x]+mp[x][i].w;
21         len[to]=len[x]+1;
22         dfs(to,x,h+1);
23         path[++tot]=x;
24         deep[tot]=h;
25     }
26 }
27 void ST(int n)
28 {
29     int i,j,x,y;
30     for(i=1;i<=n;i++) dp[i][0]=i;
31     for(j=1;(1<<j)<=n;j++)
32     {
33         for(i=1;i+(1<<j)-1<=n;i++)
34         {
35             x=dp[i][j-1];
36             y=dp[i+(1<<(j-1))][j-1];
37             dp[i][j]=deep[x]<deep[y]?x:y;
38         }
39     }
40 }
41 int query(int l,int r)
42 {
43     int len,x,y;
44     len=(int)log2(r-l+1);
45     x=dp[l][len];
46     y=dp[r-(1<<len)+1][len];
47     return deep[x]<deep[y]?x:y;
48 }
49 int lca(int x,int y)
50 {
51     int l,r,pos;
52     l=first[x];
53     r=first[y];
54     if(l>r) swap(l,r);
55     pos=query(l,r);
56     return path[pos];
57 }
58 type get_dis(int a,int b){return dis[a]+dis[b]
59     ]-2*dis[lca(a,b)];}
60 int get_len(int a,int b){return len[a]+len[b]-2*
61     len[lca(a,b)];}
62 void init(int _n)
63 {

```

```

62     n=_n;
63     for(int i=0;i<=n;i++)
64     {
65         dis[i]=0;
66         len[i]=0;
67         mp[i].clear();
68     }
69 }
70 void add_edge(int a,int b,type w=1)
71 {
72     mp[a].pb(node(b,w));
73     mp[b].pb(node(a,w));
74 }
75 void work(int rt)
76 {
77     tot=0;
78     dfs(rt,0,0);
79     ST(2*n-1);
80 }
81 int lca_root(int rt,int a,int b)
82 {
83     int fa,fb;
84     fa=lca(a,rt);
85     fb=lca(b,rt);
86     if(fa==fb) return lca(a,b);
87     else
88     {
89         if(get_dis(fa,rt)<get_dis(fb,rt)) return
                fa;
90         else return fb;
91     }
92 }
93 #undef type
94 }lca;
95 /*
96 lca.init(n);
97 lca.add_edge(a,b,w) undirected edge.
98 lca.work(rt);
99 */

```

2.4 轻重链剖分

size[] 数组，以 x 为根的子树节点个数。

top[] 数组，当前节点的所在链的顶端节点。

son[] 数组，重儿子。

deep[] 数组，当前节点的深度。

fa[] 数组，当前节点的父亲。

idx[] 数组，树中每个节点剖分后的新编号。

rnk[] 数组，idx 的逆，表示线段上中当前位置表示哪个节点。

```

1 struct HLD
2 {
3     #define type int
4     struct edge{int a,b;type v;edge(int _a,int _b,
        type _v=0):a(_a),b(_b),v(_v){}};
5     struct node{int to;type w;node(){}node(int _to,
        type _w):to(_to),w(_w){}};
6     vector<int> mp[MAX];
7     vector<edge> e;
8     int deep[MAX],fa[MAX],size[MAX],son[MAX];
9     int rnk[MAX],top[MAX],idx[MAX],tot;
10    int n,rt;
11    void init(int _n)
12    {
13        n=_n;
14        for(int i=0;i<=n;i++) mp[i].clear();
15        e.clear();
16        e.pb(edge(0,0));
17    }
18    void add_edge(int a,int b,type v=0)
19    {
20        e.pb(edge(a,b,v));
21        mp[a].pb(b);
22        mp[b].pb(a);
23    }
24    void dfs1(int x,int pre,int h)
25    {
26        int i,to;
27        deep[x]=h;
28        fa[x]=pre;
29        size[x]=1;
30        for(i=0;i<sz(mp[x]);i++)
31        {
32            to=mp[x][i];
33            if(to==pre) continue;
34            dfs1(to,x,h+1);
35            size[x]+=size[to];
36            if(son[x]==-1||size[to]>size[son[x]]) son
                [x]=to;
37        }
38    }
39    void dfs2(int x,int tp)
40    {
41        int i,to;
42        top[x]=tp;
43        idx[x]=++tot;
44        rnk[idx[x]]=x;
45        if(son[x]==-1) return;
46        dfs2(son[x],tp);
47        for(i=0;i<sz(mp[x]);i++)
48        {
49            to=mp[x][i];
50            if(to!=son[x]&&to!=fa[x]) dfs2(to,to);

```

```

51     }
52 }
53 void work(int _rt)
54 {
55     int i;
56     rt=_rt;
57     mem(son,-1);
58     tot=0;
59     dfs1(rt,0,0);
60     dfs2(rt,rt);
61 }
62 int LCA(int x,int y)
63 {
64     while(top[x]!=top[y])
65     {
66         if(deep[top[x]]<deep[top[y]]) swap(x,y);
67         x=fa[top[x]];
68     }
69     if(deep[x]>deep[y]) swap(x,y);
70     return x;
71 }
72 //node
73 void init_node()
74 {
75     build(n);
76 }
77 void modify_node(int x,int y,type val)
78 {
79     while(top[x]!=top[y])
80     {
81         if(deep[top[x]]<deep[top[y]]) swap(x,y);
82         update(idx[top[x]],idx[x],val);
83         x=fa[top[x]];
84     }
85     if(deep[x]>deep[y]) swap(x,y);
86     update(idx[x],idx[y],val);
87 }
88 type query_node(int x,int y)
89 {
90     type res=0;
91     while(top[x]!=top[y])
92     {
93         if(deep[top[x]]<deep[top[y]]) swap(x,y);
94         res+=query(idx[top[x]],idx[x]);
95         x=fa[top[x]];
96     }
97     if(deep[x]>deep[y]) swap(x,y);
98     res+=query(idx[x],idx[y]);
99     return res;
100 }
101 //path
102 void init_path()
103 {
104     v[idx[rt]]=0;

```

```

105     for(int i=1;i<n;i++)
106     {
107         if(deep[e[i].a]<deep[e[i].b]) swap(e[i].a
108             ,e[i].b);
109         v[idx[e[i].a]]=e[i].v;
110     }
111     build(n);
112 }
113 void modify_edge(int id,type val)
114 {
115     if(deep[e[id].a]>deep[e[id].b]) update(idx[e[
116         id].a],idx[e[id].a],val);
117     else update(idx[e[id].b],idx[e[id].b],val);
118 }
119 void modify_path(int x,int y,type val)
120 {
121     while(top[x]!=top[y])
122     {
123         if(deep[top[x]]<deep[top[y]]) swap(x,y);
124         update(idx[top[x]],idx[x],val);
125         x=fa[top[x]];
126     }
127     if(deep[x]>deep[y]) swap(x,y);
128     if(x!=y) update(idx[x]+1,idx[y],val);
129 }
130 type query_path(int x,int y)
131 {
132     type res=0;
133     while(top[x]!=top[y])
134     {
135         if(deep[top[x]]<deep[top[y]]) swap(x,y);
136         res+=query(idx[top[x]],idx[x]);
137         x=fa[top[x]];
138     }
139     if(deep[x]>deep[y]) swap(x,y);
140     if(x!=y) res+=query(idx[x]+1,idx[y]);
141     return res;
142 }
143 #undef type
144 }hld;
145 //*****attention!*****//
146 //hld.init(n)
147 //hld.add_edge(): undirected edge.
148 //*****//

```

2.5 并查集

2.5.1 并查集

```

1 struct dsu
2 {
3     int pre[MAX];
4     void init(int n)
5     {

```



```

6     int i;
7     for(i=1;i<=n;i++) pre[i]=i;
8 }
9 int find(int x)
10 {
11     if(pre[x]!=x) pre[x]=find(pre[x]);
12     return pre[x];
13 }
14 bool merge(int a,int b)
15 {
16     int ra,rb;
17     ra=find(a);
18     rb=find(b);
19     if(ra!=rb)
20     {
21         pre[ra]=rb;
22         return 1;
23     }
24     return 0;
25 }
26 }dsu;

```

2.5.2 map 实现并查集

```

1 struct dsu
2 {
3     unordered_map<int,int> pre;
4     void init(){pre.clear();}
5     int find(int x)
6     {
7         if(pre.count(x)) pre[x]=find(pre[x]);
8         else return x;
9         return pre[x];
10    }
11    bool merge(int a,int b)
12    {
13        int ra,rb;
14        ra=find(a);
15        rb=find(b);
16        if(ra!=rb)
17        {
18            pre[ra]=rb;
19            return 1;
20        }
21        return 0;
22    }
23 }dsu;

```

2.5.3 可撤销并查集

```

1 struct dsu
2 {
3     PII st[MAX];

```

```

4     int pre[MAX],top,sz[MAX];
5     void init(int n)
6     {
7         int i;
8         for(i=1;i<=n;i++)
9         {
10             pre[i]=i;
11             sz[i]=1;
12         }
13         top=0;
14     }
15     int find(int x)
16     {
17         while(x!=pre[x]) x=pre[x];
18         return x;
19     }
20     bool merge(int a,int b)
21     {
22         int ra,rb;
23         ra=find(a);
24         rb=find(b);
25         if(ra==rb) return 0;
26         if(sz[ra]>sz[rb]) swap(ra,rb);
27         pre[ra]=rb;
28         sz[rb]+=sz[ra];
29         st[top++]=MP(ra,rb);
30         return 1;
31     }
32     void roll_back()
33     {
34         PII now=st[--top];
35         pre[now.fi]=now.fi;
36         sz[now.se]-=sz[now.fi];
37     }
38 }dsu;

```

2.6 树状数组

2.6.1 一维单点 BIT

```

1 struct Fenwick_Tree
2 {
3     #define type int
4     type bit[MAX];
5     int n;
6     void init(int _n)
7     {
8         n=_n;
9         for(int i=0;i<=n;i++) bit[i]=0;
10    }
11    int lowbit(int x){return x&(-x);}
12    void insert(int x,type v)
13    {
14        while(x<=n)

```

```

15     {
16         bit[x]+=v;
17         x+=lowbit(x);
18     }
19 }
20 type get(int x)
21 {
22     type res=0;
23     while(x)
24     {
25         res+=bit[x];
26         x-=lowbit(x);
27     }
28     return res;
29 }
30 type ask(int l,int r)
31 {
32     if(l-1<=0) return get(r);
33     return get(r)-get(l-1);
34 }
35 #undef type
36 }tr;

```

2.6.2 一维区间 BIT

```

1 struct Fenwick_Tree
2 {
3     #define type int
4     type bit[MAX][2];
5     int n;
6     void init(int _n)
7     {
8         n=_n;
9         for(int i=0;i<=n;i++)
10         {
11             bit[i][0]=bit[i][1]=0;
12         }
13     }
14     int lowbit(int x){return x&(-x);}
15     //?????????_insert
16     void _insert(int x,type v)
17     {
18         for(int i=x;i<=n;i+=lowbit(i))
19         {
20             bit[i][0]+=v;
21             bit[i][1]+=v*(x-1);
22         }
23     }
24     void upd(int l,int r,type v)
25     {
26         _insert(l,v);
27         _insert(r+1,-v);
28     }
29     type get(int x)

```

```

30     {
31         type res=0;
32         for(int i=x;i-=lowbit(i))
33         {
34             res+=x*bit[i][0]-bit[i][1];
35         }
36         return res;
37     }
38     type ask(int l,int r)
39     {
40         if(l-1<=0) return get(r);
41         return get(r)-get(l-1);
42     }
43     #undef type
44 }tr;

```

2.6.3 二维单点 BIT

```

1 struct Fenwick_Tree
2 {
3     #define type int
4     type bit[MAX][MAX];
5     int n,m;
6     void init(int _n,int _m){n=_n;m=_m;mem(bit,0);}
7     int lowbit(int x){return x&(-x);}
8     void update(int x,int y,type v)
9     {
10         int i,j;
11         for(i=x;i<=n;i+=lowbit(i))
12         {
13             for(j=y;j<=m;j+=lowbit(j))
14             {
15                 bit[i][j]+=v;
16             }
17         }
18     }
19     type get(int x,int y)
20     {
21         type i,j,res=0;
22         for(i=x;i>0;i-=lowbit(i))
23         {
24             for(j=y;j>0;j-=lowbit(j))
25             {
26                 res+=bit[i][j];
27             }
28         }
29         return res;
30     }
31     type ask(int x1,int x2,int y1,int y2)
32     {
33         x1--;
34         y1--;
35         return get(x2,y2)-get(x1,y2)-get(x2,y1)+get(

```

```

36     }
37     #undef type
38 }tr;

```

2.7 线段树

2.7.1 线段树

```

1 struct Segment_Tree
2 {
3     #define type int
4     #define ls (id<<1)
5     #define rs (id<<1|1)
6     int n,ql,qr;
7     type a[MAX],v[MAX<<2],tag[MAX<<2],qv;
8     void pushup(int id)
9     {
10
11     }
12     void pushdown(int id)
13     {
14         if(!tag[id]) return;
15
16     }
17     void build(int l,int r,int id)
18     {
19         tag[id]=0;
20         if(l==r)
21         {
22             //init
23             return;
24         }
25         int mid=(l+r)>>1;
26         build(l,mid,ls);
27         build(mid+1,r,rs);
28         pushup(id);
29     }
30     void update(int l,int r,int id)
31     {
32         if(l>=ql&&r<=qr)
33         {
34             //do something
35             return;
36         }
37         pushdown(id);
38         int mid=(l+r)>>1;
39         if(ql<=mid) update(l,mid,ls);
40         if(qr>mid) update(mid+1,r,rs);
41         pushup(id);
42     }
43     type res;
44     void query(int l,int r,int id)
45     {
46         if(l>=ql&&r<=qr)

```

```

47     {
48         //do something
49         return;
50     }
51     pushdown(id);
52     int mid=(l+r)>>1;
53     if(ql<=mid) query(l,mid,ls);
54     if(qr>mid) query(mid+1,r,rs);
55 }
56 void build(int _n){n=_n;build(1,n,1);}
57 void upd(int l,int r,type v)
58 {
59     ql=l;
60     qr=r;
61     qv=v;
62     update(1,n,1);
63 }
64 type ask(int l,int r)//init res
65 {
66     ql=l;
67     qr=r;
68     res=0;
69     query(1,n,1);
70     return res;
71 }
72 #undef type
73 #undef ls
74 #undef rs
75 }tr;

```

2.7.2 动态开点线段树

```

1 //空间大小是nlogm,为插入的节点总数n,为区间长度m
2 struct Segment_Tree
3 {
4     #define type int
5     int root,tot,ls[MAX*20],rs[MAX*20],ql,qr,n;
6     type v[MAX*20],tag[MAX*20],qv;
7     int newnode()
8     {
9         ls[tot]=rs[tot]=0;
10        v[tot]=0;
11        tag[tot]=-1;
12        return tot++;
13    }
14    void init(int _n)
15    {
16        n=_n;
17        tot=0;
18        root=newnode();
19    }
20    void pushup(int id)
21    {
22

```

```

23 }
24 void pushdown(int id)
25 {
26     if(tag[id]==-1) return;
27     if(!ls[id]) ls[id]=newnode();
28     if(!rs[id]) rs[id]=newnode();
29
30
31     tag[id]=-1;
32 }
33 void update(int l,int r,int &id)
34 {
35     if(!id) id=newnode();
36     if(l>=ql&&r<=qr)
37     {
38         //do something
39         return;
40     }
41     pushdown(id);
42     int mid=(l+r)>>1;
43     if(ql<=mid) update(l,mid,ls[id]);
44     if(qr>mid) update(mid+1,r,rs[id]);
45     pushup(id);
46 }
47 type res;
48 void query(int l,int r,int &id)
49 {
50     if(!id) return;
51     if(l>=ql&&r<=qr)
52     {
53         //do something
54         return;
55     }
56     pushdown(id);
57     int mid=(l+r)>>1;
58     if(ql<=mid) query(l,mid,ls[id]);
59     if(qr>mid) query(mid+1,r,rs[id]);
60 }
61 void upd(int l,int r,type v)
62 {
63     ql=l;
64     qr=r;
65     qv=v;
66     update(1,n,root);
67 }
68 type ask(int l,int r)//init res
69 {
70     ql=l;
71     qr=r;
72     res=0;
73     query(1,n,root);
74     return res;
75 }
76 #undef type

```

```

77 };

```

2.7.3 线段树套线段树

```

1 struct Segment_Tree_2D
2 {
3     #define type int
4     static const int insert_num=;
5     static const int N=insert_num*20*20; //
6         insert_num*20*log(m)
7     int root[MAX<<2],tot,ls[N],rs[N],n,m;
8     int ql_in,qr_in,ql_out,qr_out;
9     type v[N],qv,tag[N];
10    void init(int _n,int _m)
11    {
12        n=_n;
13        m=_m;
14        mem(root,0);
15        ls[0]=rs[0]=0;
16        tag[0]=0;
17        v[0]=0;
18        tot=1;
19    }
20    int newnode()
21    {
22        ls[tot]=rs[tot]=0;
23        v[tot]=0;
24        tag[tot]=0;
25        return tot++;
26    }
27    void pushup(int id)
28    {
29    }
30    void pushdown(int id)
31    {
32        if(!tag[id]) return;
33        if(!ls[id]) ls[id]=newnode();
34        if(!rs[id]) rs[id]=newnode();
35
36
37        tag[id]=0;
38    }
39    void update_in(int l,int r,int &id)
40    {
41        if(!id) id=newnode();
42        if(l>=ql_in&&r<=qr_in)
43        {
44            v[id]+=qv; //must update not =
45            tag[id]+=qv;
46            return;
47        }
48        pushdown(id);
49        int mid=(l+r)>>1;

```

```

50     if(ql_in<=mid) update_in(l,mid,ls[id]);
51     if(qr_in>mid) update_in(mid+1,r,rs[id]);
52     pushup(id);
53 }
54 type res_in;
55 void query_in(int l,int r,int &id)
56 {
57     if(!id) return;
58     if(l>=ql_in&&r<=qr_in)
59     {
60         res_in+=v[id];
61         return;
62     }
63     pushdown(id);
64     int mid=(l+r)>>1;
65     type res=0;
66     if(ql_in<=mid) query_in(l,mid,ls[id]);
67     if(qr_in>mid) query_in(mid+1,r,rs[id]);
68 }
69 /*
70  ****
71  */
72 #define ls (id<<1)
73 #define rs (id<<1|1)
74 void update_out(int l,int r,int id)
75 {
76     update_in(1,m,root[id]);
77     if(l>=ql_out&&r<=qr_out) return;
78     int mid=(l+r)>>1;
79     if(ql_out<=mid) update_out(l,mid,ls);
80     if(qr_out>mid) update_out(mid+1,r,rs);
81 }
82 type res_out;
83 void query_out(int l,int r,int id)
84 {
85     if(l>=ql_out&&r<=qr_out)
86     {
87         res_in=0;
88         query_in(1,m,root[id]);
89         res_out+=res_in;
90         return;
91     }
92     int mid=(l+r)>>1;
93     if(ql_out<=mid) query_out(l,mid,ls);
94     if(qr_out>mid) query_out(mid+1,r,rs);
95 }
96 #undef ls
97 #undef rs
98 void upd(int x1,int y1,int x2,int y2,type val)
99 {
100     ql_out=x1;
101     qr_out=x2;
102     ql_in=y1;
103     qr_in=y2;

```

```

102     qv=val;
103     update_out(1,n,1);
104 }
105 type ask(int x1,int y1,int x2,int y2)
106 {
107     ql_out=x1;
108     qr_out=x2;
109     ql_in=y1;
110     qr_in=y2;
111     res_out=0;
112     query_out(1,n,1);
113     return res_out;
114 }
115 #undef type
116 }tr2d;

```

2.7.4 线段树分裂合并

```

1 struct Segment_Tree
2 {
3     #define type int
4     int s[MAX*20],top;
5     int root[MAX],tot,ls[MAX*20],rs[MAX*20],ql,qr,n;
6     type v[MAX*20],tag[MAX*20],qv;
7     void init()
8     {
9         top=0;
10        mem(root,0);
11        ls[0]=rs[0]=0;
12        v[0]=0;
13        tot=1;
14    }
15    int newnode()
16    {
17        int t;
18        if(top) t=s[--top];
19        else t=tot++;
20        ls[t]=rs[t]=0;
21        v[t]=0;
22        return t;
23    }
24    void delnode(int x)
25    {
26        s[top++]=x;
27    }
28    void pushup(int id)
29    {
30        v[id]=v[ls[id]]+v[rs[id]];
31    }
32    void pushdown(int id)
33    {
34        if(tag[id]==-1) return;
35        if(!ls[id]) ls[id]=newnode();
36        if(!rs[id]) rs[id]=newnode();

```

