

ACM-ICPC Template

tokitsukaze

2023 年 12 月 7 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	2		
1.2.1 hash	2		
1.2.2 good_hash_prime	2		
1.2.3 hash_map	2		
1.2.4 BKDRHash	2		
1.3 Manacher	3		
1.3.1 插字符	3		
1.3.2 不插字符	3		
1.4 后缀数组	4		
1.4.1 倍增 sa	4		
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	6		
1.5.3 后缀自动机	7		
1.5.4 回文自动机	8		
1.5.5 序列自动机	9		
1.6 最小表示法	9		
1.6.1 最小表示法	9		
1.6.2 最大表示法	9		
1.7 shift_and	9		
2 数据结构	10		
2.1 离散化	10		
2.2 RMQ	10		
2.2.1 一维 RMQ	10		
2.2.2 二维 RMQ	10		
2.3 单调队列	11		
2.4 并查集	11		
2.4.1 并查集	11		
2.4.2 map 实现并查集	12		
2.4.3 可撤销并查集	12		
2.5 树状数组	12		
2.5.1 一维单点 BIT	12		
2.5.2 一维区间 BIT	13		
2.5.3 二维单点 BIT	13		
2.6 线段树	14		
2.6.1 线段树	14		
2.6.2 动态开点线段树	14		
2.6.3 线段树分裂合并	15		
2.6.4 区间查询最大子段和	16		
2.6.5 矩形面积并	17		
2.7 平衡树	18		
2.7.1 Treap	18		
2.7.2 Splay 维护序列	20		
2.7.3 pbds	22		
2.8 字典树	22		
2.8.1 trie	22		
2.8.2 01trie	23		
2.9 可持久化	23		
2.9.1 可持久化线段树	23		
2.10 树套树	24		
2.10.1 线段树套线段树	24		
2.10.2 线段树套 treap	25		
2.10.3 树状数组套 treap	28		
2.11 李超树	31		
2.12 kd-tree	32		
2.13 pbds 可并堆	34		
2.14 k 叉哈夫曼树	34		
2.15 笛卡尔树	34		
2.16 析合树	35		
2.17 莫队算法	37		
2.18 ODT	37		
2.19 分块	38		
3 树	38		
3.1 LCA	38		
3.1.1 倍增 LCA	38		
3.1.2 RMQ 维护欧拉序求 LCA	39		
3.2 树链剖分	40		
3.2.1 轻重链剖分	40		
3.3 树的重心	41		
3.4 树 hash	42		
3.5 虚树	42		
3.6 Link-Cut-Tree	42		
4 图论	44		
4.1 链式前向星	44		
4.2 最短路	44		
4.2.1 dijkstra	44		
4.2.2 spfa	45		
4.2.3 floyd 求最小环	46		
4.2.4 Johnson	46		
4.2.5 同余最短路	47		
4.3 最小生成树	48		
4.3.1 kruskal	48		
4.3.2 kruskal 重构树	49		
4.3.3 prim	49		
4.4 二分图匹配	50		
4.4.1 二分图最大匹配	50		
4.4.2 二分图最大权完美匹配	51		
4.5 最大流	52		
4.5.1 dinic	52		
4.5.2 有源汇上下界网络流	53		

4.6	费用流	54	6	多项式	72
4.6.1	spfa 费用流	54	6.1	FFT	72
4.6.2	dijkstra 费用流 (dij 求 h)	55	6.2	NTT	73
4.6.3	dijkstra 费用流 (spfa 求 h)	56	6.3	FWT	74
4.7	连通性	57	6.4	拉格朗日插值	75
4.7.1	强连通分量	57	7	矩阵	75
4.7.2	边双连通分量	58	7.1	矩阵类	75
4.7.3	圆方树	59	7.2	高斯消元	76
4.8	团	59	7.2.1	浮点数方程组	76
4.8.1	最大团	59	7.2.2	异或方程组	77
4.8.2	极大团计数	60	7.3	单纯形	78
4.9	拓扑排序	61	7.4	线性基	79
4.10	2-sat	61	7.4.1	线性基	79
4.10.1	2-sat 输出任意解	61	7.4.2	带删除线性基	79
4.10.2	2-sat 字典序最小解	62	8	博弈	80
4.11	支配树	63	8.1	sg 函数	80
4.12	最小斯坦纳树	64	8.2	结论	80
5	数论	65	9	dp	81
5.1	素数筛	65	9.1	LIS	81
5.1.1	埃筛	65	9.2	LPS	81
5.1.2	线性筛	65	9.3	数位 dp	81
5.1.3	区间筛	65	10	杂项	82
5.2	逆元	65	10.1	FastIO	82
5.2.1	exgcd 求逆元	65	10.2	O(1) 快速乘	83
5.2.2	线性预处理	65	10.3	快速模	83
5.3	扩展欧几里得	65	10.4	xor_sum(1,n)	83
5.3.1	exgcd	65	10.5	约瑟夫环 kth	83
5.3.2	ax+by=c	66	10.6	判断星期几	84
5.4	中国剩余定理	66	10.7	整数三分	84
5.4.1	CRT	66	10.8	有根树与 prufer 序列的转换	84
5.4.2	exCRT	66	10.9	网格整数点正方形个数	84
5.5	组合数	66	10.10	模拟退火	84
5.5.1	打表	66	10.11	斐波那契 01 串的第 k 个字符	85
5.5.2	预处理	67	10.12	光速幂	86
5.5.3	Lucas 定理	67	10.13	倍增求等比数列和	86
5.5.4	exLucas	67	11	附录	86
5.6	欧拉函数	68	11.1	NTT 常用模数	86
5.6.1	直接求	68	11.2	线性基求交	87
5.6.2	线性筛	68			
5.7	莫比乌斯函数	68			
5.8	Berlekamp-Massey	69			
5.9	exBSGS	70			
5.10	Miller_Rabin+Pollard_rho	70			
5.11	第二类 Stirling 数	71			
5.12	原根	71			
5.13	二次剩余	72			

1 字符串

1.1 KMP

1.1.1 KMP

```

1 struct KMP
2 {
3     #define type char
4     int nex[MAX],len;
5     type t[MAX];
6     void get_next(type *s,int n)
7     {
8         int i,j;
9         for(i=1;i<n;i++) t[i]=s[i];
10        t[n+1]=0;
11        nex[0]=nex[1]=0;
12        j=0;
13        for(i=2;i<n;i++)
14        {
15            while(j&&t[j+1]!=s[i]) j=nex[j];
16            if(t[j+1]==s[i]) j++;
17            nex[i]=j;
18        }
19    }
20
21    // s[1..n], return all pos t in s
22    vector<int> match(type *s,int n)
23    {
24        int i,j;
25        vector<int> res;
26        for(i=1,j=0;i<n;i++)
27        {
28            while(j&&t[j+1]!=s[i]) j=nex[j];
29            if(t[j+1]==s[i]) j++;
30            if(j==len)
31            {
32                res.push_back(i-len+1);
33                j=nex[j];
34            }
35        }
36        return res;
37    }
38    #undef type
39 }kmp;// kmp.get_next(s,n); s[1..n]
```

1.1.2 exKMP

```

1 struct Z_Algorithm
2 {
3     char s[MAX];
4     int n,z[MAX],ex[MAX];
5     void get_z_func(char *_s,int _n) // s[0..n-1]
6     {
```

```

7         int i,j,l,r;
8         n=_n;
9         memcpy(s,_s,n);
10        z[0]=l=r=0;
11        for(i=1;i<n;i++)
12        {
13            if(i+z[i-1]-1<r) z[i]=z[i-1];
14            else
15            {
16                j=max(0,r-i+1);
17                while(i+j<n&&s[i+j]==s[j]) j++;
18                z[i]=j;
19                if(i+z[i]-1>r)
20                {
21                    l=i;
22                    r=i+z[i]-1;
23                }
24            }
25        }
26        z[0]=n;
27    }
28    void get_ex(char *t,int m) // t[0..m-1]
29    {
30        int i,j,l,r;
31        j=l=0;
32        while(j<n&&j<m&&t[j]==s[j]) j++;
33        ex[0]=j;
34        r=l+ex[0]-1;
35        for(i=1;i<m;i++)
36        {
37            if(i+z[i-1]-1<r) ex[i]=z[i-1];
38            else
39            {
40                j=max(0,r-i+1);
41                while(i+j<m&&t[i+j]==s[j]) j++;
42                ex[i]=j;
43                if(i+ex[i]-1>r)
44                {
45                    l=i;
46                    r=i+ex[i]-1;
47                }
48            }
49        }
50    }
51 }z;
52 /*
53 z[i]: lcp(s,s[i..n-1]) i=0..n-1
54 ex[i]: lcp(s,t[i..m-1]) i=0..m-1
55
56 z.get_z_func(s,n) s[0..n-1]
57 z.get_ex(t,m) t[0..m-1]
58 */
```

1.2 hash

1.2.1 hash

```

1 mt19937_64 rd(time(0));
2 struct hash_table
3 {
4     ll seed,p;
5     ll Hash[MAX],tmp[MAX];
6     void set(ll _p)
7     {
8         seed=rd()%_p;
9         p=_p;
10    }
11    void work(char *s,int n)
12    {
13        tmp[0]=1;
14        Hash[0]=0;
15        for(int i=1;i<=n;i++)
16        {
17            tmp[i]=tmp[i-1]*seed%p;
18            Hash[i]=(Hash[i-1]*seed+s[i])%p;//may
19                need change
20        }
21        ll get(int l,int r)
22        {
23            return ((Hash[r]-Hash[l-1]*tmp[r-l+1])%p+p)%p
24                ;
25        }
26    };

```

1.2.2 good_hash_prime

```

1 vector<ll> good_hash_prime={
2     50331653,100663319,201326611,
3     402653189,805306457,1610612741
4 };
5 vector<ll> get_hash_prime(int cnt=2)
6 {
7     assert(cnt<=good_hash_prime.size());
8     srand(time(0));
9     random_shuffle(good_hash_prime.begin(),
10         good_hash_prime.end());
11     return vector<ll>(good_hash_prime.begin(),
12         good_hash_prime.begin()+cnt);
13 }

```

1.2.3 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;

```

1.2.4 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    }
25 }

```

```

23     return 0;
24 }
25 void work(char *s,int n)
26 {
27     tmp[0]=1;
28     Hash[0]=0;
29     for(int i=1;i<=n;i++)
30     {
31         tmp[i]=tmp[i-1]*seed;
32         Hash[i]=Hash[i-1]*seed+s[i];
33     }
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,int n) // a[1..n]
6     {
7         int i,mid,r,res=0;
8         for(i=1;i<=n;i++)
9         {
10             p[i]=0;
11             s[2*i-1]='%';
12             s[2*i]=a[i];
13         }
14         s[n*2+1]='%';
15         mid=r=0;
16         for(i=1;i<=n;i++)
17         {
18             if(i<r) p[i]=min(p[2*mid-i],r-i);
19             else p[i]=1;
20             while(i-p[i]>=1&&i+p[i]<=n&&s[i-p[i]]==s[i+p[i]]) p[i]++;
21             if(i+p[i]>r)
22             {
23                 r=i+p[i];
24                 mid=i;
25             }
26             res=max(res,p[i]-1);
27         }
28         return res;
29     }
30 }la;

```

1.3.2 不插字符

```

1 struct Manacher
2 {
3     int p[MAX];
4     int work(char *s,int n) //s[1..n]
5     {
6         int r,mid,i,res=0;
7         //odd
8         r=mid=0;
9         for(i=0;i<=n;i++) p[i]=0;
10        for(i=1;i<=n;i++)
11        {
12            //palindrome substring s[i,i]
13            if(r>i) p[i]=min(p[2*mid-i],r-i);
14            while(i+p[i]+1<=n&&s[i+p[i]+1]==s[i-p[i]-1])
15            {
16                //palindrome substring s[i-p[i]-1,i+p[i]+1]
17                p[i]++;
18            }
19            if(i+p[i]>r)
20            {
21                r=i+p[i];
22                mid=i;
23            }
24            res=max(res,p[i]*2+1);
25        }
26        //even
27        r=mid=0;
28        for(i=0;i<=n;i++) p[i]=0;
29        for(i=2;i<=n;i++)
30        {
31            if(r>i) p[i]=min(p[2*mid-i],r-i+1);
32            while(i+p[i]<=n&&s[i+p[i]]==s[i-p[i]-1])
33            {
34                //palindrome substring s[i-p[i]-1,i+p[i]]
35                p[i]++;
36            }
37            if(i+p[i]-1>r)
38            {
39                r=i+p[i]-1;
40                mid=i;
41            }
42            res=max(res,p[i]*2);
43        }
44        return res;
45    }
46 }la;

```

1.4 后缀数组

1.4.1 倍增 sa

```

1 struct Suffix_Array
2 {
3     int s[MAX],n,SZ;
4     int c[MAX],rk[MAX],tmp[MAX],sa[MAX],h[MAX];
5     void get_sa()
6     {
7         int m,i,j,k,tot;
8         m=SZ;
9         for(i=1;i<=m;i++) c[i]=0;
10        for(i=1;i<=n;i++) c[rk[i]=s[i]]++;
11        for(i=2;i<=m;i++) c[i]+=c[i-1];
12        for(i=n;i;i--) sa[c[rk[i]]--]=i;
13        for(k=1;k<=n;k<=1)
14        {
15            tot=0;
16            for(i=n-k+1;i<=n;i++) tmp[++tot]=i;
17            for(i=1;i<=n;i++)
18            {
19                if(sa[i]>k) tmp[++tot]=sa[i]-k;
20            }
21            for(i=1;i<=m;i++) c[i]=0;
22            for(i=1;i<=n;i++) c[rk[i]]++;
23            for(i=2;i<=m;i++) c[i]+=c[i-1];
24            for(i=n;i;i--)
25            {
26                sa[c[rk[tmp[i]]]--]=tmp[i];
27                tmp[i]=0;
28            }
29            swap(rk,tmp);
30            rk[sa[1]]=1;
31            tot=1;
32            for(i=2;i<=n;i++)
33            {
34                if(sa[i]+k>n||sa[i-1]+k>n||
35                    (!(tmp[sa[i]]==tmp[sa[i-1]]&&
36                        tmp[sa[i]+k]==tmp[sa[i-1]+k])))
37                    tot++;
38                rk[sa[i]]=tot;
39            }
40            if(tot==n) break;
41            m=tot;
42        }
43        h[1]=0;
44        k=0;
45        for(i=1;i<=n;i++)
46        {
47            if(rk[i]==1) continue;
48            if(k>0) k--;
49            j=sa[rk[i]-1];
50            while(j+k<=n&&i+k<=n&&s[i+k]==s[j+k]) k
51                ++;

```

```

50         h[rk[i]]=k;
51     }
52 }
53 void work(char *_s,int _n) //s[1..n]
54 {
55     SZ=0; // char size
56     n=_n;
57     for(int i=1;i<=n;i++)
58     {
59         s[i]=_s[i];
60         SZ=max(SZ,s[i]);
61     }
62     get_sa();
63 }
64 }sa;
65 /*
66 sa[i]: s[sa[i]..n] rank is i
67 rk[i]: s[i..n] rank is rk[i]
68 h[i]: lcp(s[sa[i]..n],s[sa[i-1]..n])
69 lcp(s[i..n],s[j..n]) (i<j): min{h[i+1..j]}
70
71 sa.work(s,n) s[1..n]
72 */

```

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    }
13    #define ps(x) sa[w[s[x]]--]=x
14    #define pl(x) sa[w[s[x]]++]=x
15    inline void radix(int* v,int* s,int* t,int n,int
16        m,int n1){
17        memset(sa,0,n+1<<2); memset(c,0,m+1<<2);
18        for(int i=1;i<=n;++i) ++c[s[i]];
19        for(int i=1;i<=m;++i) w[i]=c[i]+c[i-1];
20        for(int i=n1;i--i) ps(v[i]);
21        for(int i=1;i<=m;++i) w[i]=c[i-1]+1;
22        for(int i=1;i<=n;++i) if(sa[i]>1 && t[sa[i]
23            ]-1) pl(sa[i]-1);
24        for(int i=1;i<=m;++i) w[i]=c[i];
25        for(int i=n;i--i) if(sa[i]>1 && !t[sa[i]-1])
26            ps(sa[i]-1);
27    }

```

```

24 inline void SAIS(int n,int m,int* s,int* t,int*
    p){
25     int n1=0,ch=r[1]=0,*s1=s+n; t[n]=0;
26     for(int i=n-1;i-->0) t[i]=s[i]==s[i+1]?t[i+1]:s[i]>s[i+1];
27     for(int i=2;i<=n;++i) r[i]=t[i-1]&&!t[i]?(p[
        ++n1]=i,n1):0;
28     radix(p,s,t,n,m,n1);
29     for(int i=1,x,y;i<=n;++i) if(x=r[sa[i]]){
30         if(ch<=1 || p[x+1]-p[x]!=p[y+1]-p[y]) ++
            ch;
31         else for(int j=p[x],k=p[y];j<=p[x+1];++j
            ,++k)
32             if((s[j]<<1|t[j])^(s[k]<<1|t[k])){ ++
                ch; break; }
33         s1[y=x]=ch;
34     }
35     if(ch<n1) SAIS(n1,ch,s1,t+n,p+n1);
36     else for(int i=1;i<=n1;++i) sa[s1[i]]=i;
37     for(int i=1;i<=n1;++i) s1[i]=p[sa[i]];
38     radix(s1,s,t,n,m,n1);
39 }
40 inline void get_sa(int n,const char* S,int *ssa,
    int *h){
41     int m=trans(++n,S); SAIS(n,m,s,t,p);
42     for(int i=1;i<=n;++i) r[sa[i]=sa[i+1]]=i;
43     for(int i=1,j,k=0;i<=n;++i) if(r[i]>1){
44         for(j=sa[r[i]-1];S[i+k]==S[j+k];++k);
45         if(H[r[i]]<k) --k;
46     }
47     for(int i=1;i<=n;i++)
48     {
49         ssa[i]=sa[i];
50         h[i]=H[i];
51     }
52 }
53 }sais;

```

1.5 自动机

1.5.1 AC 自动机

```

1 struct AC_Automaton
2 {
3     static const int K=;
4     int nex[MAX][K],fail[MAX],cnt[MAX],last[MAX];
5     int root,tot,pos[MAX];
6     int getid(char c){return c-;}//may need change
7     int newnode()
8     {
9         memset(nex[tot],0,sizeof nex[tot]);
10        fail[tot]=0;
11        cnt[tot]=0;
12        return tot++;

```

```

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

```



```

66     for(i=1;i<tot;i++)
67     {
68         if(!out[i]) q.push(i);
69     }
70     while(!q.empty())
71     {
72         t=q.front();
73         q.pop();
74
75         // do something
76
77         out[fail[t]]--;
78         if(out[fail[t]]==0) q.push(fail[t]);
79     }
80 }
81 int query(char *s,int n)
82 {
83     int len,now,i,res,t,tmp;
84     now=root;
85     res=0;
86     vector<pair<int,int>> del;
87     for(i=0;i<n;i++)
88     {
89         t=getid(s[i]);
90         now=nex[now][t];
91         tmp=now;
92         while(tmp&&cnt[tmp]!=-1)
93         {
94             res+=cnt[tmp];
95             del.push_back({tmp,cnt[tmp]});
96             cnt[tmp]=-1;
97             tmp=last[tmp];
98         }
99     }
100     for(auto &it:del) cnt[it.first]=it.second;
101     return res;
102 }
103 void build_fail_tree(vector<int> mp[])
104 {
105     for(int i=0;i<=tot;i++) mp[i].clear();
106     for(int i=1;i<tot;i++) mp[fail[i]].push_back(
107         i);
108 }
109 /*
110 i is the suffix for each node in the subtree(i) of
111 the fail tree
112 ac.init();
113 ac.insert(s,len,id); s[0..len-1], id:1..n
114 ac.work();
115 ac.query(s,len); s[0..len-1]
116 ac.build_fail_tree(mp);
117 */

```

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])

```

```

52     {
53         int fail_now=fail[now];
54         while(fail_now>1&&!nex[fail_now].count
            (it.fi)) fail_now=fail[fail_now];
55         if(nex[fail_now].count(it.fi)) fail[it
            .se]=nex[fail_now][it.fi];
56         else fail[it.se]=root;
57         q.push(it.se);
58     }
59 }
60 }
61 int query(VI& s,int x)
62 {
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
            [now];
71         if(nex[now].count(t)) now=nex[now][t];
72         else now=root;
73         //do something
74     }
75     return res;
76 }
77 void toptans()
78 {
79     for(int i=sz(toplist)-1;~i;i--)*do something
        */;
80 }
81 }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],
        maxlen;
6     int newnode()
7     {
8         tot++;
9         fa[tot]=len[tot]=cnt[tot]=0;
10        mem(nex[tot],0);
11        return tot;
12    }
13    void init()
14    {
15        fa[0]=len[0]=cnt[0]=0;
16        mem(nex[0],0);

```

```

17        tot=0;
18        maxlen=0;
19        last=newnode();
20    }
21    void add(int x)
22    {
23        int p,q,np,nq;
24        p=last;
25        np=last=newnode();
26        len[np]=len[p]+1;
27        maxlen=max(maxlen,len[np]);
28        cnt[last]=1;
29        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-->0) 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     static const int N=MAX;
4     static const int LOGN=log2(N)+3;
5     static const int K=26;// char size: [0,25]
6     int len[N],nex[N][K],fail[N],last,pos[N],s[N],
7         tot,n;
8     int cnt[N],dep[N];
9     int newnode(int l)
10     {
11         memset(nex[tot],0,sizeof nex[tot]);
12         fail[tot]=0;
13         dep[tot]=cnt[tot]=0;
14         len[tot]=l;

