ACM-ICPC Template

tokitsukaze

2024 年 11 月 2 日

目录

1 字符串

1.1 KMP

1.1.1 KMP

```
struct KMP
{
    #define type char
    int nex[MAX],len;
    type t[MAX];
    void get_next(type *s,int n)
    {
        int i,j;
        len=n;
        for(i=1;i<=len;i++) t[i]=s[i];
        t[len+1]=0;
        nex[0]=nex[1]=0;
        j=0;
        for(i=2;i<=len;i++)
        {
            while(j&&t[j+1]!=s[i]) j=nex[j];
            if(t[j+1]==s[i]) j++;
            nex[i]=j;
        }
    }
    vector<int> match(type *s,int n)
    {
        int i,j;
        vector<int> res;
        for(i=1,j=0;i<=n;i++)
        {
            while(j&&t[j+1]!=s[i]) j=nex[j];
            if(t[j+1]==s[i]) j++;
            if(j==len)
            {
                res.push_back(i-len+1);
                j=nex[j];
            }
        }
        return res;
    }
    #undef type
}kmp;
/*
kmp.get_next(t,len); // t[1..len]
kmp.match(s,n); // s[1..n] return all pos t in s
*/
```

1.1.2 exKMP

```
struct Z_Algorithm
{
    char s[MAX];
    int n,z[MAX],ex[MAX];
    void get_z_func(char *_s,int _n) // s[0..n-1]
    {
        int i,j,l,r;
        n=_n;
        memcpy(s,_s,n);
        z[0]=l=r=0;
        for(i=1;i<n;i++)
        {
            if(i+z[i-l]-1<r) z[i]=z[i-l];
            else
            {
                j=max(0,r-i+1);
                while(i+j<n&&s[i+j]==s[j]) j++;
                z[i]=j;
                if(i+z[i]-1>r)
                {
                    l=i;
                    r=i+z[i]-1;
                }
            }
        }
        z[0]=n;
    }
    void get_ex(char *t,int m) // t[0..m-1]
    {
        int i,j,l,r;
        j=l=0;
        while(j<n&&j<m&&t[j]==s[j]) j++;
        ex[0]=j;
        r=l+ex[0]-1;
        for(i=1;i<m;i++)
        {
            if(i+z[i-l]-1<r) ex[i]=z[i-l];
            else
            {
                j=max(0,r-i+1);
                while(i+j<m&&t[i+j]==s[j]) j++;
                ex[i]=j;
                if(i+ex[i]-1>r)
                {
                    l=i;
                    r=i+ex[i]-1;
                }
            }
        }
    }
}z;
/*
z[i]: lcp(s,s[i..n-1]) i=0..n-1
ex[i]: lcp(s,t[i..m-1]) i=0..m-1

z.get_z_func(s,n) s[0..n-1]
z.get_ex(t,m) t[0..m-1]
```

```
*/
```

1.2 hash

1.2.1 hash

```
namespace HASH
{
    #define type int
    int m,k;
    vector<type> sd,p;
    vector<vector<type>> tmp;
    void set(vector<type> _sd,vector<type> _p)
    {
        sd=_sd;
        p=_p;
        assert(sd.size()==p.size());
        k=sd.size();
    }
    void init(int _m)
    {
        int i,j;
        m=_m;
        tmp.resize(k);
        for(j=0;j<k;j++)
        {
            tmp[j].resize(m+1);
            tmp[j][0]=1;
            for(i=1;i<=m;i++) tmp[j][i]=1ll*tmp[j][i
                -1]*sd[j]%p[j];
        }
    }
    struct hash_table
    {
        vector<vector<type>> ha;
        void work(char *s,int n)
        {
            int i,j;
            assert(n<=m);
            ha.resize(k);
            for(j=0;j<k;j++)
            {
                ha[j].resize(n+1);
                ha[j][0]=0;
                for(i=1;i<=n;i++) ha[j][i]=(1ll*ha[j][
                    i-1]*sd[j]+s[i])%p[j];
            }
        }
        vector<type> get(int l,int r)
        {
            vector<type> res(k);
            for(int j=0;j<k;j++)
            {
                res[j]=(ha[j][r]-1ll*ha[j][l-1]*tmp[j
                    ][r-l+1])%p[j];
                if(res[j]<0) res[j]+=p[j];
            }
            return res;
        }
    };
    #undef type
}
HASH::hash_table ha;
/*
HASH::set({131,233},{402653189,805306457});
HASH::init(m);
ha.work(s,n); // n<=m
*/
```

1.2.2 good__hash__prime

```
vector<ll> good_hash_prime={
50331653,100663319,201326611,
402653189,805306457,1610612741
};
vector<ll> get_hash_prime(int cnt=2)
{
    assert(cnt<=good_hash_prime.size());
    srand(time(0));
    random_shuffle(good_hash_prime.begin(),
        good_hash_prime.end());
    return vector<ll>(good_hash_prime.begin(),
        good_hash_prime.begin()+cnt);
}
```

1.2.3 hash__map

```
struct hash_map
{
    #define type int
    #define TA pair<type,type>
    static const int p=999917;
    TA val[MAX];
    type w[MAX];
    int tot,head[p],nex[MAX];
    int top,st[MAX];
    hash_map(){top=tot=0;}
    void clear(){tot=0;while(top) head[st[top
        --]]=0;}
    void add(int x,TA y){val[++tot]=y;nex[tot]=head[
        x];head[x]=tot;w[tot]=0;}
    bool count(TA y)
    {
        int x=y.first%p;
        for(int i=head[x];i;i=nex[i])
        {
```

```
            if(y==val[i]) return 1;
        }
        return 0;
    }
    type& operator [](TA y)
    {
        int x=y.first%p;
        for(int i=head[x];i;i=nex[i])
        {
            if(y==val[i]) return w[i];
        }
        add(x,y);
        st[++top]=x;
        return w[tot];
    }
    #undef TA
    #undef type
}flag;
```

1.2.4 BKDRHash

```
struct BKDRHash
{
    static const ull seed=1313131;//
        31,131,1313,13131,131313
    static const int p=2000007;
    ull Hash[MAX],tmp[MAX];
    ull val[MAX];
    int last[p+10],nex[MAX],cnt;
    void init()//clear hash table
    {
        mem(last,0);
        cnt=0;
    }
    bool insert(ull x)
    {
        int u=x%p;
        for(int i=last[u];i;i=nex[i])
        {
            if(val[i]==x) return 1;
        }
        nex[++cnt]=last[u];
        last[u]=cnt;
        val[cnt]=x;
        return 0;
    }
    void work(char *s,int n)
    {
        tmp[0]=1;
        Hash[0]=0;
        for(int i=1;i<=n;i++)
        {
            tmp[i]=tmp[i-1]*seed;
            Hash[i]=Hash[i-1]*seed+s[i];
        }
    }
    ull get(int l,int r)
    {
        return Hash[r]-Hash[l-1]*tmp[r-l+1];
    }
}bkdr; //bkdr.init();
```

1.3 Manacher

1.3.1 插字符

```
struct Manacher
{
    int p[MAX<<1];
    char s[MAX<<1];
    int work(char *a,int n) // a[1..n]
    {
        int i,mid,r,res=0;
        for(i=1;i<=n;i++)
        {
            p[i]=0;
            s[2*i-1]='%';
            s[2*i]=a[i];
        }
        s[n=n*2+1]='%';
        mid=r=0;
        for(i=1;i<=n;i++)
        {
            if(i<r) p[i]=min(p[2*mid-i],r-i);
            else p[i]=1;
            while(i-p[i]>=1&&i+p[i]<=n&&s[i-p[i]]==s[
                i+p[i]]) p[i]++;
            if(i+p[i]>r)
            {
                r=i+p[i];
                mid=i;
            }
            res=max(res,p[i]-1);
        }
        return res;
    }
}la;
```

1.3.2 不插字符

```
struct Manacher
{
    int p[MAX];
    int work(char *s,int n) //s[1..n]
    {
        int r,mid,i,res=0;
        //odd
```

```
        r=mid=0;
        for(i=0;i<=n;i++) p[i]=0;
        for(i=1;i<=n;i++)
        {
            //palindrome substring s[i,i]
            if(r>i) p[i]=min(p[2*mid-i],r-i);
            while(i+p[i]+1<=n&&s[i+p[i]+1]==s[i-p[i
                ]-1])
            {
                //palindrome substring s[i-p[i]-1,i+p[
                    i]+1]
                p[i]++;
            }
            if(i+p[i]>r)
            {
                r=i+p[i];
                mid=i;
            }
            res=max(res,p[i]*2+1);
        }
        //even
        r=mid=0;
        for(i=0;i<=n;i++) p[i]=0;
        for(i=2;i<=n;i++)
        {
            if(r>i) p[i]=min(p[2*mid-i],r-i+1);
            while(i+p[i]<=n&&s[i+p[i]]==s[i-p[i]-1])
            {
                //palindrome substring s[i-p[i]-1,i+p[
                    i]]
                p[i]++;
            }
            if(i+p[i]-1>r)
            {
                r=i+p[i]-1;
                mid=i;
            }
            res=max(res,p[i]*2);
        }
        return res;
    }
}la;
/*
la.work(s,n); // s[1..n]
*/
```

1.4 后缀数组

1.4.1 倍增 sa

```
struct Suffix_Array
{
    int s[MAX],n,SZ;
    int c[MAX],rk[MAX],tmp[MAX],sa[MAX],h[MAX];
    void get_sa()
    {
        int m,i,j,k,tot;
        m=SZ;
        for(i=1;i<=m;i++) c[i]=0;
        for(i=1;i<=n;i++) c[rk[i]=s[i]]++;
        for(i=2;i<=m;i++) c[i]+=c[i-1];
        for(i=n;i;i--) sa[c[rk[i]]--]=i;
        for(k=1;k<=n;k<<=1)
        {
            tot=0;
            for(i=n-k+1;i<=n;i++) tmp[++tot]=i;
            for(i=1;i<=n;i++)
            {
                if(sa[i]>k) tmp[++tot]=sa[i]-k;
            }
            for(i=1;i<=m;i++) c[i]=0;
            for(i=1;i<=n;i++) c[rk[i]]++;
            for(i=2;i<=m;i++) c[i]+=c[i-1];
            for(i=n;i;i--)
            {
                sa[c[rk[tmp[i]]]--]=tmp[i];
                tmp[i]=0;
            }
            swap(rk,tmp);
            rk[sa[1]]=1;
            tot=1;
            for(i=2;i<=n;i++)
            {
                if(sa[i]+k>n||sa[i-1]+k>n||
                  (!(tmp[sa[i]]==tmp[sa[i-1]]&&
                    tmp[sa[i]+k]==tmp[sa[i-1]+k])))
                        tot++;
                rk[sa[i]]=tot;
            }
            if(tot==n) break;
            m=tot;
        }
        h[1]=0;
        k=0;
        for(i=1;i<=n;i++)
        {
            if(rk[i]==1) continue;
            if(k>0) k--;
            j=sa[rk[i]-1];
            while(j+k<=n&&i+k<=n&&s[i+k]==s[j+k]) k
                ++;
            h[rk[i]]=k;
        }
    }
    void work(char *_s,int _n) //s[1..n]
    {
        SZ=0; // char size
        n=_n;
```

```
        for(int i=1;i<=n;i++)
        {
            s[i]=_s[i];
            SZ=max(SZ,s[i]);
        }
        get_sa();
    }
}sa;
/*
sa[i]: s[sa[i]..n] rank is i
rk[i]: s[i..n] rank is rk[i]
h[i]: lcp(s[sa[i]..n],s[sa[i-1]..n])
lcp(s[i..n],s[j..n]) (i<j): min{h[i+1..j]}

sa.work(s,n) s[1..n]
*/
```

1.4.2 SA-IS

```
struct SA
{
    char S[MAX]; int n,m;
    int s[MAX<<1],t[MAX<<1],H[MAX],sa[MAX],r[MAX],p[
        MAX],c[MAX],w[MAX];
    inline int trans(int n,const char* S){
        int m=*max_element(S+1,S+1+n);
        for(int i=1;i<=n;++i) r[S[i]]=1;
        for(int i=1;i<=m;++i) r[i]+=r[i-1];
        for(int i=1;i<=n;++i) s[i]=r[S[i]];
        return r[m];
    }
    #define ps(x) sa[w[s[x]]--]=x
    #define pl(x) sa[w[s[x]]++]=x
    inline void radix(int* v,int* s,int* t,int n,int
         m,int n1){
        memset(sa,0,n+1<<2); memset(c,0,m+1<<2);
        for(int i=1;i<=n;++i) ++c[s[i]];
        for(int i=1;i<=m;++i) w[i]=c[i]+=c[i-1];
        for(int i=n1;i;--i) ps(v[i]);
        for(int i=1;i<=m;++i) w[i]=c[i-1]+1;
        for(int i=1;i<=n;++i) if(sa[i]>1 && t[sa[i
            ]-1]) pl(sa[i]-1);
        for(int i=1;i<=m;++i) w[i]=c[i];
        for(int i=n;i;--i) if(sa[i]>1 && !t[sa[i]-1])
             ps(sa[i]-1);
    }
    inline void SAIS(int n,int m,int* s,int* t,int*
        p){
        int n1=0,ch=r[1]=0,*s1=s+n; t[n]=0;
        for(int i=n-1;i;--i) t[i]=s[i]==s[i+1]?t[i
            +1]:s[i]>s[i+1];
        for(int i=2;i<=n;++i) r[i]=t[i-1]&&!t[i]?(p
            [++n1]=i,n1):0;
        radix(p,s,t,n,m,n1);
        for(int i=1,x,y;i<=n;++i) if(x=r[sa[i]]){
            if(ch<=1 || p[x+1]-p[x]!=p[y+1]-p[y]) ++
                ch;
            else for(int j=p[x],k=p[y];j<=p[x+1];++j
                ,++k)
                if((s[j]<<1|t[j])^(s[k]<<1|t[k])){ ++
                    ch; break; }
            s1[y=x]=ch;
        }
        if(ch<n1) SAIS(n1,ch,s1,t+n,p+n1);
        else for(int i=1;i<=n1;++i) sa[s1[i]]=i;
        for(int i=1;i<=n1;++i) s1[i]=p[sa[i]];
        radix(s1,s,t,n,m,n1);
    }
    inline void get_sa(int n,const char* S,int *ssa,
         int *h){
        int m=trans(++n,S); SAIS(n,m,s,t,p);
        for(int i=1;i<n;++i) r[sa[i]=sa[i+1]]=i;
        for(int i=1,j,k=0;i<n;++i) if(r[i]>1){
            for(j=sa[r[i]-1];S[i+k]==S[j+k];++k);
            if(H[r[i]]=k) --k;
        }
        for(int i=1;i<=n;i++)
        {
            ssa[i]=sa[i];
            h[i]=H[i];
        }
    }
}sais;
```