```

37
38
39     tag[id]=-1;
40 }
41 int split(int l,int r,int &id)
42 {
43     if(!id) return 0;
44     if(ql<=l&&r<=qr)
45     {
46         int temp=id;
47         id=0;
48         return temp;
49     }
50     int t=newnode();
51     int mid=(l+r)>>1;
52     if(ql<=mid) ls[t]=split(l,mid,ls[id]);
53     if(qr>mid) rs[t]=split(mid+1,r,rs[id]);
54     pushup(t);
55     pushup(id);
56     return t;
57 }
58 int merge(int a,int b)
59 {
60     if(!a||!b) return a+b;
61     ls[a]=merge(ls[a],ls[b]);
62     rs[a]=merge(rs[a],rs[b]);
63     if(!ls[a]&&!rs[a])
64     {
65         v[a]+=v[b]; //merge a,b to b
66     }
67     else
68     {
69         pushup(a);
70         //do something
71     }
72     delnode(b);
73     return a;
74 }
75 void update(int l,int r,int &id)
76 {
77     if(!id) id=newnode();
78     if(l>=ql&&r<=qr)
79     {
80         v[id]=(r-l+1)*qv;
81         tag[id]=qv;
82         return;
83     }
84     pushdown(id);
85     int mid=(l+r)>>1;
86     if(ql<=mid) update(l,mid,ls[id]);
87     if(qr>mid) update(mid+1,r,rs[id]);
88     pushup(id);
89 }
90 type query(int l,int r,int &id)

```

```

91 {
92     if(!id) return 0;
93     if(l>=ql&&r<=qr) return v[id];
94     int mid=(l+r)>>1;
95     type res=0;
96     if(ql<=mid) res+=query(l,mid,ls[id]);
97     if(qr>mid) res+=query(mid+1,r,rs[id]);
98     return res;
99 }
100 #undef type
101 }tr;

```

2.8 主席树

```

1 struct president_tree
2 {
3     #define type int
4     int root[MAX],ls[40*MAX],rs[40*MAX],tot,ql,qr;
5     type sum[40*MAX],qv;
6     void init()
7     {
8         mem(root,0);
9         tot=1;
10        ls[0]=rs[0]=sum[0]=0;
11    }
12    int newnode(int x)
13    {
14        ls[tot]=ls[x];
15        rs[tot]=rs[x];
16        sum[tot]=sum[x];
17        return tot++;
18    }
19    void insert(int l,int r,int &id,int pre) //set(
20        ql,ql,v)
21    {
22        id=newnode(pre);
23        sum[id]+=qv;
24        if(l==r) return;
25        int mid=(l+r)>>1;
26        if(ql<=mid) insert(l,mid,ls[id],ls[pre]);
27        else insert(mid+1,r,rs[id],rs[pre]);
28    }
29    int kindcnt(int l,int r,int id) //set(ql,qr)
30    {
31        if(ql<=l&&r<=qr) return sum[id];
32        int mid=(l+r)>>1;
33        int res=0;
34        if(ql<=mid) res+=kindcnt(l,mid,ls[id]);
35        if(qr>=mid+1) res+=kindcnt(mid+1,r,rs[id]);
36        return res;
37    }
38    int kthsmall(int l,int r,int id,int pre,int k)
39    {

```

```

39     if(l==r) return l;
40     int mid=(l+r)>>1;
41     int temp=sum[ls[id]]-sum[ls[pre]];
42     if(temp>=k) return kthsmall(l,mid,ls[id],ls[
43         pre],k);
44     else return kthsmall(mid+1,r,rs[id],rs[pre],k
45         -temp);
46 }
47 int kthbig(int l,int r,int id,int pre,int k)
48 {
49     if(l==r) return l;
50     int mid=(l+r)>>1;
51     int temp=sum[rs[id]]-sum[rs[pre]];
52     if(temp>=k) return kthbig(mid+1,r,rs[id],rs[
53         pre],k);
54     else return kthbig(l,mid,ls[id],ls[pre],k-
55         temp);
56 }
57 void set(int l,int r,type v=0){ql=l;qr=r;qv=v;}
58 }pt;

```

2.9 李超树

```

1 struct LiChao_Segment_Tree
2 {
3     #define type ll
4     #define inf -LLINF
5     #define ls (id<<1)
6     #define rs (id<<1|1)
7     #define cmp max
8     struct line
9     {
10         type k,b;
11         void init(type _k=0,type _b=0){k=_k;b=_b;}
12     }sg[MAX<<2];
13     type v[MAX<<2];
14     bool ext[MAX<<2];
15     int ql,qr,n;
16     type cal(line l,int x){return l.k*x+l.b;}
17     void pushup(int id)
18     {
19         v[id]=cmp(v[id],v[ls]);
20         v[id]=cmp(v[id],v[rs]);
21     }
22     void build(int l,int r,int id)
23     {
24         ext[id]=0;
25         sg[id].init();
26         v[id]=inf;
27         if(l==r) return;
28         int mid=(l+r)>>1;
29         build(l,mid,ls);
30         build(mid+1,r,rs);

```

```

31     }
32     void update(int l,int r,int id,line qv)
33     {
34         if(l>=ql&&r<=qr)
35         {
36             if(!ext[id])
37             {
38                 ext[id]=1;
39                 sg[id]=qv;
40             }
41             else if(cal(qv,l)>=cal(sg[id],l)&&cal(qv,
42                 r)>=cal(sg[id],r)) sg[id]=qv;
43             else if(cal(qv,l)>cal(sg[id],l)||cal(qv,r
44                 )>cal(sg[id],r))
45             {
46                 int mid=(l+r)>>1;
47                 if(cal(qv,mid)>cal(sg[id],mid)) swap(
48                     qv,sg[id]);
49                 if(cal(qv,l)>cal(sg[id],l)) update(l,
50                     mid,ls,qv);
51                 else update(mid+1,r,rs,qv);
52             }
53             v[id]=cmp(cal(sg[id],l),cal(sg[id],r));
54             if(l!=r) pushup(id);
55             //if ask min, change '>' to '<'
56         }
57     }
58     else
59     {
60         int mid=(l+r)>>1;
61         if(ql<=mid) update(l,mid,ls,qv);
62         if(qr>mid) update(mid+1,r,rs,qv);
63         v[id]=cmp(cal(sg[id],l),cal(sg[id],r));
64         if(l!=r) pushup(id);
65     }
66 }
67 type res;
68 void query(int l,int r,int id)
69 {
70     if(l>=ql&&r<=qr)
71     {
72         res=cmp(res,v[id]);
73         return;
74     }
75     res=cmp(res,cal(sg[id],max(l,ql)));
76     res=cmp(res,cal(sg[id],min(r,qr)));
77     int mid=(l+r)>>1;
78     if(ql<=mid) query(l,mid,ls);
79     if(qr>mid) query(mid+1,r,rs);
80 }
81 void build(int _n){n=_n;build(1,n,1);}
82 void upd(int l,int r,type k,type b)
83 {
84     ql=l;
85     qr=r;

```

```

81     line qv;
82     qv.init(k,b);
83     update(1,n,1,qv);
84 }
85 type ask(int l,int r)
86 {
87     ql=l;
88     qr=r;
89     res=inf;
90     query(1,n,1);
91     return res;
92 }
93 #undef type
94 #undef ls
95 #undef rs
96 #undef cmp
97 #undef inf
98 }tr;

```

2.10 平衡树

2.10.1 Treap

```

1 struct Treap
2 {
3     #define type ll
4     struct node
5     {
6         int ch[2],fix,sz,w;
7         type v;
8         node(){}
9         node(type x)
10        {
11            v=x;
12            fix=rand();
13            sz=w=1;
14            ch[0]=ch[1]=0;
15        }
16    }t[MAX];
17    int tot,root,tmp;
18    void init()
19    {
20        srand(unsigned(new char));
21        root=tot=0;
22        t[0].sz=t[0].w=0;
23        mem(t[0].ch,0);
24    }
25    inline void maintain(int k)
26    {
27        t[k].sz=t[t[k].ch[0]].sz+t[t[k].ch[1]].sz+t[k].w ;
28    }
29    inline void rotate(int &id,int k)
30    {

```

```

31        int y=t[id].ch[k^1];
32        t[id].ch[k^1]=t[y].ch[k];
33        t[y].ch[k]=id;
34        maintain(id);
35        maintain(y);
36        id=y;
37    }
38    void insert(int &id,type v)
39    {
40        if(!id) t[id=++tot]=node(v);
41        else
42        {
43            if(t[id].sz++,t[id].v==v)t[id].w++;
44            else if(insert(t[id].ch[tmp=v>t[id].v],v),
45                      t[t[id].ch[tmp]].fix>t[id].fix)
46                rotate(id,tmp^1);
47        }
48    }
49    void erase(int &id,type v)
50    {
51        if(!id)return;
52        if(t[id].v==v)
53        {
54            if(t[id].w>1) t[id].w--,t[id].sz--;
55            else
56            {
57                if(!(t[id].ch[0]&& t[id].ch[1])) id=t[
58                    id].ch[0]|t[id].ch[1];
59                else
60                {
61                    rotate(id,tmp=t[t[id].ch[0]].fix>t[
62                        t[id].ch[1]].fix);
63                    t[id].sz--;
64                    erase(t[id].ch[tmp],v);
65                }
66            }
67        }
68        else
69        {
70            t[id].sz--;
71            erase(t[id].ch[v>t[id].v],v);
72        }
73    }
74    type kth(int k)//k small
75    {
76        int id=root;
77        if(id==0) return 0;
78        while(id)
79        {

```



```

80         k-=t[t[id].ch[0]].sz+t[id].w;
81         id=t[id].ch[1];
82     }
83 }
84 }
85 int find(type key,int f)
86 {
87     int id=root,res=0;
88     while(id)
89     {
90         if(t[id].v<=key)
91         {
92             res+=t[t[id].ch[0]].sz+t[id].w;
93             if(f&&key==t[id].v) res-=t[id].w;
94             id=t[id].ch[1];
95         }
96         else id=t[id].ch[0];
97     }
98     return res;
99 }
100 type find_pre(type key)
101 {
102     type res=-LLINF;
103     int id=root;
104     while(id)
105     {
106         if(t[id].v<key)
107         {
108             res=max(res,t[id].v);
109             id=t[id].ch[1];
110         }
111         else id=t[id].ch[0];
112     }
113     return res;
114 }
115 type find_suc(type key)
116 {
117     type res=LLINF;
118     int id=root;
119     while(id)
120     {
121         if(t[id].v>key)
122         {
123             res=min(res,t[id].v);
124             id=t[id].ch[0];
125         }
126         else id=t[id].ch[1];
127     }
128     return res;
129 }
130 void insert(type v){insert(root,v);}
131 void erase(type v){erase(root,v);}
132 int upper_bound_count(type key){return find(key
,0);};//the count >=key

```

```

133 int lower_bound_count(type key){return find(key
,1);};//the count >key
134 int rank(type key){return lower_bound_count(key)
+1;}
135 #undef type
136 }t; //t.init();

```

2.11 字典树

2.11.1 trie

```

1 struct Trie
2 {
3     #define type int
4     struct trie
5     {
6         int v;
7         trie *next[26];
8         trie()
9         {
10             v=0;
11             for(int i=0;i<26;i++) next[i]=NULL;
12         }
13     }*root;
14     void insert(trie *p,char *s)
15     {
16         int i=0,t;
17         while(s[i])
18         {
19             t=s[i]-'a';
20             if(p->next[t]==NULL) p->next[t]=new trie;
21             p=p->next[t];
22             p->v++; //may need change
23             i++;
24         }
25     }
26     int find(trie *p,char *s)
27     {
28         int i=0,t;
29         while(s[i])
30         {
31             t=s[i]-'a';
32             if(p->next[t]==NULL) return 0;
33             p=p->next[t];
34             i++;
35         }
36         return p->v; //may need change
37     }
38     //r?????³sµ´®
39     void del(char *s)
40     {
41         int i=0,t,temp;
42         trie *p,*pre;
43         pre=p=root;

```

```

44 while(s[i])
45 {
46     t=s[i]-'a';
47     if(p->next[t]==NULL) return;
48     if(!s[i+1])
49     {
50         temp=p->next[t]->v;
51         p->next[t]=NULL;
52         break;
53     }
54     pre=p;
55     p=p->next[t];
56     i++;
57 }
58 i=0;
59 p=root;
60 while(s[i])
61 {
62     t=s[i]-'a';
63     if(p->next[t]==NULL) return;
64     p=p->next[t];
65     p->v-=temp;
66     i++;
67 }
68 }
69 #undef type
70 }tr;

```

2.11.2 01trie

```

1 struct Trie
2 {
3     #define type int
4     static const int mx=30;
5     int root,tot,nex[MAX*mx][2];
6     type cnt[MAX*mx];
7     int newnode()
8     {
9         mem(nex[tot],0);
10        cnt[tot]=0;
11        return tot++;
12    }
13    void init()
14    {
15        mem(nex[0],0);
16        cnt[0]=0;
17        tot=1;
18        root=newnode();
19    }
20    void upd(type x,type v)
21    {
22        int id,t,i;
23        id=root;
24        for(i=mx;~i;i--)

```

```

25    {
26        t=(x>>i)&1;
27        if(!nex[id][t]) nex[id][t]=newnode();
28        id=nex[id][t];
29        cnt[id]+=v;
30    }
31 }
32 type count(int x)
33 {
34     int id,t,i;
35     id=root;
36     for(i=mx;~i;i--)
37     {
38         t=(x>>i)&1;
39         if(!nex[id][t]) return 0;
40         id=nex[id][t];
41     }
42     return cnt[id];
43 }
44 type ask_max(type x)
45 {
46     int id,t,i;
47     type res;
48     id=root;
49     res=0;
50     for(i=mx;~i;i--)
51     {
52         t=(x>>i)&1;
53         if(nex[id][t^1]&&cnt[nex[id][t^1]]) t^=1;
54         res|=(t<<i);
55         id=nex[id][t];
56     }
57     return res;
58 }
59 type ask_min(type x)
60 {
61     int id,t,i;
62     type res;
63     id=root;
64     res=0;
65     for(i=mx;~i;i--)
66     {
67         t=(x>>i)&1;
68         if(!nex[id][t]||!cnt[nex[id][t]]) t^=1;
69         res|=(t<<i);
70         id=nex[id][t];
71     }
72     return res;
73 }
74 #undef type
75 }tr;

```

2.12 kd-tree

```

1 namespace kd_tree
2 {
3     const double alpha=0.75;
4     const int dim=2;
5     #define type int
6     const type NONE=INF; //初始值
7     struct kdtnode
8     {
9         bool exist;
10        int l,r,sz,fa,dep,x[dim],mx[dim],mn[dim];
11        type v,tag;
12        kdtnode(){ }
13        void initval()
14        {
15            sz=exist;tag=v;
16            if(exist) for(int i=0;i<dim;i++) mn[i]=mx
17                [i]=x[i];
18        }
19        void null()
20        {
21            exist=sz=0;
22            v=tag=NONE;
23            for(int i=0;i<dim;i++)
24            {
25                mx[i]=-INF;
26                mn[i]=INF;
27            }
28        }
29        void newnode(int x0,int x1,type val=NONE)
30        {
31            x[0]=x0;
32            x[1]=x1;
33            l=r=fa=0;
34            exist=1;
35            v=val;
36            initval();
37        }
38        kdtnode(int a,int b,type d=NONE){newnode(a,b,
39            d);}
40    };
41    struct KDT
42    {
43        #define ls t[id].l
44        #define rs t[id].r
45        kdtnode t[MAX];
46        int tot,idx,root;
47        inline void pushup(int id)
48        {
49            t[id].initval();
50            t[id].sz=t[ls].sz+t[rs].sz;
51            t[id].tag=min({t[ls].tag,t[rs].tag,t[id].
52                tag});
53            for(int i=0;i<dim;i++)
54            {
55                t[id].mx[i]=max(t[id].mx[i],t[ls].
56                    mx[i]);
57                t[id].mn[i]=min(t[id].mn[i],t[ls].
58                    mn[i]);
59            }
60            if(rs)
61            {
62                t[id].mx[i]=max(t[id].mx[i],t[rs].
63                    mx[i]);
64                t[id].mn[i]=min(t[id].mn[i],t[rs].
65                    mn[i]);
66            }
67        }
68        bool isbad(int id){return t[id].sz*alpha+3<
69            max(t[ls].sz,t[rs].sz);}
70        int st[MAX],top;
71        void build(int &id,int l,int r,int fa,int dep
72            =0)
73        {
74            id=0;if(l>r) return;
75            int m=(l+r)>>1; idx=dep;
76            nth_element(st+l,st+m,st+r+1,[&](int x,
77                int y){return t[x].x[idx]<t[y].x[idx
78                    ];});
79            id=st[m];
80            build(ls,l,m-1,id,(dep+1)%dim);
81            build(rs,m+1,r,id,(dep+1)%dim);
82            pushup(id);
83            t[id].dep=dep;
84            t[id].fa=fa;
85        }
86        inline void init(int n=0)
87        {
88            root=0;
89            t[0].null();
90            for(int i=1;i<=n;i++) st[i]=i;
91            if(n) build(root,1,n,0);
92            tot=n;
93        }
94        void travel(int id)
95        {
96            if(!id) return;
97            if(t[id].exist) st[++top]=id;
98            travel(ls);
99            travel(rs);
100        }
101        void rebuild(int &id,int dep)
102        {
103            top=0;travel(id);
104            build(id,1,top,t[id].fa,dep);

```

```

97     }
98     void insert(int &id,int now,int fa,int dep=0)
99     {
100         if(!id)
101         {
102             id=now;
103             t[id].dep=dep;
104             t[id].fa=fa;
105             return;
106         }
107         if(t[now].x[dep]<t[id].x[dep]) insert(ls,
108             now,id,(dep+1)%dim);
109         else insert(rs,now,id,(dep+1)%dim);
110         pushup(id);
111         if(isbad(id)) rebuild(id,t[id].dep);
112         t[id].dep=dep;
113         t[id].fa=fa;
114     }
115     inline void insert(kdtnode x){t[++tot]=x;
116         insert(root,tot,0,0);}
117     inline void del(int id)
118     {
119         if(!id) return;
120         t[id].null();
121         int x=id;
122         while(x)
123         {
124             pushup(x);
125             x=t[x].fa;
126         }
127         if(isbad(id))
128         {
129             x=t[id].fa;
130             rebuild(root==id?root:(t[x].l==id?t[x]
131                 .l:t[x].r),t[id].dep);
132         }
133     }
134     kdtnode q;
135     ll dist(ll x,ll y){return x*x+y*y;}
136     ll getdist(int id)//点离区域qt[id]最短距离
137     {
138         if(!id) return LLINF;
139         ll res=0;
140         if(q.x[0]<t[id].mn[0]) res+=dist(q.x[0]-t
141             [id].mn[0],0);
142         if(q.x[1]<t[id].mn[1]) res+=dist(q.x[1]-t
143             [id].mn[1],0);
144         if(q.x[0]>t[id].mx[0]) res+=dist(q.x[0]-t
145             [id].mx[0],0);
146         if(q.x[1]>t[id].mx[1]) res+=dist(q.x[1]-t
147             [id].mx[1],0);
148         return res;
149     }
150     kdtnode a,b;

```

```

144     inline int check(kdtnode &x)//在矩形x(a,b)内
145     {
146         int ok=1;
147         for(int i=0;i<dim;i++)
148         {
149             ok&=(x.x[i]>=a.x[i]);
150             ok&=(x.x[i]<=b.x[i]);
151         }
152         return ok;
153     }
154     inline int allin(kdtnode &x)//的子树全在矩
155         形x(a,b)内
156     {
157         int ok=1;
158         for(int i=0;i<dim;i++)
159         {
160             ok&=(x.mn[i]>=a.x[i]);
161             ok&=(x.mx[i]<=b.x[i]);
162         }
163         return ok;
164     }
165     inline int allout(kdtnode &x)//的子树全不在矩
166         形x(a,b)内
167     {
168         int ok=0;
169         for(int i=0;i<dim;i++)
170         {
171             ok|=(x.mx[i]<a.x[i]);
172             ok|=(x.mn[i]>b.x[i]);
173         }
174         return ok;
175     }
176     type res;
177     void query(int id)
178     {
179         if(!id) return;
180         if(allout(t[id])||t[id].sz==0) return;
181         if(allin(t[id]))
182         {
183             res=min(res,t[id].tag);
184             return;
185         }
186         if(check(t[id])&&t[id].exist) res=min(res
187             ,t[id].v);
188         int l,r;
189         l=ls;
190         r=rs;
191         if(t[l].tag>t[r].tag) swap(l,r);
192         if(t[l].tag<res) query(l);
193         if(t[r].tag<res) query(r);
194     }
195     inline type query(kdtnode _a,kdtnode _b)
196     {
197         a=_a;b=_b;
198         res=INF;

```