```

```

14         return tot++;
15     }
16 void init()
17 {
18     tot=n=last=0;
19     newnode(0);
20     newnode(-1);
21     s[0]=-1;
22     fail[0]=1;
23 }
24 int get_fail(int x)
25 {
26     while(s[n-len[x]-1]!=s[n]) x=fail[x];
27     return x;
28 }
29 void add(int t,int p)
30 {
31     int id,now;
32     s[++n]=t;
33     now=get_fail(last);
34     if(!nex[now][t])
35     {
36         id=newnode(len[now]+2);
37         fail[id]=nex[get_fail(fail[now])][t];
38         dep[id]=dep[fail[id]]+1;
39         nex[now][t]=id;
40     }
41     last=nex[now][t];
42     cnt[last]++;
43     pos[p]=last;
44 }
45 void topsort()
46 {
47     for(int i=tot-1;i;i--) cnt[fail[i]]+=cnt[i];
48 }
49 int st[N][LOGN];
50 void init_ST()
51 {
52     int i,j;
53     for(i=2;i<=tot;i++)
54     {
55         st[i][0]=fail[i];
56         for(j=1;j<LOGN;j++)
57         {
58             st[i][j]=st[st[i][j-1]][j-1];
59         }
60     }
61 }
62 int get_substr(int l,int r)//init_ST
63 {
64     int now,tmp,i;
65     now=pos[r];
66     for(i=LOGN-1;~i;i--)
67     {

```

```

68         if(len[st[now][i]]>=r-l+1) now=st[now][i];
69     };
70     }
71     //maybe need judge if len[now]==r-l+1
72     return now;
73 }
74 void build_tree(vector<int> mp[])// root is 0
75 {
76     for(int i=0;i<=tot+1;i++) mp[i].clear();
77     for(int i=1;i<tot;i++) mp[fail[i]].push_back(i);
78 }
79 }pam;
80 /*
81 pam.init();
82 pam.add(t,id); t is int
83 */

```

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,int n) // s[0..n-1]
2 {
3     int i,j,k,tmp;
4     i=k=0;
5     j=1;
6     while(i<n&&j<n&&k<n)
7     {
8         tmp=s[(i+k)%n]-s[(j+k)%n];
9         if(!tmp) k++;
10        else
11        {
12            if(tmp>0) i=i+k+1;
13            else j=j+k+1;
14            if(i==j) j++;

```

```

15         k=0;
16     }
17 }
18 return i<j?i:j;
19 }

```

1.6.2 最大表示法

```

1 int max_representation(char *s,int n) // s[0..n-1]
2 {
3     int i,j,k,tmp;
4     i=k=0;
5     j=1;
6     while(i<n&&j<n&&k<n)
7     {
8         tmp=s[(i+k)%n]-s[(j+k)%n];
9         if(!tmp) k++;
10        else
11        {
12            if(tmp<0) i=i+k+1;
13            else j=j+k+1;
14            if(i==j) j++;
15            k=0;
16        }
17    }
18    return i<j?i:j;
19 }

```

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

```

```

23     }
24 }
25 }

```

2 数据结构

2.1 离散化

```

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

```

2.2 RMQ

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

```

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

```

2.3 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;

```

2.4 并查集

2.4.1 并查集

```

1 struct Disjoint_Set_Union
2 {
3     int pre[MAX],sz[MAX];
4     void init(int n)
5     {
6         int i;
7         for(i=1;i<=n;i++)
8         {
9             pre[i]=i;
10            sz[i]=1;
11        }
12    }
13    int find(int x)
14    {
15        if(pre[x]!=x) pre[x]=find(pre[x]);
16        return pre[x];
17    }
18    bool merge(int a,int b)

```

```

19     {
20         int ra,rb;
21         ra=find(a);
22         rb=find(b);
23         if(ra!=rb)
24         {
25             pre[ra]=rb;
26             sz[rb]+=sz[ra];
27             return 1;
28         }
29         return 0;
30     }
31 }dsu;

```

2.4.2 map 实现并查集

```

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

```

2.4.3 可撤销并查集

```

1 struct Disjoint_Set_Union
2 {
3     pair<int,int> 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++]={ra,rb};
30     return 1;
31 }
32 void roll_back()
33 {
34     PII now=st[--top];
35     pre[now.first]=now.first;
36     sz[now.second]-=sz[now.first];
37 }
38 }dsu;

```

2.5 树状数组

2.5.1 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    type get(int x)
13    {
14        type res=0;
15        while(x)
16        {
17            res+=bit[x];
18            x-=lowbit(x);
19        }
20        return res;

```

```

21     }
22     void upd(int x,type v)
23     {
24         while(x<=n)
25         {
26             bit[x]+=v;
27             x+=lowbit(x);
28         }
29     }
30     type ask(int l,int r)
31     {
32         if(l>r) return 0;
33         if(l-1<=0) return get(r);
34         return get(r)-get(l-1);
35     }
36     #undef type
37 }tr;

```

2.5.2 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    void _insert(int x,type v)
16    {
17        for(int i=x;i<=n;i+=lowbit(i))
18        {
19            bit[i][0]+=v;
20            bit[i][1]+=v*(x-1);
21        }
22    }
23    type get(int x)
24    {
25        type res=0;
26        for(int i=x;i>=1;i-=lowbit(i))
27        {
28            res+=x*bit[i][0]-bit[i][1];
29        }
30        return res;
31    }
32    void upd(int l,int r,type v)
33    {
34        _insert(l,v);

```

```

35        _insert(r+1,-v);
36    }
37    type ask(int l,int r)
38    {
39        if(l-1<=0) return get(r);
40        return get(r)-get(l-1);
41    }
42    #undef type
43 }tr;

```

2.5.3 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>=1;i-=lowbit(i))
23        {
24            for(j=y;j>=1;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            x1,y1);
37    }
38    #undef type
39 }tr;

```


2.6 线段树

2.6.1 线段树

```

1 struct Segment_Tree
2 {
3     #define type int
4     #define ls (id<<1)
5     #define rs (id<<1|1)
6     struct node
7     {
8         type v;
9         void init()
10        {
11
12        }
13    }t[MAX<<2];
14    int n,ql,qr,qop;
15    type a[MAX],tag[MAX<<2],qv;
16    node merge(node x,node y)
17    {
18        node res;
19
20        return res;
21    }
22    void pushup(int id){t[id]=merge(t[ls],t[rs]);}
23    void pushdown(int l,int r,int id)
24    {
25        if(!tag[id]) return;
26        int mid=(l+r)>>1;
27
28    }
29    void build(int l,int r,int id)
30    {
31        tag[id]=0;
32        t[id].init();
33        if(l==r)
34        {
35            //init
36            return;
37        }
38        int mid=(l+r)>>1;
39        build(l,mid,ls);
40        build(mid+1,r,rs);
41        pushup(id);
42    }
43    void update(int l,int r,int id)
44    {
45        if(l>=ql&&r<=qr)
46        {
47
48            return;
49        }
50        pushdown(l,r,id);
51        int mid=(l+r)>>1;

```

```

52        if(ql<=mid) update(l,mid,ls);
53        if(qr>mid) update(mid+1,r,rs);
54        pushup(id);
55    }
56    node query(int l,int r,int id)
57    {
58        if(l>=ql&&r<=qr) return t[id];
59        pushdown(l,r,id);
60        int mid=(l+r)>>1;
61        if(qr<=mid) return query(l,mid,ls);
62        if(ql>mid) return query(mid+1,r,rs);
63        return merge(query(l,mid,ls),query(mid+1,r,rs));
64    }
65    void build(int _n){n=_n;build(1,n,1);}
66    void upd(int l,int r,type v)
67    {
68        ql=l;
69        qr=r;
70        qv=v;
71        update(1,n,1);
72    }
73    type ask(int l,int r)
74    {
75        ql=l;
76        qr=r;
77        return query(1,n,1).v;
78    }
79    #undef type
80    #undef ls
81    #undef rs
82 }tr;

```

2.6.2 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     static const int LOGN=31;
6     int root,tot,ls[MAX*LOGN],rs[MAX*LOGN],ql,qr,n;
7     type v[MAX*LOGN],tag[MAX*LOGN],qv;
8     int newnode()
9     {
10        ls[tot]=rs[tot]=0;
11        v[tot]=0;
12        tag[tot]=-1;
13        return tot++;
14    }
15    void build(int _n)
16    {
17        n=_n;
18        tot=0;
19        root=newnode();

```

```

20     }
21     void pushup(int id)
22     {
23
24     }
25     void pushdown(int id)
26     {
27         if(tag[id]==-1) return;
28         if(!ls[id]) ls[id]=newnode();
29         if(!rs[id]) rs[id]=newnode();
30
31
32         tag[id]=-1;
33     }
34     void update(int l,int r,int &id)
35     {
36         if(!id) id=newnode();
37         if(l>=ql&&r<=qr)
38         {
39             //do something
40             return;
41         }
42         pushdown(id);
43         int mid=(l+r)>>1;
44         if(ql<=mid) update(l,mid,ls[id]);
45         if(qr>mid) update(mid+1,r,rs[id]);
46         pushup(id);
47     }
48     type res;
49     void query(int l,int r,int &id)
50     {
51         if(!id) return;
52         if(l>=ql&&r<=qr)
53         {
54             //do something
55             return;
56         }
57         pushdown(id);
58         int mid=(l+r)>>1;
59         if(ql<=mid) query(l,mid,ls[id]);
60         if(qr>mid) query(mid+1,r,rs[id]);
61     }
62     void upd(int l,int r,type v)
63     {
64         ql=l;
65         qr=r;
66         qv=v;
67         update(1,n,root);
68     }
69     type ask(int l,int r)//init res
70     {
71         ql=l;
72         qr=r;
73         res=0;

```

```

74         query(1,n,root);
75         return res;
76     }
77     #undef type
78 };

```

2.6.3 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        tag[id]=-1;
39    }
40    int split(int l,int r,int &id)
41    {
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;

```

2.6.4 区间查询最大子段和

```

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],mx[MAX<<2],lv[MAX<<2],rv[MAX<<2],v[
8         MAX<<2],qv;
9     void pushup(int id)
10    {
11        mx[id]=max(mx[ls],mx[rs]);
12        mx[id]=max(mx[id],rv[ls]+lv[rs]);
13        lv[id]=max(lv[ls],lv[rs]+v[ls]);
14        rv[id]=max(rv[rs],rv[ls]+v[rs]);
15        v[id]=v[ls]+v[rs];
16        lv[id]=max(lv[id],v[id]);
17        rv[id]=max(rv[id],v[id]);
18        mx[id]=max({mx[id],lv[id],rv[id],v[id]});
19    }
20    void build(int l,int r,int id)
21    {
22        mx[id]=lv[id]=rv[id]=-INF;
23        v[id]=0;
24        if(l==r)
25        {
26            mx[id]=lv[id]=rv[id]=v[id]=a[l];
27            return;
28        }
29        int mid=(l+r)>>1;
30        build(l,mid,ls);
31        build(mid+1,r,rs);
32        pushup(id);
33    }
34    void update(int l,int r,int id)
35    {
36        if(l>=ql&&r<=qr)
37        {
38            mx[id]=lv[id]=rv[id]=v[id]=qv;
39            return;
40        }
41        int mid=(l+r)>>1;
42        if(ql<=mid) update(l,mid,ls);
43        if(qr>mid) update(mid+1,r,rs);
44        pushup(id);
45    }
46    type res,lmx;
47    void query(int l,int r,int id)
48    {
49        if(l>=ql&&r<=qr)

```

```

50     res=max(res,mx[id]);
51     res=max(res,lmx+lv[id]);
52     lmx=max(lmx+v[id],rv[id]);
53     return;
54 }
55 int mid=(l+r)>>1;
56 if(ql<=mid) query(l,mid,ls);
57 if(qr>mid) query(mid+1,r,rs);
58 }
59 void build(int _n){n=_n;build(1,n,1);}
60 void upd(int l,int r,type v)
61 {
62     ql=l;
63     qr=r;
64     qv=v;
65     update(1,n,1);
66 }
67 type ask(int l,int r){//init res
68 {
69     ql=l;
70     qr=r;
71     res=-INF;
72     lmx=0;
73     query(1,n,1);
74     return res;
75 }
76 #undef type
77 #undef ls
78 #undef rs
79 }tr;

```

2.6.5 矩形面积并

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const int INF=0x3f3f3f3f;
5 const int MAX=4e5+10;
6 struct Discretization
7 {
8     #define type ll
9     #define pb push_back
10    #define all(x) x.begin(),x.end()
11    vector<type> a;
12    void init(){a.clear();}
13    void add(type x){a.pb(x);}
14    void work(){sort(all(a));a.resize(unique(all(a))-a.begin());}
15    int get_pos(type x){return lower_bound(all(a),x)-a.begin()+1;}
16    type get_val(int pos){return a[pos-1];}
17    int size(){return a.size();}
18    #undef type
19    #undef pb

```

```

20    #undef all
21 }dz;
22 struct Segment_Tree
23 {
24     #define type ll
25     #define ls (id<<1)
26     #define rs (id<<1|1)
27     int n,ql,qr;
28     type a[MAX],v[MAX<<2],mn[MAX<<2],tag[MAX<<2],
29         cntmn[MAX<<2],qv;
30     void pushup(int id)
31     {
32         mn[id]=min(mn[ls],mn[rs]);
33         if(mn[ls]==mn[rs]) cntmn[id]=cntmn[ls]+cntmn[rs];
34         else cntmn[id]=(mn[ls]<mn[rs]?cntmn[ls]:cntmn[rs]);
35         v[id]=v[ls]+v[rs];
36     }
37     void maintain(int pre,int now,int id)
38     {
39         if(pre==0&&now>0) v[id]+=cntmn[id];
40         if(pre>0&&now==0) v[id]-=cntmn[id];
41     }
42     void pushdown(int l,int r,int id)
43     {
44         if(!tag[id]) return;
45         int mid=(l+r)>>1;
46         maintain(mn[ls],mn[ls]+tag[id],ls);
47         maintain(mn[rs],mn[rs]+tag[id],rs);
48         mn[ls]+=tag[id];
49         mn[rs]+=tag[id];
50         tag[ls]+=tag[id];
51         tag[rs]+=tag[id];
52         tag[id]=0;
53     }
54     void build(int l,int r,int id)
55     {
56         tag[id]=0;
57         if(l==r)
58         {
59             mn[id]=0;
60             cntmn[id]=dz.get_val(l)-dz.get_val(l-1);
61             v[id]=0;
62             return;
63         }
64         int mid=(l+r)>>1;
65         build(l,mid,ls);
66         build(mid+1,r,rs);
67         pushup(id);
68     }
69     void update(int l,int r,int id)
70     {
71         if(l>=ql&&r<=qr)

```

```

71     {
72         maintain(mn[id],mn[id]+qv,id);
73         mn[id]+=qv;
74         tag[id]+=qv;
75         return;
76     }
77     pushdown(l,r,id);
78     int mid=(l+r)>>1;
79     if(ql<=mid) update(l,mid,ls);
80     if(qr>mid) update(mid+1,r,rs);
81     pushup(id);
82 }
83 type res;
84 void query(int l,int r,int id)
85 {
86     if(l>=ql&&r<=qr)
87     {
88         res+=v[id];
89         return;
90     }
91     pushdown(l,r,id);
92     int mid=(l+r)>>1;
93     if(ql<=mid) query(l,mid,ls);
94     if(qr>mid) query(mid+1,r,rs);
95 }
96 void build(int _n){n=_n;build(1,n,1);}
97 void upd(int l,int r,type v)
98 {
99     if(l>r) return;
100     ql=l;
101     qr=r;
102     qv=v;
103     update(1,n,1);
104 }
105 type ask(int l,int r)//init res
106 {
107     ql=l;
108     qr=r;
109     res=0;
110     query(1,n,1);
111     return res;
112 }
113 #undef type
114 #undef ls
115 #undef rs
116 }tr;
117 struct node{int pos,l,r,v;};
118 ll work(vector<node> &res)
119 {
120     int i,j,pos;
121     ll ans;
122     sort(res.begin(),res.end(),[&](node x,node y){
123         if(x.pos==y.pos) return x.v>y.v;
124         return x.pos<y.pos;

```

```

125     });
126     ans=0;
127     pos=1;
128     tr.build(dz.size());
129     for(i=0;i<res.size();i++)
130     {
131         if(i) ans+=tr.ask(1,dz.size()*(res[i].pos-
132             pos);
133         tr.upd(dz.get_pos(res[i].l),dz.get_pos(res[i]
134             ].r),res[i].v);
135         pos=res[i].pos;
136     }
137     ans+=tr.ask(2,dz.size()*(res[i].pos-pos);
138     return ans;
139 }
140 int main()
141 {
142     int n,i,a,b,c,d;
143     scanf("%d",&n);
144     vector<node> x;
145     dz.init();
146     for(i=1;i<=n;i++)
147     {
148         scanf("%d%d%d%d",&a,&b,&c,&d);
149         dz.add(b);
150         dz.add(b-1);
151         dz.add(d-2);
152         dz.add(d-1);
153         x.push_back({a,b,d-1,1});
154         x.push_back({c,b,d-1,-1});
155     }
156     dz.work();
157     printf("%lld\n",work(x));
158     return 0;
159 }

```

2.7 平衡树

2.7.1 Treap

```

1 struct Treap
2 {
3     #define type int
4     static const type inf=INF;
5     struct node
6     {
7         int ch[2],fix,sz,cnt;
8         type v;
9         node(){
10             node(type x,int _sz)
11             {
12                 v=x;
13                 fix=rand();
14                 sz=cnt=_sz;

```

```

15         ch[0]=ch[1]=0;
16     }
17 }t[MAX];
18 int tot,root[MAX],rt;
19 void init(int n=1)
20 {
21     for(int i=0;i<=n;i++) root[i]=0;
22     rt=1;
23     srand(time(0));
24     tot=0;
25     t[0].sz=t[0].cnt=0;
26     memset(t[0].ch,0,sizeof t[0].ch);
27 }
28 void pushup(int id)
29 {
30     t[id].sz=t[t[id].ch[0]].sz+t[t[id].ch[1]].sz+
31         t[id].cnt;
32 }
33 void rotate(int &id,int k)
34 {
35     int y=t[id].ch[k^1];
36     t[id].ch[k^1]=t[y].ch[k];
37     t[y].ch[k]=id;
38     pushup(id);
39     pushup(y);
40     id=y;
41 }
42 void _insert(int &id,type v,int cnt)
43 {
44     if(!id)
45     {
46         id=++tot;
47         t[id]=node(v,cnt);
48         return;
49     }
50     if(t[id].v==v) t[id].cnt+=cnt;
51     else
52     {
53         int tmp=(v>t[id].v);
54         _insert(t[id].ch[tmp],v,cnt);
55         if(t[t[id].ch[tmp]].fix>t[id].fix) rotate
56             (id,tmp^1);
57     }
58     pushup(id);
59 }
60 void _erase(int &id,type v,int cnt)
61 {
62     if(!id) return;
63     if(t[id].v==v)
64     {
65         cnt=min(t[id].cnt,cnt);
66         if(t[id].cnt>cnt)

```

```

67         pushup(id);
68         return;
69     }
70     if(!(t[id].ch[0]&& t[id].ch[1]))
71     {
72         id=t[id].ch[0]+t[id].ch[1];
73         return;
74     }
75     else
76     {
77         int tmp=(t[t[id].ch[0]].fix>t[t[id].ch
78             [1]].fix);
79         rotate(id,tmp);
80         _erase(t[id].ch[tmp],v,cnt);
81         pushup(id);
82     }
83     else
84     {
85         _erase(t[id].ch[v>t[id].v],v,cnt);
86         pushup(id);
87     }
88 }
89 int _find(type key,int f)
90 {
91     int id=root[rt],res=0;
92     while(id)
93     {
94         if(t[id].v<key)
95         {
96             res+=t[t[id].ch[0]].sz+t[id].cnt;
97             if(f&&key==t[id].v) res-=t[id].cnt;
98             id=t[id].ch[1];
99         }
100         else id=t[id].ch[0];
101     }
102     return res;
103 }
104 type find_by_order(int k)//k small
105 {
106     int id=root[rt];
107     if(id==0) return 0;
108     while(id)
109     {
110         if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
111         else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
112             return t[id].v;
113         else
114         {
115             k-=t[t[id].ch[0]].sz+t[id].cnt;
116             id=t[id].ch[1];
117         }
118     }

```

```

119 int count(type key)
120 {
121     int id=root[rt];
122     while(id)
123     {
124         if(t[id].v<key)
125         {
126             if(key==t[id].v) return t[id].cnt;
127             id=t[id].ch[1];
128         }
129         else id=t[id].ch[0];
130     }
131     return 0;
132 }
133 type find_pre(type key)
134 {
135     type res=-inf;
136     int id=root[rt];
137     while(id)
138     {
139         if(t[id].v<key)
140         {
141             res=t[id].v;
142             id=t[id].ch[1];
143         }
144         else id=t[id].ch[0];
145     }
146     return res;
147 }
148 type find_nex(type key)
149 {
150     type res=inf;
151     int id=root[rt];
152     while(id)
153     {
154         if(t[id].v>key)
155         {
156             res=t[id].v;
157             id=t[id].ch[0];
158         }
159         else id=t[id].ch[1];
160     }
161     return res;
162 }
163 Treap &operator[(const int _rt)]{this->rt=_rt;
164     return *this;}
165 void insert(type v,int sz=1){_insert(root[rt],v,
166     sz);}
167 void erase(type v,int sz=1){_erase(root[rt],v,sz
168     );}
169 int upper_bound_count(type key){return _find(key
170     ,0);}//the count <=key
171 int lower_bound_count(type key){return _find(key
172     ,1);}//the count <key

```