1.5 自动机

1.5.1 AC 自动机

```
struct AC_Automaton
{
    static const int K=;
    int nex[MAX][K],fail[MAX],cnt[MAX],lst[MAX];
    int root,tot,pos[MAX];
    int getid(char c){return c-;}//may need change
    int newnode()
    {
        memset(nex[tot],0,sizeof nex[tot]);
        fail[tot]=0;
        cnt[tot]=0;
        return tot++;
    }
    void init()
    {
        tot=0;
        root=newnode();
    }
    void insert(char *s,int n,int x) // s[0..n-1]
    {
```

```
    int now,i,t;
    now=root;
    for(i=0;i<n;i++)
    {
        t=getid(s[i]);
        if(!nex[now][t]) nex[now][t]=newnode();
        now=nex[now][t];
    }
    cnt[now]++;
    pos[x]=now;
}
void work()
{
    int i,now;
    queue<int> q;
    for(i=0;i<K;i++)
    {
        if(nex[root][i]) q.push(nex[root][i]);
    }
    while(!q.empty())
    {
        now=q.front();
        q.pop();

        //suffix link
        if(cnt[fail[now]]) lst[now]=fail[now];
        else lst[now]=lst[fail[now]];

        for(i=0;i<K;i++)
        {
            if(nex[now][i])
            {
                fail[nex[now][i]]=nex[fail[now]][i
                    ];
                q.push(nex[now][i]);
            }
            else nex[now][i]=nex[fail[now]][i];
        }
    }
}
int out[MAX];
void topsort()
{
    int i,t;
    queue<int> q;
    for(i=1;i<tot;i++) out[fail[i]]++;
    for(i=1;i<tot;i++)
    {
        if(!out[i]) q.push(i);
    }
    while(!q.empty())
    {
        t=q.front();
        q.pop();

        // do something

        out[fail[t]]--;
        if(out[fail[t]]==0) q.push(fail[t]);
    }
}
int query(char *s,int n)
{
    int len,now,i,res,t,tmp;
    now=root;
    res=0;
    vector<pair<int,int>> del;
    for(i=0;i<n;i++)
    {
        t=getid(s[i]);
        now=nex[now][t];
        tmp=now;
        while(tmp!=root&&cnt[tmp]!=-1)
        {
            res+=cnt[tmp];
            del.push_back({tmp,cnt[tmp]});
            cnt[tmp]=-1;
            tmp=lst[tmp];
        }
    }
    for(auto &it:del) cnt[it.first]=it.second;
    return res;
}
void build_fail_tree(vector<int> mp[])
{
    for(int i=0;i<=tot;i++) mp[i].clear();
    for(int i=1;i<tot;i++) mp[fail[i]].push_back(
        i);
}
}ac;
/*
i is the suffix for each node in the subtree(i) of
    the fail tree

ac.init();
ac.insert(s,len,id); s[0..len-1], id:1..n
ac.work();
ac.query(s,len); s[0..len-1]
ac.build_fail_tree(mp);
*/
```

1.5.2 大字符集 AC 自动机

```
struct AC_Automaton
{
    map<int,int> nex[MAX];
    VI toplist;
    int fail[MAX],last[MAX],cnt[MAX];
```

```
    int root,tot;
    int newnode()
    {
        tot++;
        nex[tot].clear();
        return tot;
    }
    void init()
    {
        toplist.clear();
        tot=0;
        root=newnode();
    }
    void insert(VI &s)
    {
        int len,now,i;
        len=sz(s);
        now=root;
        for(i=0;i<len;i++)
        {
            int t=s[i];
            if(!nex[now].count(t)) nex[now][t]=
                newnode();
            now=nex[now][t];
        }
        cnt[now]=1;
    }
    void work()
    {
        int i,now;
        queue<int>q;
        for(auto it:nex[root])
        {
            fail[it.se]=root;
            q.push(it.se);
        }
        fail[root]=1;
        while(!q.empty())
        {
            now=q.front();
            q.pop();
            toplist.pb(now);
            //suffix link
        /* if(cnt[fail[now]]) last[now]=fail[now];
            else last[now]=last[fail[now]];*/
            cnt[now]+=cnt[fail[now]];
            for(auto it:nex[now])
            {
                int fail_now=fail[now];
                while(fail_now>1&&!nex[fail_now].count
                    (it.fi)) fail_now=fail[fail_now];
                if(nex[fail_now].count(it.fi)) fail[it
                    .se]=nex[fail_now][it.fi];
                else fail[it.se]=root;
                q.push(it.se);
            }
        }
    }
    int query(VI& s,int x)
    {
        int len,now,i,res;
        len=sz(s);
        now=root;
        res=0;
        for(i=0;i<len;i++)
        {
            int t=s[i];
            while(now>1&&!nex[now].count(t)) now=fail
                [now];
            if(nex[now].count(t)) now=nex[now][t];
            else now=root;
            //do something
        }
        return res;
    }
    void toptrans()
    {
        for(int i=sz(toplist)-1;~i;i--)/*do something
            */;
    }
}ac;
```

1.5.3 回文自动机

```
struct Palindrome_Tree
{
    static const int N=MAX;
    static const int LOGN=log2(N)+3;
    static const int K=26;// char size: [0,25]
    int len[N],nex[N][K],fail[N],last,pos[N],s[N],
        tot,n;
    int cnt[N],dep[N];
    int newnode(int l)
    {
        memset(nex[tot],0,sizeof nex[tot]);
        fail[tot]=0;
        dep[tot]=cnt[tot]=0;
        len[tot]=l;
        return tot++;
    }
    void init()
    {
        tot=n=last=0;
        newnode(0);
        newnode(-1);
        s[0]=-1;
        fail[0]=1;
    }
```

```
int get_fail(int x)
{
    while(s[n-len[x]-1]!=s[n]) x=fail[x];
    return x;
}
void add(int t,int p)
{
    int id,now;
    s[++n]=t;
    now=get_fail(last);
    if(!nex[now][t])
    {
        id=newnode(len[now]+2);
        fail[id]=nex[get_fail(fail[now])][t];
        dep[id]=dep[fail[id]]+1;
        nex[now][t]=id;
    }
    last=nex[now][t];
    cnt[last]++;
    pos[p]=last;
}
void topsort()
{
    for(int i=tot-1;i;i--) cnt[fail[i]]+=cnt[i];
}
int st[N][LOGN];
void init_ST()
{
    int i,j;
    for(i=2;i<tot;i++)
    {
        st[i][0]=fail[i];
        for(j=1;j<LOGN;j++)
        {
            st[i][j]=st[st[i][j-1]][j-1];
        }
    }
}
int get_substr(int l,int r)//init_ST
{
    int now,tmp,i;
    now=pos[r];
    for(i=LOGN-1;~i;i--)
    {
        if(len[st[now][i]]>=r-l+1) now=st[now][i
            ];
    }
    //maybe need judge if len[now]==r-l+1
    return now;
}
void build_tree(vector<int> mp[])// root is 0
{
    for(int i=0;i<=tot+1;i++) mp[i].clear();
    for(int i=1;i<tot;i++) mp[fail[i]].push_back(
        i);
}

}pam;
/*
pam.init();
pam.add(t,id); t is int
*/
```

1.5.4 序列自动机

```
int nex[MAX][26];
void work(char *s,int len)
{
    mem(nex[len],0);
    for(int i=len;i;i--)
    {
        for(int j=0;j<26;j++)
        {
            nex[i-1][j]=nex[i][j];
        }
        nex[i-1][s[i]-'a']=i;
    }
}
```

1.5.5 KMP 自动机

```
struct KMP_Automaton
{
    #define type char
    static const int K=;
    int nex[MAX],len,lst[MAX][K];
    type t[MAX];
    int get_id(char c){return c-;} //may need change
    void get_next(type *s,int n)
    {
        int i,j,k;
        len=n;
        for(i=1;i<=len;i++) t[i]=s[i];
        t[len+1]=0;
        nex[0]=nex[1]=0;
        j=0;
        for(i=2;i<=len;i++)
        {
            while(j&&t[j+1]!=s[i]) j=nex[j];
            if(t[j+1]==s[i]) j++;
            nex[i]=j;
        }
        memset(lst[0],0,sizeof lst[0]);
        for(i=1;i<=len;i++)
        {
```

```
            for(k=0;k<K;k++) lst[i][k]=lst[nex[i]][k
                ];
            if(i+1<=len) lst[i][get_id(s[i+1])]=i;
        }
    }
    vector<int> match(type *s,int n)
    {
        int i,j;
        vector<int> res;
        for(i=1,j=0;i<=n;i++)
        {
            j=lst[j][get_id(s[i])];
            if(t[j+1]==s[i]) j++;
            if(j==len)
            {
                res.push_back(i-len+1);
                j=nex[j];
            }
        }
        return res;
    }
    #undef type
}kmp;
/*
kmp.get_next(t,len); // t[1..len]
kmp.match(s,n); // s[1..n] return all pos t in s
*/
```

1.5.6 后缀自动机

```
struct Suffix_Automaton
{
    static const int N=MAX<<1;
    static const int K=26;// char size: [0,25]
    int tot,lst,nex[N][K],fa[N],len[N],cnt[N],maxlen
        ,root;
    int newnode()
    {
        tot++;
        fa[tot]=len[tot]=cnt[tot]=0;
        memset(nex[tot],0,sizeof nex[tot]);
        return tot;
    }
    void init()
    {
        fa[0]=len[0]=cnt[0]=0;
        memset(nex[0],0,sizeof nex[0]);
        tot=0;
        maxlen=0;
        root=lst=newnode();
    }
    void add(int x)
    {
        int p,q,np,nq;
        p=lst;
        np=lst=newnode();
        len[np]=len[p]+1;
        maxlen=max(maxlen,len[np]);
        cnt[lst]=1;
        while(p&&!nex[p][x])
        {
            nex[p][x]=np;
            p=fa[p];
        }
        if(p==0) fa[np]=root;
        else
        {
            q=nex[p][x];
            if(len[q]==len[p]+1) fa[np]=q;
            else
            {
                nq=newnode();
                memcpy(nex[nq],nex[q],sizeof(nex[q]));
                len[nq]=len[p]+1;
                maxlen=max(maxlen,len[nq]);
                fa[nq]=fa[q];
                fa[q]=fa[np]=nq;
                while(p&&nex[p][x]==q)
                {
                    nex[p][x]=nq;
                    p=fa[p];
                }
            }
        }
    }
    int sum[N],tp[N];
    void topsort()
    {
        int i;
        for(i=1;i<=maxlen;i++) sum[i]=0;
        for(i=1;i<=tot;i++) sum[len[i]]++;
        for(i=1;i<=maxlen;i++) sum[i]+=sum[i-1];
        for(i=1;i<=tot;i++) tp[sum[len[i]]--]=i;
        for(i=tot;i;i--) cnt[fa[tp[i]]]+=cnt[tp[i]];
    }
    void build_tree(VI mp[])
    {
        for(int i=1;i<=tot;i++) mp[i].clear();
        for(int i=1;i<=tot;i++) mp[fa[i]].pb(i);
    }
    int pos[N],id[N];
    void init_pos(char *s,int n)//s[1..n]
    {
        int now=1;
        for(int i=1;i<=tot;i++) id[i]=-1;
        for(int i=1;i<=n;i++)
        {
            now=nex[now][s[i]-'a'];
```

```
            pos[i]=now;
            id[now]=i;
        }
    }
    int st[N][21];
    void init_ST()
    {
        int i,j,x;
        for(i=1;i<=tot;i++)
        {
            x=tp[i];
            st[x][0]=fa[x];
            for(j=1;j<20;j++)
            {
                st[x][j]=st[st[x][j-1]][j-1];
            }
        }
    }
    int get_substr(int l,int r)//init_pos init_ST
    {
        int now,tmp,i;
        now=pos[r];
        for(i=19;~i;i--)
        {
            tmp=st[now][i];
            if(tmp&&len[tmp]>=r-l+1) now=tmp;
        }
        return now;
    }
}sam;
/*
sam.init();
sam.add(int x);
sam.topsort();

*/
```

1.5.7 广义后缀自动机

```
struct Suffix_Automaton
{
    static const int N=MAX<<1;
    static const int K=26;// char size: [0,25]
    int tot,lst,nex[N][K],fa[N],len[N],cnt[N],mxlen,
        root;
    int newnode()
    {
        tot++;
        fa[tot]=len[tot]=cnt[tot]=0;
        memset(nex[tot],0,sizeof nex[tot]);
        return tot;
    }
    void init()
    {
        fa[0]=len[0]=cnt[0]=0;
        memset(nex[0],0,sizeof nex[0]);
        tot=0;
        mxlen=0;
        root=lst=newnode();
    }
    void add(int x)
    {
        int p,q,np,nq;
        if(nex[lst][x])
        {
            p=lst;
            q=nex[lst][x];
            if(len[q]==len[p]+1) lst=q;
            else
            {
                nq=lst=newnode();
                memcpy(nex[nq],nex[q],sizeof(nex[q]));
                len[nq]=len[p]+1;
                mxlen=max(mxlen,len[nq]);
                fa[nq]=fa[q];
                fa[q]=nq;
                while(p&&nex[p][x]==q)
                {
                    nex[p][x]=nq;
                    p=fa[p];
                }
            }
        }
        else
        {
            p=lst;
            np=lst=newnode();
            len[np]=len[p]+1;
            mxlen=max(mxlen,len[np]);
            cnt[lst]=1;
            while(p&&!nex[p][x])
            {
                nex[p][x]=np;
                p=fa[p];
            }
            if(p==0) fa[np]=root;
            else
            {
                q=nex[p][x];
                if(len[q]==len[p]+1) fa[np]=q;
                else
                {
                    nq=newnode();
                    memcpy(nex[nq],nex[q],sizeof(nex[q
                        ]));
                    len[nq]=len[p]+1;
                    mxlen=max(mxlen,len[nq]);
                    fa[nq]=fa[q];
```

```
                fa[q]=fa[np]=nq;
                while(p&&nex[p][x]==q)
                {
                    nex[p][x]=nq;
                    p=fa[p];
                }
            }
        }
    }
    void insert(char *s,int n)
    {
        int i;
        lst=root;
        for(i=1;i<=n;i++) add(s[i]-'a');
    }
    int sum[N],tp[N];
    void topsort()
    {
        int i;
        for(i=1;i<=mxlen;i++) sum[i]=0;
        for(i=1;i<=tot;i++) sum[len[i]]++;
        for(i=1;i<=mxlen;i++) sum[i]+=sum[i-1];
        for(i=1;i<=tot;i++) tp[sum[len[i]]--]=i;
        for(i=tot;i;i--) cnt[fa[tp[i]]]+=cnt[tp[i]];
    }
}sam;
/*
sam.init();
sam.add(int x);
*/
```