```

196         query(root);
197         return res;
198     }
199     }kd;
200     #undef type
201     #undef ls
202     #undef rs
203 }
204 using namespace kd_tree;

```

2.13 虚树

```

1 VI mp[MAX];
2 void add_edge(int a,int b)
3 {
4     mp[a].pb(b);
5     mp[b].pb(a);
6 }
7 int st[MAX],top,dfn[MAX];
8 bool cmp(int x,int y){return dfn[x]<dfn[y];}
9 int build_vtree(vector<int> &a)// return root
10 {
11     int lca;
12     sort(all(a),cmp);
13     a.erase(unique(all(a)),a.end());
14     assert(a.size()>0);
15     top=0;
16     st[top++]=a[0];
17     VI tmp;
18     for(int i=1;i<sz(a);i++)
19     {
20         if(top==0){st[top++]=a[i]; continue;}
21         lca=lca(a[i],st[top-1]);
22         while(top>1&&dfn[st[top-2]]>=dfn[lca])
23             add_edge(st[top-2],st[top-1]),top--;
24         if(lca!=st[top-1]) {add_edge(lca,st[top-1]),
25             st[top-1]=lca; tmp.push_back(lca);}
26         st[top++]=a[i];
27     }
28     while(top>1) add_edge(st[top-2],st[top-1]),top--;
29     for(auto it:tmp) a.push_back(it);
30     return st[0];
31 }

```

2.14 pbds 可并堆

```

1 #include <ext/pb_ds/priority_queue.hpp>
2 using namespace __gnu_pbds;
3 __gnu_pbds::priority_queue<int,greater<int>,
4     pairing_heap_tag> q[MAX];
5 //q[i].join(q[j]) -> q[j合并到]q[i],q[j清空]

```

2.15 k 叉哈夫曼树

用两个队列代替优先队列复杂度 $O(n)$

注意：小的先进原数组有序

```

1 int a[MAX];
2 int Huffman(int k)
3 {
4     int i,res,s;
5     queue<int> q,d;
6     s=((n-1)%(k-1)?k-1-(n-1)%(k-1):0);//计算要补多少
7     while(s-->0) q.push(0);
8     for(i=1;i<=n;i++) q.push(a[i]);
9     res=0;
10    while(sz(q)+sz(d)>1)
11    {
12        s=0;
13        for(i=0;i<k;i++)
14        {
15            if(sz(q)&&sz(d))
16            {
17                if(q.front()<d.front())
18                {
19                    s+=q.front();
20                    q.pop();
21                }
22                else
23                {
24                    s+=d.front();
25                    d.pop();
26                }
27            }
28            else if(sz(q))
29            {
30                s+=q.front();
31                q.pop();
32            }
33            else if(sz(d))
34            {
35                s+=d.front();
36                d.pop();
37            }
38        }
39        res+=s;
40        d.push(s);
41    }
42    return res;
43 }

```

2.16 笛卡尔树

$O(n)$ 构造笛卡尔树返回根

性质:

1. 树中的元素满足二叉搜索树性质，要求按照中序遍历得到的序列为原数组序列
2. 树中节点满足堆性质，节点的 key 值要大于其左右子节点的 key 值

```

1 namespace Cartesian_Tree
2 {
3     int l[MAX],r[MAX],vis[MAX],stk[MAX];
4     int build(int *a,int n)
5     {
6         int i,top=0;
7         for(i=1;i<=n;i++) l[i]=0,r[i]=0,vis[i]=0;
8         for(i=1;i<=n;i++)
9         {
10             int k=top;
11             while(k>0&&a[stk[k-1]]>a[i]) k--;
12             if(k) r[stk[k-1]]=i;
13             if(k<top) l[i]=stk[k];
14             stk[k++]=i;
15             top=k;
16         }
17         for(i=1;i<=n;i++) vis[l[i]]=vis[r[i]]=1;
18         for(i=1;i<=n;i++)
19         {
20             if(!vis[i]) return i;
21         }
22     }
23 }
```

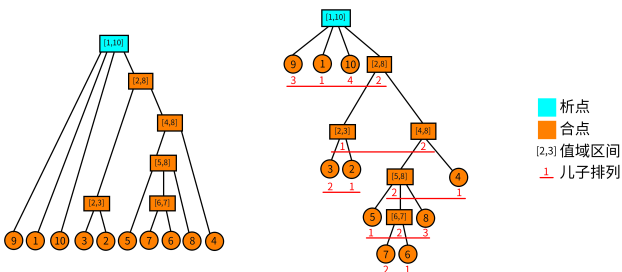
2.17 析合树

id[x]: 序列中第 x 个数在析合树上的编号。

$l[x],r[x]$: 节点 x 的作用域。

type[x]: 节点 x 的类型，0 表示析点，1 表示合点，默认叶子节点为析点。

注意: 若一个节点为合点，这个节点的儿子序列有序。析合树举例:



```

1 struct permutation_tree
2 {
3     struct RMQ
4     {
5         #define type int
6         int lg[MAX],bin[22];
7         type mx[MAX][22],mn[MAX][22];
8         void work(int n,type *v)
9         {
10             int i,j;
11             for(i=bin[0]=1;1<=(i-1)<=n;i++) bin[i]=(
12                 bin[i-1]<<1);
13             for(i=2,lg[1]=0;i<=n;i++) lg[i]=lg[i
14                 >>1]+1;
15             for(i=1;i<=n;i++) mx[i][0]=mn[i][0]=v[i];
16             for(j=1;1<=(j-1)<=n;j++)
17             {
18                 for(i=1;i+bin[j]-1<=n;i++)
19                 {
20                     mx[i][j]=max(mx[i][j-1],mx[i+bin[j]
21                         -1][j-1]);
22                     mn[i][j]=min(mn[i][j-1],mn[i+bin[j]
23                         -1][j-1]);
24                 }
25             }
26             type ask_max(int l,int r)
27             {
28                 int t=lg[r-l+1];
29                 return max(mx[l][t],mx[r-bin[t]+1][t]);
30             }
31             type ask_min(int l,int r)
32             {
33                 int t=lg[r-l+1];
34                 return min(mn[l][t],mn[r-bin[t]+1][t]);
35             }
36             #undef type
37         }rmq;
38     }struct Segment_Tree
39     {
40         #define type int
41         #define ls (id<<1)
42         #define rs (id<<1|1)
43         int n,ql,qr;
44         type mn[MAX<<2],tag[MAX<<2],qv;
45         void mdf(int id,type v){mn[id]+=v,tag[id]+=v
46             ;}
47         void pushup(int id){mn[id]=min(mn[ls],mn[rs])
48             ;}
49         void pushdown(int id)
50         {
51             if(!tag[id]) return;
52         }
```

```

47     mdf(ls,tag[id]);
48     mdf(rs,tag[id]);
49     tag[id]=0;
50 }
51 void build(int l,int r,int id)
52 {
53     tag[id]=mn[id]=0;
54     if(l==r) return;
55     int mid=(l+r)>>1;
56     build(l,mid,ls);
57     build(mid+1,r,rs);
58     pushup(id);
59 }
60 void update(int l,int r,int id)
61 {
62     if(l>=ql&&r<=qr){mdf(id,qv);return;}
63     pushdown(id);
64     int mid=(l+r)>>1;
65     if(ql<=mid) update(l,mid,ls);
66     if(qr>mid) update(mid+1,r,rs);
67     pushup(id);
68 }
69 int query(int l,int r,int id)
70 {
71     if(l==r) return l;
72     pushdown(id);
73     int mid=(l+r)>>1;
74     if(!mn[ls]) return query(l,mid,ls);
75     else query(mid+1,r,rs);
76 }
77 void build(int _n){n=_n;build(1,n,1);}
78 void upd(int l,int r,type v){ql=l;qr=r;qv=v;
79     update(1,n,1);}
80 type ask(int l,int r){ql=l;qr=r;return query
81     (1,n,1);}
82 #undef type
83 #undef ls
84 #undef rs
85 }tr;
86 bool check(int l,int r){return rmq.ask_max(l,r)-
87     rmq.ask_min(l,r)==r-l;}
88 int st[MAX],st1[MAX],st2[MAX],top,top1,top2,m[
89     MAX];
90 int tot,id[MAX],l[MAX],r[MAX],type[MAX];
91 VI mp[MAX];
92 void add_edge(int a,int b){mp[a].pb(b);}
93 int build(int n,int *a)
94 {
95     int now,i,tmp;
96     tr.build(n);
97     rmq.work(n,a);
98     for(i=0;i<=2*n;i++)
99     {
100         mp[i].clear();

```

```

97     type[i]=0;
98     }
99     top=top1=top2=0;
100     tot=0;
101     for(i=1;i<=n;i++)
102     {
103         while(top1&&a[i]<=a[st1[top1]])
104         {
105             tr.upd(st1[top1-1]+1,st1[top1],a[st1[
106                 top1]]);
107             top1--;
108         }
109         while(top2&&a[i]>=a[st2[top2]])
110         {
111             tr.upd(st2[top2-1]+1,st2[top2],-a[st2[
112                 top2]]);
113             top2--;
114         }
115         tr.upd(st1[top1]+1,i,-a[i]);
116         st1[++top1]=i;
117         tr.upd(st2[top2]+1,i,a[i]);
118         st2[++top2]=i;
119         id[i]=++tot;
120         l[tot]=r[tot]=i;
121         tmp=tr.ask(1,n);
122         now=tot;
123         while(top&&l[st[top]]>=tmp)
124         {
125             if(type[st[top]]&&check(m[st[top]],i))
126             {
127                 r[st[top]]=i;
128                 add_edge(st[top],now);
129                 now=st[top--];
130             }
131             else if(check(l[st[top]],i))
132             {
133                 type[++tot]=1;
134                 l[tot]=l[st[top]];
135                 r[tot]=i;
136                 m[tot]=l[now];
137                 add_edge(tot,st[top--]);
138                 add_edge(tot,now);
139                 now=tot;
140             }
141             else
142             {
143                 add_edge(++tot,now);
144                 do
145                 {
146                     add_edge(tot,st[top--]);
147                     }while(top&&!check(l[st[top]],i));
148                 l[tot]=l[st[top]];
149                 r[tot]=i;
150                 add_edge(tot,st[top--]);

```

```

149         now=tot;
150     }
151 }
152 st[++top]=now;
153 tr.upd(1,i,-1);
154 }
155 return st[1];
156 }
157 void work(int n,int *a)
158 {
159     int rt=build(n,a);
160 }
161 }
162 }pt;// MAX must *2

```

3 图论

3.1 链式前向星

```

1 //这里指的是边数MAX 双向边要*2
2 int head[MAX],tot;
3 struct node
4 {
5     int to,v,next;
6 }mp[MAX<<1];
7 void init()
8 {
9     mem(head,-1);
10    tot=0;
11 }
12 void add(int x,int y,int v)
13 {
14     mp[tot].v=v;
15     mp[tot].to=y;
16     mp[tot].next=head[x];
17     head[x]=tot++;
18 }

```

3.2 最短路

3.2.1 dijkstra

```

1 struct node
2 {
3     int id;
4     int v;
5     node(){}
6     node(int a,int b) :id(a),v(b){}
7     friend bool operator <(node a,node b){return a.v
8         >b.v;}
9 };
10 vector<node> mp[MAX];
11 bool flag[MAX];

```

```

11 int dis[MAX];
12 void dij(int s)
13 {
14     priority_queue<node> q;
15     node t,to;
16     mem(dis,0x3f);
17     mem(flag,0);
18     dis[s]=0;
19     q.push(node(s,0));
20     while(!q.empty())
21     {
22         t=q.top();
23         q.pop();
24         if(flag[t.id]) continue;
25         flag[t.id]=1;
26         for(int i=0;i<sz(mp[t.id]);i++)
27         {
28             to=mp[t.id][i];
29             if(dis[to.id]>dis[t.id]+to.v)
30             {
31                 dis[to.id]=dis[t.id]+to.v;
32                 q.push(node(to.id,dis[to.id]));
33             }
34         }
35     }
36 }

```

3.2.2 spfa

```

1 //最长路 变为dis-INF 松弛改成<
2 struct node
3 {
4     int id;
5     int v;
6     node(){}
7     node(int a,int b) :id(a),v(b){}
8 };
9 vector<node> mp[MAX];
10 int dis[MAX];
11 bool flag[MAX];
12 void spfa(int s)
13 {
14     queue<node> q;
15     node t,to;
16     mem(dis,0x3f);
17     mem(flag,0);
18     dis[s]=0;
19     flag[s]=1;
20     q.push(node(s,dis[s]));
21     while(!q.empty())
22     {
23         t=q.front();
24         q.pop();
25         flag[t.id]=0;

```



```

26     for(int i=0;i<sz(mp[t.id]);i++)
27     {
28         to=mp[t.id][i];
29         if(dis[to.id]>dis[t.id]+to.v)
30         {
31             dis[to.id]=dis[t.id]+to.v;
32             if(!flag[to.id])
33             {
34                 q.push(node(to.id,dis[to.id]));
35                 flag[to.id]=1;
36             }
37         }
38     }
39 }
40 }

```

3.2.3 floyd 求最小环

```

1 int mp[111][111],dis[111][111],ans;
2 void floyd(int n)
3 {
4     int i,j,k;
5     for(k=1;k<=n;k++)
6     {
7         for(i=1;i<k;i++)
8         {
9             if(mp[k][i]==INF) continue;
10            for(j=i+1;j<k;j++)
11            {
12                if(mp[k][j]==INF) continue;
13                ans=min(ans,mp[k][i]+mp[k][j]+dis[i][j]);
14            }
15        }
16        for(i=1;i<=n;i++)
17        {
18            if(dis[i][k]==INF) continue;
19            for(j=1;j<=n;j++)
20            {
21                if(dis[k][j]==INF) continue;
22                dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
23            }
24        }
25    }
26 }
27 int main()
28 {
29     int m,i,a,b,w,n;
30     while(~scanf("%d%d",&n,&m))
31     {
32         mem(mp,0x3f);
33         mem(dis,0x3f);
34         ans=INF;

```

```

35     while(m--)
36     {
37         scanf("%d%d%d",&a,&b,&w);
38         mp[a][b]=mp[b][a]=dis[a][b]=dis[b][a]=min
            (mp[a][b],w);
39     }
40     floyd(n);
41     if(ans==INF) puts("It's impossible.");
42     else printf("%d\n",ans);
43 }
44 return 0;
45 }

```

3.3 最小生成树

3.3.1 kruskal

```

1 struct dsu
2 {
3     int pre[MAX];
4     void init(int n)
5     {
6         int i;
7         for(i=1;i<=n;i++) pre[i]=i;
8     }
9     int find(int x)
10    {
11        if(pre[x]!=x) pre[x]=find(pre[x]);
12        return pre[x];
13    }
14    bool merge(int a,int b)
15    {
16        int ra,rb;
17        ra=find(a);
18        rb=find(b);
19        if(ra!=rb)
20        {
21            pre[ra]=rb;
22            return 1;
23        }
24        return 0;
25    }
26 }dsu;
27 struct node
28 {
29     int x,y,w;
30     node(){}
31     node(int a,int b,int c):x(a),y(b),w(c){}
32     friend bool operator<(node a,node b) {return a.w<b.w;}
33 };
34 vector<node> edge;
35 int kruskal(int n)
36 {

```

```

37     int res=0;
38     dsu.init(n);
39     sort(all(edge));
40     for(int i=0;i<sz(edge);i++)
41     {
42         if(dsu.merge(edge[i].x,edge[i].y)) res+=edge[
43             i].w;
44     }
45     return res;
46 }

```

3.3.2 prim

```

1 struct node
2 {
3     int id;
4     int v;
5     node(){}
6     node(int a,int b) :id(a),v(b){}
7     friend bool operator <(node a,node b){return a.v
8         >b.v;}
9 };
10 vector<node> mp[MAX];
11 bool flag[MAX];
12 int dis[MAX];
13 int prim()
14 {
15     int res=0;
16     node t,to;
17     priority_queue<node> q;
18     mem(dis,0x3f);
19     mem(flag,0);
20     dis[1]=0;
21     q.push(node(1,dis[1]));
22     while(!q.empty())
23     {
24         t=q.top();
25         q.pop();
26         if(flag[t.id]) continue;
27         flag[t.id]=1;
28         res+=dis[t.id];
29         for(int i=0;i<sz(mp[t.id]);i++)
30         {
31             to=mp[t.id][i];
32             if(!flag[to.id]&&dis[to.id]>to.v)
33             {
34                 dis[to.id]=to.v;
35                 q.push(node(to.id,dis[to.id]));
36             }
37         }
38     }
39     return res;
40 }

```

3.4 二分图匹配

3.4.1 匈牙利算法

```

1 //二分图匹配
2 /*最小点覆盖的点数最大匹配数
3 =最小路径覆盖的边数顶点数
4 =n最大匹配数-最大独立集最小路径覆盖顶点数
5 ==n最大匹配数-
6 */
7 //匈牙利算法 O(n*m)
8 struct Bipartite_Matching
9 {
10     static const int N=;
11     int n,m;
12     VI mp[N];
13     int link[N],s[N];
14     bool used[N],flag[N];
15     void init(int _n,int _m)
16     {
17         n=_n;
18         m=_m;
19         for(int i=0;i<=n;i++) mp[i].clear();
20     }
21     void add_edge(int a,int b){mp[a].pb(b);}
22     bool dfs(int x)
23     {
24         int i,to;
25         flag[x]=1;
26         for(i=0;i<sz(mp[x]);i++)
27         {
28             to=mp[x][i];
29             if(used[to]) continue;
30             used[to]=1;
31             if(link[to]==-1||dfs(link[to]))
32             {
33                 link[to]=x;
34                 s[x]=to;
35                 return 1;
36             }
37         }
38         return 0;
39     }
40     int max_match()
41     {
42         int i,res;
43         mem(link,-1);
44         mem(s,-1);
45         res=0;
46         for(i=1;i<=n;i++)
47         {
48             if(!sz(mp[i])) continue;
49             mem(used,0);
50             if(dfs(i)) res++;
51         }
52     }
53 }

```

```

52     return res;
53 }
54 int min_cover(VI &x, VI &y)
55 {
56     int i, res;
57     res = max_match();
58     mem(flag, 0);
59     mem(used, 0);
60     x.clear();
61     y.clear();
62     for(i=1; i<=n; i++)
63     {
64         if(s[i]==-1) dfs(i);
65     }
66     for(i=1; i<=n; i++)
67     {
68         if(!flag[i]) x.pb(i);
69     }
70     for(i=1; i<=m; i++)
71     {
72         if(used[i]) y.pb(i);
73     }
74     return res;
75 }
76 }bpm;

```

3.4.2 二分图带权匹配

```

1 struct KM
2 {
3     #define type int
4     #define inf INF
5     static const int N=;
6     int n, mx[N], my[N], prv[N];
7     type slk[N], lx[N], ly[N], w[N][N];
8     bool vx[N], vy[N];
9     void init(int _n)
10    {
11        n=_n;
12        for(int i=1; i<=n; i++)
13        {
14            for(int j=1; j<=n; j++)
15            {
16                w[i][j]=0;
17            }
18        }
19    }
20    void add_edge(int x, int y, type val){w[x][y]=val;};
21    void match(int y){while(y) swap(y, mx[my[y]=prv[y]]);};
22    void bfs(int x)
23    {
24        int i, y;

```

```

25    type d;
26    for(i=1; i<=n; i++)
27    {
28        vx[i]=vy[i]=0;
29        slk[i]=inf;
30    }
31    queue<int> q;
32    q.push(x);
33    vx[x]=1;
34    while(1)
35    {
36        while(!q.empty())
37        {
38            x=q.front();
39            q.pop();
40            for(y=1; y<=n; y++)
41            {
42                d=lx[x]+ly[y]-w[x][y];
43                if(!vy[y]&&d<=slk[y])
44                {
45                    prv[y]=x;
46                    if(!d)
47                    {
48                        if(!my[y]) return match(y);
49                        q.push(my[y]);
50                        vx[my[y]]=1;
51                        vy[y]=1;
52                    }
53                    else slk[y]=d;
54                }
55            }
56        }
57        d=inf+1;
58        for(i=1; i<=n; i++)
59        {
60            if(!vy[i]&&slk[i]<d)
61            {
62                d=slk[i];
63                y=i;
64            }
65        }
66        for(i=1; i<=n; i++)
67        {
68            if(vx[i]) lx[i]-=d;
69            if(vy[i]) ly[i]+=d;
70            else slk[i]-=d;
71        }
72        if(!my[y]) return match(y);
73        q.push(my[y]);
74        vx[my[y]]=1;
75        vy[y]=1;
76    }
77 }
78 type max_match()

```