```

168 int order_of_key(type key){return
169     lower_bound_count(key)+1;}
170 int size(){return t[root[rt]].sz;}
171 #undef type
172 }tr;
173 /*
174 1 treap
175 tr.init();
176 tr.insert(x);
177 tr.erase(x);
178 tr.count(x);
179 tr.order_of_key(x); // rank
180 tr.find_by_order(k); // kth
181 tr.find_pre(x);
182 tr.find_nex(x);
183 tr.upper_bound_count(x); //the count <=key
184 tr.lower_bound_count(x); //the count <key
185 n treap
186 tr.init(n);
187 tr[i].insert(x);
188 */

```

2.7.2 Splay 维护序列

```

1 struct Splay
2 {
3     #define type int
4     const type inf=INF;
5     const type zero=0;
6     struct node
7     {
8         int ch[2],fa,sz,cnt,rev,tag;
9         type v;
10    }t[MAX];
11    int tot,root;
12    type a[MAX];
13    queue<int> pool;
14    void init_null_node()
15    {
16        memset(t[0].ch,0,sizeof t[0].ch);
17        t[0].sz=t[0].cnt=t[0].fa=0;
18        t[0].v=zero;
19        t[0].mnid=0;
20    }
21    void init()
22    {
23        root=tot=0;
24        while(!pool.empty()) pool.pop();
25        init_null_node();
26        a[0]=a[1]=zero;
27        root=build(0,1,0);
28    }
29    int newnode(type v,int fa)

```

```

30 {
31     int id;
32     if(pool.size()>0)
33     {
34         id=pool.front();
35         pool.pop();
36     }
37     else id=++tot;
38     memset(t[id].ch,0,sizeof t[id].ch);
39     t[id].fa=fa;
40     t[id].sz=t[id].cnt=1;
41     t[id].tag=t[id].rev=0;
42     t[id].v=v;
43     return id;
44 }
45 void pushup(int id)
46 {
47     int ls=t[id].ch[0];
48     int rs=t[id].ch[1];
49     t[id].sz=t[ls].sz+t[rs].sz+t[id].cnt;
50
51
52 }
53 void pushdown(int id)
54 {
55     int ls=t[id].ch[0];
56     int rs=t[id].ch[1];
57     if(t[id].tag)
58     {
59         if(ls)
60         {
61             t[ls].tag=1;
62         }
63         if(rs)
64         {
65             t[rs].tag=1;
66         }
67         t[id].tag=0;
68     }
69     if(t[id].rev)
70     {
71         t[ls].rev^=1;
72         t[rs].rev^=1;
73         swap(t[ls].ch[0],t[ls].ch[1]);
74         swap(t[rs].ch[0],t[rs].ch[1]);
75         t[id].rev=0;
76     }
77 }
78
79 }
80 void insert(int pos,vector<int> nums)
81 {
82     int x,y,z,i;
83     for(i=0;i<nums.size();i++) a[i+1]=nums[i];

```

```

84     z=build(1,nums.size(),0);
85     x=find(pos);
86     y=find(pos+1);
87     splay(x,0);
88     splay(y,x);
89     t[y].ch[0]=z;
90     t[z].fa=y;
91     pushup(y);
92     pushup(x);
93 }
94 void rotate(int x)
95 {
96     int y,z,k;
97     y=t[x].fa;
98     z=t[y].fa;
99     k=(x==t[y].ch[1]);
100     t[y].ch[k]=t[x].ch[k^1];
101     if(t[x].ch[k^1]) t[t[x].ch[k^1]].fa=y;
102     t[x].ch[k^1]=y;
103     t[y].fa=x;
104     t[x].fa=z;
105     if(z) t[z].ch[y==t[z].ch[1]]=x;
106     pushup(y);
107     pushup(x);
108 }
109 void splay(int x,int goal)
110 {
111     int y,z;
112     while(t[x].fa!=goal)
113     {
114         y=t[x].fa;
115         z=t[y].fa;
116         if(z!=goal)
117         {
118             if((t[z].ch[0]==y)^(t[y].ch[0]==x))
119                 rotate(x);
120             else rotate(y);
121         }
122         rotate(x);
123     }
124     if(goal==0) root=x;
125 }
126 int kth(int k)//k small
127 {
128     int id=root;
129     while(id)
130     {
131         pushdown(id);
132         if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
133         else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
134             break;
135         else
136         {
137             k-=t[t[id].ch[0]].sz+t[id].cnt;

```



```

136         id=t[id].ch[1];
137     }
138 }
139 splay(id,0);
140 return id;
141 }
142 int find(int x){return kth(x+1);}
143 int build(int l,int r,int fa)
144 {
145     if(l>r) return 0;
146     int id,mid;
147     mid=(l+r)>>1;
148     id=newnode(a[mid],fa);
149     t[id].ch[0]=build(l,mid-1,id);
150     t[id].ch[1]=build(mid+1,r,id);
151     pushup(id);
152     return id;
153 }
154 void _del(int id)
155 {
156     if(!id) return;
157     pool.push(id);
158     _del(t[id].ch[0]);
159     _del(t[id].ch[1]);
160 }
161 void erase(int l,int r)
162 {
163     int x,fa;
164     x=split(l,r);
165     fa=t[x].fa;
166     t[fa].ch[0]=0;
167     _del(x);
168     pushup(fa);
169     pushup(t[fa].fa);
170 }
171 int split(int l,int r)
172 {
173     int x,y;
174     x=find(l-1);
175     y=find(r+1);
176     splay(x,0);
177     splay(y,x);
178     return t[y].ch[0];
179 }
180 void rev(int l,int r)
181 {
182     int x,fa;
183     x=split(l,r);
184     fa=t[x].fa;
185     t[x].rev^=1;
186     swap(t[x].ch[0],t[x].ch[1]);
187     pushup(fa);
188     pushup(t[fa].fa);
189 }

```

```

190 int ask(int l,int r)
191 {
192     int x=split(l,r);
193     return ;
194 }
195 int size(){return t[root].sz-2;}
196 #undef type
197 }tr; //tr.init();

```

2.7.3 pbds

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 tree<int,
5     null_type,
6     less<int>,
7     rb_tree_tag,
8     tree_order_statistics_node_update> tr;

```

2.8 字典树

2.8.1 trie

```

1 struct Trie
2 {
3     #define type int
4     static const int K=26;
5     int nex[MAX][K],cnt[MAX];
6     int root,tot;
7     int getid(char c){return c-'a';}
8     int newnode()
9     {
10         memset(nex[tot],0,sizeof nex[tot]);
11         cnt[tot]=0;
12         return tot++;
13     }
14     void init()
15     {
16         tot=0;
17         root=newnode();
18     }
19     void insert(char *s,int n) // s[0..n-1]
20     {
21         int now,i,t;
22         now=root;
23         for(i=0;i<n;i++)
24         {
25             t=getid(s[i]);
26             if(!nex[now][t]) nex[now][t]=newnode();
27             now=nex[now][t];
28         }
29         cnt[now]++;

```

```

30     }
31     #undef type
32 }tr;
33 /*
34 tr.init();
35 tr.insert(s,len); s[0..len-1]
36 */

```

2.8.2 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;

```

2.9 可持久化

2.9.1 可持久化线段树

```

1 struct President_Tree
2 {
3     #define type int
4     int root[MAX],ls[MAX<<5],rs[MAX<<5],tot,q1,q2,n;
5     type v[MAX<<5],qv,res;
6     void init(int _n)
7     {
8         n=_n;
9         ls[0]=rs[0]=v[0]=tot=0;
10    }
11    int copy_node(int x)
12    {
13        tot++;
14        ls[tot]=ls[x];
15        rs[tot]=rs[x];
16        v[tot]=v[x];
17        return tot;
18    }

```

```

19 void update(int l,int r,int &id,int pre)
20 {
21     if(!id) id=copy_node(pre);
22     v[id]+=qv;
23     if(l==r) return;
24     int mid=(l+r)>>1;
25     if(ql<=mid) update(l,mid,ls[id]=0,ls[pre]);
26     else update(mid+1,r,rs[id]=0,rs[pre]);
27 }
28 void query(int l,int r,int id)
29 {
30     if(!id) return;
31     if(l>=ql&&r<=qr)
32     {
33         res+=v[id];
34         return;
35     }
36     int mid=(l+r)>>1;
37     if(ql<=mid) query(l,mid,ls[id]);
38     if(qr>mid) query(mid+1,r,rs[id]);
39 }
40 int kth_small(int l,int r,int id,int pre,int k)
41 {
42     if(l==r) return l;
43     int mid=(l+r)>>1;
44     int tmp=v[ls[id]]-v[ls[pre]];
45     if(tmp>=k) return kth_small(l,mid,ls[id],ls[pre],k);
46     else return kth_small(mid+1,r,rs[id],rs[pre],k-tmp);
47 }
48 void copy_ver(int now_ver,int pre_ver)
49 {
50     root[now_ver]=root[pre_ver];
51 }
52 void create_ver(int now_ver,int pre_ver,int pos,
53                 type v)
54 {
55     root[now_ver]=0;
56     update_ver(now_ver,pre_ver,pos,v);
57 }
58 void update_ver(int now_ver,int pre_ver,int pos,
59                 type v)
60 {
61     ql=qr=pos;
62     qv=v;
63     update(1,n,root[now_ver],root[pre_ver]);
64 }
65 type ask_ver(int now_ver,int l,int r)
66 {
67     res=0;
68     if(l>r) return res;
69     ql=l;
70     qr=r;

```

```

69     query(1,n,root[now_ver]);
70     return res;
71 }
72 int ask_kth_small(int l,int r,int k)
73 {
74     return kth_small(1,n,root[r],root[l-1],k);
75 }
76 }tr;
77 /*
78 tr.init(n);
79 tr.create_ver(now_ver,pre_ver,pos,v);
80 tr.update_ver(now_ver,pre_ver,pos,v);
81 tr.copy_ver(now_ver,pre_ver);
82 tr.ask_kth_small(l,r,k);
83 */

```

2.10 树套树

2.10.1 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     tag[id]=0;
37 }
38 void update_in(int l,int r,int &id)
39 {
40     if(!id) id=newnode();
41     if(l>=ql_in&&r<=qr_in)
42     {
43         v[id]+=qv; //must update not =
44         tag[id]+=qv;
45         return;
46     }
47     pushdown(id);
48     int mid=(l+r)>>1;
49     if(ql_in<=mid) update_in(l,mid,ls[id]);
50     if(qr_in>mid) update_in(mid+1,r,rs[id]);
51     pushup(id);
52 }
53 type res_in;
54 void query_in(int l,int r,int &id)
55 {
56     if(!id) return;
57     if(l>=ql_in&&r<=qr_in)
58     {
59         res_in+=v[id];
60         return;
61     }
62     pushdown(id);
63     int mid=(l+r)>>1;
64     type res=0;
65     if(ql_in<=mid) query_in(l,mid,ls[id]);
66     if(qr_in>mid) query_in(mid+1,r,rs[id]);
67 }
68 /*
69  ****
70  */
71 #define ls (id<<1)
72 #define rs (id<<1|1)
73 void update_out(int l,int r,int id)
74 {
75     update_in(1,m,root[id]);
76     if(l>=ql_out&&r<=qr_out) return;
77     int mid=(l+r)>>1;
78     if(ql_out<=mid) update_out(l,mid,ls);
79     if(qr_out>mid) update_out(mid+1,r,rs);
80 }
81 type res_out;
82 void query_out(int l,int r,int id)
83 {
84     if(l>=ql_out&&r<=qr_out)
85     {

```

```

86         query_in(1,m,root[id]);
87         res_out+=res_in;
88         return;
89     }
90     int mid=(l+r)>>1;
91     if(ql_out<=mid) query_out(l,mid,ls);
92     if(qr_out>mid) query_out(mid+1,r,rs);
93 }
94 #undef ls
95 #undef rs
96 void upd(int x1,int y1,int x2,int y2,type val)
97 {
98     ql_out=x1;
99     qr_out=x2;
100    ql_in=y1;
101    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;

```

2.10.2 线段树套 treap

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int INF=0x3f3f3f3f;
5  const int MAX=5e4+10;
6  struct Treap
7  {
8      #define type int
9      #define inf INF
10     struct node
11     {
12         int ch[2],fix,sz,cnt;
13         type v;
14         node(){}
15         node(type x,int _sz)
16         {
17             v=x;
18             fix=rand();
19             sz=cnt=_sz;
20             ch[0]=ch[1]=0;

```

```

21     }
22     }t[MAX*40];
23     int tot,root[MAX<<2],rt;
24     void init(int n=1)
25     {
26         for(int i=0;i<=n;i++) root[i]=0;
27         rt=1;
28         srand(time(0));
29         tot=0;
30         t[0].sz=t[0].cnt=0;
31         memset(t[0].ch,0,sizeof t[0].ch);
32     }
33     void pushup(int id)
34     {
35         t[id].sz=t[t[id].ch[0]].sz+t[t[id].ch[1]].sz+
            t[id].cnt;
36     }
37     void rotate(int &id,int k)
38     {
39         int y=t[id].ch[k^1];
40         t[id].ch[k^1]=t[y].ch[k];
41         t[y].ch[k]=id;
42         pushup(id);
43         pushup(y);
44         id=y;
45     }
46     void _insert(int &id,type v,int cnt)
47     {
48         if(!id)
49         {
50             id=++tot;
51             t[id]=node(v,cnt);
52             return;
53         }
54         if(t[id].v==v) t[id].cnt+=cnt;
55         else
56         {
57             int tmp=(v>t[id].v);
58             _insert(t[id].ch[tmp],v,cnt);
59             if(t[t[id].ch[tmp]].fix>t[id].fix) rotate
                (id,tmp^1);
60         }
61         pushup(id);
62     }
63     void _erase(int &id,type v,int cnt)
64     {
65         if(!id) return;
66         if(t[id].v==v)
67         {
68             cnt=min(t[id].cnt,cnt);
69             if(t[id].cnt>cnt)
70             {
71                 t[id].cnt-=cnt;
72                 pushup(id);

```

```

73         return;
74     }
75     if(!(t[id].ch[0]&&t[id].ch[1]))
76     {
77         id=t[id].ch[0]+t[id].ch[1];
78         return;
79     }
80     else
81     {
82         int tmp=(t[t[id].ch[0]].fix>t[t[id].ch
            [1]].fix);
83         rotate(id,tmp);
84         _erase(t[id].ch[tmp],v,cnt);
85         pushup(id);
86     }
87     }
88     else
89     {
90         _erase(t[id].ch[v>t[id].v],v,cnt);
91         pushup(id);
92     }
93     }
94     int _find(type key,int f)
95     {
96         int id=root[rt],res=0;
97         while(id)
98         {
99             if(t[id].v<key)
100             {
101                 res+=t[t[id].ch[0]].sz+t[id].cnt;
102                 if(f&&key==t[id].v) res-=t[id].cnt;
103                 id=t[id].ch[1];
104             }
105             else id=t[id].ch[0];
106         }
107         return res;
108     }
109     type find_by_order(int k)//k small
110     {
111         int id=root[rt];
112         if(id==0) return 0;
113         while(id)
114         {
115             if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
116             else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
                return t[id].v;
117             else
118             {
119                 k-=t[t[id].ch[0]].sz+t[id].cnt;
120                 id=t[id].ch[1];
121             }
122         }
123     }
124     int count(type key)

```

```

125 {
126     int id=root[rt];
127     while(id)
128     {
129         if(t[id].v<key)
130         {
131             if(key==t[id].v) return t[id].cnt;
132             id=t[id].ch[1];
133         }
134         else id=t[id].ch[0];
135     }
136     return 0;
137 }
138 type find_pre(type key)
139 {
140     type res=-inf;
141     int id=root[rt];
142     while(id)
143     {
144         if(t[id].v<key)
145         {
146             res=t[id].v;
147             id=t[id].ch[1];
148         }
149         else id=t[id].ch[0];
150     }
151     return res;
152 }
153 type find_nex(type key)
154 {
155     type res=inf;
156     int id=root[rt];
157     while(id)
158     {
159         if(t[id].v>key)
160         {
161             res=t[id].v;
162             id=t[id].ch[0];
163         }
164         else id=t[id].ch[1];
165     }
166     return res;
167 }
168 Treap &operator()(const int _rt){this->rt=_rt;
169     return *this;}
170 void insert(type v,int sz=1){_insert(root[rt],v,
171     sz);}
172 void erase(type v,int sz=1){_erase(root[rt],v,sz
173     );}
174 int upper_bound_count(type key){return _find(key
175     ,0);}//the count <=key
176 int lower_bound_count(type key){return _find(key
177     ,1);}//the count <key

```

```

173 int order_of_key(type key){return
174     lower_bound_count(key)+1;}
175 int size(){return t[root[rt]].sz;}
176 }treap;
177 struct Segment_Tree
178 {
179     #define ls (id<<1)
180     #define rs (id<<1|1)
181     int n,ql,qr,qop;
182     type qv;
183     void update(int l,int r,int id)
184     {
185         if(qop==1) treap[id].insert(qv);
186         else treap[id].erase(qv);
187         if(l>=ql&&r<=qr) return;
188         int mid=(l+r)>>1;
189         if(ql<=mid) update(l,mid,ls);
190         if(qr>mid) update(mid+1,r,rs);
191     }
192     vector<int> treap_id;
193     void dfs(int l,int r,int id)
194     {
195         if(l>=ql&&r<=qr)
196         {
197             treap_id.push_back(id);
198             return;
199         }
200         int mid=(l+r)>>1;
201         if(ql<=mid) dfs(l,mid,ls);
202         if(qr>mid) dfs(mid+1,r,rs);
203     }
204     void get_treap_id(int l,int r)
205     {
206         ql=1;
207         qr=r;
208         treap_id.clear();
209         dfs(1,n,1);
210     }
211     void build(int _n){n=_n;treap.init(n<<2);}
212     void insert(int pos,type v)
213     {
214         ql=qr=pos;
215         qop=1;
216         qv=v;
217         update(1,n,1);
218     }
219     void erase(int pos,type v)
220     {
221         ql=qr=pos;
222         qop=2;
223         qv=v;
224         update(1,n,1);
225     }
226     int ask_rank(int l,int r,type v)

```

```

226 {
227     get_treap_id(l,r);
228     int res=1;
229     for(auto &rt:treap_id) res+=treap[rt].
        order_of_key(v)-1;
230     return res;
231 }
232 int ask_kth(int l,int r,int k)
233 {
234     get_treap_id(l,r);
235     l=0;
236     r=1e8;
237     while(l<r)
238     {
239         int mid=(l+r)>>1,now=1;
240         for(auto &rt:treap_id) now+=treap[rt].
            order_of_key(mid+1)-1;
241         if(now<=k) l=mid+1;
242         else r=mid;
243     }
244     return l;
245 }
246 type find_pre(int l,int r,type v)
247 {
248     get_treap_id(l,r);
249     type res=-inf;
250     for(auto &rt:treap_id) res=max(res,treap[rt].
        find_pre(v));
251     if(res==inf) return -2147483647;
252     return res;
253 }
254 type find_nex(int l,int r,type v)
255 {
256     get_treap_id(l,r);
257     type res=inf;
258     for(auto &rt:treap_id) res=min(res,treap[rt].
        find_nex(v));
259     if(res==inf) return 2147483647;
260     return res;
261 }
262 #undef type
263 #undef ls
264 #undef rs
265 }tr;
266 int a[MAX];
267 int main()
268 {
269     int n,m,i,op,l,r,k;
270     scanf("%d%d",&n,&m);
271     tr.build(n);
272     for(i=1;i<=n;i++)
273     {
274         scanf("%d",&a[i]);
275         tr.insert(i,a[i]);

```

```

276     }
277     while(m--)
278     {
279         scanf("%d",&op,&l);
280         if(op==3) scanf("%d",&k);
281         else scanf("%d",&r,&k);
282         if(op==1) printf("%d\n",tr.ask_rank(l,r,k));
283         else if(op==2) printf("%d\n",tr.ask_kth(l,r,k
            ));
284         else if(op==3)
285         {
286             tr.erase(l,a[l]);
287             a[l]=k;
288             tr.insert(l,a[l]);
289         }
290         else if(op==4) printf("%d\n",tr.find_pre(l,r,
            k));
291         else if(op==5) printf("%d\n",tr.find_nex(l,r,
            k));
292     }
293     return 0;
294 }

```

2.10.3 树状数组套 treap

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const int INF=0x3f3f3f3f;
5 const int MAX=1e5+10;
6 struct Treap
7 {
8     #define type int
9     #define inf INF
10    struct node
11    {
12        int ch[2],fix,sz,cnt;
13        type v;
14        node(){}
15        node(type x,int _sz)
16        {
17            v=x;
18            fix=rand();
19            sz=cnt=_sz;
20            ch[0]=ch[1]=0;
21        }
22    }t[MAX*20];
23    int tot,root[MAX],rt;
24    void init(int n=1)
25    {
26        for(int i=0;i<=n;i++) root[i]=0;
27        rt=1;
28        srand(time(0));
29        tot=0;