1.6 最小表示法

1.6.1 最小表示法

```
int min_representation(char *s,int n) // s[0..n-1]
{
    int i,j,k,tmp;
    i=k=0;
    j=1;
    while(i<n&&j<n&&k<n)
    {
        tmp=s[(i+k)%n]-s[(j+k)%n];
        if(!tmp) k++;
        else
        {
            if(tmp>0) i=i+k+1;
            else j=j+k+1;
            if(i==j) j++;
            k=0;
        }
    }
    return i<j?i:j;
}
```

1.6.2 最大表示法

```
int max_representation(char *s,int n) // s[0..n-1]
{
    int i,j,k,tmp;
    i=k=0;
    j=1;
    while(i<n&&j<n&&k<n)
    {
        tmp=s[(i+k)%n]-s[(j+k)%n];
        if(!tmp) k++;
        else
        {
            if(tmp<0) i=i+k+1;
            else j=j+k+1;
            if(i==j) j++;
            k=0;
        }
    }
    return i<j?i:j;
}
```

1.7 shift_and

```
void shift_and(char *s,char *t)//s[1..n],t[1..m] (n
    <=m)
{
    static const int SZ=26;
    int n,m,i;
    bitset<MAX> b[SZ],d;
    for(i=0;i<SZ;i++) b[i].reset();
    d.reset();
    n=strlen(s+1);
    m=strlen(t+1);
    for(i=1;i<=n;i++)
    {
        b[s[i]-'a'].set(i-1);//change
        //other matching character sets
    }
    for(i=1;i<=m;i++)
    {
        d<<=1;
        d.set(0);
        d&=(b[t[i]-'a']);//change
        if(d[n-1]==1)//successful match
        {

        }
    }
}
```

2 数据结构

2.1 离散化

```
struct Discretization
{
    #define type ll
    #define all(x) x.begin(),x.end()
    vector<type> a;
    void init(){a.clear();}
    void add(type x){a.push_back(x);}
    void work(){sort(all(a));a.resize(unique(all(a))
        -a.begin());}
    int get_pos(type x){return lower_bound(all(a),x)
        -a.begin()+1;}
    type get_val(int pos){return a[pos-1];}
    int size(){return a.size();}
    #undef type
    #undef all
}d;
```

2.2 RMQ

2.2.1 一维 RMQ

```
struct ST_table
{
    #define type int
    static const int N=MAX;
    static const int LOG=21;
    type v[N];
    int lg[N],bin[LOG],pmx[LOG][N],pmn[LOG][N];
    int pmax(const int &a,const int &b){return v[a]>
        v[b]?a:b;}
    int pmin(const int &a,const int &b){return v[a]<
        v[b]?a:b;}
    void work(int n,type *a)
    {
        int i,j;
        for(i=bin[0]=1;1<<(i-1)<=n;i++) bin[i]=(bin[i
            -1]<<1);
        for(i=2,lg[1]=0;i<=n;i++) lg[i]=lg[i>>1]+1;
        for(i=1;i<=n;i++)
        {
            v[i]=a[i];
            pmx[0][i]=pmn[0][i]=i;
        }
        for(j=1;1<<(j-1)<=n;j++)
        {
            for(i=1;i+bin[j]-1<=n;i++)
            {
                pmx[j][i]=pmax(pmx[j-1][i],pmx[j-1][i+
                    bin[j-1]]);
                pmn[j][i]=pmin(pmn[j-1][i],pmn[j-1][i+
                    bin[j-1]]);
            }
        }
    }
    int ask_pmax(int l,int r)
    {
        int t=lg[r-l+1];
        return pmax(pmx[t][l],pmx[t][r-bin[t]+1]);
    }
    int ask_pmin(int l,int r)
    {
        int t=lg[r-l+1];
        return pmin(pmn[t][l],pmn[t][r-bin[t]+1]);
    }
    type ask_max(int l,int r){return v[ask_pmax(l,r)
        ];}
    type ask_min(int l,int r){return v[ask_pmin(l,r)
        ];}
    #undef type
}rmq;
```

2.2.2 二维 RMQ

```
int v[302][302];
int maxx[302][302][9][9],minn[302][302][9][9];
void RMQ(int n,int m)
{
    int i,j,ii,jj;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=m;j++)
        {
            maxx[i][j][0][0]=minn[i][j][0][0]=v[i][j
                ];
        }
    }
    for(ii=0;(1<<ii)<=n;ii++)
    {
        for(jj=0;(1<<jj)<=m;jj++)
        {
            if(!(ii+jj)) continue;
            for(i=1;i+(1<<ii)-1<=n;i++)
            {
                for(j=1;j+(1<<jj)-1<=m;j++)
                {
                    if(ii)
                    {
                        minn[i][j][ii][jj]=min(minn[i][
                            j][ii-1][jj],minn[i+(1<<(ii
                            -1))][j][ii-1][jj]);
                        maxx[i][j][ii][jj]=max(maxx[i][
                            j][ii-1][jj],maxx[i+(1<<(ii
                            -1))][j][ii-1][jj]);
```

```
                }
                else
                {
                    minn[i][j][ii][jj]=min(minn[i][
                        j][ii][jj-1],minn[i][j
                        +(1<<(jj-1))][ii][jj-1]);
                    maxx[i][j][ii][jj]=max(maxx[i][
                        j][ii][jj-1],maxx[i][j
                        +(1<<(jj-1))][ii][jj-1]);
                }
            }
        }
    }
}
int ask_max(int x1,int y1,int x2,int y2)
{
    int k1=0;
    while((1<<(k1+1))<=x2-x1+1) k1++;
    int k2=0;
    while((1<<(k2+1))<=y2-y1+1) k2++;
    x2=x2-(1<<k1)+1;
    y2=y2-(1<<k2)+1;
    return max(max(maxx[x1][y1][k1][k2],maxx[x1][y2
        ][k1][k2]),max(maxx[x2][y1][k1][k2],maxx[x2
        ][y2][k1][k2]))
}
int ask_min(int x1,int y1,int x2,int y2)
{
    int k1=0;
    while((1<<(k1+1))<=x2-x1+1) k1++;
    int k2=0;
    while((1<<(k2+1))<=y2-y1+1) k2++;
    x2=x2-(1<<k1)+1;
    y2=y2-(1<<k2)+1;
    return min(min(minn[x1][y1][k1][k2],minn[x1][y2
        ][k1][k2]),min(minn[x2][y1][k1][k2],minn[x2
        ][y2][k1][k2]));
}
```

2.3 单调队列

```
struct Monotone_queue
{
    #define type int
    type v[MAX][2];//0 is min, 1 is max
    int p[MAX][2];
    int l[2],r[2];
    void clear()
    {
        l[0]=r[0]=0;
        l[1]=r[1]=0;
    }
    void insert(type x,int pos)
    {
        while(r[0]-l[0]&&v[r[0]-1][0]>=x) r[0]--;
        v[r[0]][0]=x;
        p[r[0]++][0]=pos;
        while(r[1]-l[1]&&v[r[1]-1][1]<=x) r[1]--;
        v[r[1]][1]=x;
        p[r[1]++][1]=pos;
    }
    void erase(int pos)
    {
        while(r[0]-l[0]&&p[l[0]][0]<=pos) l[0]++;
        while(r[1]-l[1]&&p[l[1]][1]<=pos) l[1]++;
    }
    type get_min(){return v[l[0]][0];}
    type get_max(){return v[l[1]][1];}
    #undef type
}dq;
```

2.4 并查集

2.4.1 并查集

```
struct Disjoint_Set_Union
{
    int pre[MAX],sz[MAX];
    void init(int n)
    {
        int i;
        for(i=1;i<=n;i++)
        {
            pre[i]=i;
            sz[i]=1;
        }
    }
    int find(int x)
    {
        if(pre[x]!=x) pre[x]=find(pre[x]);
        return pre[x];
    }
    bool merge(int x,int y,bool dir=false)
    {
        x=find(x);
        y=find(y);
        if(x==y) return 0;
        if(!dir && sz[x]>sz[y]) swap(x,y);
        pre[x]=y; // x -> y
        sz[y]+=sz[x];
        return 1;
    }
}dsu;
```

2.4.2 map 实现并查集

```
struct dsu
{
    #define type int
    unordered_map<type,type> pre;
    void init(){pre.clear();}
    type find(type x)
    {
        if(pre.count(x)) pre[x]=find(pre[x]);
        else return x;
        return pre[x];
    }
    bool merge(type a,type b)
    {
        type ra,rb;
        ra=find(a);
        rb=find(b);
        if(ra!=rb)
        {
            pre[ra]=rb;
            return 1;
        }
        return 0;
    }
    #undef type
}dsu;
```

2.4.3 可撤销并查集

```
struct Disjoint_Set_Union
{
    pair<int,int> st[MAX];
    int pre[MAX],top,sz[MAX];
    void init(int n)
    {
        int i;
        for(i=1;i<=n;i++)
        {
            pre[i]=i;
            sz[i]=1;
        }
        top=0;
    }
    int find(int x)
    {
        while(x!=pre[x]) x=pre[x];
        return x;
    }
    bool merge(int x,int y)
    {
        x=find(x);
        y=find(y);
        if(x==y) return 0;
        if(sz[x]>sz[y]) swap(x,y);
        pre[x]=y;
        sz[y]+=sz[x];
        st[top++]={x,y};
        return 1;
    }
    void roll_back()
    {
        pair<int,int> now=st[--top];
        pre[now.first]=now.first;
        sz[now.second]-=sz[now.first];
    }
}dsu;
```

2.5 树状数组

2.5.1 一维单点 BIT

```
struct Fenwick_Tree
{
    #define type int
    type bit[MAX],tmp[MAX];
    int n;
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++) bit[i]=0;
    }
    void init(int _n,type *a)
    {
        n=_n;
        tmp[0]=0;
        for(int i=1;i<=n;i++)
        {
            tmp[i]=a[i];
            tmp[i]+=tmp[i-1];
            bit[i]=tmp[i]-tmp[i-lowbit(i)];
        }
    }
    int lowbit(int x){return x&(-x);}
    type get(int x)
    {
        type res=0;
        while(x)
        {
            res+=bit[x];
            x-=lowbit(x);
        }
        return res;
    }
    void upd(int x,type v)
    {
        while(x<=n)
        {
            bit[x]+=v;
```

```
            x+=lowbit(x);
        }
    }
    type ask(int l,int r)
    {
        if(l>r) return 0;
        if(l-1<=0) return get(r);
        return get(r)-get(l-1);
    }
    #undef type
}tr;
```

2.5.2 一维区间 BIT

```
struct Fenwick_Tree
{
    #define type int
    type bit[MAX][2];
    int n;
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++)
        {
            bit[i][0]=bit[i][1]=0;
        }
    }
    int lowbit(int x){return x&(-x);}
    void _insert(int x,type v)
    {
        for(int i=x;i<=n;i+=lowbit(i))
        {
            bit[i][0]+=v;
            bit[i][1]+=v*(x-1);
        }
    }
    type get(int x)
    {
        type res=0;
        for(int i=x;i;i-=lowbit(i))
        {
            res+=x*bit[i][0]-bit[i][1];
        }
        return res;
    }
    void upd(int l,int r,type v)
    {
        _insert(l,v);
        _insert(r+1,-v);
    }
    type ask(int l,int r)
    {
        if(l-1<=0) return get(r);
        return get(r)-get(l-1);
    }
    #undef type
}tr;
```

2.5.3 二维单点 BIT

```
struct Fenwick_Tree
{
    #define type int
    type bit[MAX][MAX];
    int n,m;
    void init(int _n,int _m)
    {
        int i,j;
        n=_n;
        m=_m;
        for(i=0;i<=n;i++)
        {
            for(j=0;j<=m;j++) bit[i][j]=0;
        }
    }
    int lowbit(int x){return x&(-x);}
    type get(int x,int y)
    {
        type i,j,res=0;
        for(i=x;i>0;i-=lowbit(i))
        {
            for(j=y;j>0;j-=lowbit(j))
            {
                res+=bit[i][j];
            }
        }
        return res;
    }
    void upd(int x,int y,type v)
    {
        int i,j;
        for(i=x;i<=n;i+=lowbit(i))
        {
            for(j=y;j<=m;j+=lowbit(j))
            {
                bit[i][j]+=v;
            }
        }
    }
    type ask(int x1,int y1,int x2,int y2)
    {
        if(x1>x2||y1>y2) return 0;
        x1--;
        y1--;
        return get(x2,y2)-get(x1,y2)-get(x2,y1)+get(
            x1,y1);
    }
    #undef type
```

```
}tr;
```

2.6 线段树

2.6.1 线段树

```
struct Segment_Tree
{
    #define type int
    static const type inf=INF;
    #define ls (id<<1)
    #define rs (id<<1|1)
    int n,ql,qr,qop;
    type a[MAX],tag[MAX<<2],qv;
    struct node
    {
        type v;
        void init(){}
    }t[MAX<<2],null_node;
    node merge_node(node x,node y)
    {
        node res;

        return res;
    }
    void pushup(int id){t[id]=merge_node(t[ls],t[rs
        ]);}
    void pushdown(int l,int r,int id)
    {
        if(!tag[id]) return;
        int mid=(l+r)>>1;

    }
    void build(int l,int r,int id)
    {
        tag[id]=0;
        t[id].init();
        if(l==r)
        {
            //init
            return;
        }
        int mid=(l+r)>>1;
        build(l,mid,ls);
        build(mid+1,r,rs);
        pushup(id);
    }
    void update(int l,int r,int id)
    {
        if(l>=ql&&r<=qr)
        {

            return;
        }
        pushdown(l,r,id);
        int mid=(l+r)>>1;
        if(ql<=mid) update(l,mid,ls);
        if(qr>mid) update(mid+1,r,rs);
        pushup(id);
    }
    node query(int l,int r,int id)
    {
        if(l>=ql&&r<=qr) return t[id];
        pushdown(l,r,id);
        int mid=(l+r)>>1;
        if(qr<=mid) return query(l,mid,ls);
        if(ql>mid) return query(mid+1,r,rs);
        return merge_node(query(l,mid,ls),query(mid
            +1,r,rs));
    }
    void build(int _n)
    {
        n=_n;
        build(1,n,1);
        null_node.init();
    }
    void upd(int l,int r,type v)
    {
        if(l>r) return;
        ql=l;
        qr=r;
        qv=v;
        update(1,n,1);
    }
    type ask(int l,int r)
    {
        if(l>r) return null_node.v;
        ql=l;
        qr=r;
        return query(1,n,1).v;
    }
    #undef type
    #undef ls
    #undef rs
}tr;
/*
tr.build(n);
tr.upd(l,r,v);
tr.ask(l,r);
Segment_Tree::node res=tr.merge_node(nodex,nodey);
*/
```