```

79 {
80     int i;
81     type res;
82     for(i=1;i<=n;i++)
83     {
84         mx[i]=my[i]=ly[i]=0;
85         lx[i]=*max_element(w[i]+1,w[i]+n+1);
86     }
87     for(i=1;i<=n;i++) bfs(i);
88     res=0;
89     for(i=1;i<=n;i++) res+=lx[i]+ly[i];
90     return res;
91 }
92 #undef type
93 #undef inf
94 }km;
95 /*
96 O(n^3)
97 km.init(n);
98 km.add_edge(a,b,val); a,b: 1~n
99 */

```

3.5 最大流

3.5.1 dinic

```

1 struct Dinic
2 {
3     #define type int
4     #define inf INF
5     static const int N=;
6     struct node
7     {
8         int from,to;
9         type cap,flow;
10         node(int u,int v,type c,type f):from(u),to(v)
11             ,cap(c),flow(f){}
12     };
13     int n,s,t;
14     vector<node> edge;
15     vector<int> mp[N];
16     int vis[N],dist[N],id[N];
17     void init(int _n)
18     {
19         n=_n;
20         edge.clear();
21         for(int i=0;i<=n;i++)
22         {
23             mp[i].clear();
24             id[i]=dist[i]=vis[i]=0;
25         }
26     }
27     void add_edge(int from,int to,type cap)

```

```

28     edge.pb(node(from,to,cap,0));
29     edge.pb(node(to,from,0,0));
30     int m=edge.size();
31     mp[from].pb(m-2);
32     mp[to].pb(m-1);
33 }
34 bool bfs()
35 {
36     int i,x;
37     mem(vis,0);
38     queue<int> q;
39     q.push(s);
40     dist[s]=0;
41     vis[s]=1;
42     while(!q.empty())
43     {
44         x=q.front();
45         q.pop();
46         for(i=0;i<mp[x].size();i++)
47         {
48             node &e=edge[mp[x][i]];
49             if(!vis[e.to]&&e.cap>e.flow)
50             {
51                 vis[e.to]=1;
52                 dist[e.to]=dist[x]+1;
53                 q.push(e.to);
54             }
55         }
56     }
57     return vis[t];
58 }
59 type dfs(int x,type a)
60 {
61     if(x==t||!a) return a;
62     type flow=0,f;
63     for(int &i=id[x];i<mp[x].size();i++)
64     {
65         node &e=edge[mp[x][i]];
66         if(dist[x]+1==dist[e.to]&&(f=dfs(e.to,min
67             (a,e.cap-e.flow)))>0)
68         {
69             e.flow+=f;
70             edge[mp[x][i]^1].flow-=f;
71             flow+=f;
72             a-=f;
73             if(!a) break;
74         }
75     }
76     return flow;
77 }
78 type max_flow(int _s,int _t)
79 {
80     s=_s;
81     t=_t;

```

```

81     type res=0;
82     while(bfs())
83     {
84         for(int i=0;i<=n;i++) id[i]=0;
85         res+=dfs(s,inf);
86     }
87     return res;
88 }
89 #undef type
90 #undef inf
91 }dc;
92 /*
93 dc.init(n);
94 dc.add_edge(a,b,cap); a,b: 1~n
95 */

```

3.5.2 ISAP

```

1 struct ISAP
2 {
3     #define type int
4     #define inf INF
5     static const int N=;
6     struct node
7     {
8         int from,to;
9         type cap,flow;
10        node(){}
11        node(int u,int v,type c,type f):from(u),to(v)
12            ,cap(c),flow(f){}
13    };
14    int p[N],num[N],cur[N];
15    type d[N];
16    int t,s,n;
17    bool vis[N];
18    vector<int> mp[N];
19    vector<node> edge;
20    void init(int _n)
21    {
22        n=_n;
23        for(int i=0;i<=n;i++)
24        {
25            mp[i].clear();
26            d[i]=inf;
27            vis[i]=num[i]=cur[i]=0;
28        }
29        edge.clear();
30    }
31    void add(int from,int to,type cap)
32    {
33        edge.pb(node(from,to,cap,0));
34        edge.pb(node(to,from,0,0));
35        int m=edge.size();
36        mp[from].pb(m-2);

```

```

36        mp[to].pb(m-1);
37    }
38    bool bfs()
39    {
40        queue<int> q;
41        d[t]=0;
42        vis[t]=1;
43        q.push(t);
44        while(!q.empty())
45        {
46            int u=q.front();
47            q.pop();
48            for(int i=0;i<mp[u].size();i++)
49            {
50                node &e=edge[mp[u][i]^1];
51                if(!vis[e.from]&&e.cap>e.flow)
52                {
53                    vis[e.from]=true;
54                    d[e.from]=d[u]+1;
55                    q.push(e.from);
56                }
57            }
58        }
59        return vis[s];
60    }
61    type augment()
62    {
63        int u=t;
64        type flow=inf;
65        while(u!=s)
66        {
67            node &e=edge[p[u]];
68            flow=min(flow,e.cap-e.flow);
69            u=edge[p[u]].from;
70        }
71        u=t;
72        while(u!=s)
73        {
74            edge[p[u]].flow+=flow;
75            edge[p[u]^1].flow-=flow;
76            u=edge[p[u]].from;
77        }
78        return flow;
79    }
80    type maxflow(int _s,int _t)
81    {
82        s=_s;
83        t=_t;
84        type flow=0;
85        bfs();
86        if(d[s]>=n) return 0;
87        for(int i=0;i<n;i++)
88        {
89            if(d[i]<inf) num[d[i]]++;

```

```

90     }
91     int u=s;
92     while(d[s]<n)
93     {
94         if(u==t)
95         {
96             flow+=augment();
97             u=s;
98         }
99         bool ok=false;
100         for(int i=cur[u];i<mp[u].size();i++)
101         {
102             node &e=edge[mp[u][i]];
103             if(e.cap>e.flow&&d[u]==d[e.to]+1)
104             {
105                 ok=true;
106                 p[e.to]=mp[u][i];
107                 cur[u]=i;
108                 u=e.to;
109                 break;
110             }
111         }
112         if(!ok)
113         {
114             type mn=n-1;
115             for(int i=0;i<mp[u].size();i++)
116             {
117                 node &e=edge[mp[u][i]];
118                 if(e.cap>e.flow) mn=min(mn,d[e.to])
119                     ;
120             }
121             if(--num[d[u]]==0) break;
122             num[d[u]=mn+1]++;
123             cur[u]=0;
124             if(u!=s) u=edge[p[u]].from;
125         }
126         return flow;
127     }
128     #undef type
129     #undef inf
130 }isap;

```

3.5.3 high-level-preflow-push

```

1 struct High_Level_Preflow_Push
2 {
3     static const int N=10010;
4     struct node{int v,cap,index;};
5     vector<node> edge[N];
6     vector<int> List[N];
7     vector<list<int>::iterator> listit;
8     list<int> dlist[N];

```

```

9     int highest,highestActive,vis[N],excess[N],
        height[N],n;
10    void init(int _n)
11    {
12        n=_n;
13        for(int i=0;i<=n;i++) edge[i].clear();
14    }
15    void add_edge(int u,int v,int cap)
16    {
17        edge[u].push_back(node{v,cap,edge[v].size()})
18        ;
19        edge[v].push_back(node{u,0,edge[u].size()-1})
20        ;
21    }
22    void globalRelabel(int t)
23    {
24        int u,i,hp,v,index;
25        queue<int> q;
26        for(i=0;i<=n;i++)
27        {
28            height[i]=n;
29            List[i].clear();
30            dlist[i].clear();
31            vis[i]=0;
32        }
33        height[t]=0;
34        q.push(t);
35        while(!q.empty())
36        {
37            u=q.front();
38            q.pop();
39            for(i=0;i<edge[u].size();i++)
40            {
41                v=edge[u][i].v;
42                index=edge[u][i].index;
43                if(height[v]==n&&edge[v][index].cap>0)
44                {
45                    height[v]=height[u]+1;
46                    vis[height[v]]++;
47                    q.push(hp=v);
48                }
49            }
50        }
51        for(i=0;i<=n;i++)
52        {
53            if(height[i]<n)
54            {
55                listit[i]=dlist[height[i]].insert(
56                    dlist[height[i]].begin(),i);
57                if(excess[i]>0) List[height[i]].
                    push_back(i);
58            }
59        }
60        highest=height[hp];

```

```

58     highestActive=height[hp];
59 }
60 void push(int u,node &e)
61 {
62     int v,df;
63     v=e.v;
64     df=min(excess[u],e.cap);
65     e.cap=e.cap-df;
66     edge[v][e.index].cap=edge[v][e.index].cap+df;
67     excess[u]=excess[u]-df;
68     excess[v]=excess[v]+df;
69     if(excess[v]>0&&excess[v]<=df) List[height[v]
70         ].push_back(v);
71 }
72 void discharge(int u)
73 {
74     int i,nh,v,cap,h;
75     nh=n;
76     for(i=0;i<edge[u].size();i++)
77     {
78         v=edge[u][i].v;
79         cap=edge[u][i].cap;
80         if(cap>0)
81         {
82             if(height[u]==height[v]+1)
83             {
84                 push(u,edge[u][i]);
85                 if(excess[u]==0) return;
86             }
87             else nh=min(nh,height[v]+1);
88         }
89     }
90     h=height[u];
91     if(vis[h]==1)
92     {
93         for(i=h;i<=highest;i++)
94         {
95             for(list<int>::iterator it=dlist[i].
96                 begin();it!=dlist[i].end();it++)
97             {
98                 vis[height[*it]]--;
99                 height[*it]=n;
100             }
101             dlist[i].clear();
102         }
103         highest=h-1;
104     }
105     else
106     {
107         vis[h]--;
108         listit[u]=dlist[h].erase(listit[u]);
109         height[u]=nh;
110         if(nh==n) return;
111         vis[nh]++;

```

```

110         listit[u]=dlist[nh].insert(dlist[nh].
111             begin(),u);
112         highestActive=nh;
113         highest=max(highest,highestActive);
114         List[nh].push_back(u);
115     }
116 }
117 int maxflow(int s,int e)
118 {
119     int i,u;
120     if(s==e) return 0;
121     highestActive=0;
122     highest=0;
123     for(i=0;i<n;i++) height[i]=vis[i]=excess[i]
124         ]=0;
125     height[s]=n;
126     listit.resize(n);
127     for(i=0;i<n;i++)
128     {
129         if(i!=s)
130         {
131             listit[i]=dlist[height[i]].insert(
132                 dlist[height[i]].begin(),i);
133         }
134     }
135     vis[0]=n-1;
136     excess[s]=INF;
137     excess[e]=-INF;
138     for(i=0;i<edge[s].size();i++) push(s,edge[s][
139         i]);
140     globalRelabel(e);
141     while(highestActive>=0)
142     {
143         if(List[highestActive].empty()==1)
144         {
145             highestActive--;
146             continue;
147         }
148         u=List[highestActive].back();
149         List[highestActive].pop_back();
150         discharge(u);
151     }
152     return excess[e]+INF;
153 }
154 }hlpp;

```

3.6 最小费用最大流

3.6.1 spfa 费用流

```

1 struct MCMF
2 {
3     #define type int
4     #define inf INF

```

```

5  static const int N=;
6  struct node
7  {
8      int from,to;
9      type cap,flow,cost;
10     node(){}
11     node(int u,int v,type c,type f,type co):from(
12         u),to(v),cap(c),flow(f),cost(co){}
13 };
14 int n,s,t;
15 vector<node> edge;
16 vector<int> mp[N];
17 int vis[N],id[N];
18 type d[N],a[N];
19 void init(int _n)
20 {
21     n=_n;
22     for(int i=0;i<=n;i++) mp[i].clear();
23     edge.clear();
24 }
25 void add_edge(int from,int to,type cap,type cost
26     =0)
27 {
28     edge.pb(node(from,to,cap,0,cost));
29     edge.pb(node(to,from,0,0,-cost));
30     int m=edge.size();
31     mp[from].pb(m-2);
32     mp[to].pb(m-1);
33 }
34 bool spfa(type& flow,type& cost)
35 {
36     for(int i=0;i<=n;i++)
37     {
38         d[i]=inf;
39         vis[i]=0;
40     }
41     d[s]=0;vis[s]=1;id[s]=0;a[s]=inf;
42     queue<int> q;
43     q.push(s);
44     while(!q.empty())
45     {
46         int x=q.front();
47         q.pop();
48         vis[x]=0;
49         for(int i=0;i<mp[x].size();i++)
50         {
51             node& e=edge[mp[x][i]];
52             int to=e.to;
53             if(e.cap>e.flow&& d[to]>d[x]+e.cost)
54             {
55                 d[to]=d[x]+e.cost;
56                 a[to]=min(a[x],e.cap-e.flow);
57                 id[to]=mp[x][i];
58                 if(!vis[to])

```

```

57         {
58             vis[to]=1;
59             q.push(to);
60         }
61     }
62 }
63 }
64 if(d[t]==inf) return false;
65 flow+=a[t];
66 cost+=a[t]*d[t];
67 int x=t;
68 while(x!=s)
69 {
70     edge[id[x]].flow+=a[t];
71     edge[id[x]^1].flow-=a[t];
72     x=edge[id[x]].from;
73 }
74 return true;
75 }
76 pair<type,type> mincost_maxflow(int _s,int _t)
77 {
78     type flow=0,cost=0;
79     s=_s;
80     t=_t;
81     while(spfa(flow,cost));
82     return MP(cost,flow);
83 }
84 #undef type
85 #undef inf
86 }mcmf;
87 /*
88 mcmf.init(n);
89 mcmf.add_edge(a,b,cap,cost); a,b: 1~n
90 */

```

3.6.2 dijkstra 费用流

```

1  struct MCMF_dij
2  {
3      #define type int
4      #define inf INF
5      #define PTI pair<type,int>
6      static const int N=;
7      struct node
8      {
9          int from,to;
10         type flow,cost;
11         node(){}
12         node(int u,int v,type f,type co):from(u),to(v
13             ),flow(f),cost(co){}
14     };
15     int n,s,t,id[N];
16     vector<node> edge;
17     vector<int> mp[N];

```



```

17  type dis[N],h[N];
18  void init(int _n)
19  {
20      n=_n;
21      for(int i=0;i<=n;i++) mp[i].clear();
22      edge.clear();
23  }
24  void add_edge(int from,int to,type cap,type cost
25      =0)
26  {
27      edge.pb(node(from,to,cap,cost));
28      edge.pb(node(to,from,0,-cost));
29      int m=edge.size();
30      mp[from].pb(m-2);
31      mp[to].pb(m-1);
32  }
33  bool dij(type& flow,type& cost)
34  {
35      for(int i=0;i<=n;i++) dis[i]=inf;
36      dis[s]=0;id[s]=0;
37      priority_queue<PTI ,vector<PTI>,greater<PTI>
38          > q;
39      q.push(MP(type(0),s));
40      while(!q.empty())
41      {
42          PTI x=q.top();
43          q.pop();
44          if(x.fi!=dis[x.se]) continue;
45          if(x.se==t) break;
46          for(int i=0;i<mp[x.se].size();i++)
47          {
48              node& e=edge[mp[x.se][i]];
49              int to=e.to;
50              type now_cost=e.cost+h[x.se]-h[to];
51              if(e.flow>0&&dis[to]>dis[x.se]+
52                  now_cost)
53              {
54                  dis[to]=dis[x.se]+now_cost;
55                  q.push(MP(dis[to],to));
56                  e.from=x.se;
57                  id[to]=mp[x.se][i];
58              }
59          }
60      }
61      if(dis[t]==inf) return false;
62      for(int i=0;i<=n;i++) h[i]=min(h[i]+dis[i],
63          inf);
64      type new_flow=inf;
65      int x=t;
66      while(x!=s)
67      {
68          new_flow=min(new_flow,edge[id[x]].flow);
69          x=edge[id[x]].from;
70      }

```

```

67      flow+=new_flow;
68      cost+=new_flow*h[t];
69      x=t;
70      while(x!=s)
71      {
72          edge[id[x]].flow-=new_flow;
73          edge[id[x]^1].flow+=new_flow;
74          x=edge[id[x]].from;
75      }
76      return true;
77  }
78  pair<type,type> mincost_maxflow(int _s,int _t)
79  {
80      type flow=0,cost=0;
81      for(int i=0;i<=n;i++) h[i]=0;
82      s=_s;
83      t=_t;
84      while(dij(flow,cost));
85      return MP(cost,flow);
86  }
87  #undef type
88  #undef inf
89  #undef PTI
90 }mcmf;

```

3.7 强连通分量

```

1  int scc,top,tot;
2  vector<int> mp[MAX];
3  int low[MAX],dfn[MAX],belong[MAX];
4  int stk[MAX],flag[MAX];
5  void init(int n)
6  {
7      int i;
8      for(i=1;i<=n;i++)
9      {
10          mp[i].clear();
11          low[i]=0;
12          dfn[i]=0;
13          stk[i]=0;
14          flag[i]=0;
15      }
16      scc=top=tot=0;
17  }
18  void tarjan(int x)
19  {
20      int to,i,temp;
21      stk[top++]=x;
22      flag[x]=1;
23      low[x]=dfn[x]=++tot;
24      for(i=0;i<mp[x].size();i++)
25      {
26          to=mp[x][i];

```

```

27     if(!dfn[to])
28     {
29         tarjan(to);
30         low[x]=min(low[x],low[to]);
31     }
32     else if(flag[to]) low[x]=min(low[x],dfn[to]);
33 }
34 if(low[x]==dfn[x])
35 {
36     scc++;
37     do
38     {
39         temp=stk[--top];
40         flag[temp]=0;
41         belong[temp]=scc;
42     }while(temp!=x);
43 }
44 }

```

3.8 双联通分量

3.8.1 边双连通

```

1 namespace Tarjan
2 {
3     int bcc,top,tot,n;
4     vector<int> mp[MAX];
5     vector<PII > bridge;
6     int low[MAX],dfn[MAX],belong[MAX],fa[MAX];
7     int stk[MAX];
8     int cut[MAX],add_block[MAX];
9     void dfs(int x,int pre)
10    {
11        int to,i,tmp,k,son;
12        stk[top++]=x;
13        low[x]=dfn[x]=++tot;
14        fa[x]=pre;
15        son=k=0;
16        for(auto to:mp[x])
17        {
18            if(to==pre&&!k)
19            {
20                k++;
21                continue;
22            }
23            if(!dfn[to])
24            {
25                son++;
26                dfs(to,x);
27                low[x]=min(low[x],low[to]);
28                if(x!=pre&&low[to]>=dfn[x])
29                {
30                    cut[x]=1;
31                    add_block[x]++;

```

```

32                }
33                if(low[to]>dfn[x]) bridge.pb(MP(x,to))
34                ;
35            }
36            else low[x]=min(low[x],dfn[to]);
37        }
38        if(x==pre&&son>1)
39        {
40            cut[x]=1;
41            add_block[x]=son-1;
42        }
43        if(low[x]==dfn[x])
44        {
45            bcc++;
46            do
47            {
48                tmp=stk[--top];
49                belong[tmp]=bcc;
50            }while(tmp!=x);
51        }
52    }
53    void work(int _n,vector<int> e[])
54    {
55        n=_n;
56        for(int i=1;i<=n;i++)
57        {
58            mp[i]=e[i];
59            low[i]=dfn[i]=fa[i]=stk[i]=0;
60            cut[i]=add_block[i]=0;
61        }
62        bcc=top=tot=0;
63        bridge.clear();
64        for(int i=1;i<=n;i++)
65        {
66            if(!dfn[i]) dfs(i,i);
67        }
68    }
69    void rebuild(vector<int> e[])
70    {
71        int i,t;
72        for(i=1;i<=n;i++) e[i].clear();
73        for(i=1;i<=n;i++)
74        {
75            t=fa[i];
76            if(belong[i]!=belong[t])
77            {
78                e[belong[i]].pb(belong[t]);
79                e[belong[t]].pb(belong[i]);
80            }
81        }
82    }

```

3.9 团

3.9.1 最大团