```

```

30     t[0].sz=t[0].cnt=0;
31     memset(t[0].ch,0,sizeof t[0].ch);
32 }
33 void pushup(int id)
34 {
35     t[id].sz=t[t[id].ch[0]].sz+t[t[id].ch[1]].sz+
        t[id].cnt;
36 }
37 void rotate(int &id,int k)
38 {
39     int y=t[id].ch[k^1];
40     t[id].ch[k^1]=t[y].ch[k];
41     t[y].ch[k]=id;
42     pushup(id);
43     pushup(y);
44     id=y;
45 }
46 void _insert(int &id,type v,int cnt)
47 {
48     if(!id)
49     {
50         id=++tot;
51         t[id]=node(v,cnt);
52         return;
53     }
54     if(t[id].v==v) t[id].cnt+=cnt;
55     else
56     {
57         int tmp=(v>t[id].v);
58         _insert(t[id].ch[tmp],v,cnt);
59         if(t[t[id].ch[tmp]].fix>t[id].fix) rotate
            (id,tmp^1);
60     }
61     pushup(id);
62 }
63 void _erase(int &id,type v,int cnt)
64 {
65     if(!id) return;
66     if(t[id].v==v)
67     {
68         cnt=min(t[id].cnt,cnt);
69         if(t[id].cnt>cnt)
70         {
71             t[id].cnt-=cnt;
72             pushup(id);
73             return;
74         }
75         if(!(t[id].ch[0]&& t[id].ch[1]))
76         {
77             id=t[id].ch[0]+t[id].ch[1];
78             return;
79         }
80         else
81         {

```

```

82         int tmp=(t[t[id].ch[0]].fix>t[t[id].ch
            [1]].fix);
83         rotate(id,tmp);
84         _erase(t[id].ch[tmp],v,cnt);
85         pushup(id);
86     }
87 }
88 else
89 {
90     _erase(t[id].ch[v>t[id].v],v,cnt);
91     pushup(id);
92 }
93 }
94 int _find(type key,int f)
95 {
96     int id=root[rt],res=0;
97     while(id)
98     {
99         if(t[id].v<key)
100         {
101             res+=t[t[id].ch[0]].sz+t[id].cnt;
102             if(f&&key==t[id].v) res-=t[id].cnt;
103             id=t[id].ch[1];
104         }
105         else id=t[id].ch[0];
106     }
107     return res;
108 }
109 type find_by_order(int k)//k small
110 {
111     int id=root[rt];
112     if(id==0) return 0;
113     while(id)
114     {
115         if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
116         else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
            return t[id].v;
117         else
118         {
119             k-=t[t[id].ch[0]].sz+t[id].cnt;
120             id=t[id].ch[1];
121         }
122     }
123 }
124 int count(type key)
125 {
126     int id=root[rt];
127     while(id)
128     {
129         if(t[id].v<key)
130         {
131             if(key==t[id].v) return t[id].cnt;
132             id=t[id].ch[1];
133         }

```



```

134         else id=t[id].ch[0];
135     }
136     return 0;
137 }
138 type find_pre(type key)
139 {
140     type res=-inf;
141     int id=root[rt];
142     while(id)
143     {
144         if(t[id].v<key)
145         {
146             res=t[id].v;
147             id=t[id].ch[1];
148         }
149         else id=t[id].ch[0];
150     }
151     return res;
152 }
153 type find_nex(type key)
154 {
155     type res=inf;
156     int id=root[rt];
157     while(id)
158     {
159         if(t[id].v>key)
160         {
161             res=t[id].v;
162             id=t[id].ch[0];
163         }
164         else id=t[id].ch[1];
165     }
166     return res;
167 }
168 Treap &operator[](const int _rt){this->rt=_rt;
169     return *this;}
170 void insert(type v,int sz=1){_insert(root[rt],v,
171     sz);}
172 void erase(type v,int sz=1){_erase(root[rt],v,sz
173     );}
174 int upper_bound_count(type key){return _find(key
175     ,0);};//the count <=key
176 int lower_bound_count(type key){return _find(key
177     ,1);};//the count <key
178 int order_of_key(type key){return
179     lower_bound_count(key)+1;}
180 int size(){return t[root[rt]].sz;}
181 }treap;
182 struct Fenwick_Tree
183 {
184     int n;
185     void init(int _n)
186     {
187         n=_n;

```

```

182     treap.init(n);
183 }
184 int lowbit(int x){return x&(-x);}
185 type get(int x,type v)
186 {
187     type res=0;
188     while(x)
189     {
190         res+=treap[x].lower_bound_count(v);
191         x-=lowbit(x);
192     }
193     return res;
194 }
195 void insert(int x,type v)
196 {
197     while(x<=n)
198     {
199         treap[x].insert(v);
200         x+=lowbit(x);
201     }
202 }
203 void erase(int x,type v)
204 {
205     while(x<=n)
206     {
207         treap[x].erase(v);
208         x+=lowbit(x);
209     }
210 }
211 int ask_kth(int ql,int qr,int k)
212 {
213     int l,r,mid;
214     l=0;
215     r=1e9;
216     while(l<r)
217     {
218         mid=(l+r)>>1;
219         if(get(qr,mid+1)-get(ql-1,mid+1)+1<=k) l=
220             mid+1;
221         else r=mid;
222     }
223     return l;
224 }
225 #undef type
226 }tr;
227 int a[MAX];
228 int main()
229 {
230     int n,m,i,l,r,k;
231     char op[3];
232     scanf("%d%d",&n,&m);
233     tr.init(n);
234     for(i=1;i<=n;i++)

```

```

235     scanf("%d",&a[i]);
236     tr.insert(i,a[i]);
237 }
238 while(m--)
239 {
240     scanf("%s%d",op,&l);
241     if(op[0]=='Q')
242     {
243         scanf("%d%d",&r,&k);
244         printf("%d\n",tr.ask_kth(l,r,k));
245     }
246     else
247     {
248         scanf("%d",&k);
249         tr.erase(l,a[l]);
250         a[l]=k;
251         tr.insert(l,a[l]);
252     }
253 }
254 return 0;
255 }

```

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

```

```

80     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;

```

2.12 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;

```

```

31            x[1]=x1;
32            l=r=fa=0;
33            exist=1;
34            v=val;
35            initval();
36        }
37        kdtnode(int a,int b,type d=NONE){newnode(a,b,
38            d);}
39    };
40    struct KDT
41    {
42        #define ls t[id].l
43        #define rs t[id].r
44        kdtnode t[MAX];
45        int tot,idx,root;
46        inline void pushup(int id)
47        {
48            t[id].initval();
49            t[id].sz+=t[ls].sz+t[rs].sz;
50            t[id].tag=min({t[ls].tag,t[rs].tag,t[id].
51                tag});
52            for(int i=0;i<dim;i++)
53            {
54                if(ls)
55                {
56                    t[id].mx[i]=max(t[id].mx[i],t[ls].
57                        mx[i]);
58                    t[id].mn[i]=min(t[id].mn[i],t[ls].
59                        mn[i]);
60                }
61                if(rs)
62                {
63                    t[id].mx[i]=max(t[id].mx[i],t[rs].
64                        mx[i]);
65                    t[id].mn[i]=min(t[id].mn[i],t[rs].
66                        mn[i]);
67                }
68            }
69        }
70        bool isbad(int id){return t[id].sz*alpha+3<
71            max(t[ls].sz,t[rs].sz);}
72        int st[MAX],top;
73        void build(int &id,int l,int r,int fa,int dep
74            =0)
75        {
76            id=0;if(l>r) return;
77            int m=(l+r)>>1; idx=dep;
78            nth_element(st+l,st+m,st+r+1,&)(int x,
79                int y){return t[x].x[idx]<t[y].x[idx
80                    ];});
81            id=st[m];
82            build(ls,l,m-1,id,(dep+1)%dim);
83            build(rs,m+1,r,id,(dep+1)%dim);
84            pushup(id);

```

```

75         t[id].dep=dep;
76         t[id].fa=fa;
77     }
78     inline void init(int n=0)
79     {
80         root=0;
81         t[0].null();
82         for(int i=1;i<=n;i++) st[i]=i;
83         if(n) build(root,1,n,0);
84         tot=n;
85     }
86     void travel(int id)
87     {
88         if(!id) return;
89         if(t[id].exist) st[++top]=id;
90         travel(ls);
91         travel(rs);
92     }
93     void rebuild(int &id,int dep)
94     {
95         top=0;travel(id);
96         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         {

```

```

127         x=t[id].fa;
128         rebuild(root==id?root:(t[x].l==id?t[x]
129             ].l:t[x].r),t[id].dep);
130     }
131     kdtnode q;
132     ll dist(ll x,ll y){return x*x+y*y;}
133     ll getdist(int id)//点离区域qt[id]最短距离
134     {
135         if(!id) return LLINF;
136         ll res=0;
137         if(q.x[0]<t[id].mn[0]) res+=dist(q.x[0]-t
138             [id].mn[0],0);
139         if(q.x[1]<t[id].mn[1]) res+=dist(q.x[1]-t
140             [id].mn[1],0);
141         if(q.x[0]>t[id].mx[0]) res+=dist(q.x[0]-t
142             [id].mx[0],0);
143         if(q.x[1]>t[id].mx[1]) res+=dist(q.x[1]-t
144             [id].mx[1],0);
145         return res;
146     }
147     kdtnode a,b;
148     inline int check(kdtnode &x)//在矩形x(a,b)内
149     {
150         int ok=1;
151         for(int i=0;i<dim;i++)
152         {
153             ok&=(x.x[i]>=a.x[i]);
154             ok&=(x.x[i]<=b.x[i]);
155         }
156         return ok;
157     }
158     inline int allin(kdtnode &x)//的子树全在矩
159         形x(a,b)内
160     {
161         int ok=1;
162         for(int i=0;i<dim;i++)
163         {
164             ok&=(x.mn[i]>=a.x[i]);
165             ok&=(x.mx[i]<=b.x[i]);
166         }
167         return ok;
168     }
169     inline int allout(kdtnode &x)//的子树全不在矩
170         形x(a,b)内
171     {
172         int ok=0;
173         for(int i=0;i<dim;i++)
174         {
175             ok|=(x.mx[i]<a.x[i]);
176             ok|=(x.mn[i]>b.x[i]);
177         }
178         return ok;
179     }
180     type res;

```

```

175 void query(int id)
176 {
177     if(!id) return;
178     if(allout(t[id]) || t[id].sz==0) return;
179     if(allin(t[id]))
180     {
181         res=min(res,t[id].tag);
182         return;
183     }
184     if(check(t[id])&&t[id].exist) res=min(res
        ,t[id].v);
185     int l,r;
186     l=ls;
187     r=rs;
188     if(t[l].tag>t[r].tag) swap(l,r);
189     if(t[l].tag<res) query(l);
190     if(t[r].tag<res) query(r);
191 }
192 inline type query(kdtnode _a,kdtnode _b)
193 {
194     a=_a;b=_b;
195     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 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>,
    pairing_heap_tag> q[MAX];
4 //q[i].join(q[j]) -> q[j]c^μq[i],q[j]

```

2.14 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 struct k_Huffman
2 {
3     #define type ll
4     type work(int n,int k,type *a)// a[1..n], sorted
5     {
6         int i;
7         type res,s;

```

```

8 queue<type> q,d;
9 s=((n-1)%(k-1)?k-1-(n-1)%(k-1):0);//
    21902190
10 while(s--) q.push(0);
11 for(i=1;i<=n;i++) q.push(a[i]);
12 res=0;
13 while(q.size()+d.size()>1)
14 {
15     s=0;
16     for(i=0;i<k;i++)
17     {
18         if(q.size()&&d.size())
19         {
20             if(q.front()<=d.front())
21             {
22                 s+=q.front();
23                 q.pop();
24             }
25             else
26             {
27                 s+=d.front();
28                 d.pop();
29             }
30         }
31         else if(q.size())
32         {
33             s+=q.front();
34             q.pop();
35         }
36         else if(d.size())
37         {
38             s+=d.front();
39             d.pop();
40         }
41     }
42     res+=s;
43     d.push(s);
44 }
45 return res;
46 }
47 #undef type
48 }hfm;

```

2.15 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 struct 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]=r[i]=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 }ct;

```

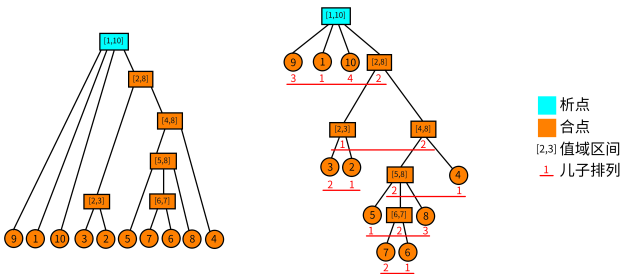
2.16 析合树

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,qv;
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         mdf(ls,tag[id]);
53         mdf(rs,tag[id]);
54         tag[id]=0;
55     }
56     void build(int l,int r,int id)
57     {
58         tag[id]=mn[id]=0;
59         if(l==r) return;
60         int mid=(l+r)>>1;
61         build(l,mid,ls);
62         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();
101         type[i]=0;
102     }
103     top=top1=top2=0;
104     tot=0;
105     for(i=1;i<=n;i++)
106     {

```

```

107     }
108     while(top&&a[i]>=a[st2[top2]])
109     {
110         tr.upd(st2[top2-1]+1,st2[top2],-a[st2[
111             top2]]);
112         top2--;
113     }
114     tr.upd(st1[top1]+1,i,-a[i]);
115     st1[++top1]=i;
116     tr.upd(st2[top2]+1,i,a[i]);
117     st2[++top2]=i;
118     id[i]=++tot;
119     l[tot]=r[tot]=i;
120     tmp=tr.ask(1,n);
121     now=tot;
122     while(top&&l[st[top]]>=tmp)
123     {
124         if(type[st[top]]&&check(m[st[top]],i))
125         {
126             r[st[top]]=i;
127             add_edge(st[top],now);
128             now=st[top--];
129         }
130         else if(check(l[st[top]],i))
131         {
132             type[++tot]=1;
133             l[tot]=l[st[top]];
134             r[tot]=i;
135             m[tot]=l[now];
136             add_edge(tot,st[top--]);
137             add_edge(tot,now);
138             now=tot;
139         }
140         else
141         {
142             add_edge(++tot,now);
143             do
144             {
145                 add_edge(tot,st[top--]);
146             }while(top&&!check(l[st[top]],i));
147             l[tot]=l[st[top]];
148             r[tot]=i;
149             add_edge(tot,st[top--]);
150             now=tot;
151         }
152     }
153     st[++top]=now;
154     tr.upd(1,i,-1);
155 }
156 return st[1];
157 }
158 void work(int n,int *a)// a[1..n]
159 {
160     int rt=build(n,a);

```

```

160     }
161 }pt; // MAX must *2
162

```

```

47 }mo; // mo.init(n,q)

```

2.17 莫队算法

```

1 struct MO_Algorithm
2 {
3     #define type int
4     struct query_info{int id,l,r;type v;};
5     vector<query_info> qst;
6     int n,q;
7     type ans[MAX],res;
8     void init(int _n,int _q)
9     {
10         qst.clear();
11         n=_n;
12         q=_q;
13     }
14     void add_query(int id,int l,int r,int v=0) {qst.
15         pb(query_info{id,l,r,v});}
16     int a[MAX];
17     void add(int x,int k)
18     {
19     }
20     void del(int x,int k)
21     {
22     }
23 }
24 void work()
25 {
26     int i,l,r,sq;
27     sq=sqrt(n);
28     sort(all(qst),[&](query_info a,query_info b){
29         if(a.l/sq!=b.l/sq) return a.l/sq<b.l/sq;
30         return a.r<b.r;
31     });
32
33     for(i=1;i<=q;i++) ans[i]=0;
34     l=1;
35     r=0;
36     res=0;
37     for(auto q:qst)
38     {
39         while(l<q.l) del(l++);
40         while(l>q.l) add(--l);
41         while(r<q.r) add(++r);
42         while(r>q.r) del(r--);
43         ans[q.id]+=res;
44     }
45 }
46 #undef type

```

2.18 ODT

```

1 struct ODT
2 {
3     #define type int
4     #define init_val 0
5     struct ODT_node
6     {
7         int l,r,id;
8         mutable type v;
9         ODT_node(const int &l,const int &r,const
10             type &v): l(l),r(r),v(v){}
11         inline bool operator<(const ODT_node &o)const
12             {return l<o.l;}
13     };
14     int n;
15     set<ODT_node> odt;
16     typedef set<ODT_node>::iterator odt_iter;
17     set<int> mp[MAX];
18     void init(int _n)
19     {
20         n=_n;
21         odt.clear();
22         odt.insert(ODT_node(1,n,init_val));
23         for(int i=1;i<=n;i++) mp[i].clear();
24     }
25     odt_iter find(int x){return --odt.upper_bound((
26         ODT_node){x,0,init_val});}
27     odt_iter split(int x)
28     {
29         if(x>n) return odt.end();
30         odt_iter it=find(x);
31         if(it->l==x) return it;
32         int l=it->l,r=it->r;
33         type v=it->v;
34         odt.erase(it);
35         odt.insert(ODT_node(l,x-1,v));
36         return odt.insert(ODT_node(x,r,v)).first;
37     }
38     void assign(int l,int r,type v)
39     {
40         odt_iter itr=split(r+1),itl=split(l);
41         odt.erase(itl,itr);
42         odt.insert(ODT_node(l,r,v));
43     }
44     int work(int pos,int v)
45     {
46         auto it=find(pos);
47         if(it->v <= v) return sz(odt);
48         int pre=pos;
49         for(;it!=odt.end();it++)

```



```

47     {
48         if(it->v>=v) pre=it->r;
49         else break;
50     }
51     assign(pos,pre,v);
52     return sz(odt);
53 }
54 #undef init_val
55 #undef type
56 }odt;

```

2.19 分块

```

1 struct Block
2 {
3     #define type int
4     static const int N=;
5     static const int size=sqrt(N);
6     static const int num=N/size+1;
7
8     void init()
9     {
10
11     }
12     void point_modify(int x,type v)
13     {
14         int id=(x-1)/size+1;
15         int pos=x%size;
16
17     }
18     void block_modify(int id,type v)
19     {
20
21     }
22     type point_query(int x)
23     {
24         int id=(x-1)/size+1;
25         int pos=x%size;
26
27     }
28     type block_query(int id)
29     {
30
31     }
32     void upd(int l,int r,type x)
33     {
34         while(l<=r&&r%blocks!=0) point_modify(r,x),r
35             --;
36         while(l<=r&&l%blocks!=1) point_modify(l,x),l
37             ++;
38         while(l<=r)
39         {
40             int id=(l-1)/size+1;

```

```

39         block_modify(id,x);
40         l+=size;
41     }
42 }
43 type ask(int l,int r)
44 {
45     type ans=0;
46     while(l<=r&&r%blocks!=0) ans+=point_query(r),
47         r--;
48     while(l<=r&&l%blocks!=1) ans+=point_query(l),
49         l++;
50     while(l<=r)
51     {
52         int id=(l-1)/size+1;
53         ans+=block_query(id);
54         l+=size;
55     }
56     return ans;
57 }
58 #undef type
59 }blk;

```

3 树

3.1 LCA

3.1.1 倍增 LCA

```

1 struct LCA
2 {
3     static const int N=MAX;
4     static const int LOGN=log2(N)+3;
5     int fa[N][LOGN],dep[N],limt,bin[LOGN];
6     void dfs(int x,int pre,vector<int> mp[])
7     {
8         int i;
9         for(i=1;bin[i]<=dep[x];i++) fa[x][i]=fa[fa[x]
10             ][i-1][i-1];
11         for(auto &to:mp[x])
12         {
13             if(to==pre) continue;
14             dep[to]=dep[x]+1;
15             fa[to][0]=x;
16             dfs(to,x,mp);
17         }
18     }
19     void work(int n,int root,vector<int> mp[])
20     {
21         int i;
22         for(limt=1;(1<<(limt-1))<n;limt++);
23         for(i=bin[0]=1;i<=limt;i++) bin[i]=(bin[i-1]
24             <<1);

```

```

23     for(i=0;i<=n;i++) memset(fa[i],0,sizeof fa[i
24         ]);
25     dep[root]=0;
26     dfs(root,-1,mp);
27 }
28 int go(int x,int d)
29 {
30     for(int i=0;i<=limt&&d;i++)
31     {
32         if(bin[i]&d)
33         {
34             d^=bin[i];
35             x=fa[x][i];
36         }
37     }
38     return x;
39 }
40 int lca(int x,int y)
41 {
42     if(dep[x]<dep[y]) swap(x,y);
43     x=go(x,dep[x]-dep[y]);
44     if(x==y) return x;
45     for(int i=limt;~i;i--)
46     {
47         if(fa[x][i]!=fa[y][i])
48         {
49             x=fa[x][i];
50             y=fa[y][i];
51         }
52     }
53     return fa[x][0];
54 }lca;
55 /*
56 O(nlogn)-O(logn)
57 lca.work(n,root,mp);
58 */

```

3.1.2 RMQ 维护欧拉序求 LCA

```

1 struct LCA
2 {
3     static const int N=MAX;
4     static const int LOG2N=log2(2*N)+3;
5     #define type int
6     struct node{int to;type w;};
7     type dis[N];
8     int path[2*N],deep[2*N],first[N],len[N],tot,n;
9     int dp[2*N][LOG2N];
10    vector<node> mp[N];
11    void init(int _n)
12    {
13        n=_n;
14        for(int i=0;i<=n;i++)

```