2.6.2 动态开点线段树

```
struct Segment_Tree
{
    #define type int
    static const int LOG=31;
```

```
5   struct node
6   {
7       type v;
8       void init(){v=0;}
9   }t[MAX*LOG],null_node;
10  int root[MAX],ls[MAX*LOG],rs[MAX*LOG],rt,tot,qop
        ;
11  int st[MAX*LOG],top;
12  type ql,qr,qv,n,tag[MAX*LOG];
13  Segment_Tree &operator[](const int _rt){this->rt
        =_rt;return *this;}
14  void init(type _n)
15  {
16      n=_n;
17      rt=1;
18      tot=top=0;
19      t[0].init();
20      ls[0]=rs[0]=0;
21      null_node.init();
22  }
23  void build(){root[rt]=0;}
24  void delnode(int id){st[top++]=id;}
25  int newnode()
26  {
27      int id;
28      id=top?st[--top]:++tot;
29      t[id].init();
30      ls[id]=rs[id]=0;
31      tag[id]=0;
32      return id;
33  }
34  node merge_node(node x,node y)
35  {
36      node res;
37      //
38      return res;
39  }
40  void pushup(int id){t[id]=merge_node(t[ls[id]],t
        [rs[id]]);}
41  void pushdown(type l,type r,int id)
42  {
43      if(!tag[id]) return;
44      if(!ls[id]) ls[id]=newnode();
45      if(!rs[id]) rs[id]=newnode();
46
47
48      tag[id]=0;
49  }
50  void update(type l,type r,int &id)
51  {
52      if(!id) id=newnode();
53      if(l>=ql&&r<=qr)
54      {
55          //
56          return;
57      }
58      pushdown(l,r,id);
59      type mid=(l+r)>>1;
60      if(ql<=mid) update(l,mid,ls[id]);
61      if(qr>mid) update(mid+1,r,rs[id]);
62      pushup(id);
63  }
64  node query(type l,type r,int id)
65  {
66      if(!id) return null_node;
67      if(l>=ql&&r<=qr) return t[id];
68      pushdown(l,r,id);
69      type mid=(l+r)>>1;
70      if(qr<=mid) return query(l,mid,ls[id]);
71      if(ql>mid) return query(mid+1,r,rs[id]);
72      return merge_node(query(l,mid,ls[id]),query(
            mid+1,r,rs[id]));
73  }
74  int split(type l,type r,int &id)
75  {
76      int x;
77      if(!id) return 0;
78      if(l>=ql&&r<=qr)
79      {
80          x=id;
81          id=0;
82          return x;
83      }
84      x=newnode();
85      int mid=(l+r)>>1;
86      if(ql<=mid) ls[x]=split(l,mid,ls[id]);
87      if(qr>mid) rs[x]=split(mid+1,r,rs[id]);
88      pushup(x);
89      pushup(id);
90      return x;
91  }
92  int merge(type l,type r,int x,int y)
93  {
94      if(!x||!y) return x+y;
95      if(l==r)
96      {
97          //merge info x <- y
98          delnode(y);
99          return x;
100     }
101     type mid=(l+r)>>1;
102     ls[x]=merge(l,mid,ls[x],ls[y]);
103     rs[x]=merge(mid+1,r,rs[x],rs[y]);
104     pushup(x);
105     delnode(y);
106     return x;
107 }
108 void split_segtree(type l,type r,int new_tree)
```

```
    {
        ql=l;
        qr=r;
        root[new_tree]=split(1,n,root[rt]);
    }
    void merge_segtree(int y)
    {
        root[rt]=merge(1,n,root[rt],root[y]);
        root[y]=0;
    }
    void upd(type l,type r,type v)
    {
        ql=l;
        qr=r;
        qv=v;
        update(1,n,root[rt]);
    }
    type ask(type l,type r)
    {
        ql=l;
        qr=r;
        return query(1,n,root[rt]).v;
    }
    #undef type
}tr;
/*
tr.init(n);
tr.build();
tr.upd(l,r,v);
tr.ask(l,r);
Segment_Tree::node res=tr.merge_node(nodex,nodey);
---------
tr[x].build();
tr[x].merge_segtree(y);
tr[x].split_segtree(l,r,new_tree);
*/
```

2.6.3 区间查询最大子段和

```
struct node
{
    type sum,lmx,rmx,mx;
    void init(){sum=0;lmx=rmx=mx=-inf;}
}t[MAX<<2],null_node;
node merge_node(node x,node y)
{
    node res;
    res.sum=x.sum+y.sum;
    res.lmx=max(x.lmx,x.sum+y.lmx);
    res.rmx=max(y.rmx,y.sum+x.rmx);
    res.mx=max({x.mx,y.mx,x.rmx+y.lmx});
    return res;
}
```

2.6.4 矩形面积并

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int INF=0x3f3f3f3f;
const int MAX=4e5+10;
struct Discretization
{
    #define type ll
    #define pb push_back
    #define all(x) x.begin(),x.end()
    vector<type> a;
    void init(){a.clear();}
    void add(type x){a.pb(x);}
    void work(){sort(all(a));a.resize(unique(all(a))
        -a.begin());}
    int get_pos(type x){return lower_bound(all(a),x)
        -a.begin()+1;}
    type get_val(int pos){return a[pos-1];}
    int size(){return a.size();}
    #undef type
    #undef pb
    #undef all
}dz;
struct Segment_Tree
{
    #define type ll
    #define ls (id<<1)
    #define rs (id<<1|1)
    int n,ql,qr;
    type a[MAX],v[MAX<<2],mn[MAX<<2],tag[MAX<<2],
        cntmn[MAX<<2],qv;
    void pushup(int id)
    {
        mn[id]=min(mn[ls],mn[rs]);
        if(mn[ls]==mn[rs]) cntmn[id]=cntmn[ls]+cntmn[
            rs];
        else cntmn[id]=(mn[ls]<mn[rs]?cntmn[ls]:cntmn
            [rs]);
        v[id]=v[ls]+v[rs];
    }
    void maintain(int pre,int now,int id)
    {
        if(pre==0&&now>0) v[id]+=cntmn[id];
        if(pre>0&&now==0) v[id]-=cntmn[id];
    }
    void pushdown(int l,int r,int id)
    {
        if(!tag[id]) return;
        int mid=(l+r)>>1;
        maintain(mn[ls],mn[ls]+tag[id],ls);
        maintain(mn[rs],mn[rs]+tag[id],rs);
        mn[ls]+=tag[id];
```

```
        mn[rs]+=tag[id];
        tag[ls]+=tag[id];
        tag[rs]+=tag[id];
        tag[id]=0;
    }
    void build(int l,int r,int id)
    {
        tag[id]=0;
        if(l==r)
        {
            mn[id]=0;
            cntmn[id]=dz.get_val(l)-dz.get_val(l-1);
            v[id]=0;
            return;
        }
        int mid=(l+r)>>1;
        build(l,mid,ls);
        build(mid+1,r,rs);
        pushup(id);
    }
    void update(int l,int r,int id)
    {
        if(l>=ql&&r<=qr)
        {
            maintain(mn[id],mn[id]+qv,id);
            mn[id]+=qv;
            tag[id]+=qv;
            return;
        }
        pushdown(l,r,id);
        int mid=(l+r)>>1;
        if(ql<=mid) update(l,mid,ls);
        if(qr>mid) update(mid+1,r,rs);
        pushup(id);
    }
    type res;
    void query(int l,int r,int id)
    {
        if(l>=ql&&r<=qr)
        {
            res+=v[id];
            return;
        }
        pushdown(l,r,id);
        int mid=(l+r)>>1;
        if(ql<=mid) query(l,mid,ls);
        if(qr>mid) query(mid+1,r,rs);
    }
    void build(int _n){n=_n;build(1,n,1);}
    void upd(int l,int r,type v)
    {
        if(l>r) return;
        ql=l;
        qr=r;
        qv=v;
        update(1,n,1);
    }
    type ask(int l,int r)//init res
    {
        ql=l;
        qr=r;
        res=0;
        query(1,n,1);
        return res;
    }
    #undef type
    #undef ls
    #undef rs
}tr;
struct node{int pos,l,r,v;};
ll work(vector<node> &res)
{
    int i,j,pos;
    ll ans;
    sort(res.begin(),res.end(),[&](node x,node y){
        if(x.pos==y.pos) return x.v>y.v;
        return x.pos<y.pos;
    });
    ans=0;
    pos=1;
    tr.build(dz.size());
    for(i=0;i<res.size();i++)
    {
        if(i) ans+=tr.ask(1,dz.size())*(res[i].pos-
            pos);
        tr.upd(dz.get_pos(res[i].l),dz.get_pos(res[i
            ].r),res[i].v);
        pos=res[i].pos;
    }
    ans+=tr.ask(2,dz.size())*(res[i].pos-pos);
    return ans;
}
int main()
{
    int n,i,a,b,c,d;
    scanf("%d",&n);
    vector<node> x;
    dz.init();
    for(i=1;i<=n;i++)
    {
        scanf("%d%d%d%d",&a,&b,&c,&d);
        dz.add(b);
        dz.add(b-1);
        dz.add(d-2);
        dz.add(d-1);
        x.push_back({a,b,d-1,1});
        x.push_back({c,b,d-1,-1});
    }
```

```
    dz.work();
    printf("%lld\n",work(x));
    return 0;
}
```

2.6.5 线段树维护 hash

```
struct node
{
    type len,ha;
    void init(){len=ha=0;}
}t[MAX<<2],null_node;
node merge_node(node x,node y)
{
    node res;
    res.len=x.len+y.len;
    res.ha=x.ha*qpow(bs,y.len)+y.ha;
    return res;
}
```

2.7 平衡树

2.7.1 Treap

```
struct Treap
{
    #define type int
    static const type inf=INF;
    struct node
    {
        int ch[2],fix,sz,cnt;
        type v;
        node(){}
        node(type x,int _sz)
        {
            v=x;
            fix=rand();
            sz=cnt=_sz;
            ch[0]=ch[1]=0;
        }
    }t[MAX];
    int tot,root[MAX],rt;
    void init(int n=1)
    {
        for(int i=0;i<=n;i++) root[i]=0;
        rt=1;
        srand(time(0));
        tot=0;
        t[0].sz=t[0].cnt=0;
        memset(t[0].ch,0,sizeof t[0].ch);
    }
    void pushup(int id)
    {
        t[id].sz=t[t[id].ch[0]].sz+t[t[id].ch[1]].sz+
            t[id].cnt;
    }
    void rotate(int &id,int k)
    {
        int y=t[id].ch[k^1];
        t[id].ch[k^1]=t[y].ch[k];
        t[y].ch[k]=id;
        pushup(id);
        pushup(y);
        id=y;
    }
    void _insert(int &id,type v,int cnt)
    {
        if(!id)
        {
            id=++tot;
            t[id]=node(v,cnt);
            return;
        }
        if(t[id].v==v) t[id].cnt+=cnt;
        else
        {
            int tmp=(v>t[id].v);
            _insert(t[id].ch[tmp],v,cnt);
            if(t[t[id].ch[tmp]].fix>t[id].fix) rotate
                (id,tmp^1);
        }
        pushup(id);
    }
    void _erase(int &id,type v,int cnt)
    {
        if(!id) return;
        if(t[id].v==v)
        {
            cnt=min(t[id].cnt,cnt);
            if(t[id].cnt>cnt)
            {
                t[id].cnt-=cnt;
                pushup(id);
                return;
            }
            if(!(t[id].ch[0]&&t[id].ch[1]))
            {
                id=t[id].ch[0]+t[id].ch[1];
                return;
            }
            else
            {
                int tmp=(t[t[id].ch[0]].fix>t[t[id].ch
                    [1]].fix);
                rotate(id,tmp);
                _erase(t[id].ch[tmp],v,cnt);
                pushup(id);
```

```
        }
    }
    else
    {
        _erase(t[id].ch[v>t[id].v],v,cnt);
        pushup(id);
    }
}
int _find(type key,int f)
{
    int id=root[rt],res=0;
    while(id)
    {
        if(t[id].v<=key)
        {
            res+=t[t[id].ch[0]].sz+t[id].cnt;
            if(f&&key==t[id].v) res-=t[id].cnt;
            id=t[id].ch[1];
        }
        else id=t[id].ch[0];
    }
    return res;
}
type find_by_order(int k)//k small
{
    int id=root[rt];
    if(id==0) return 0;
    while(id)
    {
        if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
        else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
            return t[id].v;
        else
        {
            k-=t[t[id].ch[0]].sz+t[id].cnt;
            id=t[id].ch[1];
        }
    }
    return 0;
}
int count(type key)
{
    int id=root[rt];
    while(id)
    {
        if(t[id].v<key)
        {
            if(key==t[id].v) return t[id].cnt;
            id=t[id].ch[1];
        }
        else id=t[id].ch[0];
    }
    return 0;
}
type find_pre(type key)
{
    type res=-inf;
    int id=root[rt];
    while(id)
    {
        if(t[id].v<key)
        {
            res=t[id].v;
            id=t[id].ch[1];
        }
        else id=t[id].ch[0];
    }
    return res;
}
type find_nex(type key)
{
    type res=inf;
    int id=root[rt];
    while(id)
    {
        if(t[id].v>key)
        {
            res=t[id].v;
            id=t[id].ch[0];
        }
        else id=t[id].ch[1];
    }
    return res;
}
Treap &operator[](const int _rt){this->rt=_rt;
    return *this;}
void insert(type v,int sz=1){_insert(root[rt],v,
    sz);}
void erase(type v,int sz=1){_erase(root[rt],v,sz
    );}
int upper_bound_count(type key){return _find(key
    ,0);}//the count <=key
int lower_bound_count(type key){return _find(key
    ,1);}//the count <key
int order_of_key(type key){return
    lower_bound_count(key)+1;}
int size(){return t[root[rt]].sz;}
#undef type
}tr;
/*
1 treap
tr.init();
tr.insert(x);
tr.erase(x);
tr.count(x);
tr.order_of_key(x); // rank
tr.find_by_order(k); // kth
tr.find_pre(x);
```