```

1 struct Maximum_Clique
2 {
3     static const int N=;
4     vector<int> sol; // vertex of maximum clique
5     int mp[N][N/30+1],s[N][N/30+1];
6     int n,ans,dp[N];
7     void init(int _n)
8     {
9         n=_n;
10        for(int i=0;i<=n;i++)
11        {
12            dp[i]=0;
13            mem(mp[i],0);
14        }
15    }
16    void add_edge(int a,int b) //0~n-1
17    {
18        if(a>b) swap(a,b);
19        if(a==b) return;
20        mp[a][b/32]|=(1<<(b%32));
21    }
22    bool dfs(int x,int k)
23    {
24        int c=0,d=0;
25        for(int i=0;i<(n+31)/32;i++)
26        {
27            s[k][i]=mp[x][i];
28            if(k!=1) s[k][i]&=s[k-1][i];
29            c+=__builtin_popcount(s[k][i]);
30        }
31        if(c==0)
32        {
33            if(k>ans)
34            {
35                ans=k;
36                sol.clear();
37                sol.pb(x);
38                return 1;
39            }
40            return 0;
41        }
42        for(int i=0;i<(n+31)/32;i++)
43        {
44            for(int a=s[k][i];a;d++)
45            {
46                if(k+(c-d)<=ans) return 0;
47                int lb=a&(-a),lg=0;
48                a^=lb;
49                while(lb!=1)
50                {
51                    lb=(unsigned int)(lb)>>1;

```

```

52                lg++;
53            }
54            int u=i*32+lg;
55            if(k+dp[u]<=ans) return 0;
56            if(dfs(u,k+1))
57            {
58                sol.pb(x);
59                return 1;
60            }
61        }
62    }
63    return 0;
64    }
65    int maximum_clique()
66    {
67        ans=0;
68        for(int i=n-1;i>=0;i--)
69        {
70            dfs(i,1);
71            dp[i]=ans;
72        }
73        return ans;
74    }
75    }mcp;
76    /*
77    undirected graph
78    mcp.init(n);
79    mcp.add_edge(a,b); a,b: 0~n-1
80    */

```

3.9.2 极大团计数

```

1 struct Bron_Kerbosch
2 {
3     static const int N=;
4     bitset<N> MASK,ZERO,mp[N];
5     int n,cnt_clique;
6     void init(int _n)
7     {
8         n=_n;
9         for(int i=0;i<=n;i++) mp[i].reset();
10        ZERO.reset();
11        MASK=ZERO;
12        MASK.flip();
13    }
14    void add_edge(int a,int b) //0~n-1 , undir
15    {
16        if(a==b) return;
17        mp[a][b]=mp[b][a]=1;
18    }
19    void dfs(bitset<N> now,bitset<N> some,bitset<N>
20        none)

```

```

21     if(some.none() && none.none()) // one maximal
22         clique
23     {
24         cnt_clique++;
25         return;
26     }
27     bitset<N> r = some;
28     bool fi = 1;
29     for(int i = 0; i < n; i++)
30     {
31         if(!r[i]) continue;
32         if(fi)
33         {
34             fi = 0;
35             r &= mp[i] ^ MASK;
36         }
37         now[i] = 1;
38         dfs(now, some & mp[i], none & mp[i]);
39         now[i] = 0;
40         some[i] = 0;
41         none[i] = 1;
42     }
43     int count_maximal_clique()
44     {
45         cnt_clique = 0;
46         bitset<N> now;
47         dfs(now, MASK, ZERO);
48         return cnt_clique;
49     }
50 }bk;
51 /*
52 undirected graph
53 bk.init(n);
54 bk.add_edge(a,b); a,b: 0~n-1
55 */

```

3.10 拓扑排序

```

1 VI mp[MAX];
2 int degree[MAX];
3 VI toplist;
4 bool topsort(int n)
5 {
6     int i, x;
7     queue<int> q;
8     for(i = 1; i <= n; i++)
9     {
10         if(!degree[i]) q.push(i);
11     }
12     toplist.clear();
13     while(!q.empty())
14     {

```

```

15         x = q.front();
16         q.pop();
17         toplist.pb(x);
18         for(auto to : mp[x])
19         {
20             degree[to]--;
21             if(!degree[to]) q.push(to);
22         }
23     }
24     return ! (sz(toplist) == n);
25 }

```

3.11 2-sat

3.11.1 2-sat 输出任意解

```

1 // 判断是否有解 输出任意一组解 O(n+m)
2 int scc, top, tot;
3 vector<int> mp[MAX];
4 int low[MAX], dfn[MAX], belong[MAX];
5 int stk[MAX], flag[MAX];
6 int pos[MAX], degree[MAX], ans[MAX], outflag[MAX], cnt;
7 vector<int> dag[MAX];
8 void init(int n)
9 {
10     int i;
11     for(i = 0; i < 2*n; i++)
12     {
13         mp[i].clear();
14         dag[i].clear();
15         low[i] = 0;
16         dfn[i] = 0;
17         stk[i] = 0;
18         flag[i] = 0;
19         degree[i] = 0;
20         outflag[i] = 0;
21     }
22     scc = top = tot = 0;
23 }
24 void tarjan(int x)
25 {
26     int to, i, temp;
27     stk[top++] = x;
28     flag[x] = 1;
29     low[x] = dfn[x] = ++tot;
30     for(i = 0; i < mp[x].size(); i++)
31     {
32         to = mp[x][i];
33         if(!dfn[to])
34         {
35             tarjan(to);
36             low[x] = min(low[x], low[to]);
37         }
38         else if(flag[to]) low[x] = min(low[x], dfn[to]);

```

```

39     }
40     if(low[x]==dfn[x])
41     {
42         scc++;
43         do
44         {
45             temp=stk[--top];
46             flag[temp]=0;
47             belong[temp]=scc;
48         }while(temp!=x);
49     }
50 }
51 void add(int x,int y)
52 {
53     mp[x].pb(y);
54 }
55 void topsort(int n)
56 {
57     int i,t;
58     queue<int> q;
59     cnt=0;
60     for(i=1;i<=scc;i++)
61     {
62         if(degree[i]==0) q.push(i);
63         outflag[i]=0;
64     }
65     while(!q.empty())
66     {
67         t=q.front();
68         q.pop();
69         if(outflag[t]==0)
70         {
71             outflag[t]=1;
72             outflag[pos[t]]=2;
73         }
74         for(i=0;i<sz(dag[t]);i++)
75         {
76             int to=dag[t][i];
77             degree[to]--;
78             if(degree[to]==0) q.push(to);
79         }
80     }
81 }
82 void bulddag(int n)
83 {
84     int i,j,to;
85     for(i=0;i<2*n;i++)
86     {
87         for(j=0;j<sz(mp[i]);j++)
88         {
89             to=mp[i][j];
90             if(belong[i]!=belong[to])
91             {
92                 degree[belong[i]]++;

```

```

93                 dag[belong[to]].pb(belong[i]);
94             }
95         }
96     }
97 }
98 void twosat(int n)
99 {
100     int i;
101     for(i=0;i<2*n;i++)
102     {
103         if(!dfn[i]) tarjan(i);
104     }
105     for(i=0;i<n;i++)
106     {
107         if(belong[2*i]==belong[2*i+1])//无解
108         {
109             puts("NO");
110             return;
111         }
112         pos[belong[2*i]]=belong[2*i+1];
113         pos[belong[2*i+1]]=belong[2*i];
114     }
115     bulddag(n);
116     topsort(n);
117     cnt=0;
118     for(i=0;i<2*n;i++)
119     {
120         if(outflag[belong[i]]==1) ans[cnt++]=i+1;
121     }
122     for(i=0;i<cnt;i++)
123     {
124         printf("%d\n",ans[i]);
125     }
126 }

```

3.11.2 2-sat 字典序最小解

```

1 //判断是否有解 输出字典序最小的解O(n*m)
2 vector<int> mp[MAX];
3 bool flag[MAX];
4 int cnt,s[MAX];
5 void init(int n)
6 {
7     int i;
8     for(i=0;i<2*n;i++)
9     {
10         mp[i].clear();
11     }
12     mem(flag,0);
13 }
14 bool dfs(int x)
15 {
16     int i;
17     if(flag[x^1]) return 0;

```

```

18     if(flag[x]) return 1;
19     s[cnt++]=x;
20     flag[x]=1;
21     for(i=0;i<sz(mp[x]);i++)
22     {
23         if(!dfs(mp[x][i])) return 0;
24     }
25     return 1;
26 }
27 void twosat(int n)
28 {
29     int i;
30     for(i=0;i<2*n;i++)
31     {
32         if(!flag[i]&&!flag[i^1])
33         {
34             cnt=0;
35             if(!dfs(i))
36             {
37                 while(cnt) flag[s[--cnt]]=0;
38                 if(!dfs(i^1))//无解
39                 {
40                     puts("NO");
41                     return;
42                 }
43             }
44         }
45     }
46     for(i=0;i<2*n;i+=2)
47     {
48         if(flag[i]) printf("%d\n",i+1);
49         else printf("%d\n",i+2);
50     }
51 }

```

3.12 支配树

```

1 struct dominator_tree
2 {
3     int n,tot,dfn[MAX],best[MAX],semi[MAX],idom[MAX],id[MAX],fa[MAX];
4     VI nex[MAX],pre[MAX],tmp[MAX],son[MAX];
5     void init(int _n)
6     {
7         n=_n;
8         for(int i=0;i<=n;i++)
9         {
10             nex[i].clear();
11             pre[i].clear();
12             tmp[i].clear();
13             son[i].clear();
14             dfn[i]=0;
15             idom[i]=semi[i]=best[i]=fa[i]=i;

```

```

16         }
17     }
18     void add_edge(int x,int y)
19     {
20         nex[x].pb(y);
21         pre[y].pb(x);
22     }
23     int ckmin(int x,int y){return dfn[semi[x]]<dfn[semi[y]]?x:y;}
24     int getfa(int k)
25     {
26         if(k==fa[k]) return k;
27         int ret=getfa(fa[k]);
28         best[k]=ckmin(best[fa[k]],best[k]);
29         return fa[k]=ret;
30     }
31     void dfs(int x)
32     {
33         dfn[x]=++tot;
34         id[tot]=x;
35         for(auto to:nex[x])
36         {
37             if(dfn[to]) continue;
38             dfs(to);
39             son[x].pb(to);
40         }
41     }
42     void tarjan(VI mp[])
43     {
44         int i,j,k;
45         for(i=tot;i-->0)
46         {
47             k=id[i];
48             for(auto to:pre[k])
49             {
50                 if(!dfn[to]) continue;
51                 if(dfn[to]<dfn[k])
52                 {
53                     if(dfn[to]<dfn[semi[k]]) semi[k]=to;
54                 }
55                 else
56                 {
57                     getfa(to);
58                     semi[k]=semi[ckmin(best[to],k)];
59                 }
60             }
61             if(k!=semi[k]) tmp[semi[k]].pb(k);
62             for(auto to:tmp[k])
63             {
64                 getfa(to);
65                 if(semi[best[to]]==k) idom[to]=k;
66                 else idom[to]=best[to];
67             }

```

```

68     for(auto to:son[k]) fa[to]=k;
69 }
70 for(i=2;i<=tot;i++)
71 {
72     k=id[i];
73     if(idom[k]!=semi[k]) idom[k]=idom[idom[k]
74         ]];
75     if (k!=idom[k])
76     {
77         mp[idom[k]].pb(k); //add edge
78     }
79 }
80 }
81 void work(int rt,VI mp[])
82 {
83     for(int i=0;i<=n;i++) mp[i].clear();
84     tot=0;
85     dfs(rt);
86     tarjan(mp);
87 }
88 }dt;
89 /*
90 dt.init(n);
91 dt.add_edge(a,b); // DAG
92 dt.work(rt,mp);
93 */

```

4 数论

4.1 素数筛

4.1.1 埃筛

```

1 //x is a prime if prime[x]==x(x>=2)
2 int p[MAX],tot,prime[MAX];
3 void init(int n)
4 {
5     int i,j;
6     tot=0;
7     mem(prime,0);
8     prime[1]=1;
9     for(i=2;i<=n;i++)
10     {
11         if(prime[i]) continue;
12         p[tot++]=i;
13         for(j=i;j<=n;j+=i)
14         {
15             if(!prime[j]) prime[j]=i;
16         }
17     }
18 }

```

4.1.2 线性筛

```

1 //x is a prime if prime[x]==x(x>=2)
2 int p[MAX],tot,prime[MAX];
3 void init(int n)
4 {
5     int i,j;
6     tot=0;
7     mem(prime,0);
8     prime[1]=1;
9     for(i=2;i<=n;i++)
10     {
11         if(!prime[i]) prime[i]=p[tot++]=i;
12         for(j=0;j<tot&&p[j]*i<=n;j++)
13         {
14             prime[i*p[j]]=p[j];
15             if(i%p[j]==0) break;
16         }
17     }
18 }

```

4.1.3 区间筛

```

1 //O(r-l+1)
2 ll p[MAX],tot;
3 bool flag[MAX],prime[MAX];
4 void init(ll l,ll r)
5 {
6     ll i,j,sq=sqrt(r+0.5);
7     tot=0;
8     for(i=0;i<=sq;i++) flag[i]=1;
9     for(i=1;i<=r;i++) prime[i-1]=1;
10    if(l==0) prime[0]=prime[1]=0;
11    if(l==1) prime[0]=0;
12    for(i=2;i<=sq;i++)
13    {
14        if(!flag[i]) continue;
15        for(j=i+i;j<=sq;j+=i) flag[j]=0;
16        for(j=max(2LL,(l+i-1)/i)*i;j<=r;j+=i) prime[j-1]=0;
17    }
18    for(i=1;i<=r;i++)
19    {
20        if(prime[i-1]) p[tot++]=i;
21    }
22 }

```

4.2 扩展欧几里得

4.2.1 exgcd

```

1 /*解
2 xa+yb=gcd(a,b) 返回值为

```

```

3 gcd(a,b) 其中一组解为
4 x y 通解
5 :
6     x1=x+b/gcd(a,b)*t
7     y1=y-a/gcd(a,b)*t
8     (为任意整数t)
9 */
10 ll exgcd(ll a,ll b,ll &x,ll &y)
11 {
12     if(b==0)
13     {
14         x=1;
15         y=0;
16         return a;
17     }
18     ll g,t;
19     g=exgcd(b,a%b,x,y);
20     t=x;
21     x=y;
22     y=t-a/b*y;
23     return g;
24 }

```

4.2.2 $ax+by=c$

```

1 /*
2 xa+yb=c 求正整数最小的一组解
3 x 有解条件
4 c%gcd(a,b)==0
5 */
6 ll linear_equation(ll a,ll b,ll c,ll &x,ll &y)
7 {
8     ll g,t;
9     g=exgcd(a,b,x,y);
10    if(!c) x=y=0;
11    else if((!a&&!b&&c)||c%g) return -1;//no
12           solution
13    else if(!a&&b) x=1,y=c/b;
14    else if(a&&!b) x=c/a,y=-c/a;
15    else
16    {
17        a/=g,b/=g,c/=g;
18        x*=c,y*=c;
19        t=x;
20        x%=b;
21        if(x<=0) x+=b;//or x<0
22        ll k=(t-x)/b;
23        y+=k*a;
24    }
25    return g;
26 }

```

4.2.3 exgcd 求逆元

```

1 /*扩展欧几里得求逆元条件
2
3 :gcd(a,mod)==1 如果
4 gcd(a,mod)!=1 返回 -1
5 */
6 ll inv(ll a,ll p)
7 {
8     ll g,x,y;
9     g=exgcd(a,p,x,y);
10    return g==1?(x+p)%p:-1;
11 }

```

4.3 中国剩余定理

4.3.1 CRT

```

1 //是除数m 是余数r 是除数的pLCM也就是答案的循环节()
2 int CRT(int *m,int *r,int n)
3 {
4     int p=m[0],res=r[0],x,y,g;
5     for(int i=1;i<n;i++)
6     {
7         g=exgcd(p,m[i],x,y);
8         if((r[i]-res)%g) return -1;//无解
9         x=(r[i]-res)/g*x%(m[i]/g);
10        res+=x*p;
11        p=p/g*m[i];
12        res%=p;
13    }
14    return res>0?res:res+p;
15 }

```

4.3.2 exCRT

```

1 namespace exCRT
2 {
3     ll excrt(VL a,VL b)//res=a_i(mod b_i)
4     {
5         ll x,y,k,g,c,p,res,bg;
6         assert(sz(a)==sz(b));
7         assert(sz(a)>0);
8         p=b[0];
9         res=a[0];
10        for(int i=1;i<sz(a);i++)
11        {
12            c=(a[i]-res*b[i]+b[i])%b[i];
13            g=exgcd(p,b[i],x,y);
14            bg=b[i]/g;
15            if(c%g!=0) return -1;
16            x=(x*(c/g))%bg;
17            res+=x*p;

```



```

18         p*=bg;
19         res=(res%p+p)%p;
20     }
21     return (res%p+p)%p;
22 }
23 };

```

4.4 组合数

4.4.1 打表

```

1 ll C[1010][1010];
2 void init(int n)
3 {
4     int i,j;
5     for(i=(C[0][0]=1);i<=n;i++)
6     {
7         for(j=(C[i][0]=1);j<=n;j++)
8         {
9             C[i][j]=(C[i-1][j]+C[i-1][j-1])%mod;
10        }
11    }
12 }

```

4.4.2 预处理

```

1 ll pow2(ll a,ll b)
2 {
3     ll res=1;
4     while(b)
5     {
6         if(b&1) res=res*a%mod;
7         a=a*a%mod;
8         b>>=1;
9     }
10    return res;
11 }
12 ll inv(ll x){return pow2(x,mod-2);}
13 ll fac[MAX],invfac[MAX];
14 void init(int n)
15 {
16     fac[0]=1;
17     for(int i=1;i<=n;i++) fac[i]=fac[i-1]*i%mod;
18     invfac[n]=inv(fac[n]);
19     for(int i=n-1;~i;i--) invfac[i]=invfac[i+1]*(i
        +1)%mod;
20 }
21 ll C(ll n,ll m)
22 {
23     if(m>n||m<0||n<0) return 0;
24     return fac[n]*invfac[m]%mod*invfac[n-m]%mod;
25 }

```

4.4.3 Lucas 定理

```

1 //C(n,m) n,m<=1e18 p<=1e5
2 //p must be a prime number
3 ll Lucas(ll n,ll m,ll p)
4 {
5     if(m==0) return 1;
6     return C(n%p,m%p)*Lucas(n/p,m/p,p)%p;
7 }

```

4.4.4 exLucas

```

1 namespace exLucas
2 {
3     ll pow2(ll a,ll b,ll p)
4     {
5         ll res=1;
6         while(b>0)
7         {
8             if(b&1) res=res*a%p;
9             a=a*a%p;
10            b>>=1;
11        }
12        return res;
13    }
14    ll inv(ll a,ll p)
15    {
16        ll g,x,y,res;
17        g=exgcd(a,p,x,y);
18        res=(g==1?(x+p)%p:-1);
19        assert(res!=-1);
20        return res;
21    }
22    map<ll,pair<VL,VL> > mp;
23    map<PLL,VL > fac;
24    void init(VL mod_list)
25    {
26        ll i,j,p;
27        mp.clear();
28        fac.clear();
29        for(auto mod_i:mod_list)
30        {
31            p=mod_i;
32            VL a,b;
33            for(i=2;i*i<=p;i++)
34            {
35                if(p%i) continue;
36                b.pb(1LL);
37                while(p%i==0) b[sz(b)-1]*=i,p/=i;
38                a.pb(i);
39            }
40            if(p>1) a.pb(p),b.pb(p);
41            mp[mod_i]=MP(a,b);
42            for(i=0;i<sz(a);i++)

```