```

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

```

```

68     }
69     type get_dis(int a,int b){return dis[a]+dis[b
70         ]-2*dis[lca(a,b)];}
71     int get_len(int a,int b){return len[a]+len[b]-2*
72         len[lca(a,b)];}
73     void work(int rt)
74     {
75         tot=0;
76         dfs(rt,0,0);
77         ST(2*n-1);
78     }
79     int lca_root(int rt,int x,int y)
80     {
81         int fx,fy;
82         fx=lca(x,rt);
83         fy=lca(y,rt);
84         if(fx==fy) return lca(x,y);
85         else
86         {
87             if(get_len(fx,rt)<get_len(fy,rt)) return
88                 fx;
89             else return fy;
90         }
91     }
92     #undef type
93 }lca;
94 /*
95 O(n*log(n))-O(1)
96
97 lca.init(n);
98 lca.add_edge(a,b,w);
99 lca.work(rt);
100 */

```

3.2 树链剖分

3.2.1 轻重链剖分

```

1 struct Heavy_Light-Decomposition
2 {
3     #define type int
4     struct edge{int a,b;type v;};
5     struct node{int to;type w;};
6     vector<int> mp[MAX];
7     vector<edge> e;
8     int dep[MAX],fa[MAX],sz[MAX],son[MAX];
9     int id[MAX],top[MAX],dfn[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.push_back({0,0,0});

```

```

17    }
18    void add_edge(int a,int b,type v=0)
19    {
20        e.push_back({a,b,v});
21        mp[a].push_back(b);
22        mp[b].push_back(a);
23    }
24    void dfs1(int x,int pre,int h)
25    {
26        int i,to;
27        dep[x]=h;
28        fa[x]=pre;
29        sz[x]=1;
30        for(i=0;i<mp[x].size();i++)
31        {
32            to=mp[x][i];
33            if(to==pre) continue;
34            dfs1(to,x,h+1);
35            sz[x]+=sz[to];
36            if(son[x]==-1||sz[to]>sz[son[x]]) son[x]=
37                to;
38        }
39    }
40    void dfs2(int x,int tp)
41    {
42        int i,to;
43        dfn[x]=++tot;
44        id[dfn[x]]=x;
45        top[x]=tp;
46        if(son[x]==-1) return;
47        dfs2(son[x],tp);
48        for(i=0;i<mp[x].size();i++)
49        {
50            to=mp[x][i];
51            if(to!=son[x]&&to!=fa[x]) dfs2(to,to);
52        }
53    }
54    void work(int _rt)
55    {
56        rt=_rt;
57        for(int i=0;i<=n;i++) son[i]=-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(dep[top[x]]<dep[top[y]]) swap(x,y);
67            x=fa[top[x]];
68        }
69        if(dep[x]>dep[y]) swap(x,y);
70        return x;

```

```

70     }
71     //node
72     void init_node(type *v)
73     {
74         for(int i=1;i<=n;i++) tr.a[dfn[i]]=v[i];
75         tr.build(n);
76     }
77     void upd_node(int x,int y,type v)
78     {
79         while(top[x]!=top[y])
80         {
81             if(dep[top[x]]<dep[top[y]]) swap(x,y);
82             tr.upd(dfn[top[x]],dfn[x],v);
83             x=fa[top[x]];
84         }
85         if(dep[x]>dep[y]) swap(x,y);
86         tr.upd(dfn[x],dfn[y],v);
87     }
88     type ask_node(int x,int y)
89     {
90         type res=0;
91         while(top[x]!=top[y])
92         {
93             if(dep[top[x]]<dep[top[y]]) swap(x,y);
94             res+=tr.ask(dfn[top[x]],dfn[x]);
95             x=fa[top[x]];
96         }
97         if(dep[x]>dep[y]) swap(x,y);
98         res+=tr.ask(dfn[x],dfn[y]);
99         return res;
100     }
101     //path
102     void init_path()
103     {
104         tr.a[dfn[rt]]=0;
105         for(int i=1;i<n;i++)
106         {
107             if(dep[e[i].a]<dep[e[i].b]) swap(e[i].a,e[i].b);
108             tr.a[dfn[e[i].a]]=e[i].v;
109         }
110         tr.build(n);
111     }
112     void upd_edge(int id,type v)
113     {
114         if(dep[e[id].a]>dep[e[id].b]) tr.upd(dfn[e[id].a],dfn[e[id].a],v);
115         else tr.upd(dfn[e[id].b],dfn[e[id].b],v);
116     }
117     void upd_path(int x,int y,type v)
118     {
119         while(top[x]!=top[y])
120         {
121             if(dep[top[x]]<dep[top[y]]) swap(x,y);

```

```

122         tr.upd(dfn[top[x]],dfn[x],v);
123         x=fa[top[x]];
124     }
125     if(dep[x]>dep[y]) swap(x,y);
126     if(x!=y) tr.upd(dfn[x]+1,dfn[y],v);
127 }
128 type ask_path(int x,int y)
129 {
130     type res=0;
131     while(top[x]!=top[y])
132     {
133         if(dep[top[x]]<dep[top[y]]) swap(x,y);
134         res+=tr.ask(dfn[top[x]],dfn[x]);
135         x=fa[top[x]];
136     }
137     if(dep[x]>dep[y]) swap(x,y);
138     if(x!=y) res+=tr.ask(dfn[x]+1,dfn[y]);
139     return res;
140 }
141 // sub tree
142 void upd_subtree(int x,type v){tr.upd(dfn[x],dfn[x]+sz[x]-1,v);}
143 type ask_subtree(int x){return tr.ask(dfn[x],dfn[x]+sz[x]-1);}
144 #undef type
145 }hld;
146 /*
147 hld.init(n)
148 hld.add_edge(a,b,v=0); a <-> b
149 hld.work(root);
150 */

```

3.3 树的重心

```

1 struct Tree_Centroid
2 {
3     VI centroid;
4     int sz[MAX],w[MAX],n;
5     void dfs(VI *mp,int x,int fa)
6     {
7         sz[x]=1;
8         w[x]=0;
9         for(int i=0;i<sz(mp[x]);i++)
10         {
11             int to=mp[x][i];
12             if(to==fa) continue;
13             dfs(mp,to,x);
14             sz[x]+=sz[to];
15             w[x]=max(w[x],sz[to]);
16         }
17         w[x]=max(w[x],n-sz[x]);
18         if(w[x]<=n/2) centroid.pb(x);
19     }

```

```

20 VI get_tree_centroid(int _n,VI *mp,int root)
21 {
22     n=_n;
23     centroid.clear();
24     dfs(mp,root,0);
25     return centroid;
26 }
27 }trct;

```

3.4 树 hash

```

1 struct Tree_Hash
2 {
3     const ull mask=std::chrono::steady_clock::now().
4         time_since_epoch().count();
5     ull shift(ull x)
6     {
7         x^=mask;
8         x^=x<<13;
9         x^=x>>7;
10        x^=x<<17;
11        x^=mask;
12        return x;
13    }
14    ull hash[MAX];
15    void dfs(VI *mp,int x,int fa)
16    {
17        hash[x]=1;
18        for(auto to:mp[x])
19        {
20            if(to==fa) continue;
21            dfs(mp,to,x);
22            hash[x]+=shift(hash[to]);
23        }
24    }
25    void get_tree_hash(VI *mp,int root)
26    {
27        dfs(mp,root,0);
28    }
29 }trha;

```

3.5 虚树

```

1 struct Virtual_Tree
2 {
3     int st[MAX],top;
4     int build_vtree(VI &a,VI vtree_mp[])// return
5         root
6     {
7         int now_lca;
8         sort(all(a), [&](int x,int y){return lca.dfn[x]
9             <lca.dfn[y];});
10        a.erase(unique(all(a)),a.end());

```

```

9        assert(sz(a)>0);
10        top=0;
11        st[top++]=a[0];
12        VI tmp;
13        for(int i=1;i<sz(a);i++)
14        {
15            if(top==0)
16            {
17                st[top++]=a[i];
18                continue;
19            }
20            now_lca=lca.lca(a[i],st[top-1]);
21            while(top>1&&lca.dfn[st[top-2]]>=lca.dfn[
22                now_lca])
23            {
24                vtree_mp[st[top-2]].pb(st[top-1]);
25                top--;
26            }
27            if(now_lca!=st[top-1])
28            {
29                vtree_mp[now_lca].pb(st[top-1]);
30                st[top-1]=now_lca;
31                tmp.push_back(now_lca);
32            }
33            st[top++]=a[i];
34        }
35        while(top>1)
36        {
37            vtree_mp[st[top-2]].pb(st[top-1]);
38            top--;
39        }
40        for(auto it:tmp) a.push_back(it);
41        return st[0];
42    }
43    void clear_vtree(VI &a,VI vtree_mp[])
44    {
45        for(auto it:a) vtree_mp[it].clear();
46    }
47 }vt; // need lca and dfn

```

3.6 Link-Cut-Tree

```

1 struct Link_Cut_Tree
2 {
3     #define type int
4     const type inf=INF;
5     struct node
6     {
7         int ch[2],fa,sz,rev,tag;
8         type v,sum;
9     }t[MAX];
10    int tot,root,st[MAX];
11    void maintain(int id,type v)

```

```

12 {
13     t[id].v=v;
14     t[id].sum=v;
15 }
16 void pushup(int id)
17 {
18     int ls=t[id].ch[0];
19     int rs=t[id].ch[1];
20     t[id].sz=t[ls].sz+t[rs].sz+1;
21     t[id].sum=t[ls].sum+t[rs].sum+t[id].v;
22 }
23 void pushdown(int id)
24 {
25     int ls=t[id].ch[0];
26     int rs=t[id].ch[1];
27     if(t[id].tag)
28     {
29         if(ls)
30         {
31             maintain(ls,t[id].v);
32             t[ls].tag=1;
33         }
34         if(rs)
35         {
36             maintain(rs,t[id].v);
37             t[rs].tag=1;
38         }
39         t[id].tag=0;
40     }
41     if(t[id].rev)
42     {
43         t[ls].rev^=1;
44         t[rs].rev^=1;
45         swap(t[ls].ch[0],t[ls].ch[1]);
46         swap(t[rs].ch[0],t[rs].ch[1]);
47         t[id].rev=0;
48     }
49 }
50 int newnode(type v,int fa)
51 {
52     int id=++tot;
53     memset(t[id].ch,0,sizeof t[id].ch);
54     t[id].fa=fa;
55     t[id].sz=1;
56     t[id].tag=t[id].rev=0;
57     maintain(id,v);
58     return id;
59 }
60 int is_splay_root(int x)
61 {
62     int fa=t[x].fa;
63     return t[fa].ch[0]!=x&&t[fa].ch[1]!=x;
64 }
65 void rotate(int x)

```

```

66 {
67     int y,z,k;
68     y=t[x].fa;
69     z=t[y].fa;
70     k=(x==t[y].ch[1]);
71     if(!is_splay_root(y)) t[z].ch[y==t[z].ch[1]]=
        x;
72     t[y].ch[k]=t[x].ch[k^1];
73     if(t[x].ch[k^1]) t[t[x].ch[k^1]].fa=y;
74     t[x].ch[k^1]=y;
75     t[y].fa=x;
76     t[x].fa=z;
77     pushup(y);
78 }
79 void splay(int x)
80 {
81     int y,z,top;
82     top=0;
83     y=x;
84     st[++top]=y;
85     while(!is_splay_root(y))
86     {
87         y=t[y].fa;
88         st[++top]=y;
89     }
90     while(top>0) pushdown(st[top--]);
91     while(!is_splay_root(x))
92     {
93         y=t[x].fa;
94         z=t[y].fa;
95         if(!is_splay_root(y))
96         {
97             if((t[z].ch[0]==y)^(t[y].ch[0]==x))
98                 rotate(x);
99             else rotate(y);
100         }
101         rotate(x);
102     }
103     pushup(x);
104 }
105 void init_null_node()
106 {
107     memset(t[0].ch,0,sizeof t[0].ch);
108     t[0].sz=t[0].fa=0;
109     t[0].v=t[0].sum=0;
110 }
111 void init(int n,type *v)
112 {
113     int i;
114     tot=0;
115     init_null_node();
116     for(i=1;i<=n;i++) newnode(v[i],0);
117 }
118 void init(int n)

```

```

118 {
119     int i;
120     tot=0;
121     init_null_node();
122     for(i=1;i<=n;i++) newnode(0,0);
123 }
124 int access(int x)
125 {
126     int fa=0;
127     while(x)
128     {
129         splay(x);
130         t[x].ch[1]=fa;
131         pushup(x);
132         fa=x;
133         x=t[x].fa;
134     }
135     return fa;
136 }
137 int findroot(int x)
138 {
139     access(x);
140     splay(x);
141     pushdown(x);
142     while(t[x].ch[0])
143     {
144         x=t[x].ch[0];
145         pushdown(x);
146     }
147     splay(x);
148     return x;
149 }
150 void makeroot(int x)
151 {
152     x=access(x);
153     swap(t[x].ch[0],t[x].ch[1]);
154     t[x].rev^=1;
155 }
156 int split(int x,int y)
157 {
158     makeroot(x);
159     access(y);
160     splay(y);
161     return y;
162 }
163 void link(int x,int y)
164 {
165     makeroot(x);
166     splay(x);
167     if(findroot(y)!=x) t[x].fa=y;
168 }
169 void cut(int x,int y)
170 {
171     makeroot(x);

```

```

172     if(findroot(y)==x&&t[y].fa==x&&t[y].ch[0]==0)
173     {
174         t[y].fa=t[x].ch[1]=0;
175         pushup(x);
176     }
177 }
178 int is_connect(int x,int y)
179 {
180     makeroot(x);
181     return findroot(y)==x;
182 }
183 void upd_node(int x,type v)
184 {
185     splay(x);
186     maintain(x,v);
187 }
188 #undef type
189 }lct;
190 /*
191 lct.init(n);
192 lct.init(n,*v); v[1..n]
193 */

```

4 图论

4.1 链式前向星

```

1 int head[MAX],tot;
2 struct node{int to,nex;int v} mp[MAXE];
3 void init()
4 {
5     memset(head,-1,sizeof head);
6     tot=0;
7 }
8 void add_edge(int x,int y,int v)
9 {
10     mp[tot].v=v;
11     mp[tot].to=y;
12     mp[tot].nex=head[x];
13     head[x]=tot++;
14 }

```

4.2 最短路

4.2.1 dijkstra

```

1 struct Dijkstra
2 {
3     #define type int
4     const type inf=INF;
5     struct node
6     {
7         int id;

```

```

8     type v;
9     friend bool operator <(node a,node b){return
        a.v>b.v;}
10 };
11 static const int N=MAX;
12 vector<node> mp[N];
13 type dis[N];
14 int n,vis[N];
15 void init(int _n)
16 {
17     n=_n;
18     for(int i=0;i<=n;i++) mp[i].clear();
19 }
20 void add_edge(int x,int y,type v){mp[x].
    push_back({y,v});}
21 void work(int s)
22 {
23     int i,to;
24     type w;
25     priority_queue<node> q;
26     for(i=0;i<=n;i++)
27     {
28         dis[i]=inf;
29         vis[i]=0;
30     }
31     dis[s]=0;
32     q.push({s,type(0)});
33     while(!q.empty())
34     {
35         node t=q.top();
36         q.pop();
37         if(vis[t.id]) continue;
38         vis[t.id]=1;// this node has already been
            extended
39         for(auto &it:mp[t.id])
40         {
41             to=it.id;
42             w=it.v;
43             if(dis[to]>dis[t.id]+w)
44             {
45                 dis[to]=dis[t.id]+w;
46                 if(!vis[to]) q.push({to,dis[to]});
47             }
48         }
49     }
50 }
51 #undef type
52 }dij;

```

4.2.2 spfa

```

1 struct SPFA
2 {
3     #define type int

```

```

4 #define inf INF
5 #define PTI pair<type,int>
6 static const int N=MAX;
7 vector<pair<int,type> > mp[N];
8 type dis[N];
9 int n,vis[N],cnt[N];
10 void init(int _n)
11 {
12     n=_n;
13     for(int i=0;i<=n;i++) mp[i].clear();
14 }
15 void add_edge(int x,int y,type v){ mp[x].pb(MP(y
    ,v));}
16 bool work(int s)
17 {
18     int i,x,to;
19     type w;
20     queue<int> q;
21     for(i=0;i<=n;i++)
22     {
23         dis[i]=inf;
24         vis[i]=cnt[i]=0;
25     }
26     dis[s]=0;
27     vis[s]=1;
28     q.push(s);
29     while(!q.empty())
30     {
31         x=q.front();
32         q.pop();
33         vis[x]=0;
34         for(auto it:mp[x])
35         {
36             to=it.fi;
37             w=it.se;
38             if(dis[to]>dis[x]+w)
39             {
40                 dis[to]=dis[x]+w;
41                 cnt[to]=cnt[x]+1;
42                 if(cnt[to]>n)
43                 {
44                     // cnt is edge counts of
                        current short path
45                     // if cnt >= (sum of node), the
                        graph exists negative ring
46                     return false;
47                 }
48                 if(!vis[to])
49                 {
50                     q.push(to);
51                     vis[to]=1;
52                 }
53             }
54         }

```



```

55     }
56     return true;
57 }
58 #undef type
59 #undef inf
60 #undef PTI
61 }spfa;

```

4.2.3 floyd 求最小环

```

1 struct Floyd
2 {
3     #define type int
4     const type inf=INF;
5     static const int N=;
6     int n;
7     type mp[N][N],dis[N][N];
8     type min_circle_3;// len(circle)>=3
9     type min_circle; // len(circle)>=1
10    void init(int _n)
11    {
12        int i,j;
13        n=_n;
14        for(i=1;i<=n;i++)
15        {
16            for(j=1;j<=n;j++)
17            {
18                mp[i][j]=dis[i][j]=inf;
19            }
20        }
21    }
22    void add_edge(int x,int y,type w)
23    {
24        w=min(mp[x][y],w);
25        mp[x][y]=dis[x][y]=w;
26    }
27    void work()
28    {
29        int i,j,k;
30        min_circle_3=inf;
31        for(k=1;k<=n;k++)
32        {
33            for(i=1;i<k;i++)
34            {
35                if(mp[i][k]==inf) continue;
36                for(j=1;j<k;j++)
37                {
38                    if(i==j||mp[k][j]==inf||dis[j][i]==
39                        inf) continue;
40                    min_circle_3=min(min_circle_3,mp[i
41                        ][k]+mp[k][j]+dis[j][i]);
42                }
43            }
44        }
45        for(i=1;i<=n;i++)

```

```

43    {
44        if(dis[i][k]==inf) continue;
45        for(j=1;j<=n;j++)
46        {
47            if(dis[k][j]==inf) continue;
48            dis[i][j]=min(dis[i][j],dis[i][k]+
49                dis[k][j]);
50        }
51    }
52    min_circle=inf;
53    for(i=1;i<=n;i++) min_circle=min(min_circle,
54        dis[i][i]);
55    }
56    #undef type
57 }floyd;
58 /*
59 floyd.init(n);
60 floyd.add_edge(x,y,w); x,y [1..n] x->y
61 floyd.work();
62 */

```

4.2.4 Johnson

```

1 struct Johnson
2 {
3     #define type int
4     #define inf INF
5     #define PTI pair<type,int>
6     static const int N=3000+10;
7     vector<pair<int,type> > mp[N];
8     type dis[N],h[N];
9     int n,vis[N],cnt[N];
10    bool spfa_flag;
11    void init(int _n)
12    {
13        n=_n;
14        spfa_flag=false;
15        for(int i=0;i<=n;i++) mp[i].clear();
16    }
17    void add_edge(int x,int y,type v){mp[x].pb(MP(y,
18        v));}
19    bool spfa(int s)
20    {
21        int i,x,to;
22        type w;
23        queue<int> q;
24        for(i=0;i<=n;i++)
25        {
26            h[i]=inf;
27            vis[i]=cnt[i]=0;
28        }
29        h[s]=0;
30        vis[s]=1;

```

```

30     q.push(s);
31     while(!q.empty())
32     {
33         x=q.front();
34         q.pop();
35         vis[x]=0;
36         for(auto it:mp[x])
37         {
38             to=it.fi;
39             w=it.se;
40             if(h[to]>h[x]+w)
41             {
42                 h[to]=h[x]+w;
43                 cnt[to]=cnt[x]+1;
44                 if(cnt[to]>n)
45                 {
46                     // cnt is edge counts of
47                     // current short path
48                     // if cnt >= (sum of node), the
49                     // graph exists negative ring
50                     return false;
51                 }
52                 if(!vis[to])
53                 {
54                     q.push(to);
55                     vis[to]=1;
56                 }
57             }
58         }
59         spfa_flag=true;
60         return true;
61     }
62     void dij(int s)
63     {
64         assert(spfa_flag);
65         int i,to;
66         type w;
67         priority_queue<PTI ,vector<PTI>,greater<PTI>
68             > q;
69         for(i=0;i<=n;i++)
70         {
71             dis[i]=inf;
72             vis[i]=0;
73         }
74         dis[s]=0;
75         q.push(MP(type(0),s));
76         while(!q.empty())
77         {
78             PTI t=q.top();
79             q.pop();
80             if(vis[t.se]) continue;
81             vis[t.se]=1;
82             for(auto it:mp[t.se])

```