```
tr.find_nex(x);
tr.upper_bound_count(x); //the count <=key
tr.lower_bound_count(x); //the count <key

n treap
tr.init(n);
tr[i].insert(x);
*/
```

2.7.2 Splay 维护序列

```
struct Splay
{
    #define type int
    const type inf=INF;
    const type zero=0;
    struct node
    {
        int ch[2],fa,sz,cnt,rev,tag;
        type v;
    }t[MAX];
    int tot,root;
    type a[MAX];
    queue<int> pool;
    void init_null_node()
    {
        memset(t[0].ch,0,sizeof t[0].ch);
        t[0].sz=t[0].cnt=t[0].fa=0;
        t[0].v=zero;
        t[0].mnid=0;
    }
    void init()
    {
        root=tot=0;
        while(!pool.empty()) pool.pop();
        init_null_node();
        a[0]=a[1]=zero;
        root=build(0,1,0);
    }
    int newnode(type v,int fa)
    {
        int id;
        if(pool.size()>0)
        {
            id=pool.front();
            pool.pop();
        }
        else id=++tot;
        memset(t[id].ch,0,sizeof t[id].ch);
        t[id].fa=fa;
        t[id].sz=t[id].cnt=1;
        t[id].tag=t[id].rev=0;
        t[id].v=v;
        return id;
    }
    void pushup(int id)
    {
        int ls=t[id].ch[0];
        int rs=t[id].ch[1];
        t[id].sz=t[ls].sz+t[rs].sz+t[id].cnt;


    }
    void pushdown(int id)
    {
        int ls=t[id].ch[0];
        int rs=t[id].ch[1];
        if(t[id].tag)
        {
            if(ls)
            {

                t[ls].tag=1;
            }
            if(rs)
            {

                t[rs].tag=1;
            }
            t[id].tag=0;
        }
        if(t[id].rev)
        {
            t[ls].rev^=1;
            t[rs].rev^=1;
            swap(t[ls].ch[0],t[ls].ch[1]);
            swap(t[rs].ch[0],t[rs].ch[1]);
            t[id].rev=0;
        }
    }
    void insert(int pos,vector<int> nums)
    {
        int x,y,z,i;
        for(i=0;i<nums.size();i++) a[i+1]=nums[i];
        z=build(1,nums.size(),0);
        x=find(pos);
        y=find(pos+1);
        splay(x,0);
        splay(y,x);
        t[y].ch[0]=z;
        t[z].fa=y;
        pushup(y);
        pushup(x);
    }
    void rotate(int x)
    {
        int y,z,k;
        y=t[x].fa;
```

```
        z=t[y].fa;
        k=(x==t[y].ch[1]);
        t[y].ch[k]=t[x].ch[k^1];
        if(t[x].ch[k^1]) t[t[x].ch[k^1]].fa=y;
        t[x].ch[k^1]=y;
        t[y].fa=x;
        t[x].fa=z;
        if(z) t[z].ch[y==t[z].ch[1]]=x;
        pushup(y);
        pushup(x);
    }
    void splay(int x,int goal)
    {
        int y,z;
        while(t[x].fa!=goal)
        {
            y=t[x].fa;
            z=t[y].fa;
            if(z!=goal)
            {
                if((t[z].ch[0]==y)^(t[y].ch[0]==x))
                    rotate(x);
                else rotate(y);
            }
            rotate(x);
        }
        if(goal==0) root=x;
    }
    int kth(int k)//k small
    {
        int id=root;
        while(id)
        {
            pushdown(id);
            if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
            else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
                break;
            else
            {
                k-=t[t[id].ch[0]].sz+t[id].cnt;
                id=t[id].ch[1];
            }
        }
        splay(id,0);
        return id;
    }
    int find(int x){return kth(x+1);}
    int build(int l,int r,int fa)
    {
        if(l>r) return 0;
        int id,mid;
        mid=(l+r)>>1;
        id=newnode(a[mid],fa);
        t[id].ch[0]=build(l,mid-1,id);
        t[id].ch[1]=build(mid+1,r,id);
        pushup(id);
        return id;
    }
    void _del(int id)
    {
        if(!id) return;
        pool.push(id);
        _del(t[id].ch[0]);
        _del(t[id].ch[1]);
    }
    void erase(int l,int r)
    {
        int x,fa;
        x=split(l,r);
        fa=t[x].fa;
        t[fa].ch[0]=0;
        _del(x);
        pushup(fa);
        pushup(t[fa].fa);
    }
    int split(int l,int r)
    {
        int x,y;
        x=find(l-1);
        y=find(r+1);
        splay(x,0);
        splay(y,x);
        return t[y].ch[0];
    }
    void rev(int l,int r)
    {
        int x,fa;
        x=split(l,r);
        fa=t[x].fa;
        t[x].rev^=1;
        swap(t[x].ch[0],t[x].ch[1]);
        pushup(fa);
        pushup(t[fa].fa);
    }
    int ask(int l,int r)
    {
        int x=split(l,r);
        return ;
    }
    int size(){return t[root].sz-2;}
    #undef type
}tr; //tr.init();
```

2.7.3 FHQ-Treap 维护序列

```
struct FHQ_Treap
{
    #define type int
```

```
    static const type inf=INF;
    struct node
    {
        int ls,rs,fix,sz;
        type v;
        node(){}
        node(type x,int _sz)
        {
            v=x;
            fix=rand();
            sz=_sz;
            ls=rs=0;
        }
    }t[MAX];
    int st[MAX];
    int revtag[MAX];
    int rt,tot;
    void init()
    {
        rt=0;
        srand(time(0));
        tot=0;
        t[0].sz=0;
        t[0].ls=t[0].rs=0;
    }
    int newnode(type v)
    {
        t[++tot]=node(v,1);
        revtag[tot]=0;
        return tot;
    }
    void pushup(int id)
    {
        t[id].sz=t[t[id].ls].sz+t[t[id].rs].sz+1;
    }
    void pushdown(int id)
    {
        if(revtag[id])
        {
            int ls=t[id].ls;
            int rs=t[id].rs;
            swap(t[ls].ls,t[ls].rs);
            swap(t[rs].ls,t[rs].rs);
            revtag[ls]^=revtag[id];
            revtag[rs]^=revtag[id];
            revtag[id]=0;
        }
    }
    int build(vector<type> &a)
    {
        int id,k,top=0;
        for(auto &it:a)
        {
            id=newnode(it);
            k=top;
            while(k>0&&t[st[k-1]].fix>t[id].fix)
                pushup(st[--k]);
            if(k) t[st[k-1]].rs=id;
            if(k<top) t[id].ls=st[k];
            st[k++]=id;
            top=k;
        }
        while(top>0) pushup(st[--top]);
        return st[0];
    }
    void split(int id,int pos,int &x,int &y)
    {
        if(!id)
        {
            x=y=0;
            return;
        }
        pushdown(id);
        int tmp=t[t[id].ls].sz+1;
        if(tmp<=pos)
        {
            x=id;
            split(t[id].rs,pos-tmp,t[id].rs,y);
        }
        else
        {
            y=id;
            split(t[id].ls,pos,x,t[id].ls);
        }
        pushup(id);
    }
    int merge(int x,int y)
    {
        if(!x||!y) return x|y;
        int id;
        if(t[x].fix<t[y].fix)
        {
            pushdown(x);
            id=x;
            t[x].rs=merge(t[x].rs,y);
        }
        else
        {
            pushdown(y);
            id=y;
            t[y].ls=merge(x,t[y].ls);
        }
        pushup(id);
        return id;
    }
    void insert(int pos,type v)
    {
        int ra,rb;
```

```
        split(rt,pos-1,ra,rb);
        rt=merge(merge(ra,newnode(v)),rb);
    }
    void insert(int pos,vector<type> a)
    {
        int ra,rb;
        split(rt,pos-1,ra,rb);
        rt=merge(merge(ra,build(a)),rb);
    }
    void erase(int pos,type v)
    {
        int ra,rb,rc;
        split(rt,pos,ra,rc);
        split(ra,pos-1,ra,rb);
        rt=merge(ra,rc);
    }
    void rev(int l,int r)
    {
        int ra,rb,rc;
        split(rt,r,ra,rc);
        split(ra,l-1,ra,rb);
        revtag[rb]^=1;
        swap(t[rb].ls,t[rb].rs);
        rt=merge(merge(ra,rb),rc);
    }
    vector<int> res;
    void dfs(int id)
    {
        if(!id) return;
        pushdown(id);
        dfs(t[id].ls);
        res.push_back(t[id].v);
        dfs(t[id].rs);
    }
    int size(){return t[rt].sz;}
    #undef type
}tr;
```

2.7.4 pbds

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
tree<int,
    null_type,
    less<int>,
    rb_tree_tag,
    tree_order_statistics_node_update> tr;
```

2.8 字典树

2.8.1 trie

```
struct Trie
{
    #define type int
    static const int K=26;
    int nex[MAX][K],cnt[MAX];
    int root,tot;
    int getid(char c){return c-'a';}
    int newnode()
    {
        tot++;
        memset(nex[tot],0,sizeof nex[tot]);
        cnt[tot]=0;
        return tot;
    }
    void init()
    {
        tot=0;
        memset(nex[0],0,sizeof nex[0]);
        cnt[0]=0;
        root=newnode();
    }
    void insert(char *s,int n) // s[0..n-1]
    {
        int now,i,t;
        now=root;
        for(i=0;i<n;i++)
        {
            t=getid(s[i]);
            if(!nex[now][t]) nex[now][t]=newnode();
            now=nex[now][t];
        }
        cnt[now]++;
    }
    #undef type
}tr;
/*
tr.init();
tr.insert(s,len); s[0..len-1]
*/
```

2.8.2 01trie

```
struct Trie
{
    #define type int
    static const int LOG=30;
    static const int K=LOG+2;
    static const type inf=INF;
    int rt,root[MAX],tot,ch[MAX*K][2],cnt[MAX*K];
    Trie &operator[](const int _rt){this->rt=_rt;
        return *this;}
    int newnode()
    {
```

```
    tot++;
    memset(ch[tot],0,sizeof ch[tot]);
    cnt[tot]=0;
    return tot;
}
void init(int n=1)
{
    ch[0][0]=ch[0][1]=cnt[0]=0;
    tot=1;
    for(int i=0;i<=n;i++) root[i]=0;
    rt=1;
}
void insert(type x)
{
    int id,t,i;
    if(!root[rt]) root[rt]=newnode();
    id=root[rt];
    for(i=LOG;~i;i--)
    {
        cnt[id]++;
        t=(x>>i)&1;
        if(!ch[id][t]) ch[id][t]=newnode();
        id=ch[id][t];
    }
    cnt[id]++;
}
type ask_max(type x)
{
    int id,i;
    type res,t;
    if(!root[rt]) return -inf;
    id=root[rt];
    res=0;
    for(i=LOG;~i;i--)
    {
        t=(x>>i)&1;
        if(ch[id][t^1]&&cnt[ch[id][t^1]]) t^=1;
        res|=(t<<i);
        id=ch[id][t];
    }
    return res^x;
}
type ask_min(type x)
{
    int id,i;
    type res,t;
    if(!root[rt]) return inf;
    id=root[rt];
    res=0;
    for(i=LOG;~i;i--)
    {
        t=(x>>i)&1;
        if(!ch[id][t]||!cnt[ch[id][t]]) t^=1;
        res|=(t<<i);
        id=ch[id][t];
    }
    return res^x;
}
#undef type
}tr;
```

2.9 可持久化

2.9.1 可持久化线段树

```
struct Persistent_Segment_Tree
{
    #define type int
    int root[MAX],ls[MAX<<5],rs[MAX<<5],tot,ql,qr,n;
    type v[MAX<<5],qv,res;
    void init(int _n)
    {
        n=_n;
        root[0]=0;
        ls[0]=rs[0]=v[0]=tot=0;
    }
    int copy_node(int x)
    {
        tot++;
        ls[tot]=ls[x];
        rs[tot]=rs[x];
        v[tot]=v[x];
        return tot;
    }
    void update(int l,int r,int &id,int pre)
    {
        if(!id) id=copy_node(pre);
        v[id]+=qv;
        if(l==r) return;
        int mid=(l+r)>>1;
        if(ql<=mid) update(l,mid,ls[id]=0,ls[pre]);
        else update(mid+1,r,rs[id]=0,rs[pre]);
    }
    void query_valsum(int l,int r,int id,int pre)
    {
        if(!id) return;
        if(l>=ql&&r<=qr)
        {
            res+=v[id]-v[pre];
            return;
        }
        int mid=(l+r)>>1;
        if(ql<=mid) query_valsum(l,mid,ls[id],ls[pre
            ]);
        if(qr>mid) query_valsum(mid+1,r,rs[id],rs[pre
            ]);
    }
    int kth_small(int l,int r,int id,int pre,int k)
```

```
    {
        if(l==r) return l;
        int mid=(l+r)>>1;
        int tmp=v[ls[id]]-v[ls[pre]];
        if(tmp>=k) return kth_small(l,mid,ls[id],ls[
            pre],k);
        else return kth_small(mid+1,r,rs[id],rs[pre],
            k-tmp);
    }
    void update_ver(int now_ver,int pre_ver,int pos,
        type v)
    {
        ql=qr=pos;
        qv=v;
        update(1,n,root[now_ver],root[pre_ver]);
    }
    void copy_ver(int now_ver,int pre_ver)
    {
        root[now_ver]=root[pre_ver];
    }
    void create_ver(int now_ver,int pre_ver,int pos,
        type v)
    {
        root[now_ver]=0;
        update_ver(now_ver,pre_ver,pos,v);
    }
    int ask_kth_small(int l,int r,int k)
    {
        return kth_small(1,n,root[r],root[l-1],k);
    }
    type ask_valsum(int pre_ver,int now_ver,int
        val_l,int val_r)
    {
        ql=val_l;
        qr=val_r;
        res=0;
        query_valsum(1,n,root[now_ver],root[pre_ver
            -1]);
        return res;
    }
    #undef type
}tr;
/*
tr.init(n);
tr.create_ver(now_ver,pre_ver,pos,v);
tr.copy_ver(now_ver,pre_ver);
tr.ask_kth_small(l,r,k);
tr.ask_valsum(l,r,val_l,val_r);
*/
```