```

43     {
44         if(fac.count(MP(a[i],b[i]))) continue;
45         VL fac_tmp=VL(b[i]+1);
46         fac_tmp[0]=1;
47         for(j=1;j<=b[i];j++)
48         {
49             if(j%a[i]) fac_tmp[j]=fac_tmp[j-1]*
                    j%b[i];
50             else fac_tmp[j]=fac_tmp[j-1];
51         }
52         fac[MP(a[i],b[i])]=fac_tmp;
53     }
54 }
55
56 ll cal_fac(ll n,ll x,ll p)
57 {
58     if(!n) return 1LL;
59     ll res=1;
60     assert(fac.count(MP(x,p)));
61     res=res*fac[MP(x,p)][p-1]%p;
62     res=pow2(res,n/p,p);
63     res=res*fac[MP(x,p)][n%p]%p;
64     return res*cal_fac(n/x,x,p)%p;
65 }
66
67 ll multilucas(ll n,ll m,ll x,ll p)
68 {
69     if(m>n) return 0;
70     ll i,cnt;
71     cnt=0;
72     for(i=n;i/=x) cnt+=i/x;
73     for(i=m;i/=x) cnt-=i/x;
74     for(i=n-m;i/=x) cnt-=i/x;
75     return pow2(x,cnt,p)* \
76             cal_fac(n,x,p)%p* \
77             inv(cal_fac(m,x,p),p)%p* \
78             inv(cal_fac(n-m,x,p),p)%p;
79 }
80
81 ll C(ll n,ll m,ll p)
82 {
83     if(m>n||m<0||n<0) return 0;
84     ll i,res;
85     VL a,b,resa;
86     assert(mp.count(p));
87     a=mp[p].fi;
88     b=mp[p].se;
89     for(i=0;i<sz(a);i++) resa.pb(multilucas(n,m,a
90         [i],b[i]));
91     res=exCRT::excrt(resa,b);
92     assert(res!=-1);
93     return res;
94 }
95
96 };//exLucas::init(VL{});

```

4.5 欧拉函数

$\leq n$ 且与 n 互质的数的和: $n * \phi(n) / 2$

4.5.1 直接求

```

1 //O(sqrt(n))
2 int euler(int n)
3 {
4     int ans,i;
5     ans=n;
6     for(i=2;i*i<=n;i++)
7     {
8         if(n%i==0)
9         {
10             ans=ans-ans/i;
11             while(n%i==0) n/=i;
12         }
13     }
14     if(n>1) ans=ans-ans/n;
15     return ans;
16 }

```

4.5.2 线性筛

```

1 int prime[MAX],phi[MAX],tot;
2 bool flag[MAX];
3 void init(int n)
4 {
5     int i,j,k;
6     tot=0;
7     mem(flag,0);
8     phi[0]=0;
9     phi[1]=1;
10    for(i=2;i<=n;i++)
11    {
12        if(!flag[i])
13        {
14            prime[tot++]=i;
15            phi[i]=i-1;
16        }
17        for(j=0;j<tot&&i*prime[j]<=n;j++)
18        {
19            k=i*prime[j];
20            flag[k]=1;
21            if(i%prime[j]==0)
22            {
23                phi[k]=phi[i]*prime[j];
24                break;
25            }
26            else phi[k]=phi[i]*(prime[j]-1);
27        }
28    }

```

4.6 莫比乌斯函数

```

1 int mo[MAX], prime[MAX], tot;
2 bool flag[MAX];
3 void initmo(int n)
4 {
5     int i, j;
6     mem(flag, 0);
7     mem(mo, 0);
8     tot = 0;
9     mo[1] = 1;
10    for(i = 2; i <= n; i++)
11    {
12        if(!flag[i])
13        {
14            prime[tot++] = i;
15            mo[i] = -1;
16        }
17        for(j = 0; j < tot && prime[j] * i <= n; j++)
18        {
19            flag[i * prime[j]] = 1;
20            if(i % prime[j] == 0)
21            {
22                mo[prime[j] * i] = 0;
23                break;
24            }
25            mo[prime[j] * i] = -mo[i];
26        }
27    }
28 }

```

4.7 Berlekamp-Massey

```

1 //Berlekamp-Massey
2 typedef vector<int> VI;
3 namespace linear_seq
4 {
5     #define rep(i, a, n) for (int i = a; i < n; i++)
6     #define SZ(x) ((int)(x).size())
7     const ll mod = 1e9 + 7;
8     ll powmod(ll a, ll b) { ll res = 1; a %= mod; assert(b >= 0); for(;; b >>= 1) { if(b & 1) res = res * a % mod; a = a * a % mod; } return res; }
9     const int N = 10010;
10    ll res[N], base[N], _c[N], _md[N];
11    vector<int> Md;
12    void mul(ll *a, ll *b, int k)
13    {
14        rep(i, 0, k + k) _c[i] = 0;
15        rep(i, 0, k) if (a[i]) rep(j, 0, k) _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;

```

```

16    for (int i = k + k - 1; i >= k; i--) if (_c[i])
17        rep(j, 0, SZ(Md)) _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] * _md[Md[j]]) % mod;
18    rep(i, 0, k) a[i] = _c[i];
19 }
20 int solve(ll n, VI a, VI b) {
21     ll ans = 0, pnt = 0;
22     int k = SZ(a);
23     assert(SZ(a) == SZ(b));
24     rep(i, 0, k) _md[k - 1 - i] = -a[i]; _md[k] = 1;
25     Md.clear();
26     rep(i, 0, k) if (_md[i] != 0) Md.push_back(i);
27     rep(i, 0, k) res[i] = base[i] = 0;
28     res[0] = 1;
29     while ((1ll << pnt) <= n) pnt++;
30     for (int p = pnt; p >= 0; p--) {
31         mul(res, res, k);
32         if ((n >> p) & 1) {
33             for (int i = k - 1; i >= 0; i--) res[i + 1] = res[i]; res[0] = 0;
34             rep(j, 0, SZ(Md)) res[Md[j]] = (res[Md[j]] - res[k] * _md[Md[j]]) % mod;
35         }
36     }
37     rep(i, 0, k) ans = (ans + res[i] * b[i]) % mod;
38     if (ans < 0) ans += mod;
39     return ans;
40 }
41 VI BM(VI s) {
42     VI C(1, 1), B(1, 1);
43     int L = 0, m = 1, b = 1;
44     rep(n, 0, SZ(s)) {
45         ll d = 0;
46         rep(i, 0, L + 1) d = (d + (1ll) C[i] * s[n - i]) % mod;
47         if (d == 0) ++m;
48         else if (2 * L <= n) {
49             VI T = C;
50             ll c = mod - d * powmod(b, mod - 2) % mod; // 222
51             while (SZ(C) < SZ(B) + m) C.pb(0);
52             rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
53             L = n + 1 - L; B = T; b = d; m = 1;
54         } else {
55             ll c = mod - d * powmod(b, mod - 2) % mod; // 222
56             while (SZ(C) < SZ(B) + m) C.pb(0);
57             rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
58             ++m;
59         }
60     }
61     return C;
62 }
63 int gao(VI a, ll n)
64 {

```

```

65     VI c=BM(a);
66     c.erase(c.begin());
67     rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
68     return solve(n,c,VI(a.begin(),a.begin()+SZ(c)
69         ));
70 };//linear_seq::gao(VI{},n-1)

```

4.8 exBSGS

```

1 //a^x=b (mod c)
2 ll exBSGS(ll a,ll b,ll c)
3 {
4     ll i,g,d,num,now,sq,t,x,y;
5     if(c==1) return b?-1:(a!=1);
6     if(b==1) return a?0:-1;
7     if(a%c==0) return b?-1:1;
8     num=0;
9     d=1;
10    while((__gcd(a,c))>1)
11    {
12        if(b%g) return -1;
13        num++;
14        b/=g;
15        c/=g;
16        d=(d*a/g)%c;
17        if(d==b) return num;
18    }
19    mp.clear();
20    sq=ceil(sqrt(c));
21    t=1;
22    for(i=0;i<sq;i++)
23    {
24        if(!mp.count(t)) mp[t]=i;
25        else mp[t]=min(mp[t],i);
26        t=t*a%c;
27    }
28    for(i=0;i<sq;i++)
29    {
30        exgcd(d,c,x,y);
31        x=(x*b%c+c)%c;
32        if(mp.count(x)) return i*sq+mp[x]+num;
33        d=d*t%c;
34    }
35    return -1;
36 }

```

4.9 Miller_Rabin+Pollard_rho

```

1 const int S=20;
2 mt19937 rd(time(0));
3 ll mul2(ll a,ll b,ll p)
4 {

```

```

5     ll res=0;
6     while(b)
7     {
8         if(b&1) res=(res+a)%p;
9         a=(a+a)%p;
10        b>>=1;
11    }
12    return res;
13 }
14 ll pow2(ll a,ll b,ll p)
15 {
16     ll res=1;
17     while(b)
18     {
19         if(b&1) res=mul2(res,a,p);
20         a=mul2(a,a,p);
21         b>>=1;
22     }
23     return res;
24 }
25 int check(ll a,ll n,ll x,ll t)//一定是合数返回不一定
    返回1,0
26 {
27     ll now,nex,i;
28     now=nex=pow2(a,x,n);
29     for(i=1;i<=t;i++)
30     {
31         now=mul2(now,now,n);
32         if(now==1&&nex!=1&&nex!=n-1) return 1;
33         nex=now;
34     }
35     if(now!=1) return 1;
36     return 0;
37 }
38 int Miller_Rabin(ll n)
39 {
40     if(n<2) return 0;
41     if(n==2) return 1;
42     if((n&1)==0) return 0;
43     ll x,t,i;
44     x=n-1;
45     t=0;
46     while((x&1)==0) x>>=1,t++;
47     for(i=0;i<S;i++)
48     {
49         if(check(rd()%(n-1)+1,n,x,t)) return 0;
50     }
51     return 1;
52 }
53 ll Pollard_rho(ll x,ll c)
54 {
55     ll i,k,g,t,y;
56     i=1;
57     k=2;

```

```

58     y=t*rd()%x;
59     while(1)
60     {
61         i++;
62         t=(mul2(t,t,x)+c)%x;
63         g=__gcd(y-t+x,x);
64         if(g!=1&&g!=x) return g;
65         if(y==t) return x;
66         if(i==k)
67         {
68             y=t;
69             k+=k;
70         }
71     }
72 }
73 vector<ll> fac;
74 void findfac(ll n)
75 {
76     if(Miller_Rabin(n))
77     {
78         fac.pb(n);
79         return;
80     }
81     ll t=n;
82     while(t>=n) t=Pollard_rho(t,rd()%(n-1)+1);
83     findfac(t);
84     findfac(n/t);
85 }
86 void work(ll x)
87 {
88     fac.clear();
89     findfac(x);
90 }

```

4.10 第二类 Stirling 数

```

1 //dp[i][j]表示j个元素划分到个不可区分的非空盒子里的方案
  数。ik
2 ll dp[MAX][MAX];
3 void init()
4 {
5     ll i,j;
6     mem(dp,0);
7     dp[1][1]=1;
8     for(i=2;i<MAX;i++)
9     {
10         for(j=1;j<=i;j++)
11         {
12             dp[i][j]=(dp[i-1][j-1]+j*dp[i-1][j])%mod;
13         }
14     }
15 }

```

4.11 原根

原根性质

1. 一个数 m 如果有原根，则其原根个数为 $\phi[\phi[m]]$ 。
若 m 为素数，则其原根个数为 $\phi[\phi[m]] = \phi[m-1]$ 。
2. 有原根的数只有 $2, 4, p^n, 2 * p^n$ (p 为质数, n 为正整数)
3. 一个数的最小原根的大小是 $O(n^{0.25})$ 的
4. 如果 g 为 n 的原根，则 g^d 为 n 的原根的充要条件是 $\gcd(d, \phi[n]) = 1$

指标法则

1. $I(a * b) = I(a) + I(b) \pmod{p-1}$
2. $I(a^k) = k * I(a) \pmod{p-1}$

```

1 int p[MAX],tot,prime[MAX];
2 void init(int n)
3 {
4     int i,j;
5     tot=0;
6     mem(prime,0);
7     prime[1]=1;
8     for(i=2;i<=n;i++)
9     {
10         if(!prime[i]) prime[i]=p[tot++]=i;
11         for(j=0;j<tot&&p[j]*i<=n;j++)
12         {
13             prime[i*p[j]]=p[j];
14             if(i%p[j]==0) break;
15         }
16     }
17 }
18 ll pow2(ll a,ll b,ll p)
19 {
20     ll res=1;
21     while(b)
22     {
23         if(b&1) res=res*a%p;
24         a=a*a%p;
25         b>>=1;
26     }
27     return res;
28 }
29 int tp[MAX];
30 int find_root(int x)//求素数原根
31 {
32     if(x==2) return 1;
33     int f,phi=x-1;
34     tp[0]=0;
35     for(int i=0;phi&&i<tot;i++)
36     {
37         if(phi%p[i]==0)

```

```

38     {
39         tp[++tp[0]]=p[i];
40         while(phi%p[i]==0) phi/=p[i];
41     }
42 }
43 if(phi!=1) tp[++tp[0]]=phi;
44 phi=x-1;
45 for(int g=2;g<=x-1;g++)
46 {
47     f=1;
48     for(int i=1;i<=tp[0];i++)
49     {
50         if(pow2(g,phi/tp[i],x)==1)
51         {
52             f=0;
53             break;
54         }
55     }
56     if(f) return g;
57 }
58 return 0;
59 }
60 int I[MAX];
61 void get_I(int p)//求指标表
62 {
63     int g,now;
64     g=find_root(p);
65     now=1;
66     for(int i=1;i<p;i++)
67     {
68         now=now*g%p;
69         I[now]=i;
70     }
71 }

```

4.12 二次剩余

```

1 //O(log^2)
2 struct Tonelli_Shanks
3 {
4     ll mul2(ll a,ll b,ll p)
5     {
6         ll res=0;
7         a%=p;
8         while(b)
9         {
10             if(b&1)
11             {
12                 res+=a;
13                 if(res>=p) res-=p;
14             }
15             a+=a;
16             if(a>=p) a-=p;

```

```

17         b>>=1;
18     }
19     return res;
20 }
21 ll pow2(ll a,ll b,ll p)
22 {
23     ll res=1;
24     while(b)
25     {
26         if(b&1) res=mul2(res,a,p);
27         a=mul2(a,a,p);
28         b>>=1;
29     }
30     return res;
31 }
32 ll sqrt(ll n,ll p)
33 {
34     if(p==2) return (n&1)?1:-1;
35     if(pow2(n,p>>1,p)!=1) return -1;
36     if(p&2) return pow2(n,(p+1)>>2,p);
37     ll q,z,c,r,t,tmp,s,i,m;
38     s=__builtin_ctzll(p^1);
39     q=p>>s;
40     z=2;
41     for(;pow2(z,p>>1,p)==1;z++);
42     c=pow2(z,q,p);
43     r=pow2(n,(q+1)>>1,p);
44     t=pow2(n,q,p);
45     for(m=s;t!=1;)
46     {
47         for(i=0,tmp=t;tmp!=1;i++) tmp=tmp*tmp%p;
48         for(;i<--m;) c=c*c%p;
49         r=r*c%p;
50         c=c*c%p;
51         t=t*c%p;
52     }
53     return r;
54 }
55 }ts;

```

5 多项式

5.1 FFT

```

1 namespace FFT
2 {
3     #define rep(i,a,b) for(int i=(a);i<=(b);i++)
4     const double pi=acos(-1);
5     const int maxn=(1<<19)+10;
6     struct cp
7     {
8         double a,b;
9         cp(){

```

```

58     rep(i,0,l2)y[i]=cp(c[i],0);
59     rep(i,l1+1,K)x[i]=cp(0,0);
60     rep(i,l2+1,K)y[i]=cp(0,0);
61     fft(x,K,0);fft(y,K,0);
62     rep(i,0,K)z[i]=x[i]*y[i];
63     fft(z,K,1);
64     rep(i,0,l1+l2)a[i]=(l1)(z[i].a+0.5);
65 }
66 };

```

```

1 namespace NTT
2 {
3     const int g=3;
4     const int p=998244353;
5     int wn[35];
6     int pow2(int a,int b)
7     {
8         int res=1;
9         while(b)
10         {
11             if(b&1) res=1ll*res*a%p;
12             a=1ll*a*a%p;
13             b>>=1;
14         }
15         return res;
16     }
17     void getwn()
18     {
19         for(int i=0;i<25;i++) wn[i]=pow2(g,(p-1)/(1ll
20             <<i));
21     }
22     void ntt(VI &a,int len,int f)
23     {
24         int i,j=0,t,k,w,id;
25         for(i=1;i<len-1;i++)
26         {
27             for(t=len;j^=t>>=1,~j&t;);
28             if(i<j) swap(a[i],a[j]);
29         }
30         for(i=1,id=1;i<len;i<=1,id++)
31         {
32             t=i<<1;
33             for(j=0;j<len;j+=t)
34             {
35                 for(k=0,w=1;k<i;k++,w=1ll*w*wn[id]%p)
36                 {
37                     int x=a[j+k],y=1ll*w*a[j+k+i]%p;
38                     a[j+k]=x+y;
39                     if(a[j+k]>=p) a[j+k]-=p;
40                     a[j+k+i]=x-y;
41                     if(a[j+k+i]<0) a[j+k+i]+=p;

```

```

41         }
42     }
43 }
44 if(f)
45 {
46     for(i=1,j=len-1;i<j;i++,j--) swap(a[i],a[
47         j]);
48     int inv=pow2(len,p-2);
49     for(i=0;i<len;i++) a[i]=1ll*a[i]*inv%p;
50 }
51 void qpow(VI &a,int b)//limt: sz(a)*b is small
52 {
53     int len,i,l1;
54     l1=sz(a);
55     for(len=1;len<=(l1+1)*b-1;len<=&=1);
56     a.resize(len+1);
57     for(i=l1;i<len;i++) a[i]=0;
58     ntt(a,len,0);
59     for(i=0;i<len;i++) a[i]=pow2(a[i],b);
60     ntt(a,len,1);
61     a.resize((l1+1)*b-1);
62 }
63 void mul(VI &res,VI a,VI b)
64 {
65     int len,i,l1,l2;
66     l1=sz(a);
67     l2=sz(b);
68     for(len=1;len<=l1+l2;len<=&=1);
69     a.resize(len+1);
70     b.resize(len+1);
71     for(i=l1+1;i<len;i++) a[i]=0;
72     for(i=l2+1;i<len;i++) b[i]=0;
73     ntt(a,len,0);ntt(b,len,0);
74     res.resize(len);
75     for(i=0;i<len;i++) res[i]=1ll*a[i]*b[i]%p;
76     ntt(res,len,1);
77     res.resize(l1+l2-1);
78 }
79 };//NTT::getwn();

```

5.3 FWT

```

1 namespace FWT
2 {
3     ll inv2;//对2的逆元p
4     const ll p=1e9+7;
5     ll pow2(ll a,ll b)
6     {
7         ll res=1;
8         while(b)
9         {
10             if(b&1) res=res*a%p;

```

```

11         a=a*a%p;
12         b>>=1;
13     }
14     return res;
15 }
16 void fwt(ll *a,int n,int f,int v)
17 {
18     for(int d=1;d<n;d<=&=1)
19     {
20         for(int m=d<<1,i=0;i<n;i+=m)
21         {
22             for(int j=0;j<d;j++)
23             {
24                 ll x=a[i+j],y=a[i+j+d];
25                 if(!v)
26                 {
27                     if(f==1) a[i+j]=(x+y)%p,a[i+j+d]
28                         =(x-y+p)%p;//xor
29                     else if(f==2) a[i+j]=(x+y)%p;//
30                         and
31                     else if(f==3) a[i+j+d]=(x+y)%p;
32                         //or
33                 }
34                 else
35                 {
36                     if(f==1) a[i+j]=(x+y)*inv2%p,a[
37                         i+j+d]=(x-y+p)%p*inv2%p;//
38                         xor
39                     else if(f==2) a[i+j]=(x-y+p)%p;
40                         //and
41                     else if(f==3) a[i+j+d]=(y-x+p)%
42                         p;//or
43                 }
44             }
45         }
46     }
47 }
48 //结果存在a
49 void XOR(ll *a,ll *b,int n)
50 {
51     int len;
52     for(len=1;len<=n;len<=&=1);
53     fwt(a,len,1,0);
54     fwt(b,len,1,0);
55     for(int i=0;i<len;i++) a[i]=a[i]*b[i]%p;
56     inv2=pow2(2,p-2);
57     fwt(a,len,1,1);
58 }
59 void AND(ll *a,ll *b,int n)
60 {
61     int len;
62     for(len=1;len<=n;len<=&=1);
63     fwt(a,len,2,0);

```



```

58     fwt(b,len,2,0);
59     for(int i=0;i<len;i++) a[i]=a[i]*b[i]%p;
60     fwt(a,len,2,1);
61 }
62 void OR(ll *a,ll *b,int n)
63 {
64     int len;
65     for(len=1;len<=n;len<=1);
66     fwt(a,len,3,0);
67     fwt(b,len,3,0);
68     for(int i=0;i<len;i++) a[i]=a[i]*b[i]%p;
69     fwt(a,len,3,1);
70 }
71 };

```

5.4 拉格朗日插值