```

81         {
82             to=it.fi;
83             w=it.se+h[t.se]-h[to];
84             if(dis[to]>dis[t.se]+w)
85             {
86                 dis[to]=dis[t.se]+w;
87                 if(!vis[to]) q.push(MP(dis[to],to))
88                 ;
89             }
90         }
91         for(i=0;i<=n;i++)
92         {
93             if(dis[i]==inf) continue;
94             dis[i]-=h[s]-h[i];
95         }
96     }
97     #undef type
98     #undef inf
99     #undef PTI
100 }js;

```

4.2.5 同余最短路

```

1 struct Dijkstra
2 {
3     #define type ll
4     const type inf=LLINF;
5     struct node
6     {
7         int id;
8         type v;
9         friend bool operator <(node a,node b){return
10             a.v>b.v;}
11 };
12 static const int N=MAX;
13 vector<node> mp[N];
14 type dis[N];
15 int n,vis[N];
16 void init(int _n)
17 {
18     n=_n;
19     for(int i=0;i<=n;i++) mp[i].clear();
20 }
21 void add_edge(int x,int y,type v){ mp[x].
22     push_back({y,v});}
23 void work(int s)
24 {
25     int i,to;
26     type w;
27     priority_queue<node> q;
28     for(i=0;i<=n;i++)
29     {
30         dis[i]=inf;

```

```

29     vis[i]=0;
30 }
31 dis[s]=0;
32 q.push({s,type(0)});
33 while(!q.empty())
34 {
35     node t=q.top();
36     q.pop();
37     if(vis[t.id]) continue;
38     vis[t.id]=1; // this node has already been
39                 extended
40     for(auto &it:mp[t.id])
41     {
42         to=it.id;
43         w=it.v;
44         if(dis[to]>dis[t.id]+w)
45         {
46             dis[to]=dis[t.id]+w;
47             if(!vis[to]) q.push({to,dis[to]});
48         }
49     }
50 }
51 #undef type
52 }dij;
53 // [l,r] b sum{ai*xi}=b, xi>=0
54 ll congruent_short_path(vector<int> &a,ll l,ll r)
55 {
56     int n,i,j;
57     ll res;
58     n=a.size();
59     sort(a.begin(),a.end());
60     a.resize(unique(a.begin(),a.end())-a.begin());
61     dij.init(a[0]);
62     for(i=0;i<a[0];i++)
63     {
64         for(j=1;j<n;j++)
65         {
66             dij.add_edge(i,(i+a[j])%a[0],a[j]);
67         }
68     }
69     dij.work(0);
70     res=0;
71     for(i=0;i<a[0];i++)
72     {
73         if(dij.dis[i]<=r) res+=(r-dij.dis[i])/a[0]+1;
74         if(dij.dis[i]<=l-1) res-=((l-1)-dij.dis[i])/a
75             [0]+1;
76     }
77     return res;
78 }

```

4.3 最小生成树

4.3.1 kruskal

```

1 struct Disjoint_Set_Union
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 Kruskal
28 {
29     #define type int
30     #define inf INF
31     struct edge{int x,y,type w;};
32     vector<edge> e;
33     void init(){e.clear();}
34     void add_edge(int a,int b,type w){e.push_back({a
35         ,b,w});}
36     type work(int n)
37     {
38         int i,cnt;
39         type res=0;
40         dsu.init(n);
41         sort(e.begin(),e.end(),[&](edge x,edge y){
42             return x.w<y.w;
43         });
44         for(auto &it:e)
45         {
46             if(dsu.merge(it.x,it.y)) res+=it.w;
47         }
48         cnt=0;
49         for(i=1;i<=n;i++) cnt+=dsu.find(i)==i;
50         if(cnt!=1) return inf; // no connect
51         return res;
52     }
53 }

```

```

51     }
52     #undef type
53     #undef inf
54 }krsk;

```

4.3.2 kruskal 重构树

```

1 struct Disjoint_Set_Union
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 Kruskal_Tree
28 {
29     #define type int
30     #define inf INF
31     struct edge{int x,y;type w;};
32     vector<edge> e;
33     void init(){e.clear();}
34     void add_edge(int x,int y,type w){e.push_back({x,y,w});}
35     int build_kruskal_tree(int n,vector<int> *mp,
36                             type *w)
37     {
38         int rt,x,y,i;
39         for(i=1;i<=2*n-1;i++)
40         {
41             mp[i].clear();
42             w[i]=0;
43         }
44         dsu.init(2*n-1);
45         sort(e.begin(),e.end(),[&](edge x,edge y){

```

```

46         }));
47         rt=n;
48         for(auto &it:e)
49         {
50             x=dsu.find(it.x);
51             y=dsu.find(it.y);
52             if(x==y) continue;
53             rt++;
54             w[rt]=it.w;
55             mp[rt].push_back(x);
56             mp[rt].push_back(y);
57             dsu.merge(x,rt);
58             dsu.merge(y,rt);
59         }
60         return rt;
61     }
62     #undef type
63     #undef inf
64 }krsk;

```

4.3.3 prim

```

1 struct Prim
2 {
3     #define type int
4     const type inf=INF;
5     struct node{int id;type w;};
6     static const int N=MAX;
7     vector<node> mp[N];
8     type dis[N];
9     int n,vis[N];
10    void init(int _n)
11    {
12        n=_n;
13        for(int i=0;i<=n;i++) mp[i].clear();
14    }
15    void add_edge(int x,int y,type w)
16    {
17        mp[x].push_back({y,w});
18        mp[y].push_back({x,w});
19    }
20    type work()
21    {
22        int i,x,cnt;
23        type res,mn;
24        for(i=0;i<=n;i++)
25        {
26            dis[i]=inf;
27            vis[i]=0;
28        }
29        x=1;
30        for(auto &to:mp[x]) dis[to.id]=min(dis[to.id],to.w);
31        res=0;

```

```

32     cnt=0;
33     while(1)
34     {
35         mn=inf;
36         vis[x]=1;
37         cnt++;
38         for(i=1;i<=n;i++)
39         {
40             if(!vis[i]&&dis[i]<mn)
41             {
42                 mn=dis[i];
43                 x=i;
44             }
45         }
46         if(mn==inf) break;
47         res+=mn;
48         for(auto &to:mp[x]) dis[to.id]=min(dis[to
         .id],to.w);
49     }
50     if(cnt<n) return inf; // no connect
51     return res;
52 }
53 #undef type
54 }prim;
55 /*
56 O(n^2+m)
57
58 prim.init(n);
59 prim.add_edge(x,y,w); x<->y
60 prim.work();
61 */

```

4.4 二分图匹配

4.4.1 二分图最大匹配

```

1  /*二分图 特 点
2  μ, 二分图匹配=二分图最大匹配
3  • 二分图, 二分图匹配=二分图最大匹配-二分图
4  b^二分图=二分图 • 二分图, 二分图匹配=二分图最大匹配-二分图
5  */
6  struct Bipartite_Matching
7  {
8      static const int N=;
9      static const int M=;
10     int n,m;
11     vector<int> mp[N];
12     int flag[N],s[N],link[M],used[M];
13     void init(int _n,int _m)
14     {
15         n=_n;
16         m=_m;
17         for(int i=0;i<=n;i++) mp[i].clear();
18     }

```

```

19     void add_edge(int a,int b){mp[a].push_back(b);}
20     bool dfs(int x,int timetag)
21     {
22         int i,to;
23         flag[x]=1;
24         for(i=0;i<mp[x].size();i++)
25         {
26             to=mp[x][i];
27             if(used[to]==timetag) continue;
28             used[to]=timetag;
29             if(link[to]==-1||dfs(link[to],timetag))
30             {
31                 link[to]=x;
32                 s[x]=to;
33                 return 1;
34             }
35         }
36         return 0;
37     }
38     int max_match()
39     {
40         int i,res;
41         memset(link,-1,sizeof link);
42         memset(s,-1,sizeof s);
43         memset(used,0,sizeof used);
44         res=0;
45         for(i=1;i<=n;i++)
46         {
47             if(mp[i].size()==0) continue;
48             if(dfs(i,i)) res++;
49         }
50         return res;
51     }
52     int min_cover(vector<int> &x,vector<int> &y)
53     {
54         int i,res;
55         res=max_match();
56         memset(flag,0,sizeof flag);
57         memset(used,0,sizeof used);
58         x.clear();
59         y.clear();
60         for(i=1;i<=n;i++)
61         {
62             if(s[i]==-1) dfs(i);
63         }
64         for(i=1;i<=n;i++)
65         {
66             if(!flag[i]) x.push_back(i);
67         }
68         for(i=1;i<=m;i++)
69         {
70             if(used[i]) y.push_back(i);
71         }
72         return res;

```

4.4.2 二分图最大权完美匹配

```
while(1)
{
    while(!q.empty())
    {
        x=q.front();
        q.pop();
        for(y=1;y<=n;y++)
        {
            d=lx[x]+ly[y]-w[x][y];
            if(!vy[y]&&d<=slk[y])
            {
                pre[y]=x;
                if(!d)
                {
                    if(!my[y]) return match(y);
                    q.push(my[y]);
                    vx[my[y]]=1;
                    vy[y]=1;
                }
                else slk[y]=d;
            }
        }
    }
    d=inf+1;
    for(i=1;i<=n;i++)
    {
        if(!vy[i]&&slk[i]<d)
        {
            d=slk[i];
            y=i;
        }
    }
    for(i=1;i<=n;i++)
    {
        if(vx[i]) lx[i]-=d;
        if(vy[i]) ly[i]+=d;
        else slk[i]-=d;
    }
    if(!my[y]) return match(y);
    q.push(my[y]);
    vx[my[y]]=1;
    vy[y]=1;
}

// int max_match()
// {
//     int i,j;
//     type res;
//     for(i=1;i<=n;i++)
//     {
//         mx[i]=my[i]=ly[i]=0;
//         lx[i]=*max_element(w[i]+1,w[i]+n+1);
//     }
//     for(i=1;i<=n;i++) bfs(i);
// }
```

```

98     res=0;
99     for(i=1;i<=n;i++)
100     {
101         // 2»
102         res+=w[i][mx[i]];
103     }
104     return res;
105 }
106 #undef type
107 }km;
108 /*
109 O(n^3)
110 km.init(n);
111 km.add_edge(a,b,val); a,b: 1~n
112 */

```

4.5 最大流

4.5.1 dinic

```

1 struct Dinic
2 {
3     #define type int
4     const type 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    {
29        edge.push_back(node(from,to,cap,0));
30        edge.push_back(node(to,from,0,0));
31        int m=edge.size();
32        mp[from].push_back(m-2);
33        mp[to].push_back(m-1);
34    }
35 }

```

```

34 bool bfs()
35 {
36     int i,x;
37     memset(vis,0,sizeof vis);
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;
82     type res=0;
83     while(bfs())
84     {
85         for(int i=0;i<=n;i++) id[i]=0;
86         res+=dfs(s,inf);
87     }
88 }

```

```

87     return res;
88 }
89 #undef type
90 }dc;
91 /*
92 O(n^2*m)
93 bipartite graph: O(m*sqrt(n))
94
95 dc.init(n);
96 dc.add_edge(a,b,cap); a,b: 1~n
97 */

```

4.5.2 有源汇上下界网络流

```

1 struct Dinic
2 {
3     #define type int
4     const type 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,s1,t1;
14    vector<node> edge;
15    vector<int> mp[N];
16    int vis[N],dist[N],id[N];
17    type in[N],out[N];
18    void init(int _n)
19    {
20        s1=_n+1;
21        t1=s1+1;
22        n=t1;
23        assert(n<N);
24        edge.clear();
25        for(int i=0;i<=n;i++)
26        {
27            mp[i].clear();
28            id[i]=dist[i]=0;
29            in[i]=out[i]=0;
30        }
31    }
32    void add_edge(int from,int to,type lcap,type
33        rcap)
34    {
35        edge.push_back(node(from,to,rcap-lcap,0));
36        edge.push_back(node(to,from,0,0));
37        in[to]+=lcap;
38        out[from]+=lcap;
39        int m=edge.size();
40        mp[from].push_back(m-2);

```

```

39        mp[to].push_back(m-1);
40    }
41    bool bfs()
42    {
43        int i,x;
44        for(int i=0;i<=n;i++) vis[i]=0;
45        queue<int>q;
46        q.push(s);
47        dist[s]=0;
48        vis[s]=1;
49        while(!q.empty())
50        {
51            x=q.front();
52            q.pop();
53            for(i=0;i<mp[x].size();i++)
54            {
55                node &e=edge[mp[x][i]];
56                if(!vis[e.to]&&e.cap>e.flow)
57                {
58                    vis[e.to]=1;
59                    dist[e.to]=dist[x]+1;
60                    q.push(e.to);
61                }
62            }
63        }
64        return vis[t];
65    }
66    type dfs(int x,type a)
67    {
68        if(x==t||!a) return a;
69        type flow=0,f;
70        for(int &i=id[x];i<mp[x].size();i++)
71        {
72            node &e=edge[mp[x][i]];
73            if(dist[x]+1==dist[e.to]&&(f=dfs(e.to,min
74                (a,e.cap-e.flow)))>0)
75            {
76                e.flow+=f;
77                edge[mp[x][i]^1].flow-=f;
78                flow+=f;
79                a-=f;
80                if(!a) break;
81            }
82        }
83        return flow;
84    }
85    type dinic(int _s,int _t)
86    {
87        int i;
88        s=_s;
89        t=_t;
90        type res=0;
91        while(bfs())
92        {

```



```

92     for(i=0;i<=n;i++) id[i]=0;
93     res+=dfs(s,inf);
94 }
95 return res;
96 }
97 type min_flow(int _s,int _t)
98 {
99     int i;
100    s=s1;
101    t=t1;
102    for(i=0;i<s1;i++)
103    {
104        if(in[i]>out[i]) add_edge(s,i,0,in[i]-out[i]);
105        else if(in[i]<out[i]) add_edge(i,t,0,out[i]-in[i]);
106    }
107    add_edge(_t,_s,0,inf);
108    dinic(s,t);
109    for(i=0;i<mp[s].size();i++)
110    {
111        node &e=edge[mp[s][i]];
112        if(e.cap-e.flow!=0) return -1;
113    }
114    s=_s;
115    t=_t;
116    type res;
117    for(i=0;i<mp[t].size();i++)
118    {
119        node &e=edge[mp[t][i]];
120        if(e.to==s)
121        {
122            res=e.flow;
123            e.flow=edge[mp[t][i]^1].flow=0;
124            e.cap=edge[mp[t][i]^1].cap=0;
125        }
126    }
127    res-=dinic(t,s);
128    return res;
129 }
130 type max_flow(int _s,int _t)
131 {
132     int i;
133     s=s1;
134     t=t1;
135     for(i=0;i<s1;i++)
136     {
137         if(in[i]>out[i]) add_edge(s,i,0,in[i]-out[i]);
138         else if(in[i]<out[i]) add_edge(i,t,0,out[i]-in[i]);
139     }
140     add_edge(_t,_s,0,inf);
141     dinic(s,t);

```

```

142     for(i=0;i<mp[s].size();i++)
143     {
144         node &e=edge[mp[s][i]];
145         if(e.cap-e.flow!=0) return -1;
146     }
147     s=_s;
148     t=_t;
149     type res;
150     for(i=0;i<mp[t].size();i++)
151     {
152         node &e=edge[mp[t][i]];
153         if(e.to==s)
154         {
155             res=e.flow;
156             e.flow=edge[mp[t][i]^1].flow=0;
157             e.cap=edge[mp[t][i]^1].cap=0;
158         }
159     }
160     res+=dinic(s,t);
161     return res;
162 }
163 #undef type
164 }dc;
165 /*
166 dc.init(n);
167 dc.add_edge(a,b,lcap,rcap); a,b: 1~n
168 dc.min_flow(s,t);
169 dc.max_flow(s,t);
170 */

```

4.6 费用流

4.6.1 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    {

```

```

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
    =0)
25 {
26     edge.pb(node(from,to,cap,0,cost));
27     edge.pb(node(to,from,0,0,-cost));
28     int m=edge.size();
29     mp[from].pb(m-2);
30     mp[to].pb(m-1);
31 }
32 bool spfa(type& flow,type& cost)
33 {
34     for(int i=0;i<=n;i++)
35     {
36         d[i]=inf;
37         vis[i]=0;
38     }
39     d[s]=0;vis[s]=1;id[s]=0;a[s]=inf;
40     queue<int> q;
41     q.push(s);
42     while(!q.empty())
43     {
44         int x=q.front();
45         q.pop();
46         vis[x]=0;
47         for(int i=0;i<mp[x].size();i++)
48         {
49             node& e=edge[mp[x][i]];
50             int to=e.to;
51             if(e.cap>e.flow&&d[to]>d[x]+e.cost)
52             {
53                 d[to]=d[x]+e.cost;
54                 a[to]=min(a[x],e.cap-e.flow);
55                 id[to]=mp[x][i];
56                 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 */

```

4.6.2 dijkstra 费用流 (dij 求 h)

```

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),flow(f),cost(co){}
13    };
14    int n,s,t,id[N];
15    vector<node> edge;
16    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
        =0)
25    {
26        edge.push_back(node(from,to,cap,cost));
27        edge.push_back(node(to,from,0,-cost));
28        int m=edge.size();
29        mp[from].push_back(m-2);
30        mp[to].push_back(m-1);
31    }

```

```

32 bool dij()
33 {
34     int i,x,to;
35     type cost,now_cost;
36     for(i=0;i<=n;i++) dis[i]=inf;
37     dis[s]=0;id[s]=0;
38     priority_queue<PTI ,vector<PTI>,greater<PTI>
39         > q;
40     q.push({type(0),s});
41     while(!q.empty())
42     {
43         PTI tmp=q.top();
44         q.pop();
45         cost=tmp.first;
46         x=tmp.second;
47         if(cost>dis[x]) continue;
48         for(i=0;i<mp[x].size();i++)
49         {
50             node& e=edge[mp[x][i]];
51             to=e.to;
52             type now_cost=e.cost+h[x]-h[to];
53             if(e.flow>0&&dis[to]>dis[x]+now_cost)
54             {
55                 dis[to]=dis[x]+now_cost;
56                 q.push({dis[to],to});
57                 e.from=x;
58                 id[to]=mp[x][i];
59             }
60         }
61         return dis[t]!=inf;
62     }
63     pair<type,type> mincost_maxflow(int _s,int _t)
64     {
65         int i;
66         type flow=0,cost=0;
67         for(int i=0;i<=n;i++) h[i]=0;
68         s=_s;
69         t=_t;
70         while(dij())
71         {
72             for(i=0;i<=n;i++) h[i]+=dis[i];
73             type new_flow=inf;
74             for(i=t;i!=s;i=edge[id[i]].from)
75             {
76                 new_flow=min(new_flow,edge[id[i]].flow
77                     );
78             }
79             for(i=t;i!=s;i=edge[id[i]].from)
80             {
81                 edge[id[i]].flow-=new_flow;
82                 edge[id[i]^1].flow+=new_flow;
83             }
84             flow+=new_flow;

```

```

84         cost+=new_flow*h[t];
85     }
86     return {cost,flow};
87 }
88 #undef type
89 #undef inf
90 #undef PTI
91 }mcmf; //upper: O(nmlog(nm) + max_flow*mlogm)

```

4.6.3 dijkstra 费用流 (spfa 求 h)

```

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],vis[N];
16    vector<node> edge;
17    vector<int> mp[N];
18    type dis[N],h[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.push_back(node(from,to,cap,cost));
29        edge.push_back(node(to,from,0,-cost));
30        int m=edge.size();
31        mp[from].push_back(m-2);
32        mp[to].push_back(m-1);
33    }
34    void spfa()
35    {
36        int i,x,to;
37        for(i=0;i<=n;i++)
38        {
39            h[i]=inf;
40            vis[i]=0;
41        }
42        queue<int> q;
43        q.push(s);

```

```

42     h[s]=0;
43     vis[s]=1;
44     while(!q.empty())
45     {
46         x=q.front();
47         q.pop();
48         vis[x]=0;
49         for(i=0;i<mp[x].size();i++)
50         {
51             node &e=edge[mp[x][i]];
52             to=e.to;
53             if(e.flow>0&&h[to]>h[x]+e.cost)
54             {
55                 h[to]=h[x]+e.cost;
56                 if(!vis[to])
57                 {
58                     vis[to]=1;
59                     q.push(to);
60                 }
61             }
62         }
63     }
64 }
65 bool dij()
66 {
67     int i,x,to;
68     type cost,now_cost;
69     for(i=0;i<=n;i++) dis[i]=inf;
70     dis[s]=0;id[s]=0;
71     priority_queue<PTI ,vector<PTI>,greater<PTI>
72         > q;
73     q.push({type(0),s});
74     while(!q.empty())
75     {
76         PTI tmp=q.top();
77         q.pop();
78         cost=tmp.first;
79         x=tmp.second;
80         if(cost>dis[x]) continue;
81         for(i=0;i<mp[x].size();i++)
82         {
83             node& e=edge[mp[x][i]];
84             to=e.to;
85             type now_cost=e.cost+h[x]-h[to];
86             if(e.flow>0&&dis[to]>dis[x]+now_cost)
87             {
88                 dis[to]=dis[x]+now_cost;
89                 q.push({dis[to],to});
90                 e.from=x;
91                 id[to]=mp[x][i];
92             }
93         }
94     }
95     return dis[t]!=inf;