2.9.2 可持久化 01trie

```
struct Persistent_01Trie
{
    #define type int
    static const int LOG=30;
    static const int K=LOG+2;
    int root[MAX],tot,nex[MAX*K][2],cnt[MAX*K];
    void init()
    {
        root[0]=0;
        nex[0][0]=nex[0][1]=cnt[0]=tot=0;
    }
    int newnode()
    {
        tot++;
        memset(nex[tot],0,sizeof nex[tot]);
        cnt[tot]=0;
        return tot;
    }
    void update(type x,int &rt,int pre)
    {
        int id,t,i;
        if(!rt) rt=newnode();
        id=rt;
        for(i=LOG;~i;i--)
        {
            cnt[id]=cnt[pre]+1;
            t=(x>>i)&1;
            nex[id][t^1]=nex[pre][t^1];
            nex[id][t]=newnode();
            id=nex[id][t];
            pre=nex[pre][t];
        }
        cnt[id]=cnt[pre]+1;
    }
    type query_max(type x,int rt,int pre)
    {
        int id,i;
        type res,t;
        id=rt;
        res=0;
        for(i=LOG;~i;i--)
        {
            t=(x>>i)&1;
            if(cnt[nex[id][t^1]]-cnt[nex[pre][t
                ^1]]>0) t^=1;
            res|=(t<<i);
            id=nex[id][t];
            pre=nex[pre][t];
        }
        return res^x;
    }
    void copy_ver(int now_ver,int pre_ver)
    {
        root[now_ver]=root[pre_ver];
    }
    void create_ver(int now_ver,int pre_ver,type x)
```

```
    {
        root[now_ver]=0;
        update(x,root[now_ver],root[pre_ver]);
    }
    type ask_max(int pre_ver,int now_ver,type x)
    {
        assert(pre_ver>0);
        return query_max(x,root[now_ver],root[pre_ver
            -1]);
    }
    #undef type
}tr;
/*
tr.init();
tr.create_ver(now_ver,pre_ver,v);
tr.copy_ver(now_ver,pre_ver);
tr.ask_max(l,r,v);
*/
```

2.9.3 可持久化数组

```
struct Persistent_Array
{
    #define type int
    struct node{int ls,rs;type v;}t[MAX<<5];
    int root[MAX],tot,qx,n;
    type a[MAX],qv;
    void init(int _n)
    {
        n=_n;
        tot=root[0]=0;
        t[0]={0,0,0};
    }
    void build(int l,int r,int &id)
    {
        if(!id) id=++tot;
        t[id]={0,0,0};
        if(l==r)
        {
            t[id]={0,0,a[l]};
            return;
        }
        int mid=(l+r)>>1;
        build(l,mid,t[id].ls);
        build(mid+1,r,t[id].rs);
    }
    int copy_node(int x)
    {
        t[++tot]=t[x];
        return tot;
    }
    void update(int l,int r,int &id,int pre)
    {
        if(!id) id=copy_node(pre);
        if(l==r)
        {
            t[id].v=qv;
            return;
        }
        int mid=(l+r)>>1;
        if(qx<=mid) update(l,mid,t[id].ls=0,t[pre].ls
            );
        else update(mid+1,r,t[id].rs=0,t[pre].rs);
    }
    type query(int l,int r,int id)
    {
        if(!id) return 0;
        if(l==r) return t[id].v;
        int mid=(l+r)>>1;
        if(qx<=mid) return query(l,mid,t[id].ls);
        else return query(mid+1,r,t[id].rs);
    }
    void build_ver(int now_ver){build(1,n,root[
        now_ver]=0);}
    void clear_ver(int now_ver){root[now_ver]=0;}
    void copy_ver(int now_ver,int pre_ver){root[
        now_ver]=root[pre_ver];}
    void create_ver(int now_ver,int pre_ver,int pos,
        type v)
    {
        root[now_ver]=0;
        qx=pos;
        qv=v;
        update(1,n,root[now_ver],root[pre_ver]);
    }
    void update_ver(int now_ver,int pos,type v)
    {
        qx=pos;
        qv=v;
        update(1,n,root[now_ver],root[now_ver]);
    }
    type ask(int now_ver,int pos)
    {
        qx=pos;
        return query(1,n,root[now_ver]);
    }
    #undef type
}pa;
/*
pa.init(n);
pa.build_ver(now_ver);
pa.clear_ver(now_ver);
pa.copy_ver(now_ver,pre_ver);
pa.create_ver(now_ver,pre_ver,pos,v);
pa.update_ver(now_ver,pos,v);
pa.ask(now_ver,pos);
*/
```

2.9.4 可持久化并查集

```
struct Persistent_DSU
{
    struct node{int ls,rs,fa,dep;}t[MAX<<5];
    int root[MAX],tot,n,qx,qop,qv;
    void build(int l,int r,int &id)
    {
        if(!id) id=++tot;
        t[id]={0,0,0,0};
        if(l==r)
        {
            t[id]={0,0,l,1};
            return;
        }
        int mid=(l+r)>>1;
        build(l,mid,t[id].ls);
        build(mid+1,r,t[id].rs);
    }
    void init(int _n,int init_ver)
    {
        n=_n;
        tot=root[0]=0;
        t[0]={0,0,0,0};
        build(1,n,root[init_ver]);
    }
    int copy_node(int x)
    {
        t[++tot]=t[x];
        return tot;
    }
    void update(int l,int r,int &id,int pre)
    {
        if(!id) id=copy_node(pre);
        if(l==r)
        {
            if(qop==1) t[id].fa=qv;
            else t[id].dep+=qv;
            return;
        }
        int mid=(l+r)>>1;
        if(qx<=mid) update(l,mid,t[id].ls=0,t[pre].ls
            );
        else update(mid+1,r,t[id].rs=0,t[pre].rs);
    }
    int query(int l,int r,int id)
    {
        if(!id) return 0;
        if(l==r) return id;
        int mid=(l+r)>>1;
        if(qx<=mid) return query(l,mid,t[id].ls);
        else return query(mid+1,r,t[id].rs);
    }
    void create_ver(int now_ver,int pre_ver,int pos,
        int v,int op)
    {
        root[now_ver]=0;
        qx=pos;
        qv=v;
        qop=op;
        update(1,n,root[now_ver],root[pre_ver]);
    }
    void update_ver(int now_ver,int pos,int v,int op
        )
    {
        qx=pos;
        qv=v;
        qop=op;
        update(1,n,root[now_ver],root[now_ver]);
    }
    int ask_id(int now_ver,int pos)
    {
        qx=pos;
        return query(1,n,root[now_ver]);
    }
    void clear_ver(int now_ver){root[now_ver]=0;}
    void copy_ver(int now_ver,int pre_ver){root[
        now_ver]=root[pre_ver];}
    int find_id(int now_ver,int x)
    {
        int id;
        while(1)
        {
            id=ask_id(now_ver,x);
            if(x==t[id].fa) return id;
            x=t[id].fa;
        }
        return 0;
    }
    int find(int now_ver,int x){return t[find_id(
        now_ver,x)].fa;}
    bool merge(int now_ver,int pre_ver,int a,int b)
    {
        int raid,rbid;
        raid=find_id(pre_ver,a);
        rbid=find_id(pre_ver,b);
        if(t[raid].fa==t[rbid].fa)
        {
            copy_ver(now_ver,pre_ver);
            return 0;
        }
        if(t[raid].dep>t[rbid].dep) swap(raid,rbid);
        if(now_ver==pre_ver) update_ver(now_ver,t[
            raid].fa,t[rbid].fa,1);
        else create_ver(now_ver,pre_ver,t[raid].fa,t[
            rbid].fa,1);
```

```
        if(t[raid].dep==t[rbid].dep) update_ver(
            now_ver,t[rbid].fa,1,2);
        return 1;
    }
}dsu;
/*
dsu.init(n,init_ver);
dsu.find(now_ver,x);
dsu.merge(now_ver,pre_ver,a,b);
dsu.clear_ver(now_ver);
dsu.copy_ver(now_ver,pre_ver);
*/
```

2.10 树套树

2.10.1 线段树套线段树

```
struct Segment_Tree_2D
{
    #define type int
    static const int insert_num=;
    static const int N=insert_num*20*20; //
        insert_num*20*log(m)
    int root[MAX<<2],tot,ls[N],rs[N],n,m;
    int ql_in,qr_in,ql_out,qr_out;
    type v[N],qv,tag[N];
    void init(int _n,int _m)
    {
        n=_n;
        m=_m;
        mem(root,0);
        ls[0]=rs[0]=0;
        tag[0]=0;
        v[0]=0;
        tot=1;
    }
    int newnode()
    {
        ls[tot]=rs[tot]=0;
        v[tot]=0;
        tag[tot]=0;
        return tot++;
    }
    void pushup(int id)
    {

    }
    void pushdown(int id)
    {
        if(!tag[id]) return;
        if(!ls[id]) ls[id]=newnode();
        if(!rs[id]) rs[id]=newnode();

        tag[id]=0;
    }
    void update_in(int l,int r,int &id)
    {
        if(!id) id=newnode();
        if(l>=ql_in&&r<=qr_in)
        {
            v[id]+=qv; //must update not =
            tag[id]+=qv;
            return;
        }
        pushdown(id);
        int mid=(l+r)>>1;
        if(ql_in<=mid) update_in(l,mid,ls[id]);
        if(qr_in>mid) update_in(mid+1,r,rs[id]);
        pushup(id);
    }
    type res_in;
    void query_in(int l,int r,int &id)
    {
        if(!id) return;
        if(l>=ql_in&&r<=qr_in)
        {
            res_in+=v[id];
            return;
        }
        pushdown(id);
        int mid=(l+r)>>1;
        type res=0;
        if(ql_in<=mid) query_in(l,mid,ls[id]);
        if(qr_in>mid) query_in(mid+1,r,rs[id]);
    }
    /*
        ********************************************
        */
    #define ls (id<<1)
    #define rs (id<<1|1)
    void update_out(int l,int r,int id)
    {
        update_in(1,m,root[id]);
        if(l>=ql_out&&r<=qr_out) return;
        int mid=(l+r)>>1;
        if(ql_out<=mid) update_out(l,mid,ls);
        if(qr_out>mid) update_out(mid+1,r,rs);
    }
    type res_out;
    void query_out(int l,int r,int id)
    {
        if(l>=ql_out&&r<=qr_out)
        {
            res_in=0;
            query_in(1,m,root[id]);
            res_out+=res_in;
            return;
```

```
        }
        int mid=(l+r)>>1;
        if(ql_out<=mid) query_out(l,mid,ls);
        if(qr_out>mid) query_out(mid+1,r,rs);
    }
    #undef ls
    #undef rs
    void upd(int x1,int y1,int x2,int y2,type val)
    {
        ql_out=x1;
        qr_out=x2;
        ql_in=y1;
        qr_in=y2;
        qv=val;
        update_out(1,n,1);
    }
    type ask(int x1,int y1,int x2,int y2)
    {
        ql_out=x1;
        qr_out=x2;
        ql_in=y1;
        qr_in=y2;
        res_out=0;
        query_out(1,n,1);
        return res_out;
    }
    #undef type
}tr2d;
```

2.10.2 线段树套 treap

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int INF=0x3f3f3f3f;
const int MAX=5e4+10;
struct Treap
{
    #define type int
    #define inf INF
    struct node
    {
        int ch[2],fix,sz,cnt;
        type v;
        node(){}
        node(type x,int _sz)
        {
            v=x;
            fix=rand();
            sz=cnt=_sz;
            ch[0]=ch[1]=0;
        }
    }t[MAX*40];
    int tot,root[MAX<<2],rt;
    void init(int n=1)
    {
        for(int i=0;i<=n;i++) root[i]=0;
        rt=1;
        srand(time(0));
        tot=0;
        t[0].sz=t[0].cnt=0;
        memset(t[0].ch,0,sizeof t[0].ch);
    }
    void pushup(int id)
    {
        t[id].sz=t[t[id].ch[0]].sz+t[t[id].ch[1]].sz+
            t[id].cnt;
    }
    void rotate(int &id,int k)
    {
        int y=t[id].ch[k^1];
        t[id].ch[k^1]=t[y].ch[k];
        t[y].ch[k]=id;
        pushup(id);
        pushup(y);
        id=y;
    }
    void _insert(int &id,type v,int cnt)
    {
        if(!id)
        {
            id=++tot;
            t[id]=node(v,cnt);
            return;
        }
        if(t[id].v==v) t[id].cnt+=cnt;
        else
        {
            int tmp=(v>t[id].v);
            _insert(t[id].ch[tmp],v,cnt);
            if(t[t[id].ch[tmp]].fix>t[id].fix) rotate
                (id,tmp^1);
        }
        pushup(id);
    }
    void _erase(int &id,type v,int cnt)
    {
        if(!id) return;
        if(t[id].v==v)
        {
            cnt=min(t[id].cnt,cnt);
            if(t[id].cnt>cnt)
            {
                t[id].cnt-=cnt;
                pushup(id);
                return;
            }
            if(!(t[id].ch[0]&&t[id].ch[1]))
```

```
        {
            id=t[id].ch[0]+t[id].ch[1];
            return;
        }
        else
        {
            int tmp=(t[t[id].ch[0]].fix>t[t[id].ch[1]].fix);
            rotate(id,tmp);
            _erase(t[id].ch[tmp],v,cnt);
            pushup(id);
        }
    }
    else
    {
        _erase(t[id].ch[v>t[id].v],v,cnt);
        pushup(id);
    }
}
int _find(type key,int f)
{
    int id=root[rt],res=0;
    while(id)
    {
        if(t[id].v<key)
        {
            res+=t[t[id].ch[0]].sz+t[id].cnt;
            if(f&&key==t[id].v) res-=t[id].cnt;
            id=t[id].ch[1];
        }
        else id=t[id].ch[0];
    }
    return res;
}
type find_by_order(int k)//k small
{
    int id=root[rt];
    if(id==0) return 0;
    while(id)
    {
        if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
        else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
            return t[id].v;
        else
        {
            k-=t[t[id].ch[0]].sz+t[id].cnt;
            id=t[id].ch[1];
        }
    }
}
int count(type key)
{
    int id=root[rt];
    while(id)
    {
        if(t[id].v<key)
        {
            if(key==t[id].v) return t[id].cnt;
            id=t[id].ch[1];
        }
        else id=t[id].ch[0];
    }
    return 0;
}
type find_pre(type key)
{
    type res=-inf;
    int id=root[rt];
    while(id)
    {
        if(t[id].v<key)
        {
            res=t[id].v;
            id=t[id].ch[1];
        }
        else id=t[id].ch[0];
    }
    return res;
}
type find_nex(type key)
{
    type res=inf;
    int id=root[rt];
    while(id)
    {
        if(t[id].v>key)
        {
            res=t[id].v;
            id=t[id].ch[0];
        }
        else id=t[id].ch[1];
    }
    return res;
}
Treap &operator[](const int _rt){this->rt=_rt;return *this;}
void insert(type v,int sz=1){_insert(root[rt],v,sz);}
void erase(type v,int sz=1){_erase(root[rt],v,sz);}
int upper_bound_count(type key){return _find(key,0);}//the count <=key
int lower_bound_count(type key){return _find(key,1);}//the count <key
int order_of_key(type key){return lower_bound_count(key)+1;}
int size(){return t[root[rt]].sz;}
}treap;
```