```

1 namespace polysum {
2     #define rep(i,a,n) for (int i=a;i<n;i++)
3     #define per(i,a,n) for (int i=n-1;i>=a;i--)
4     const int D=101000;
5     ll a[D],tmp[D],f[D],g[D],p[D],p1[D],p2[D],b[D],h
6         [D][2],C[D];
7     ll powmod(ll a,ll b){ll res=1;a%=mod;assert(b
8         >=0);for(;b>=1){if(b&1)res=res*a%mod;a=a*
9         a%mod;}return res;}
10    ll calcn(int d,ll *a,ll n) { // a[0].. a[d] a[n]
11        if (n<=d) return a[n];
12        p1[0]=p2[0]=1;
13        rep(i,0,d+1) {
14            ll t=(n-i+mod)%mod;
15            p1[i+1]=p1[i]*t%mod;
16        }
17        rep(i,0,d+1) {
18            ll t=(n-d+i+mod)%mod;
19            p2[i+1]=p2[i]*t%mod;
20        }
21        ll ans=0;
22        rep(i,0,d+1) {
23            ll t=g[i]*g[d-i]%mod*p1[i]%mod*p2[d-i]%
24                mod*a[i]%mod;
25            if ((d-i)&1) ans=(ans-t+mod)%mod;
26            else ans=(ans+t)%mod;
27        }
28        return ans;
29    }
30    void init(int M) {
31        f[0]=f[1]=g[0]=g[1]=1;
32        rep(i,2,M+5) f[i]=f[i-1]*i%mod;
33        g[M+4]=powmod(f[M+4],mod-2);
34        per(i,1,M+4) g[i]=g[i+1]*(i+1)%mod;
35    }
36 }

```

```

32 ll polysum(ll n,ll *a,ll m) { // a[0].. a[m] \
33     sum_{i=0}^{n-1} a[i]
34     rep(i,0,m+1) tmp[i]=a[i];
35     tmp[m+1]=calcn(m,tmp,m+1);
36     rep(i,1,m+2) tmp[i]=(tmp[i-1]+tmp[i])%mod;
37     return calcn(m+1,tmp,n-1);
38 }
39 ll qpolysum(ll R,ll n,ll *a,ll m) { // a[0].. a[
40     m] \sum_{i=0}^{n-1} a[i]*R^i
41     if (R==1) return polysum(n,a,m);
42     a[m+1]=calcn(m,a,m+1);
43     ll r=powmod(R,mod-2),p3=0,p4=0,c,ans;
44     h[0][0]=0;h[0][1]=1;
45     rep(i,1,m+2) {
46         h[i][0]=(h[i-1][0]+a[i-1])*r%mod;
47         h[i][1]=h[i-1][1]*r%mod;
48     }
49     rep(i,0,m+2) {
50         ll t=g[i]*g[m+1-i]%mod;
51         if (i&1) p3=((p3-h[i][0]*t)%mod+mod)%mod,
52             p4=((p4-h[i][1]*t)%mod+mod)%mod;
53         else p3=(p3+h[i][0]*t)%mod,p4=(p4+h[i
54             ][1]*t)%mod;
55     }
56     c=powmod(p4,mod-2)*(mod-p3)%mod;
57     rep(i,0,m+2) h[i][0]=(h[i][0]+h[i][1]*c)%mod;
58     rep(i,0,m+2) C[i]=h[i][0];
59     ans=(calcn(m,C,n)*powmod(R,n)-c)%mod;
60     if (ans<0) ans+=mod;
61     return ans;
62 }
63 // polysum::init();

```

6 矩阵

6.1 矩阵类

```

1 const ll mod=1e9+7;
2 struct Matrix
3 {
4     ll c[6][6],n;
5     Matrix(){}
6     Matrix(ll a,ll v=0)
7     {
8         int i,j;
9         n=a;
10        for(i=0;i<n;i++)
11        {
12            for(j=0;j<n;j++)
13            {
14                c[i][j]=v;
15            }
16        }
17    }
18 }

```

```

17     }
18     Matrix operator *(const Matrix &b)const
19     {
20         int i,j,k;
21         Matrix res(n);
22         for(k=0;k<n;k++)
23         {
24             for(i=0;i<n;i++)
25             {
26                 if(!c[i][k]) continue;
27                 for(j=0;j<n;j++)
28                 {
29                     res.c[i][j]+=c[i][k]*b.c[k][j];
30                     if(res.c[i][j]>=mod) res.c[i][j]%=
mod;
31                 }
32             }
33         }
34         return res;
35     }
36 };
37 Matrix matpow2(Matrix a,ll b)
38 {
39     Matrix res(a.n);
40     for(int i=0;i<a.n;i++)
41     {
42         res.c[i][i]=1;
43     }
44     while(b)
45     {
46         if(b&1) res=res*a;
47         a=a*a;
48         b>>=1;
49     }
50     return res;
51 }

```

6.2 高斯消元

6.2.1 同余方程

```

1 namespace Gauss
2 {
3     int p;
4     ll mp[905][905],sol[905];
5     void set_mod(int _p)
6     {
7         p=_p;
8         mem(mp,0);
9         mem(sol,0);
10    }
11    ll pow2(ll a,ll b)
12    {
13        ll res=1;

```

```

14    while(b)
15    {
16        if(b&1) res=res*a%p;
17        a=a*a%p;
18        b>>=1;
19    }
20    return res;
21 }
22 ll inv(ll x){return pow2(x,p-2);}
23 ll lcm(ll a,ll b){return a/__gcd(a,b)*b;}
24 int gauss(int n,int m)
25 {
26     int r,c,id,i,j;
27     ll tmp,ta,tb;
28     r=c=0;
29     while(r<n&&c<m)
30     {
31         id=r;
32         for(i=r+1;i<n;i++)
33         {
34             if(abs(mp[i][c])>abs(mp[id][c])) id=i;
35         }
36         if(id!=r)
37         {
38             for(i=0;i<m;i++) swap(mp[r][i],mp[id
][i]);
39         }
40         if(abs(mp[r][c])!=0)
41         {
42             for(i=r+1;i<n;i++)
43             {
44                 if(abs(mp[i][c])==0) continue;
45                 tmp=lcm(abs(mp[i][c]),abs(mp[r][c]
));
46                 ta=tmp/abs(mp[i][c]);
47                 tb=tmp/abs(mp[r][c]);
48                 if(mp[i][c]*mp[r][c]<0) tb=-tb;
49                 for(j=c;j<m;j++)
50                 {
51                     mp[i][j]=(mp[i][j]*ta-mp[r][j]*
tb)%p;
52                     if(mp[i][j]<0) mp[i][j]+=p;
53                 }
54             }
55             r++;
56         }
57         c++;
58     }
59     for(i=r;i<n;i++)
60     {
61         if(mp[i][m]!=0) return -1;//no solution
62     }
63     // if(r<m) return m-r;//multi solution
64     for(i=m-1;~i;i--)

```

```

65     {
66         tmp=mp[i][m];
67         for(j=i+1;j<m;j++)
68         {
69             if(mp[i][j]==0) continue;
70             tmp=(tmp-mp[i][j]*sol[j])%p;
71             if(tmp<0) tmp+=p;
72         }
73         sol[i]=tmp*inv(mp[i][i])%p;
74     }
75     return 0;
76 }
77 }
78 using namespace Gauss;
79 //set_mod();

```

6.2.2 同余方程 mod=2

```

1 //同余方程 mod=2 异或加速O(n*n*m)
2 int mat[22][MAX];
3 int Gauss(int n,int m)//是未知数个数n 是方程个数m
4 {
5     int i,j;
6     for(i=1,j=1;i<=n&&j<=m;j++)
7     {
8         int k=i;
9         while(k<=n&&!mat[k][j]) k++;
10        if(mat[k][j])
11        {
12            for(int r=1;r<=m+1;r++) swap(mat[i][r],
13                mat[k][r]);
14            for(int r=1;r<=n;r++)
15            {
16                if(r!=i&&mat[r][j])
17                {
18                    for(k=1;k<=m+1;k++)
19                    {
20                        mat[r][k]^=mat[i][k];
21                    }
22                }
23            }
24            i++;
25        }
26        //第m列是等号右边+1
27        for(j=i;j<=n;j++)
28        {
29            if(mat[j][m+1]) return -1;//无解
30        }
31        return m-i+1;//返回解的个数
32    }

```

6.3 单纯形

```

1 typedef double db;
2 typedef vector<db> VD;
3 typedef vector<VD> VVD;
4 typedef vector<int> VI;
5 struct Simplex
6 {
7     int m,n;
8     VI B,N;
9     VVD D;
10    Simplex(){}
11    Simplex(const VVD &A,const VD &b,const VD &c):m(
12        sz(b)),n(sz(c)),N(n+1),B(m),D(m+2,VD(n+2))
13    {
14        int i,j;
15        for(i=0;i<m;i++)
16        {
17            for(j=0;j<n;j++)
18            {
19                D[i][j]=A[i][j];
20            }
21        }
22        for(i=0;i<m;i++)
23        {
24            B[i]=n+i;
25            D[i][n]=-1;
26            D[i][n+1]=b[i];
27        }
28        for(j=0;j<n;j++)
29        {
30            N[j]=j;
31            D[m][j]=-c[j];
32        }
33        N[n]=-1;
34        D[m+1][n]=1;
35    }
36    void Pivot(int r,int s)
37    {
38        int i,j;
39        for(i=0;i<m+2;i++)
40        {
41            if(i==r) continue;
42            for(j=0;j<n+2;j++)
43            {
44                if(j==s) continue;
45                D[i][j]-=D[r][j]*D[i][s]/D[r][s];
46            }
47        }
48        for(j=0;j<n+2;j++)
49        {
50            if (j!=s) D[r][j]/=D[r][s];
51        }
52        for(i=0;i<m+2;i++)

```

```

52     {
53         if(i!=r) D[i][s]/=-D[r][s];
54     }
55     D[r][s]=1.0/D[r][s];
56     swap(B[r],N[s]);
57 }
58 bool simplex(int phase)
59 {
60     int i,j,s,r;
61     int x=phase==1?m+1:m;
62     while(1)
63     {
64         s=-1;
65         for(j=0;j<=n;j++)
66         {
67             if(phase==2&&N[j]==-1) continue;
68             if(s==-1||D[x][j]<D[x][s]||D[x][j]==D[
                x][s]&&N[j]<N[s]) s=j;
69         }
70         if(D[x][s]>-eps) return 1;
71         r=-1;
72         for(i=0;i<m;i++)
73         {
74             if(D[i][s]<eps) continue;
75             if(r==-1||D[i][n+1]/D[i][s]<D[r][n+1]/
                D[r][s]) r=i;
76             if(D[i][n+1]/D[i][s]==D[r][n+1]/D[r][s]
                &&B[i]<B[r]) r=i;
77         }
78         if(r==-1) return 0;
79         Pivot(r,s);
80     }
81 }
82 db work(VD &res)
83 {
84     int i,j,k,r,s;
85     r=0;
86     for(i=1;i<m;i++)
87     {
88         if(D[i][n+1]<D[r][n+1]) r=i;
89     }
90     if(D[r][n+1]<-eps)
91     {
92         Pivot(r,n);
93         if(!simplex(1)||D[m+1][n+1]<-eps) return
            -numeric_limits<db>::infinity();//no
            solution
94         for(i=0;i<m;i++)
95         {
96             if(B[i]!=-1) continue;
97             s=-1;
98             for(j=0;j<=n;j++)
99             {

```

```

100                 if(s==-1||D[i][j]<D[i][s]||D[i][j]
                    ]==D[i][s]&&N[j]<N[s]) s=j;
101             }
102             Pivot(i,s);
103         }
104     }
105     if(!simplex(2)) return numeric_limits<db>::
        infinity();//solution is INF
106     res=VD(n);
107     for(i=0;i<m;i++)
108     {
109         if(B[i]<n) res[B[i]]=D[i][n+1];
110     }
111     return D[m][n+1];
112 }
113 };
114 /*
115 sum(A[i]*res[i])<=B,res[i]>=0
116 MAX(sum(C[i]*res[i]))
117 */

```

6.4 线性基

```

1 struct Base
2 {
3     #define type ll
4     #define mx 60
5     type d[mx+3];
6     int p[mx+3],cnt;
7     void init()
8     {
9         memset(d,0,sizeof(d));
10        cnt=0;
11    }
12    bool insert(type x,int pos=0)
13    {
14        int i;
15        for(i=mx;~i;i--)
16        {
17            if(!(x&(1LL<<i))) continue;
18            if(!d[i])
19            {
20                cnt++;
21                d[i]=x;
22                p[i]=pos;
23                break;
24            }
25            if(p[i]<pos)
26            {
27                swap(d[i],x);
28                swap(p[i],pos);
29            }
30            x^=d[i];

```

```

31     }
32     return x>0;
33 }
34 type query_max(int pos=-1)
35 {
36     int i;
37     type res=0;
38     for(i=mx;~i;i--)
39     {
40         if(p[i]>=pos)
41         {
42             if((res^d[i])>res) res^=d[i];
43         }
44     }
45     return res;
46 }
47 type query_min(int pos=-1)
48 {
49     for(int i=0;i<=mx;i++)
50     {
51         if(d[i]&& p[i]>=pos) return d[i];
52     }
53     return 0;
54 }
55 void merge(Base x)
56 {
57     if(cnt<x.cnt)
58     {
59         swap(cnt,x.cnt);
60         swap(d,x.d);
61         swap(p,x.p);
62     }
63     for(int i=mx;~i;i--)
64     {
65         if(x.d[i]) insert(x.d[i]);
66     }
67 }
68 //kth min
69 //first use rebuild()
70 type tp[mx+3];
71 void rebuild()
72 {
73     int i,j;
74     cnt=0;
75     for(i=mx;~i;i--)
76     {
77         for(j=i-1;~j;j--)
78         {
79             if(d[i]&(1LL<<j)) d[i]^=d[j];
80         }
81     }
82     for(i=0;i<=mx;i++)
83     {
84         if(d[i]) tp[cnt++]=d[i];

```

```

85     }
86 }
87 type kth(type k)
88 {
89     type res=0;
90     if(k>=(1LL<<cnt)) return -1;
91     for(int i=mx;~i;i--)
92     {
93         if(k&(1LL<<i)) res^=tp[i];
94     }
95     return res;
96 }
97 };

```

7 博弈

7.1 SG 函数

f[m]: 可改变当前状态的方式, N 为方式的种类, 要先从小到大 sort

sg[]: sg 表

flag[m]: 为 x 后继状态的集合

7.1.1 sg 表

```

1 int f[111],sg[MAX];
2 void SG(int n,int m)
3 {
4     int i,j,flag[111];
5     mem(sg,0);
6     for(i=1;i<=n;i++)
7     {
8         mem(flag,0);
9         for(j=0;f[j]<=i&&j<m;j++)
10         {
11             flag[sg[i-f[j]]]=1;
12         }
13         for(j=0;;j++)
14         {
15             if(!flag[j])
16             {
17                 sg[i]=j;
18                 break;
19             }
20         }
21     }
22 }

```

7.1.2 记忆化搜索求 sg 函数

```

1 int f[105],sg[MAX],m;
2 int dfs(int x)
3 {
4     int i,j,flag[105];
5     if(sg[x]!=-1) return sg[x];
6     mem(flag,0);
7     for(i=1;i<=m;i++)
8     {
9         if(x>=f[i])
10        {
11            dfs(x-f[i]);
12            flag[sg[x-f[i]]]=1;
13        }
14    }
15    for(i=0;;i++)
16    {
17        if(!flag[i])
18        {
19            j=i;
20            break;
21        }
22    }
23    return sg[x]=j;
24 }

```

7.2 结论

1. 阶梯博弈

0 层为终点的阶梯博弈，等价于奇数层的 nim，偶数层的移动不影响结果

2.SJ 定理

对于任意一个 Anti-SG 游戏，如果我们规定当局面中所有的单一游戏的 SG 值为 0 时，游戏结束。

先手必胜当且仅当：

- (1) 游戏的 SG 函数不为 0 且游戏中某个单一游戏的 SG 函数大于 1；
- (2) 游戏的 SG 函数为 0 且游戏中没有单一游戏的 SG 函数大于 1。

8 dp

8.1 LIS

```

1 //最长上升子序列(>)nlogn 返回长度
2 //最长下降子序列(<) 把原数组取负数
3 int a[MAX],b[MAX];

```

```

4 int LIS(int n)
5 {
6     int i;
7     mem(b,0x3f);
8     for(i=0;i<n;i++)
9     {
10        *lower_bound(b,b+n,a[i])=a[i];//最长不下降子序
11        列(>=)改为upper_bound
12    }
13    return lower_bound(b,b+n,INF)-b;
14 }

```

8.2 LPS

```

1 //最长回文子序列
2 //dp[i][j] 表示s[i..j] 最长回文子序列的长度
3 int dp[2222][2222];
4 void LPS(char *s,int n)
5 {
6     int i,j,len;
7     for(i=1;i<=n;i++) dp[i][i]=1;
8     for(len=2;len<=n;len++)
9     {
10        for(i=1;i<=n-len+1;i++)
11        {
12            j=i+len-1;
13            if(s[i]==s[j]) dp[i][j]=dp[i+1][j-1]+2;
14            else dp[i][j]=max({dp[i+1][j],dp[i][j-1],
15                               dp[i+1][j-1]});
16        }
17    }
18 }

```

8.3 数位 dp

```

1 const int DIG=20+2;
2 ll dp[DIG][2];
3 ll gao(ll x)
4 {
5     const int base=10;
6     int p[DIG],tot=0;
7     if(x==-1) return 0;
8     while(1)
9     {
10        p[tot++]=x%base;
11        x/=base;
12        if(!x) break;
13    }
14    function<ll(int,int,int,int)> dfs=[&](int pos,
15        int lead,int sta,int limt)->ll
16    {
17        if(pos==0) return 1;
18    }
19 }

```

```

31     for(;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
32     if(ch=='.'){
33         double tmp=1; ch=nc();
34         for(;ch>='0'&&ch<='9';ch=nc())tmp/=10.0,x
            +=tmp*(ch-'0');
35     }
36     if(sign)x=-x;
37     return true;
38 }
39 inline bool read(char *s){
40     char ch=nc();
41     for(;blank(ch);ch=nc());
42     if(!ch)return false;
43     for(;!blank(ch)&&!ch=='\n';ch=nc())*s++=ch;
44     *s=0;
45     return true;
46 }
47 inline bool read(char &c){
48     for(c=nc();blank(c);c=nc());
49     if(!ch){c=-1;return false;}
50     return true;
51 }
52 template<class T,class... U>bool read(T& h,U&...
    t){return read(h)&&read(t...);}
53 //fwrite->print
54 struct Ostream_fwrite{
55     char *buf,*p1,*pend;
56     Ostream_fwrite(){buf=new char[BUF_SIZE];p1=
        buf;pend=buf+BUF_SIZE;}
57 // void out(char ch){putchar(ch);}
58 void out(char ch){if(p1==pend){fwrite(buf,1,
        BUF_SIZE,stdout);p1=buf;}*p1++=ch;}
59 template<class T>void print(T x){
60     static char s[33],*s1;s1=s;
61     if(!x)*s1++='0';if(x<0)out('-'),x=-x;
62     while(x)*s1++=x%10+'0',x/=10;
63     while(s1--!=s)out(*s1);
64 }
65 void print(double x,int y){
66     static ll mul[]=
67         {1,10,100,1000,10000,100000,1000000,10000000,100000000,
68             1000000000,10000000000,100000000000,
69             1000000000000,10000000000000,100000000000000,
70             1000000000000000,10000000000000000,
71             100000000000000000,1000000000000000000};
72     if(x<-1e-12)out('-'),x=-x;
73     ll x2=(ll)floor(x);if(!y&&x-x2>=0.5)++x2;
74     x-=x2;x*=mul[y];
75     ll x3=(ll)floor(x);if(y&&x-x3>=0.5)++x3;
76     print(x2);
77     if(y>0){out('.');for(size_t i=1;i<y&&x3*
        mul[i]<mul[y];out('0'),++i);print(x3);

```

9.1 FastIO