```

```

95     }
96     pair<type,type> mincost_maxflow(int _s,int _t)
97     {
98         int i;
99         type flow=0,cost=0;
100        s=_s;
101        t=_t;
102        spfa();
103        while(dij())
104        {
105            for(i=0;i<=n;i++) h[i]+=dis[i];
106            type new_flow=inf;
107            for(i=t;i!=s;i=edge[id[i]].from)
108            {
109                new_flow=min(new_flow,edge[id[i]].flow
110                    );
111            }
112            for(i=t;i!=s;i=edge[id[i]].from)
113            {
114                edge[id[i]].flow-=new_flow;
115                edge[id[i]^1].flow+=new_flow;
116            }
117            flow+=new_flow;
118            cost+=new_flow*h[t];
119        }
120        return {cost,flow};
121    }
122    #undef type
123    #undef inf
124    #undef PTI
125 }mcmf; // upper: O(nm + max_flow*mlogm)

```

4.7 连通性

4.7.1 强连通分量

```

1 struct Strongly_Connected_Components
2 {
3     int scc_cnt,tot;
4     int low[MAX],dfn[MAX],col[MAX],sz[MAX];
5     int st[MAX],top,flag[MAX];
6     vector<int> mp[MAX];
7     void dfs(int x)
8     {
9         int tmp;
10        st[top++]=x;
11        flag[x]=1;
12        low[x]=dfn[x]=++tot;
13        for(auto &to:mp[x])
14        {
15            if(!dfn[to])
16            {
17                dfs(to);
18                low[x]=min(low[x],low[to]);

```

```

19     }
20     else if(flag[to]) low[x]=min(low[x],dfn[
        to]);
21 }
22 if(low[x]==dfn[x])
23 {
24     scc_cnt++;
25     do
26     {
27         tmp=st[--top];
28         flag[tmp]=0;
29         col[tmp]=scc_cnt;
30         sz[scc_cnt]++;
31     }while(tmp!=x);
32 }
33 }
34 void work(int n,vector<int> *_mp)
35 {
36     int i;
37     for(i=1;i<=n;i++)
38     {
39         col[i]=sz[i]=flag[i]=0;
40         mp[i]=_mp[i];
41     }
42     scc_cnt=top=tot=0;
43     for(i=1;i<=n;i++)
44     {
45         if(col[i]) continue;
46         dfs(i);
47     }
48 }
49 void rebuild(int n,vector<int> *g)
50 {
51     int i;
52     for(i=1;i<=n;i++) g[i].clear();
53     for(i=1;i<=n;i++)
54     {
55         for(auto &to:mp[i])
56         {
57             if(col[i]==col[to]) continue;
58             g[col[i]].push_back(col[to]);
59         }
60     }
61 }
62 }scc;

```

4.7.2 边双连通分量

```

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;

```

```

60     }
61     bcc=top=tot=0;
62     bridge.clear();
63     for(int i=1;i<=n;i++)
64     {
65         if(!dfn[i]) dfs(i,i);
66     }
67 }
68 void rebuild(vector<int> e[])
69 {
70     int i,t;
71     for(i=1;i<=n;i++) e[i].clear();
72     for(i=1;i<=n;i++)
73     {
74         t=fa[i];
75         if(belong[i]!=belong[t])
76         {
77             e[belong[i]].pb(belong[t]);
78             e[belong[t]].pb(belong[i]);
79         }
80     }
81 }
82 }

```

4.7.3 圆方树

```

1  #include <cstdio>
2  #include <vector>
3  #include <algorithm>
4
5  const int MN = 100005;
6
7  int N, M, cnt;
8  std::vector<int> G[MN], T[MN * 2];
9
10 int dfn[MN], low[MN], dfc;
11 int stk[MN], tp;
12
13 void Tarjan(int u) {
14     printf(" Enter : %d\n", u);
15     low[u] = dfn[u] = ++dfc; // low 0'j003»~μ±% dfn
16     stk[++tp] = u; // 000g00%
17     for (auto v : G[u]) { // 000± u 0000000μ
18         if (!dfn[v]) { // 000δ00'0 •
19             Tarjan(v); // 00μ
20             low[u] = std::min(low[u], low[v]); //
21             δ00+Í0 • low dmin
22             if (low[v] == dfn[u]) { // 00000h000±%, u
23                 Í0j0+°0,μl • -
24                 ++cnt; // 00r0000%μ
25                 printf(" Found a New BCC %d.\n", cnt
26                     - N);
27                 // 0+0r000%«μ u
28                 j000g00000000μf-²¢ • %l±

```

```

25     for (int x = 0; x != v; --tp) {
26         x = stk[tp];
27         T[cnt].push_back(x);
28         T[x].push_back(cnt);
29         printf(" BCC %d has vertex %d\n",
30             cnt - N, x);
31     }
32     // 000u 0000XÇ000g1±`μ«²»f0
33     T[cnt].push_back(u);
34     T[u].push_back(cnt);
35     printf(" BCC %d has vertex %d\n",
36         cnt - N, u);
37 }
38 else low[u] = std::min(low[u], dfn[v]); //
39     0v00+Í0dfn dmin
40 }
41 printf(" Exit : %d : low = %d\n", u, low[u]);
42 printf(" Stack:\n ");
43 for (int i = 1; i <= tp; ++i) printf("%d, ", stk
44     [i]);
45 puts("");
46 }
47
48 int main() {
49     scanf("%d%d", &N, &M);
50     cnt = N; // 0+μ / 0000 • %μ N '¿ª
51     for (int i = 1; i <= M; ++i) {
52         int u, v;
53         scanf("%d%d", &u, &v);
54         G[u].push_back(v); // 0+000%
55         G[v].push_back(u);
56     }
57     // 000é'1
58     for (int u = 1; u <= N; ++u)
59         if (!dfn[u]) Tarjan(u), --tp;
60     // Tarjan 0g000h00000g¹,´,£-½«
61     return 0;
62 }

```

4.8 团

4.8.1 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 */

```

4.8.2 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    {
22        if(some.none()&&none.none())//one maximal
23            clique
24        {
25            cnt_clique++;
26            return;
27        }
28        bitset<N> r=some;
29        bool fi=1;
30        for(int i=0;i<n;i++)
31        {
32            if(!r[i]) continue;
33            if(fi)
34            {
35                fi=0;

```

```

34         r&=mp[i]^MASK;
35     }
36     now[i]=1;
37     dfs(now,some&mp[i],none&mp[i]);
38     now[i]=0;
39     some[i]=0;
40     none[i]=1;
41 }
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 */

```

4.9 拓扑排序

```

1 vector<int> topsort(vector<int> &node,vector<int>
2   mp[],int *in)
3 {
4     queue<int> q;
5     for(auto &it:node)
6     {
7         if(!in[it]) q.push(it);
8     }
9     vector<int> toplist;
10    while(!q.empty())
11    {
12        int x=q.front();
13        q.pop();
14        toplist.push_back(x);
15        for(auto &to:mp[x])
16        {
17            in[to]--;
18            if(!in[to]) q.push(to);
19        }
20    }
21    return toplist;

```

4.10 2-sat

4.10.1 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 builddag(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 builddag(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 }

```

4.10.2 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 }

```

4.11 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     vector<int> 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(vector<int> *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                 }
56                 else
57                 {
58                     getfa(to);
59                     semi[k]=semi[ckmin(best[to],k)];
60                 }
61             }
62             if(k!=semi[k]) tmp[semi[k]].pb(k);
63             for(auto &to:tmp[k])
64             {
65                 getfa(to);
66                 if(semi[best[to]]==k) idom[to]=k;
67                 else idom[to]=best[to];
68             }
69             for(auto &to:son[k]) fa[to]=k;
70         }
71         for(i=2;i<=tot;i++)
72         {
73             k=id[i];
74             if(idom[k]!=semi[k]) idom[k]=idom[idom[k]];
75             if (k!=idom[k])
76             {
77                 mp[idom[k]].push_back(k); //add edge
78             }
79         }
80     }
81     void work(int rt,vector<int> *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 */

```

4.12 最小斯坦纳树

```

1 struct Minimum_Steiner_Tree
2 {
3     #define type int
4     const type inf=INF;
5     static const int N=;
6     static const int K=;
7     struct node
8     {
9         int id;
10        type v;
11        friend bool operator <(node a,node b){return
12            a.v>b.v;}
13    };
14    vector<node> mp[N];
15    type dp[(1<<K)+3][N];
16    int n,vis[N];
17    void init(int _n)
18    {
19        n=_n;
20        for(int i=1;i<=n;i++) mp[i].clear();
21    }
22    void add_edge(int x,int y,type v){ mp[x].
23        push_back({y,v});}
24    void dijkstra(int s)
25    {
26        int i,to;
27        type w;
28        priority_queue<node> q;
29        for(i=1;i<=n;i++)
30        {
31            vis[i]=0;
32            if(dp[s][i]!=inf) q.push({i,dp[s][i]});
33        }
34        while(!q.empty())
35        {
36            node t=q.top();
37            q.pop();
38            if(vis[t.id]) continue;
39            vis[t.id]=1;
40            for(auto &t:mp[t.id])

```

```

39        {
40            to=t.id;
41            w=t.v;
42            if(dp[s][to]>dp[s][t.id]+w)
43            {
44                dp[s][to]=dp[s][t.id]+w;
45                if(!vis[to]) q.push({to,dp[s][to]});
46            }
47        }
48    }
49 }
50 type work(vector<int> key_node)
51 {
52     int s,t,i,k;
53     type res;
54     k=key_node.size();
55     for(s=0;s<(1<<k);s++)
56     {
57         for(i=1;i<=n;i++) dp[s][i]=inf;
58     }
59     for(i=0;i<k;i++) dp[(1<<i)][key_node[i]]=0;
60     for(s=0;s<(1<<k);s++)
61     {
62         for(t=s&(s-1);t=t&(t-1))
63         {
64             if(t<(s^t)) break;
65             for(i=1;i<=n;i++)
66             {
67                 dp[s][i]=min(dp[s][i],dp[t][i]+dp[s
68                     ^t][i]);
69             }
70             dijkstra(s);
71         }
72         res=inf;
73         for(i=1;i<=n;i++) res=min(res,dp[(1<<k)-1][i
74             ]);
75         return res;
76     }
77     #undef type
78 }stn;
79 /*
80 minimum spanning tree including all k key_node
81 O(n*3^k + m*log(m)*2^k)
82 stn.init(n);
83 stn.add_edge(a,b,w);
84 stn.work(key_node);
85 */

```

5 数论

5.1 素数筛

5.1.1 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 }
```

5.1.2 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     memset(prime,0,sizeof prime);
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 }
```

5.1.3 区间筛

```

1 //O(r-l+1)
2 const int N=2e7+10;
3 ll p[N],tot;
4 bool vis[N],prime[N];
5 void init(ll l,ll r)
```

```

6 {
7     ll i,j,sq=sqrt(r+0.5);
8     tot=0;
9     for(i=0;i<=sq;i++) vis[i]=1;
10    for(i=1;i<=r;i++) prime[i-1]=1;
11    if(l==0) prime[0]=prime[1]=0;
12    if(l==1) prime[0]=0;
13    for(i=2;i<=sq;i++)
14        {
15            if(!vis[i]) continue;
16            for(j=i+i;j<=sq;j+=i) vis[j]=0;
17            for(j=max(2LL,(l+i-1)/i)*i;j<=r;j+=i) prime[j-1]=0;
18        }
19    for(i=1;i<=r;i++)
20        {
21            if(prime[i-1]) p[tot++]=i;
22        }
23 }
```

5.2 逆元

5.2.1 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 }
```

5.2.2 线性预处理

```

1 void getinv(int n,ll p)
2 {
3     ll i;
4     inv[1]=1;
5     for(i=2;i<=n;i++) inv[i]=(p-p/i)*inv[p%i]%p;
6 }
```

5.3 扩展欧几里得

5.3.1 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 }

```

5.3.2 $ax+by=c$

```

1 /*
2 xa+yb=c
3 x,y
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
        solution
12    else if(!a&&b) x=1,y=c/b;
13    else if(a&&!b) x=c/a,y=-c/a;
14    else
15    {
16        a/=g,b/=g,c/=g;
17        x*=c,y*=c;
18        t=x;
19        x%=b;
20        if(x<=0) x+=b;//or x<0
21        ll k=(t-x)/b;
22        y+=k*a;
23    }
24    return g;
25 }

```

5.4 中国剩余定理

5.4.1 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 }

```

5.4.2 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 };

```

5.5 组合数

5.5.1 打表

```

1 ll C[1010][1010];
2 void init(int n)
3 {
4     int i,j;
5     C[0][0]=1;
6     for(i=1;i<=n;i++)

```

```

7   {
8       C[i][0]=1;
9       for(j=1;j<=n;j++)
10      {
11          C[i][j]=(C[i-1][j]+C[i-1][j-1])%mod;
12      }
13  }
14 }

```

5.5.2 预处理

```

1  ll qpow(ll a,ll b)
2  {
3      ll res=1;
4      while(b>0)
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 qpow(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
20         +1)%mod;
21 }
22 ll C(ll n,ll m)
23 {
24     if(m>n||m<0||n<0) return 0;
25     return fac[n]*invfac[m]%mod*invfac[n-m]%mod;
26 }
27 ll A(ll n,ll m)
28 {
29     if(m>n||m<0||n<0) return 0;
30     return fac[n]*invfac[n-m]%mod;

```

5.5.3 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  }

```

5.5.4 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]*
50                         j%b[i];
51                     else fac_tmp[j]=fac_tmp[j-1];

```

```

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 ll multilucas(ll n,ll m,ll x,ll p)
67 {
68     if(m>n) return 0;
69     ll i,cnt;
70     cnt=0;
71     for(i=n;i/=x) cnt+=i/x;
72     for(i=m;i/=x) cnt-=i/x;
73     for(i=n-m;i/=x) cnt-=i/x;
74     return pow2(x,cnt,p)* \
75         cal_fac(n,x,p)%p* \
76         inv(cal_fac(m,x,p),p)%p* \
77         inv(cal_fac(n-m,x,p),p)%p;
78 }
79 ll C(ll n,ll m,ll p)
80 {
81     if(m>n||m<0||n<0) return 0;
82     ll i,res;
83     VL a,b,resa;
84     assert(mp.count(p));
85     a=mp[p].fi;
86     b=mp[p].se;
87     for(i=0;i<sz(a);i++) resa.pb(multilucas(n,m,a
88         [i],b[i]));
89     res=exCRT::excrt(resa,b);
90     assert(res!=-1);
91     return res;
92 }
93 };//exLucas::init(VL{});

```

5.6 欧拉函数

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

5.6.1 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 }

```

5.6.2 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    }
29 }

```

5.7 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  }

```

5.8 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
9          >=0); for(;b>=1){if(b&1)res=res*a%mod;a=a
10          *a%mod;}return res;}
11      const int N=10010;
12      ll res[N],base[N],_c[N],_md[N];
13      vector<int> Md;
14      void mul(ll *a,ll *b,int k)
15      {
16          rep(i,0,k+k) _c[i]=0;
17          rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i
18              +j]+a[i]*b[j])%mod;
19          for (int i=k+k-1;i>=k;i--) if (_c[i])
20              rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[
21                  j]]-_c[i]*_md[Md[j]])%mod;
22          rep(i,0,k) a[i]=_c[i];
23      }
24      int solve(ll n,VI a,VI b){
25          ll ans=0,pnt=0;
26          int k=SZ(a);
27          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[
34                  i];res[0]=0;
35              rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]
36                  ]-res[k]*_md[Md[j]])%mod;
37          }
38      }
39      rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
40      if (ans<0) ans+=mod;
41      return ans;
42  }
43  VI BM(VI s){
44      VI C(1,1),B(1,1);
45      int L=0,m=1,b=1;
46      rep(n,0,SZ(s)){
47          ll d=0;
48          rep(i,0,L+1) d=(d+(1ll)C[i]*s[n-i])%mod;
49          if(d==0) ++m;
50          else if(2*L<n){
51              VI T=C;
52              ll c=mod-d*powmod(b,mod-2)%mod;
53              while (SZ(C)<SZ(B)+m) C.pb(0);
54              rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%
55                  mod;
56              L=n+1-L; B=T; b=d; m=1;
57          } else {
58              ll c=mod-d*powmod(b,mod-2)%mod;
59              while (SZ(C)<SZ(B)+m) C.pb(0);
60              rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%
61                  mod;
62              ++m;
63          }
64      }
65      return C;
66  }
67  int gao(VI a,ll n)
68  {
69      VI c=BM(a);
70      c.erase(c.begin());
71      rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
72      return solve(n,c,VI(a.begin(),a.begin()+SZ(c)
73          ));
74  }
75  };//linear_seq::gao(VI{ },n-1)

```


5.9 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 }

```

5.10 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 }

```

5.11 第二类 Stirling 数

```

1 //dp[i][j]表示j个元素划分到i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 }

```

5.12 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 }

```

5.13 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;

```

6 多项式

6.1 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(){}
10        cp(double _x,double _y){a=_x,b=_y;}
11        cp operator +(const cp &o)const{return (cp){a
12            +o.a,b+o.b};}
13        cp operator -(const cp &o)const{return (cp){a
14            -o.a,b-o.b};}
15        cp operator *(const cp &o)const{return (cp){a
16            *o.a-b*o.b,b*o.a+a*o.b};}
17        cp operator *(const double &o)const{return (
18            cp){a*o,b*o};}

```

```

15     cp operator !()const{return (cp){a,-b};}
16 }x[maxn],y[maxn],z[maxn],w[maxn];
17 void fft(cp x[],int k,int v)
18 {
19     int i,j,l;
20     for(i=0,j=0;i<k;i++)
21     {
22         if(i>j)swap(x[i],x[j]);
23         for(l=k>>1;(j^=1)<l;l>>=1);
24     }
25     w[0]=(cp){1,0};
26     for(i=2;i<=k;i<=1)
27     {
28         cp g=(cp){cos(2*pi/i),(v?-1:1)*sin(2*pi/i
29             )});
30         for(j=(i>>1);j>=0;j-=2)w[j]=w[j>>1];
31         for(j=1;j<i>>1;j+=2)w[j]=w[j-1]*g;
32         for(j=0;j<k;j+=i)
33         {
34             cp *a=x+j,*b=a+(i>>1);
35             for(l=0;l<i>>1;l++)
36             {
37                 cp o=b[l]*w[l];
38                 b[l]=a[l]-o;
39                 a[l]=a[l]+o;
40             }
41         }
42         if(v)for(i=0;i<k;i++)x[i]=(cp){x[i].a/k,x[i].
43             b/k};
44     }
45     // a=b*c
46     // b[0..11]
47     // c[0..12]
48 void mul(int *a,int *b,int *c,int l1,int l2)
49 {
50     if(l1<128&&l2<128)
51     {
52         rep(i,0,l1+l2)a[i]=0;
53         rep(i,0,l1)rep(j,0,l2)a[i+j]+=b[i]*c[j];
54         return;
55     }
56     int K;
57     for(K=1;K<=l1+l2;K<=1);
58     rep(i,0,l1)x[i]=cp(b[i],0);
59     rep(i,0,l2)y[i]=cp(c[i],0);
60     rep(i,l1+1,K)x[i]=cp(0,0);
61     rep(i,l2+1,K)y[i]=cp(0,0);
62     fft(x,K,0);fft(y,K,0);
63     rep(i,0,K)z[i]=x[i]*y[i];
64     fft(z,K,1);
65     rep(i,0,l1+l2)a[i]=(ll)(z[i].a+0.5);
66 }

```

```

67 };

```

6.2 NTT

```

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>0)
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     assert(p==mod);
20     for(int i=0;i<25;i++) wn[i]=pow2(g,(p-1)/(1ll
21         <<i));
22 }
23 void ntt(vector<int> &a,int len,int f)
24 {
25     int i,j=0,t,k,w,id;
26     for(i=1;i<len-1;i++)
27     {
28         for(t=len;j^=t>>=1,~j&t;);
29         if(i<j) swap(a[i],a[j]);
30     }
31     for(i=1,id=1;i<len;i<=1,id++)
32     {
33         t=i<<1;
34         for(j=0;j<len;j+=t)
35         {
36             for(k=0,w=1;k<i;k++,w=1ll*w*wn[id]%p)
37             {
38                 int x=a[j+k],y=1ll*w*a[j+k+i]%p;
39                 a[j+k]=x+y;
40                 if(a[j+k]>=p) a[j+k]-=p;
41                 a[j+k+i]=x-y;
42                 if(a[j+k+i]<0) a[j+k+i]+=p;
43             }
44         }
45     }
46     if(f)
47     {
48         for(i=1,j=len-1;i<j;i++,j--) swap(a[i],a[
49             j]);
50     }
51 }

```

```

48     int inv=pow2(len,p-2);
49     for(i=0;i<len;i++) a[i]=1ll*a[i]*inv%p;
50 }
51 }
52 vector<int> qpow(vector<int> a,int b)//limt: sz(
53     a)*b is small
54 {
55     int len,i,l1;
56     l1=a.size();
57     for(len=1;len<=(l1+1)*b-1;len<=1);
58     a.resize(len,0);
59     ntt(a,len,0);
60     vector<int> res(len);
61     for(i=0;i<len;i++) res[i]=pow2(a[i],b);
62     ntt(res,len,1);
63     res.resize((l1+1)*b-1);
64     return res;
65 }
66 vector<int> mul(vector<int> a,vector<int> b)
67 {
68     int len,i,l1,l2;
69     l1=a.size();
70     l2=b.size();
71     for(len=1;len<l1+l2;len<=1);
72     a.resize(len,0);
73     b.resize(len,0);
74     ntt(a,len,0);ntt(b,len,0);
75     vector<int> res(len);
76     for(i=0;i<len;i++) res[i]=1ll*a[i]*b[i]%p;
77     ntt(res,len,1);
78     res.resize(l1+l2-1);
79     return res;
80 }
81 //get kth
82 vector<int> merge_generating_functions(vector<
83     vector<int>> &dp,int k)
84 {
85     int i,j;
86     priority_queue<pair<int,int>> q;
87     for(i=0;i<dp.size();i++) q.push({-dp[i].size
88         (),i});
89     while(q.size()>1)
90     {
91         i=q.top().second;
92         q.pop();
93         j=q.top().second;
94         q.pop();
95         dp[i]=mul(dp[i],dp[j]);
96         if(dp[i].size()>k) dp[i].resize(k+1);
97         q.push({-dp[i].size(),i});
98     }
99     return dp[q.top().second];
100 }