```
struct Segment_Tree
{
    #define ls (id<<1)
    #define rs (id<<1|1)
    int n,ql,qr,qop;
    type qv;
    void update(int l,int r,int id)
    {
        if(qop==1) treap[id].insert(qv);
        else treap[id].erase(qv);
        if(l>=ql&&r<=qr) return;
        int mid=(l+r)>>1;
        if(ql<=mid) update(l,mid,ls);
        if(qr>mid) update(mid+1,r,rs);
    }
    vector<int> treap_id;
    void dfs(int l,int r,int id)
    {
        if(l>=ql&&r<=qr)
        {
            treap_id.push_back(id);
            return;
        }
        int mid=(l+r)>>1;
        if(ql<=mid) dfs(l,mid,ls);
        if(qr>mid) dfs(mid+1,r,rs);
    }
    void get_treap_id(int l,int r)
    {
        ql=l;
        qr=r;
        treap_id.clear();
        dfs(1,n,1);
    }
    void build(int _n){n=_n;treap.init(n<<2);}
    void insert(int pos,type v)
    {
        ql=qr=pos;
        qop=1;
        qv=v;
        update(1,n,1);
    }
    void erase(int pos,type v)
    {
        ql=qr=pos;
        qop=2;
        qv=v;
        update(1,n,1);
    }
    int ask_rank(int l,int r,type v)
    {
        get_treap_id(l,r);
        int res=1;
        for(auto &rt:treap_id) res+=treap[rt].
            order_of_key(v)-1;
        return res;
    }
    int ask_kth(int l,int r,int k)
    {
        get_treap_id(l,r);
        l=0;
        r=1e8;
        while(l<r)
        {
            int mid=(l+r)>>1,now=1;
            for(auto &rt:treap_id) now+=treap[rt].
                order_of_key(mid+1)-1;
            if(now<=k) l=mid+1;
            else r=mid;
        }
        return l;
    }
    type find_pre(int l,int r,type v)
    {
        get_treap_id(l,r);
        type res=-inf;
        for(auto &rt:treap_id) res=max(res,treap[rt].
            find_pre(v));
        if(res==-inf) return -2147483647;
        return res;
    }
    type find_nex(int l,int r,type v)
    {
        get_treap_id(l,r);
        type res=inf;
        for(auto &rt:treap_id) res=min(res,treap[rt].
            find_nex(v));
        if(res==inf) return 2147483647;
        return res;
    }
    #undef type
    #undef ls
    #undef rs
}tr;
int a[MAX];
int main()
{
    int n,m,i,op,l,r,k;
    scanf("%d%d",&n,&m);
    tr.build(n);
    for(i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
        tr.insert(i,a[i]);
    }
    while(m--)
    {
```

```
        scanf("%d%d",&op,&l);
        if(op==3) scanf("%d",&k);
        else scanf("%d%d",&r,&k);
        if(op==1) printf("%d\n",tr.ask_rank(l,r,k));
        else if(op==2) printf("%d\n",tr.ask_kth(l,r,k
            ));
        else if(op==3)
        {
            tr.erase(l,a[l]);
            a[l]=k;
            tr.insert(l,a[l]);
        }
        else if(op==4) printf("%d\n",tr.find_pre(l,r,
            k));
        else if(op==5) printf("%d\n",tr.find_nex(l,r,
            k));
    }
    return 0;
}
```

2.10.3 树状数组套 treap

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int INF=0x3f3f3f3f;
const int MAX=1e5+10;
struct Treap
{
    #define type int
    #define inf INF
    struct node
    {
        int ch[2],fix,sz,cnt;
        type v;
        node(){}
        node(type x,int _sz)
        {
            v=x;
            fix=rand();
            sz=cnt=_sz;
            ch[0]=ch[1]=0;
        }
    }t[MAX*20];
    int tot,root[MAX],rt;
    void init(int n=1)
    {
        for(int i=0;i<=n;i++) root[i]=0;
        rt=1;
        srand(time(0));
        tot=0;
        t[0].sz=t[0].cnt=0;
        memset(t[0].ch,0,sizeof t[0].ch);
    }
    void pushup(int id)
    {
        t[id].sz=t[t[id].ch[0]].sz+t[t[id].ch[1]].sz+
            t[id].cnt;
    }
    void rotate(int &id,int k)
    {
        int y=t[id].ch[k^1];
        t[id].ch[k^1]=t[y].ch[k];
        t[y].ch[k]=id;
        pushup(id);
        pushup(y);
        id=y;
    }
    void _insert(int &id,type v,int cnt)
    {
        if(!id)
        {
            id=++tot;
            t[id]=node(v,cnt);
            return;
        }
        if(t[id].v==v) t[id].cnt+=cnt;
        else
        {
            int tmp=(v>t[id].v);
            _insert(t[id].ch[tmp],v,cnt);
            if(t[t[id].ch[tmp]].fix>t[id].fix) rotate
                (id,tmp^1);
        }
        pushup(id);
    }
    void _erase(int &id,type v,int cnt)
    {
        if(!id) return;
        if(t[id].v==v)
        {
            cnt=min(t[id].cnt,cnt);
            if(t[id].cnt>cnt)
            {
                t[id].cnt-=cnt;
                pushup(id);
                return;
            }
            if(!(t[id].ch[0]&&t[id].ch[1]))
            {
                id=t[id].ch[0]+t[id].ch[1];
                return;
            }
            else
            {
                int tmp=(t[t[id].ch[0]].fix>t[t[id].ch
                    [1]].fix);
                rotate(id,tmp);
```

```
            _erase(t[id].ch[tmp],v,cnt);
            pushup(id);
        }
    }
    else
    {
        _erase(t[id].ch[v>t[id].v],v,cnt);
        pushup(id);
    }
}
int _find(type key,int f)
{
    int id=root[rt],res=0;
    while(id)
    {
        if(t[id].v<key)
        {
            res+=t[t[id].ch[0]].sz+t[id].cnt;
            if(f&&key==t[id].v) res-=t[id].cnt;
            id=t[id].ch[1];
        }
        else id=t[id].ch[0];
    }
    return res;
}
type find_by_order(int k)//k small
{
    int id=root[rt];
    if(id==0) return 0;
    while(id)
    {
        if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];
        else if(t[t[id].ch[0]].sz+t[id].cnt>=k)
            return t[id].v;
        else
        {
            k-=t[t[id].ch[0]].sz+t[id].cnt;
            id=t[id].ch[1];
        }
    }
}
int count(type key)
{
    int id=root[rt];
    while(id)
    {
        if(t[id].v<key)
        {
            if(key==t[id].v) return t[id].cnt;
            id=t[id].ch[1];
        }
        else id=t[id].ch[0];
    }
    return 0;
}
type find_pre(type key)
{
    type res=-inf;
    int id=root[rt];
    while(id)
    {
        if(t[id].v<key)
        {
            res=t[id].v;
            id=t[id].ch[1];
        }
        else id=t[id].ch[0];
    }
    return res;
}
type find_nex(type key)
{
    type res=inf;
    int id=root[rt];
    while(id)
    {
        if(t[id].v>key)
        {
            res=t[id].v;
            id=t[id].ch[0];
        }
        else id=t[id].ch[1];
    }
    return res;
}
Treap &operator[](const int _rt){this->rt=_rt;
    return *this;}
void insert(type v,int sz=1){_insert(root[rt],v,
    sz);}
void erase(type v,int sz=1){_erase(root[rt],v,sz
    );}
int upper_bound_count(type key){return _find(key
    ,0);}//the count <=key
int lower_bound_count(type key){return _find(key
    ,1);}//the count <key
int order_of_key(type key){return
    lower_bound_count(key)+1;}
int size(){return t[root[rt]].sz;}
}treap;
struct Fenwick_Tree
{
    int n;
    void init(int _n)
    {
        n=_n;
        treap.init(n);
    }
    int lowbit(int x){return x&(-x);}
```

```
    type get(int x,type v)
    {
        type res=0;
        while(x)
        {
            res+=treap[x].lower_bound_count(v);
            x-=lowbit(x);
        }
        return res;
    }
    void insert(int x,type v)
    {
        while(x<=n)
        {
            treap[x].insert(v);
            x+=lowbit(x);
        }
    }
    void erase(int x,type v)
    {
        while(x<=n)
        {
            treap[x].erase(v);
            x+=lowbit(x);
        }
    }
    int ask_kth(int ql,int qr,int k)
    {
        int l,r,mid;
        l=0;
        r=1e9;
        while(l<r)
        {
            mid=(l+r)>>1;
            if(get(qr,mid+1)-get(ql-1,mid+1)+1<=k) l=
                mid+1;
            else r=mid;
        }
        return l;
    }
    #undef type
}tr;
int a[MAX];
int main()
{
    int n,m,i,l,r,k;
    char op[3];
    scanf("%d%d",&n,&m);
    tr.init(n);
    for(i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
        tr.insert(i,a[i]);
    }
    while(m--)
    {
        scanf("%s%d",op,&l);
        if(op[0]=='Q')
        {
            scanf("%d%d",&r,&k);
            printf("%d\n",tr.ask_kth(l,r,k));
        }
        else
        {
            scanf("%d",&k);
            tr.erase(l,a[l]);
            a[l]=k;
            tr.insert(l,a[l]);
        }
    }
    return 0;
}
```

2.11 李超树

```
struct LiChao_Segment_Tree
{
    #define type ll
    static const type inf=LLINF;
    #define ls (id<<1)
    #define rs (id<<1|1)
    static const int MIN_TAG=0;
    static const int MAX_TAG=1;
    int TAG;
    #define init_val (TAG==MIN_TAG?inf:-inf)
    struct line{type k,b;}sg[MAX<<2];
    type v[MAX<<2];
    bool ext[MAX<<2];
    int ql,qr,n;
    type cal(const line &l,const int &x){return l.k*
        x+l.b;}
    int sgn(const type &x){return x==0?0:(x>0?1:-1)
        ;}
    int cmp_min(const type &x,const type &y){return
        sgn(y-x);}
    int cmp_max(const type &x,const type &y){return
        sgn(x-y);}
    int cmp(const type &x,const type &y){return TAG
        ==MIN_TAG?cmp_min(x,y):cmp_max(x,y);}
    type ckres(const type &x,const type &y){return
        TAG==MIN_TAG?min(x,y):max(x,y);}
    void build(int l,int r,int id)
    {
        ext[id]=0;
        v[id]=init_val;
        sg[id]={0,init_val};
        if(l==r) return;
```

```
    int mid=(l+r)>>1;
    build(l,mid,ls);
    build(mid+1,r,rs);
}
void pushup(int id)
{
    v[id]=ckres(v[id],v[ls]);
    v[id]=ckres(v[id],v[rs]);
}
void pushdown(int l,int r,int id,line qv)
{
    int cl,cr,mid;
    if(!ext[id])
    {
        ext[id]=1;
        sg[id]=qv;
        goto pushdown_end;
    }
    mid=(l+r)>>1;
    if(cmp(cal(qv,mid),cal(sg[id],mid))==1) swap(qv,sg[id]);
    cl=cmp(cal(qv,l),cal(sg[id],l));
    cr=cmp(cal(qv,r),cal(sg[id],r));
    if(cl>=0&&cr>=0)
    {
        sg[id]=qv;
        goto pushdown_end;
    }
    if(cl==1) pushdown(l,mid,ls,qv);
    else if(cr==1) pushdown(mid+1,r,rs,qv);
pushdown_end:
    v[id]=ckres(cal(sg[id],l),cal(sg[id],r));
    if(l!=r) pushup(id);
}
void update(int l,int r,int id,line qv)
{
    if(l>=ql&&r<=qr)
    {
        pushdown(l,r,id,qv);
        return;
    }
    int mid=(l+r)>>1;
    if(ql<=mid) update(l,mid,ls,qv);
    if(qr>mid) update(mid+1,r,rs,qv);
    v[id]=ckres(cal(sg[id],l),cal(sg[id],r));
    pushup(id);
}
type res;
void query(int l,int r,int id)
{
    if(l>=ql&&r<=qr)
    {
        res=ckres(v[id],res);
        return;
    }
    res=ckres(res,cal(sg[id],max(l,ql)));
    res=ckres(res,cal(sg[id],min(r,qr)));
    int mid=(l+r)>>1;
    if(ql<=mid) query(l,mid,ls);
    if(qr>mid) query(mid+1,r,rs);
}
void build(int _n,int _TAG){n=_n;TAG=_TAG;build(1,n,1);}
void upd(int l,int r,type k,type b)
{
    ql=l;
    qr=r;
    update(1,n,1,{k,b});
}
type ask(int l,int r)
{
    ql=l;
    qr=r;
    res=init_val;
    query(1,n,1);
    return res;
}
void dfs(int l,int r,int id)
{
    cout<<l<<" "<<r<<" "<<sg[id].k<<" "<<sg[id].b<<endl;
    if(l==r) return;
    int mid=(l+r)>>1;
    dfs(l,mid,ls);
    dfs(mid+1,r,rs);
}
#undef type
#undef init_val
#undef ls
#undef rs
}tr;
/*
upd:O(log^2), ask:O(log)

tr.build(n,LiChao_Segment_Tree::MIN_TAG);
tr.build(n,LiChao_Segment_Tree::MAX_TAG);
tr.upd(l,r,k,b);
tr.ask(l,r);
*/
```