```

1 namespace fastIO{
2     #define BUF_SIZE 100000
3     #define OUT_SIZE 100000
4     #define ll long long
5     //fread->read
6     bool IOerror=0;
7     // inline char nc(){char ch=getchar();if(ch===-1)
8         IOerror=1;return ch;}
9     inline char nc(){
10         static char buf[BUF_SIZE],*p1=buf+BUF_SIZE,*
11             pend=buf+BUF_SIZE;
12         if(p1==pend){
13             p1=buf;pend=buf+fread(buf,1,BUF_SIZE,
14                 stdin);
15             if(pend==p1){IOerror=1;return -1;}
16         }
17         return *p1++;
18     }
19     inline bool blank(char ch){return ch==' '||ch=='\n'
20         ||ch=='\r' ||ch=='\t';}
21     template<class T> inline bool read(T &x){
22         bool sign=0;char ch=nc();x=0;
23         for(;;blank(ch);ch=nc());
24         if(IOerror)return false;
25         if(ch=='-')sign=1,ch=nc();
26         for(;;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
27         if(sign)x=-x;
28         return true;
29     }
30     inline bool read(double &x){
31         bool sign=0;char ch=nc();x=0;
32         for(;;blank(ch);ch=nc());
33         if(IOerror)return false;
34         if(ch=='-')sign=1,ch=nc();

```

```

        ;}
74     }
75     void print(char *s){while(*s)out(*s++);}
76     void print(const char *s){while(*s)out(*s++)
        ;}
77     void flush(){if(p1!=buf){fwrite(buf,1,p1-buf,
        stdout);p1=buf;}}
78     ~Ostream_fwrite(){flush();}
79     }Ostream;
80     template<class T>void print(T x){Ostream.print(x
        );}
81     inline void print(char x){Ostream.out(x);}
82     inline void print(char *s){Ostream.print(s);}
83     inline void print(string s){Ostream.print(s.
        c_str());}
84     inline void print(const char *s){Ostream.print(s
        );}
85     inline void print(double x,int y){Ostream.print(
        x,y);}
86     template<class T,class... U>void print(const T&
        h,const U&... t){print(h);print(t...);}
87     void println(){print('\n');}
88     template<class T,class... U>void println(const T
        & h,const U&... t){print(h);println(t...);}
89     inline void flush(){Ostream.flush();}
90     #undef ll
91     #undef OUT_SIZE
92     #undef BUF_SIZE
93 };
94 using namespace fastIO;

```

9.2 $O(1)$ 快速乘

```

1 ll mul2(ll x,ll y,ll p)
2 {
3     ll res=(x*y-ll((long double)x/p*y+1.0e-8)*p);
4     return res<0?res+p:res;
5 }

```

9.3 快速模

```

1 typedef long long i64;
2 typedef unsigned long long u64;
3 typedef __uint128_t u128;
4 const int word_bits=sizeof(u64)*8;
5 struct FastMod
6 {
7     static u64 mod,inv,r2;
8     u64 x;
9     FastMod():x(0){}
10    FastMod(u64 n):x(init(n)){}
11    static u64 modulus(){return mod;}

```

```

12    static u64 init(u64 w){return reduce(u128(w)*r2)
        ;}
13    static void set_mod(u64 m)
14    {
15        mod=m;
16        assert(mod&1);
17        inv=m;
18        for(int i=0;i<5;i++) inv*=2-inv*m;
19        r2=-u128(m)%m;
20    }
21    static u64 reduce(u128 x)
22    {
23        u64 y=u64(x>>word_bits)-u64((u128(u64(x)*inv)
        *mod)>>word_bits);
24        return i64(y)<0?y+mod:y;
25    }
26    FastMod& operator+=(FastMod rhs)
27    {
28        x+=rhs.x-mod;
29        if(i64(x)<0) x+=mod;
30        return *this;
31    }
32    FastMod operator+(FastMod rhs)const {return
        FastMod(*this)+=rhs;}
33    FastMod& operator*=(FastMod rhs)
34    {
35        x=reduce(u128(x)*rhs.x);
36        return *this;
37    }
38    FastMod operator*(FastMod rhs)const {return
        FastMod(*this)*=rhs;}
39    u64 get()const {return reduce(x);}
40 };
41 u64 FastMod::mod,FastMod::inv,FastMod::r2;
42 // FastMod::set_mod(p);

```

9.4 xor_sum(1,n)

```

1 ll xor_sum(ll n)
2 {
3     ll t=n&3;
4     if (t&1) return t/2ull^1;
5     return t/2ull^n;
6 }

```

9.5 约瑟夫环 kth

```

1 ll kth(ll n,ll m,ll k)
2 {
3     if(m==1) return k;
4     ll res=(m-1)%(n-k+1);
5     for(ll i=n-k+2,stp=0;i<=n;i+=stp,res+=stp*m)
6     {

```



```

7     if(res+m>=i)
8     {
9         res=(res+m)%i;
10        i++;
11        stp=0;
12    }
13    else
14    {
15        stp=(i-res-2)/(m-1);
16        if(i+stp>n)
17        {
18            res+=(n-(i-1))*m;
19            break;
20        }
21    }
22 }
23 return res+1;
24 }

```

9.6 判断星期几

```

1 int judge(int y,int m,int d)
2 {
3     int res;
4     if(m==1||m==2) m+=12,y--;
5     if((y<1752)||((y==1752&&m<9)||((y==1752&&m==9&&d
6         <3))) res=(d+2*m+3*(m+1)/5+y+y/4+5)%7;
7     else res=(d+2*m+3*(m+1)/5+y+y/4-y/100+y/400)%7;
8     return res+1;
9 }

```

9.7 离散化

```

1 struct Discretization
2 {
3     #define type ll
4     vector<type> a;
5     void init(){a.clear();}
6     void add(type x){a.pb(x);}
7     void work(){sort(all(a));a.resize(unique(all(a))
8         -a.begin());}
9     int get_pos(type x){return lower_bound(all(a),x)
10        -a.begin()+1;}
11     type get_val(int pos){return a[pos-1];}
12     int size(){return a.size();}
13     #undef type
14 }d;

```

9.8 网格整数点正方形个数

```

1 struct node
2 {

```

```

3     int x,y;
4     void input(){scanf("%d%d",&x,&y);}
5 }p[511];
6 int main()
7 {
8     int n,i,j,ans;
9     while(~scanf("%d",&n))
10    {
11        map<pair<int,int>,int> mp;
12        for(i=0;i<n;i++)
13        {
14            p[i].input();
15            mp[MP(p[i].x,p[i].y)]=1;
16        }
17        ans=0;
18        for(i=0;i<n;i++)
19        {
20            for(j=i+1;j<n;j++)
21            {
22                int a,b,c,d,e,f,g,h;
23                a=p[i].x;
24                b=p[i].y;
25                c=p[j].x;
26                d=p[j].y;
27                e=a+b+c-d;
28                f=-a+b+c+d;
29                g=a-b+c+d;
30                h=a+b-c+d;
31                if(abs(e%2)+abs(f%2)+abs(g%2)+abs(h%2)
32                    ==0)
33                {
34                    if(mp[MP(e/2,f/2)]&&mp[MP(g/2,h/2)]
35                        ) ans++;
36                }
37            }
38        }
39        printf("%d\n",ans/2);
40    }
41    return 0;
42 }

```

9.9 模拟退火

```

1 /*简单版
2 1. 模拟退火求费马点->复杂版
3
4 2. 求矩形区域内一点到各点距离之和最短时间复杂度
5
6 cnt*c1*c2*n
7
8 */
9 int sgn(double x)
10 {

```

```

11     if(fabs(x)<eps) return 0;
12     else return x>0?1:-1;
13 }
14 struct Point
15 {
16     double x,y;
17     Point(){}
18     Point(double a,double b)
19     {
20         x=a;
21         y=b;
22     }
23     void input()
24     {
25         scanf("%lf%lf",&x,&y);
26     }
27 };
28 typedef Point Vector;
29 Vector operator -(Vector a,Vector b){return Vector(
30     a.x-b.x,a.y-b.y);}
31 double dot(Vector a,Vector b){return a.x*b.x+a.y*b.
32     y;}
33 double dist(Point a,Point b){return sqrt(dot(a-b,a-
34     b));}
35 double lx,ly;//矩形区域(0,0)-(lx,ly)
36 int check(double x,double y)
37 {
38     if(sgn(x)<0||sgn(y)<0||sgn(x-lx)>0||sgn(y-ly)>0)
39         return 1;
40     return 0;
41 }
42 double Rand(double r,double l)
43 {
44     return(rand()%((int)(1-r)*1000))/(1000.0+r);
45 }
46 double getres(Point t,Point *p,int n)//求距离之和
47 {
48     double res=0;
49     for(int i=0;i<n;i++)
50     {
51         res+=dist(t,p[i]);
52     }
53     return res;
54 }
55 pair<Point,double> SA(Point *p,int n)//模拟退火
56 {
57     srand(time(0));//重置随机种子
58     const double k=0.85;//退火常数
59     const int c1=30;//随机取点的个数
60     const int c2=50;//退火次数
61     Point q[c1+10];//随机取点
62     double dis[c1+10];//每个点的计算结果
63     int i,j;
64     for(i=1;i<=c1;i++)

```

```

61     {
62         q[i]=Point(Rand(0,lx),Rand(0,ly));
63         dis[i]=getres(q[i],p,n);
64     }
65     double tmax=max(lx,ly);
66     double tmin=1e-3;
67     // int cnt计算外层循环次数=0;//
68     while(tmax>tmin)
69     {
70         for(i=1;i<=c1;i++)
71         {
72             for(j=1;j<=c2;j++)
73             {
74                 double ang=Rand(0,2*PI);
75                 Point z;
76                 z.x=q[i].x+cos(ang)*tmax;
77                 z.y=q[i].y+sin(ang)*tmax;
78                 if(check(z.x,z.y)) continue;
79                 double temp=getres(z,p,n);
80                 if(temp<dis[i])
81                 {
82                     dis[i]=temp;
83                     q[i]=z;
84                 }
85             }
86         }
87         // cnt++;
88         tmax*=k;
89     }
90     // cout<<cnt*c1*c2*n<<endl时间复杂度;//
91     int pos=1;
92     for(i=2;i<=c1;i++)
93     {
94         if(dis[i]<dis[pos])
95         {
96             pos=i;
97         }
98     }
99     pair<Point,double> res;
100     res=make_pair(q[pos],dis[pos]);
101     return res;
102 }

```

9.10 矩形面积并

```

1 struct node
2 {
3     ll l,r,h;
4     int tag;
5     friend bool operator <(node a,node b)
6     {
7         return a.h<b.h;
8     }

```

```

9 }seg[MAX<<1]; // 线段
10 ll x[MAX<<1]; // 横坐标离散化
11 struct Segment_Tree
12 {
13     #define ls (id<<1)
14     #define rs (id<<1|1)
15     ll n,ql,qr,qv;
16     ll cover[MAX<<3],len[MAX<<3]; // 注意这里要开倍8
17     void build(ll _n)
18     {
19         mem(cover,0);
20         mem(len,0);
21         n=_n;
22     }
23     void callen(int id,int l,int r)
24     {
25         if(cover[id]) len[id]=x[r+1]-x[l]; // 被整段覆盖
26         else if(l==r) len[id]=0; // 不是一条线段
27         else len[id]=len[ls]+len[rs]; // 是一条线段但又没有被整段覆盖
28     }
29     void update(int l,int r,int id)
30     {
31         if(l>=ql&&r<=qr)
32         {
33             cover[id]+=qv; // 覆盖情况
34             callen(id,l,r);
35             return;
36         }
37         int mid=(l+r)>>1;
38         if(ql<=mid) update(l,mid,ls);
39         if(qr>mid) update(mid+1,r,rs);
40         callen(id,l,r);
41     }
42 }tree;
43 int main()
44 {
45     int n,i,tot,l,r,cnt;
46     ll x1,y1,x2,y2,ans;
47     while(~scanf("%d",&n)&&n)
48     {
49         tot=0;
50         mem(x,0);
51         for(i=0;i<n;i++)
52         {
53             scanf("%lld%lld%lld%lld",&x1,&y1,&x2,&y2)
54             ;
55             // 矩形的左下和右上坐标
56             x[tot]=x1;
57             seg[tot].tag=-1;
58             seg[tot].l=x1;
59             seg[tot].r=x2;
60             seg[tot++].h=y1;

```

```

// 上边界

```

```

61         x[tot]=x2;
62         seg[tot].tag=1;
63         seg[tot].l=x1;
64         seg[tot].r=x2;
65         seg[tot++].h=y2;
66         // 下边界
67     }
68     sort(seg,seg+tot); // 线段按纵坐标升序
69     sort(x,x+tot); // 横坐标升序
70     cnt=unique(x,x+tot)-x;
71     tree.build(cnt-1);
72     ans=0;
73     for(i=0;i<tot;i++)
74     {
75         if(i) ans+=(seg[i].h-seg[i-1].h)*tree.len
76             [1];
77         tree.ql=lower_bound(x,x+cnt-1,seg[i].l)-x
78             ;
79         tree.qr=lower_bound(x,x+cnt-1,seg[i].r)-x
80             -1;
81         tree.qv=seg[i].tag;
82         tree.update(0,cnt-1,1);
83     }
84     printf("%lld\n",ans);
85     return 0;
86 }

```

9.11 维护不同颜色最值和次值

```

1 struct Segment_Tree
2 {
3     #define type int // may need change
4     #define NONE INF // may need change
5     #define ls (id<<1)
6     #define rs (id<<1|1)
7     int n,ql,qr;
8     pair<type,type> a[MAX],mx[2][MAX<<2],mn[2][MAX
9         <<2];
10     pair<type,type> rmx[2],rmn[2];
11     void upmx(int id)
12     {
13         type tx[4]={mx[0][id].fi,mx[1][id].fi,rmx[0].
14             fi,rmx[1].fi};
15         type tc[4]={mx[0][id].se,mx[1][id].se,rmx[0].
16             se,rmx[1].se};
17         int idx[4]={0,1,2,3},cnt=0;
18         sort(idx,idx+4,[&](int a,int b){return tx[a]>
19             tx[b]});
20         for(int i=0;i<4&&cnt<2;i++)
21         {
22             if(i==0||tc[idx[i]]!=tc[idx[i-1]])
23             {

```

```

20         rm[x][cnt].fi=tx[idx[i]];
21         rm[x][cnt++].se=tc[idx[i]];
22     }
23 }
24 for(int i=cnt;i<2;i++) rm[x][i]=MP(-NONE,-NONE)
25 ;
26 void upmn(int id)
27 {
28     type tx[4]={mn[0][id].fi,mn[1][id].fi,rmn[0].
29         fi,rmn[1].fi};
30     type tc[4]={mn[0][id].se,mn[1][id].se,rmn[0].
31         se,rmn[1].se};
32     int idx[4]={0,1,2,3},cnt=0;
33     sort(idx,idx+4,[&](int a,int b){return tx[a]<
34         tx[b];});
35     for(int i=0;i<4&&cnt<2;i++)
36     {
37         if(i==0||tc[idx[i]]!=tc[idx[i-1]])
38         {
39             rmn[cnt].fi=tx[idx[i]];
40             rmn[cnt++].se=tc[idx[i]];
41         }
42     }
43     for(int i=cnt;i<2;i++) rmn[i]=MP(NONE,-NONE);
44 }
45 void pushup(int id)
46 {
47     mx[0][id]=mx[0][ls];
48     rm[x][0]=mx[0][rs];
49     mx[1][id]=mx[1][ls];
50     rm[x][1]=mx[1][rs];
51     upmx(id);
52     mx[0][id]=rm[x][0];
53     mx[1][id]=rm[x][1];
54
55     mn[0][id]=mn[0][ls];
56     rmn[0]=mn[0][rs];
57     mn[1][id]=mn[1][ls];
58     rmn[1]=mn[1][rs];
59     upmn(id);
60     mn[0][id]=rmn[0];
61     mn[1][id]=rmn[1];
62 }
63 void build(int l,int r,int id)
64 {
65     mx[1][id]=MP(-NONE,-NONE);
66     mn[1][id]=MP(NONE,-NONE);
67     if(l==r)
68     {
69         mx[0][id]=mn[0][id]=a[l];
70         return;
71     }
72     int mid=(l+r)>>1;

```

```

70     build(l,mid,ls);
71     build(mid+1,r,rs);
72     pushup(id);
73 }
74 void update(int l,int r,int id)//only l==r
75 {
76     mx[1][id]=MP(-NONE,-NONE);
77     mn[1][id]=MP(NONE,-NONE);
78     if(l>=ql&&r<=qr)
79     {
80         mx[0][id]=mn[0][id]=a[l];
81         return;
82     }
83     int mid=(l+r)>>1;
84     if(ql<=mid) update(l,mid,ls);
85     if(qr>mid) update(mid+1,r,rs);
86     pushup(id);
87 }
88 void query(int l,int r,int id)
89 {
90     if(l>=ql&&r<=qr)
91     {
92         upmx(id);
93         upmn(id);
94         return ;
95     }
96     int mid=(l+r)>>1;
97     if(ql<=mid) query(l,mid,ls);
98     if(qr>mid) query(mid+1,r,rs);
99 }
100 void build(int _n){n=_n;build(1,n,1);}
101 void upd(int l,int r)
102 {
103     ql=l;
104     qr=r;
105     update(1,n,1);
106 }
107 void ask(int l,int r)
108 {
109     rm[x][0]=rm[x][1]=MP(-NONE,-NONE);
110     rmn[0]=rmn[1]=MP(NONE,-NONE);
111     ql=l;
112     qr=r;
113     query(1,n,1);
114
115     // something
116 }
117 #undef type
118 #undef NONE
119 #undef ls
120 #undef rs
121 }tr;

```

10 附录

10.1 NTT 常用模数

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

4179340454199820289,29,57,3

10.2 树 hash

Method I

Formula

$$f_{now} = size_{now} \times \sum f_{son_{now,i}} \times seed^{i-1}$$

Notes

其中 f_x 为以节点 x 为根的子树对应的哈希值。特殊地，我们令叶子节点的哈希值为 1。

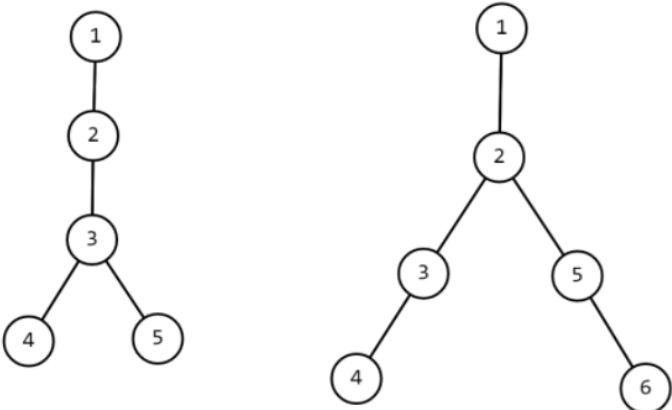
$size_x$ 表示以节点 x 为根的子树大小。

$son_{x,i}$ 表示 x 所有子节点以 f 作为关键字排序后排名第 i 的儿子。

$seed$ 为选定的一个合适的种子（最好是质数，对字符串 hash 有了解的人一定不陌生）

上述哈希过程中，可以适当取模避免溢出或加快运行速度。

Hack



上图中，可以计算出两棵树的哈希值均为 $60(1 + seed)$ 。

Method II

Formula

$$f_{now} = \bigoplus f_{son_{now,i}} \times seed + size_{son_{now,i}}$$

Notes

其中 f_x 为以节点 x 为根的子树对应的哈希值。特殊地，我们令叶子节点的哈希值为 1。

$size_x$ 表示以节点 x 为根的子树大小。

$son_{x,i}$ 表示 x 所有子节点之一（不用排序）。

$seed$ 为选定的一个合适的质数。

\bigoplus 表示异或和。

Hack

由于异或的性质，如果一个节点下有多棵本质相同的子树，这种哈希值将无法分辨该种子树出现 1, 3, 5, ... 次的情况。

Method III

Formula

$$f_{now} = 1 + \sum f_{son_{now,i}} \times prime(size_{son_{now,i}})$$

Notes

其中 f_x 为以节点 x 为根的子树对应的哈希值。

$size_x$ 表示以节点 x 为根的子树大小。

$son_{x,i}$ 表示 x 所有子节点之一（不用排序）。

$prime(i)$ 表示第 i 个质数。

10.3 线性基求交

```

1  LBasis intersection(const LBasis &a, const LBasis &
   b){
2      LBasis ans, c = b, d = b;
3      ans.init();
4      for (int i = 0; i <= 32; i++){
5          ll x = a.d[i];
6          if(!x)continue;
7          int j = i;
8          ll T = 0;
9          for(; j >= 0; --j){
10             if((x >> j) & 1)
11                 if(c.d[j]) {x ^= c.d[j]; T ^= d.d[j];}
12                 else break;
13             }
14             if(!x) ans.d[i] = T;
15             else {c.d[j] = x; d.d[j] = T;}
16         }
17         return ans;
18     }

```