```

```

99 };//NTT::getwn();

```

6.3 FWT

```

1 namespace FWT
2 {
3     const int p=998244353;
4     const ll inv2=(p+1)/2;
5     void fwt(vector<int> &a,int n,int f,int v)
6     {
7         for(int d=1;d<n;d<=1)
8         {
9             for(int m=d<<1,i=0;i<n;i+=m)
10             {
11                 for(int j=0;j<d;j++)
12                 {
13                     ll x=a[i+j],y=a[i+j+d];
14                     if(!v)
15                     {
16                         if(f==1) a[i+j]=(x+y)%p,a[i+j+d]
17                             =(x-y+p)%p;//xor
18                         else if(f==2) a[i+j]=(x+y)%p;//
19                             and
20                         else if(f==3) a[i+j+d]=(x+y)%p;
21                             //or
22                     }
23                     else
24                     {
25                         if(f==1) a[i+j]=(x+y)*inv2%p,a[
26                             i+j+d]=(x-y+p)%p*inv2%p;//
27                             xor
28                         else if(f==2) a[i+j]=(x-y+p)%p;
29                             //and
30                         else if(f==3) a[i+j+d]=(y-x+p)%
31                             p;//or
32                     }
33                 }
34             }
35         }
36     }
37 }
38
39 vector<int> XOR(vector<int> a,vector<int> b)
40 {
41     int n,len;
42     n=a.size();
43     for(len=1;len<n;len<=1);
44     a.resize(len,0);
45     b.resize(len,0);
46     fwt(a,len,1,0);
47     fwt(b,len,1,0);
48     for(int i=0;i<len;i++) a[i]=1ll*a[i]*b[i]%p;
49     fwt(a,len,1,1);
50     return a;
51 }

```

```

43 }
44 vector<int> AND(vector<int> a,vector<int> b)
45 {
46     int n,len;
47     n=a.size();
48     for(len=1;len<n;len<=&=1);
49     a.resize(len,0);
50     b.resize(len,0);
51     fwt(a,len,2,0);
52     fwt(b,len,2,0);
53     for(int i=0;i<len;i++) a[i]=1ll*a[i]*b[i]%p;
54     fwt(a,len,2,1);
55     return a;
56 }
57 vector<int> OR(vector<int> a,vector<int> b)
58 {
59     int n,len;
60     n=a.size();
61     for(len=1;len<n;len<=&=1);
62     a.resize(len,0);
63     b.resize(len,0);
64     fwt(a,len,3,0);
65     fwt(b,len,3,0);
66     for(int i=0;i<len;i++) a[i]=1ll*a[i]*b[i]%p;
67     fwt(a,len,3,1);
68     return a;
69 }
70 };

```

6.4 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
        [D][2],C[D];
6     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;}
7     ll calcn(int d,ll *a,ll n) { // a[0].. a[d] a[n]
8         if (n<=d) return a[n];
9         p1[0]=p2[0]=1;
10        rep(i,0,d+1) {
11            ll t=(n-i+mod)%mod;
12            p1[i+1]=p1[i]*t%mod;
13        }
14        rep(i,0,d+1) {
15            ll t=(n-d+i+mod)%mod;
16            p2[i+1]=p2[i]*t%mod;
17        }
18        ll ans=0;
19        rep(i,0,d+1) {

```

```

20            ll t=g[i]*g[d-i]%mod*p1[i]%mod*p2[d-i]%
                mod*a[i]%mod;
21            if ((d-i)&1) ans=(ans-t+mod)%mod;
22            else ans=(ans+t)%mod;
23        }
24        return ans;
25    }
26    void init(int M) {
27        f[0]=f[1]=g[0]=g[1]=1;
28        rep(i,2,M+5) f[i]=f[i-1]*i%mod;
29        g[M+4]=powmod(f[M+4],mod-2);
30        per(i,1,M+4) g[i]=g[i+1]*(i+1)%mod;
31    }
32    ll polysum(ll n,ll *a,ll m) { // a[0].. a[m] \
        sum_{i=0}^{n-1} a[i]
33        rep(i,0,m+1) tmp[i]=a[i];
34        tmp[m+1]=calcn(m,tmp,m+1);
35        rep(i,1,m+2) tmp[i]=(tmp[i-1]+tmp[i])%mod;
36        return calcn(m+1,tmp,n-1);
37    }
38    ll qpolysum(ll R,ll n,ll *a,ll m) { // a[0].. a[
        m] \sum_{i=0}^{n-1} a[i]*R^i
39        if (R==1) return polysum(n,a,m);
40        a[m+1]=calcn(m,a,m+1);
41        ll r=powmod(R,mod-2),p3=0,p4=0,c,ans;
42        h[0][0]=0;h[0][1]=1;
43        rep(i,1,m+2) {
44            h[i][0]=(h[i-1][0]+a[i-1])*r%mod;
45            h[i][1]=h[i-1][1]*r%mod;
46        }
47        rep(i,0,m+2) {
48            ll t=g[i]*g[m+1-i]%mod;
49            if (i&1) p3=((p3-h[i][0]*t)%mod+mod)%mod,
                p4=((p4-h[i][1]*t)%mod+mod)%mod;
50            else p3=(p3+h[i][0]*t)%mod,p4=(p4+h[i]
                ][1]*t)%mod;
51        }
52        c=powmod(p4,mod-2)*(mod-p3)%mod;
53        rep(i,0,m+2) h[i][0]=(h[i][0]+h[i][1]*c)%mod;
54        rep(i,0,m+2) C[i]=h[i][0];
55        ans=(calcn(m,C,n)*powmod(R,n)-c)%mod;
56        if (ans<0) ans+=mod;
57        return ans;
58    }
59 } // polysum::init();

```

7 矩阵

7.1 矩阵类

```

1 const ll mod=1e9+7;
2 struct Matrix
3 {

```

```

4 static const int N=;
5 int n;
6 ll c[N][N];
7 Matrix(){}
8 Matrix(int _n,ll v=0)
9 {
10     int i,j;
11     n=_n;
12     for(i=0;i<n;i++)
13     {
14         for(j=0;j<n;j++)
15         {
16             c[i][j]=v;
17         }
18     }
19 }
20 void init_identity_matrix() {for(int i=0;i<n;i
    ++)) c[i][i]=1;}
21 Matrix operator *(const Matrix &b)const
22 {
23     int i,j,k;
24     Matrix res(n);
25     for(k=0;k<n;k++)
26     {
27         for(i=0;i<n;i++)
28         {
29             if(!c[i][k]) continue;
30             for(j=0;j<n;j++)
31             {
32                 res.c[i][j]+=c[i][k]*b.c[k][j];
33                 if(res.c[i][j]>=mod) res.c[i][j]%=
34                     mod;
35             }
36         }
37     }
38     return res;
39 };
40 Matrix matpow2(Matrix a,ll b)
41 {
42     Matrix res(a.n);
43     res.init_identity_matrix();
44     while(b)
45     {
46         if(b&1) res=res*a;
47         a=a*a;
48         b>>=1;
49     }
50     return res;
51 }

```

7.2 高斯消元

7.2.1 浮点数方程组

```

1 struct Gauss
2 {
3     const double eps=1e-7;
4     double mp[][];
5     int gauss(int n,int m)
6     {
7         int i,j,k,pos,r;
8         double tmp;
9         r=0;
10        for(k=1;k<=m;k++)
11        {
12            pos=r+1;
13            if(pos>n) return -1; // no solution
14            for(i=pos+1;i<=n;i++)
15            {
16                if(fabs(mp[i][k])>fabs(mp[pos][k]))
17                    pos=i;
18            }
19            if(fabs(mp[pos][k])<eps) continue;
20            r++;
21            swap(mp[pos],mp[r]);
22            tmp=mp[r][k];
23            for(j=k;j<=m+1;j++) mp[r][j]/=tmp;
24            for(i=r+1;i<=n;i++)
25            {
26                tmp=mp[i][k];
27                for(j=k;j<=m+1;j++)
28                {
29                    mp[i][j]-=mp[r][j]*tmp;
30                }
31            }
32            return r;
33        }
34        int work(int n,int m,double *res)
35        {
36            int i,j,cnt;
37            cnt=gauss(n,m);
38            if(cnt==-1) return -1;
39            for(i=cnt+1;i<=n;i++)
40            {
41                if(fabs(mp[i][m+1])>eps)
42                {
43                    // no solution
44                    return -1;
45                }
46            }
47            if(cnt<m)
48            {
49                // multi solution
50                return 1;

```

```

51     }
52     res[m]=mp[m][m+1];
53     for(i=m-1;i>=1;i--)
54     {
55         res[i]=mp[i][m+1];
56         for(j=i+1;j<=m;j++)
57         {
58             res[i]-=mp[i][j]*res[j];
59         }
60     }
61     return 0;
62 }
63 }gs;
64 /*
65 (mp[1][1]*x1) + (mp[1][2]*x2) + ... + (mp[1][m]*xm)
66     = mp[1][m+1]
67 (mp[2][1]*x1) + (mp[1][2]*x2) + ... + (mp[2][m]*xm)
68     = mp[2][m+1]
69 ...
70 (mp[n][1]*x1) + (mp[n][2]*x2) + ... + (mp[n][m]*xm)
71     = mp[n][m+1]
72
73 x x x x | x
74 0 x x x | x
75 0 0 x x | x
76 0 0 0 x | x
77
78 O(n*m^2) m<=n
79
80 gs.work(n,m,res); mp[1..n][1..m+1], res[1..m]
81
82 -1: no solution
83 0 : one solution
84 1 : multi solution
85 */

```

7.2.2 异或方程组

```

1 struct Gauss
2 {
3     bitset<> mp[];
4     int gauss_jordan(int n,int m)
5     {
6         int i,j,k,pos,r;
7         r=0;
8         for(k=1;k<=m;k++)
9         {
10             pos=r+1;
11             if(pos>n) return -1; // no solution
12             while(pos<n&&!mp[pos][k]) pos++;
13             if(!mp[pos][k]) continue;
14             r++;
15             swap(mp[pos],mp[r]);
16             for(i=1;i<=n;i++)

```

```

17         {
18             if(i!=r&&mp[i][k]) mp[i]^=mp[r];
19         }
20     }
21     return r;
22 }
23 int work(int n,int m,int *res)
24 {
25     int i,j,cnt;
26     cnt=gauss_jordan(n,m);
27     if(cnt==-1) return -1;
28     for(i=cnt+1;i<=n;i++)
29     {
30         if(mp[i][m+1])
31         {
32             // no solution
33             return -1;
34         }
35     }
36     if(cnt<m)
37     {
38         // multi solution
39         return 1;
40     }
41     for(i=1;i<=m;i++) res[i]=mp[i][m+1];
42     return 0;
43 }
44 }gs;
45 /*
46 (mp[1][1]*x1) xor (mp[1][2]*x2) xor ... xor (mp[1][
47     m]*xm) = mp[1][m+1]
48 (mp[2][1]*x1) xor (mp[1][2]*x2) xor ... xor (mp[2][
49     m]*xm) = mp[2][m+1]
50 ...
51 (mp[n][1]*x1) xor (mp[n][2]*x2) xor ... xor (mp[n][
52     m]*xm) = mp[n][m+1]
53
54 a:0/1 x:0/1
55
56 x 0 0 0 | x
57 0 x 0 0 | x
58 0 0 x 0 | x
59 0 0 0 x | x
60
61 O((n*m^2)/64)
62
63 gs.work(n,m,res); mp[1..n][1..m+1], res[1..m]
64
65 -1: no solution
66 0 : one solution
67 1 : multi solution
68 */

```


7.3 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(
        sz(b)),n(sz(c)),N(n+1),B(m),D(m+2,VD(n+2))
12    {
13        int i,j;
14        for(i=0;i<m;i++)
15        {
16            for(j=0;j<n;j++)
17            {
18                D[i][j]=A[i][j];
19            }
20        }
21        for(i=0;i<m;i++)
22        {
23            B[i]=n+i;
24            D[i][n]=-1;
25            D[i][n+1]=b[i];
26        }
27        for(j=0;j<n;j++)
28        {
29            N[j]=j;
30            D[m][j]=-c[j];
31        }
32        N[n]=-1;
33        D[m+1][n]=1;
34    }
35    void Pivot(int r,int s)
36    {
37        int i,j;
38        for(i=0;i<m+2;i++)
39        {
40            if(i==r) continue;
41            for(j=0;j<n+2;j++)
42            {
43                if(j==s) continue;
44                D[i][j]-=D[r][j]*D[i][s]/D[r][s];
45            }
46        }
47        for(j=0;j<n+2;j++)
48        {
49            if (j!=s) D[r][j]/=D[r][s];
50        }
51        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]
101            == D[i][s] && N[j] < N[s]) s = j;
102     }
103     Pivot(i,s);
104 }
105 if(!simplex(2)) return numeric_limits<db>::
106     infinity();//solution is INF
107 res = VD(n);
108 for(i=0; i<m; i++)
109 {
110     if(B[i] < n) res[B[i]] = D[i][n+1];
111 }
112 return D[m][n+1];
113 }
114 };
115 /*
116 sum(A[i]*res[i]) <= B, res[i] >= 0
117 MAX(sum(C[i]*res[i]))
118 */

```

7.4 线性基

7.4.1 线性基

```

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

```

```

30    }
31    return res;
32 }
33 void merge(Base x)
34 {
35     for(int i=mx; ~i; i--)
36     {
37         if(x.d[i]) insert(x.d[i]);
38     }
39 }
40 #undef type
41 }bs;

```

7.4.2 带删除线性基

```

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 };

```

8 博弈

8.1 sg 函数

```

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

```

8.2 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。

3.k-nim

问题描述: 给定 n 堆石子, 每堆 a_i 个, 每次可以从最多 k 堆中拿走任意个, 先拿完的人胜利

结论: 先手必败, 当且仅当将每个 a_i 写成二进制, 对每一个二进制位, 这一位为 1 的 i 的个数为 s , $s \bmod (k+1) = 0$

4. 树上删边博弈

结论: 叶子 sg 为 0, 非叶子 sg 为所有子节点的 $sg+1$ 的异或和

9 dp

9.1 LIS

```

1 struct LIS
2 {
3     #define type int
4     const type inf=INF;
5     vector<int> work(vector<type> a,bool strict)
6     {
7         int i,pos,len,n;
8         n=a.size();
9         vector<type> b(n,inf);
10        vector<int> tmp(n),res;
11        for(i=0;i<n;i++)
12        {
13            // strict: lower_bound
14            //not strict: upper_bound
15            if(strict) pos=lower_bound(b.begin(),b.
16                end(),a[i])-b.begin();
17            else pos=upper_bound(b.begin(),b.end(),a[
18                i])-b.begin();
19            b[pos]=a[i];
20            tmp[i]=pos;
21        }
22        len=lower_bound(b.begin(),b.end(),inf)-b.
23            begin();
24        for(i=n-1;~i;i--)
25        {
26            if(!len) break;
27            if(tmp[i]+1==len)
28            {
29                len--;
30                res.push_back(i);
31            }
32        }
33        return res;
34    }
35    #undef type

```

```

33 }lis;

```

9.2 LPS

```

1 int dp[105][105];
2 void LPS(char *s,int n) // s[1..n]
3 {
4     int i,len,l,r;
5     memset(dp,0,sizeof dp);
6     for(i=1;i<=n;i++) dp[i][i]=1;
7     for(len=2;len<=n;len++)
8     {
9         for(l=1;l+len-1<=n;l++)
10        {
11            r=l+len-1;
12            if(s[l]==s[r]) dp[l][r]=dp[l+1][r-1]+2;
13            else dp[l][r]=max(dp[l+1][r],dp[l][r-1]);
14        }
15    }
16 }

```

9.3 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 limit)->ll
16    {
17        if(pos==-1) return ;
18        if(!limit&&!lead&&dp[pos][sta]!=-1) return dp[
19            pos][sta];
20        ll res=0;
21        for(int i=(limit?p[pos]:base-1);~i;i--)
22        {
23            res+=dfs(pos-1,lead&&i==0&&pos,,limit&&i==
24                p[pos]);
25        }
26        if(!limit&&!lead) dp[pos][sta]=res;
27        return res;
28    };
29    return dfs(tot-1,1,0,1);
30 }

```

```

43     for(; !blank(ch)&&!IOerror;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(IOerror){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         {1,10,100,1000,10000,100000,1000000,10000000,100000000,1000000000,10000000000,100000000000,1000000000000,10000000000000,100000000000000,1000000000000000,10000000000000000,100000000000000000,1000000000000000000,10000000000000000000};
68         ,1000000000000000000LL,1000000000000000000LL,
69         1000000000000000000LL,100000000000000000LL
        ,100000000000000000LL
        ,100000000000000000LL};
70     if(x<-1e-12)out('-'),x=-x;
71     ll x2=(ll)floor(x);if(!y&&x-x2>=0.5)++x2;
        x-=x2;x*=mul[y];
72     ll x3=(ll)floor(x);if(y&&x-x3>=0.5)++x3;
        print(x2);
73     if(y>0){out('.');for(size_t i=1;i<y&&x3*
        mul[i]<mul[y];out('0'),++i);print(x3)
        ;}
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;

```

10.2 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 }

```

10.3 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);

```

10.4 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 }

```

10.5 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 }

```

10.6 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 }

```

10.7 整数三分

```

1 while(l<r)
2 {
3     lm=l+(r-l)/3;
4     rm=r-(r-l)/3;
5     tl=cal(lm);
6     tr=cal(rm);
7     // tl>tr:min tl<tr:max
8     if(tl>tr) l=lm+1;
9     else r=rm-1;
10 }

```

10.8 有根树与 prufer 序列的转换

```

1 //TODO

```

10.9 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     return 0;
41 }

```

10.10 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++)
65     {
66         q[i]=Point(Rand(0,lx),Rand(0,ly));
67         dis[i]=getres(q[i],p,n);
68     }
69     double tmax=max(lx,ly);
70     double tmin=1e-3;
71     // int cnt计算外层循环次数=0;
72     while(tmax>tmin)
73     {
74         for(i=1;i<=c1;i++)
75         {
76             for(j=1;j<=c2;j++)
77             {

```

```

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 }

```

10.11 斐波那契 01 串的第 k 个字符

```

1 int fib[MAX],tot;
2 void init_fib(ll limt)
3 {
4     assert(limt>=0);
5     tot=1;
6     fib[0]=fib[1]=1;
7     for(int i=2;;i++)
8     {
9         if(fib[i-1]>limt-fib[i-2])
10        {
11            tot=i-1;
12            break;
13        }
14        fib[i]=fib[i-1]+fib[i-2];
15    }
16 }
17 //S(0)="0", S(1)="1", S(i)=S(i-1)+S(i-2)
18 //FibString: S(i) [i>=1]
19 int get_kth_FibString(ll k)
20 {
21     int i;

```



```

22     for(i=tot;i>=2;i--)
23     {
24         if(k>fib[i]) k-=fib[i];
25     }
26     return k==1;
27 }
28 int is_fib(ll x)
29 {
30     for(int i=1;i<=tot;i++)
31     {
32         if(x==fib[i]) return 1;
33     }
34     return 0;
35 }

```

10.12 光速幂

```

1 struct light_speed_pow
2 {
3     #define type int
4     int n,sq;
5     type res[MAX][2],val;
6     void init(int _n,type _val)
7     {
8         n=_n;
9         val=_val;
10        sq=sqrt(n)+1;
11        res[0][0]=res[0][1]=1;
12        for(int i=1;i<=sq;i++) res[i][0]=1ll*res[i-1][0]*val%mod;
13        for(int i=1;i<=sq;i++) res[i][1]=1ll*res[i-1][1]*res[sq][0]%mod;
14    }
15    type qpow(int exp)
16    {
17        if(exp<=sq) return res[exp][0];
18        return 1ll*res[exp/sq][1]*res[exp-exp/sq*sq][0]%mod;
19    }
20    #undef type
21 }lsp;
22 /*
23 val^exp
24 O(sqrt exp)-O(1)
25 */

```

10.13 倍增求等比数列和

```

1 ll cal(ll a0,ll q,ll n,ll p)
2 {
3     int i;
4     ll tmp[33],sum[33],res,now;
5     tmp[0]=0;

```

```

6     tmp[1]=q;
7     for(i=2;i<=30;i++) tmp[i]=tmp[i-1]*tmp[i-1]%p;
8     sum[0]=1;
9     for(i=1;i<=30;i++) sum[i]=sum[i-1]*(1+tmp[i])%p;
10    res=0;
11    now=1;
12    for(i=30;~i;i--)
13    {
14        if((n>>i)&1)
15        {
16            res=(res+now*sum[i])%p;
17            now=(now*tmp[i+1])%p;
18        }
19    }
20    return a0*res%p;
21 }

```

11 附录

11.1 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

11.2 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     }
```