2.12 kd-tree

```
namespace kd_tree
{
    const double alpha=0.75;
    const int dim=2;
    #define type int
```

```
const type NONE=INF; //初始值
struct kdtnode
{
    bool exist;
    int l,r,sz,fa,dep,x[dim],mx[dim],mn[dim];
    type v,tag;
    kdtnode(){}
    void initval()
    {
        sz=exist;tag=v;
        if(exist) for(int i=0;i<dim;i++) mn[i]=mx[i]=x[i];
    }
    void null()
    {
        exist=sz=0;
        v=tag=NONE;
        for(int i=0;i<dim;i++)
        {
            mx[i]=-INF;
            mn[i]=INF;
        }
    }
    void newnode(int x0,int x1,type val=NONE)
    {
        x[0]=x0;
        x[1]=x1;
        l=r=fa=0;
        exist=1;
        v=val;
        initval();
    }
    kdtnode(int a,int b,type d=NONE){newnode(a,b,d);}
};
struct KDT
{
    #define ls t[id].l
    #define rs t[id].r
    kdtnode t[MAX];
    int tot,idx,root;
    inline void pushup(int id)
    {
        t[id].initval();
        t[id].sz+=t[ls].sz+t[rs].sz;
        t[id].tag=min({t[ls].tag,t[rs].tag,t[id].tag});
        for(int i=0;i<dim;i++)
        {
            if(ls)
            {
                t[id].mx[i]=max(t[id].mx[i],t[ls].mx[i]);
                t[id].mn[i]=min(t[id].mn[i],t[ls].mn[i]);
            }
            if(rs)
            {
                t[id].mx[i]=max(t[id].mx[i],t[rs].mx[i]);
                t[id].mn[i]=min(t[id].mn[i],t[rs].mn[i]);
            }
        }
    }
    bool isbad(int id){return t[id].sz*alpha+3<max(t[ls].sz,t[rs].sz);}
    int st[MAX],top;
    void build(int &id,int l,int r,int fa,int dep=0)
    {
        id=0;if(l>r) return;
        int m=(l+r)>>1; idx=dep;
        nth_element(st+l,st+m,st+r+1,[&](int x,int y){return t[x].x[idx]<t[y].x[idx];});
        id=st[m];
        build(ls,l,m-1,id,(dep+1)%dim);
        build(rs,m+1,r,id,(dep+1)%dim);
        pushup(id);
        t[id].dep=dep;
        t[id].fa=fa;
    }
    inline void init(int n=0)
    {
        root=0;
        t[0].null();
        for(int i=1;i<=n;i++) st[i]=i;
        if(n) build(root,1,n,0);
        tot=n;
    }
    void travel(int id)
    {
        if(!id) return;
        if(t[id].exist) st[++top]=id;
        travel(ls);
        travel(rs);
    }
    void rebuild(int &id,int dep)
    {
        top=0;travel(id);
        build(id,1,top,t[id].fa,dep);
    }
    void insert(int &id,int now,int fa,int dep=0)
    {
        if(!id)
        {
```

```
            id=now;
            t[id].dep=dep;
            t[id].fa=fa;
            return;
        }
        if(t[now].x[dep]<t[id].x[dep]) insert(ls,now,id,(dep+1)%dim);
        else insert(rs,now,id,(dep+1)%dim);
        pushup(id);
        if(isbad(id)) rebuild(id,t[id].dep);
        t[id].dep=dep;
        t[id].fa=fa;
    }
    inline void insert(kdtnode x){t[++tot]=x;insert(root,tot,0,0);}
    inline void del(int id)
    {
        if(!id) return;
        t[id].null();
        int x=id;
        while(x)
        {
            pushup(x);
            x=t[x].fa;
        }
        if(isbad(id))
        {
            x=t[id].fa;
            rebuild(root==id?root:(t[x].l==id?t[x].l:t[x].r),t[id].dep);
        }
    }
    kdtnode q;
    ll dist(ll x,ll y){return x*x+y*y;}
    ll getdist(int id)//点离区域qt[id最短距离]
    {
        if(!id) return LLINF;
        ll res=0;
        if(q.x[0]<t[id].mn[0]) res+=dist(q.x[0]-t[id].mn[0],0);
        if(q.x[1]<t[id].mn[1]) res+=dist(q.x[1]-t[id].mn[1],0);
        if(q.x[0]>t[id].mx[0]) res+=dist(q.x[0]-t[id].mx[0],0);
        if(q.x[1]>t[id].mx[1]) res+=dist(q.x[1]-t[id].mx[1],0);
        return res;
    }
    kdtnode a,b;
    inline int check(kdtnode &x)//在矩形x(a,b)内
    {
        int ok=1;
        for(int i=0;i<dim;i++)
        {
            ok&=(x.x[i]>=a.x[i]);
            ok&=(x.x[i]<=b.x[i]);
        }
        return ok;
    }
    inline int allin(kdtnode &x)//的子树全在矩形x(a,b)内
    {
        int ok=1;
        for(int i=0;i<dim;i++)
        {
            ok&=(x.mn[i]>=a.x[i]);
            ok&=(x.mx[i]<=b.x[i]);
        }
        return ok;
    }
    inline int allout(kdtnode &x)//的子树全不在矩形x(a,b)内
    {
        int ok=0;
        for(int i=0;i<dim;i++)
        {
            ok|=(x.mx[i]<a.x[i]);
            ok|=(x.mn[i]>b.x[i]);
        }
        return ok;
    }
    type res;
    void query(int id)
    {
        if(!id) return;
        if(allout(t[id])||t[id].sz==0) return;
        if(allin(t[id]))
        {
            res=min(res,t[id].tag);
            return;
        }
        if(check(t[id])&&t[id].exist) res=min(res,t[id].v);
        int l,r;
        l=ls;
        r=rs;
        if(t[l].tag>t[r].tag) swap(l,r);
        if(t[l].tag<res) query(l);
        if(t[r].tag<res) query(r);
    }
    inline type query(kdtnode _a,kdtnode _b)
    {
        a=_a;b=_b;
        res=INF;
        query(root);
        return res;
    }
}kd;
#undef type
```

```
    #undef ls
    #undef rs
}
using namespace kd_tree;
```

2.13 可并堆

2.13.1 pbds 可并堆

```
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
__gnu_pbds::priority_queue<int,greater<int>,
    pairing_heap_tag> q[MAX];
//q[i].join(q[j]) -> q[j]c º¢µ½q[i],q[j???]
```

2.13.2 左偏树 (支持打 tag, 不支持删除)

```
struct Leftist_Tree
{
    #define type ll
    struct node
    {
        int ls,rs,d,id;
        type v;
        void init(){ls=rs=d=id=v=0;}
    }t[MAX];
    int root[MAX],rt,tot,sz[MAX];
    type tagadd[MAX],tagmul[MAX];
    leftist_tree &operator[](const int _rt){this->rt
        =_rt;return *this;}
    void init(int n=1)
    {
        rt=1;
        tot=0;
        t[0].init();
        sz[0]=0;
        for(int i=0;i<=n;i++) root[i]=0;
    }
    int newnode(type v,int p)
    {
        int id=++tot;
        t[id].init();
        t[id].id=p;
        t[id].v=v;
        sz[id]=1;
        tagadd[id]=0;
        tagmul[id]=1;
        return id;
    }
    void pushdown(int id)
    {
        int ls=t[id].ls;
        int rs=t[id].rs;
        if(tagmul[id]!=1)
        {
            if(ls)
            {
                t[ls].v*=tagmul[id];
                tagmul[ls]*=tagmul[id];
                tagadd[ls]*=tagmul[id];
            }
            if(rs)
            {
                t[rs].v*=tagmul[id];
                tagmul[rs]*=tagmul[id];
                tagadd[rs]*=tagmul[id];
            }
            tagmul[id]=1;
        }
        if(tagadd[id])
        {
            if(ls)
            {
                t[ls].v+=tagadd[id];
                tagadd[ls]+=tagadd[id];
            }
            if(rs)
            {
                t[rs].v+=tagadd[id];
                tagadd[rs]+=tagadd[id];
            }
            tagadd[id]=0;
        }
    }
    int merge(int x,int y)
    {
        if(x==y) return x;
        if(!x||!y) return x+y;
        if(t[x].v>t[y].v) swap(x,y); // > small, <
            big
        pushdown(x);
        t[x].rs=merge(t[x].rs,y);
        if(t[t[x].rs].d>t[t[x].ls].d) swap(t[x].ls,t[
            x].rs);
        t[x].d=t[t[x].rs].d+1;
        sz[x]=sz[t[x].ls]+sz[t[x].rs]+1;
        return x;
    }
    void clear(){root[rt]=0;}
    void join(int y)
    {
        if(rt==y) return;
        root[rt]=merge(root[rt],root[y]);
        root[y]=0;
    }
    void push(type v,int p=0){root[rt]=merge(root[rt
        ],newnode(v,p));}
```

```
    void pop()
    {
        int x=root[rt];
        pushdown(x);
        root[rt]=merge(t[x].ls,t[x].rs);
    }
    node top(){return t[root[rt]];}
    bool empty(){return !root[rt];}
    int size(){return sz[root[rt]];}
    void upd(int op,type v)
    {
        int x=root[rt];
        if(op==0)
        {
            tagadd[x]+=v;
            t[x].v+=v;
        }
        else
        {
            tagadd[x]*=v;
            tagmul[x]*=v;
            t[x].v*=v;
        }
    }
    #undef type
}q;
/*
q.init();
q.init(n);
q[x].push(x);
q[x].push(x,id);
q[x].top();
q[x].pop();
q[x].join(y);
q[x].clear();
*/
```

2.13.3 左偏树 (支持删除, 不支持打 tag)

```
struct leftist_tree
{
    #define type int
    struct node
    {
        int ls,rs,d,fa,id;
        type v;
        void init(){ls=rs=d=fa=id=v=0;}
    }t[MAX];
    int root[MAX],rt,tot,pos[MAX];
    int st[MAX],sttop;
    leftist_tree &operator[](const int _rt){this->rt
        =_rt;return *this;}
    void init(int n=1)
    {
        rt=1;
        tot=0;
        t[0].init();
        for(int i=0;i<=n;i++) root[i]=0;
    }
    int newnode(type v,int p)
    {
        int id=sttop?st[--sttop]:++tot;
        t[id].init();
        t[id].v=v;
        t[id].id=p;
        pos[p]=id;
        return id;
    }
    void delnode(int id){st[sttop++]=id;}
    int merge(int x,int y)
    {
        if(x==y) return x;
        if(!x||!y) return x+y;
        if(t[x].v<t[y].v) swap(x,y); // > small, <
            big
        t[x].rs=merge(t[x].rs,y);
        t[t[x].rs].fa=x;
        if(t[t[x].rs].d>t[t[x].ls].d) swap(t[x].ls,t[
            x].rs);
        t[x].d=t[t[x].rs].d+1;
        return x;
    }
    void erase_by_id(int id)
    {
        int x,y;
        x=pos[id];
        if(root[rt]==x)
        {
            pop();
            return;
        }
        delnode(x);
        pos[id]=0;
        y=merge(t[x].ls,t[x].rs);
        t[y].fa=t[x].fa;
        if(t[t[x].fa].ls==x) t[t[x].fa].ls=y;
        else if(t[t[x].fa].rs==x) t[t[x].fa].rs=y;
        x=t[y].fa;
        while(x)
        {
            if(t[x].d==t[t[x].rs].d+1) break;
            t[x].d=t[t[x].rs].d+1;
            x=t[x].fa;
        }
    }
    void del(int x)
    {
        if(!x) return;
```

```
        del(t[x].ls);
        del(t[x].rs);
        delnode(x);
    }
    void clear(){del(root[rt]);root[rt]=0;}
    void join(int y)
    {
        if(rt==y) return;
        root[rt]=merge(root[rt],root[y]);
        root[y]=0;
    }
    void push(type v,int p=0){root[rt]=merge(root[rt
        ],newnode(v,p));}
    void pop()
    {
        int x=root[rt];
        pos[t[x].id]=0;
        root[rt]=merge(t[x].ls,t[x].rs);
        delnode(x);
    }
    node top(){return t[root[rt]];}
    bool empty(){return !root[rt];}
    void dfs(int x,type v)
    {
        if(!x) return;
        t[x].v+=v;
        dfs(t[x].ls,v);
        dfs(t[x].rs,v);
    }
    void updall(type v){dfs(root[rt],v);}
    node get_by_id(int id){return t[pos[id]];}
    #undef type
}q;
/*
q.init();
q.init(n);
q[x].push(x);
q[x].push(x,id);
q[x].top();
q[x].pop();
q[x].join(y);
q[x].clear();
q[x].erase_by_id(id);
*/
```

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

```
struct k_Huffman
{
    #define type ll
    type work(int n,int k,type *a)// a[1..n], sorted
    {
        int i;
        type res,s;
        queue<type> q,d;
        s=((n-1)%(k-1)?k-1-(n-1)%(k-1):0);//
            ????Ç?????¼²¹¶0
        while(s--) q.push(0);
        for(i=1;i<=n;i++) q.push(a[i]);
        res=0;
        while(q.size()+d.size()>1)
        {
            s=0;
            for(i=0;i<k;i++)
            {
                if(q.size()&&d.size())
                {
                    if(q.front()<=d.front())
                    {
                        s+=q.front();
                        q.pop();
                    }
                    else
                    {
                        s+=d.front();
                        d.pop();
                    }
                }
                else if(q.size())
                {
                    s+=q.front();
                    q.pop();
                }
                else if(d.size())
                {
                    s+=d.front();
                    d.pop();
                }
            }
            res+=s;
            d.push(s);
        }
        return res;
    }
    #undef type
}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

```
struct Cartesian_Tree
{
    int l[MAX],r[MAX],vis[MAX],stk[MAX];
    int build(int *a,int n)
    {
        int i,top=0;
        for(i=1;i<=n;i++) l[i]=r[i]=vis[i]=0;
        for(i=1;i<=n;i++)
        {
            int k=top;
            while(k>0&&a[stk[k-1]]>a[i]) k--;
            if(k) r[stk[k-1]]=i;
            if(k<top) l[i]=stk[k];
            stk[k++]=i;
            top=k;
        }
        for(i=1;i<=n;i++) vis[l[i]]=vis[r[i]]=1;
        for(i=1;i<=n;i++)
        {
            if(!vis[i]) return i;
        }
    }
}ct;
```

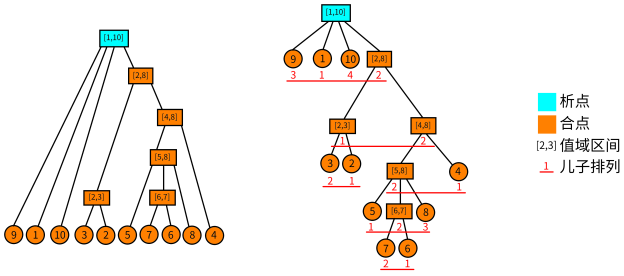
2.16 析合树

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：




```
struct Permutation_Tree
{
    struct RMQ
    {
        #define type int
        int lg[MAX],bin[22];
        type mx[MAX][22],mn[MAX][22];
        void work(int n,type *v)
        {
            int i,j;
            for(i=bin[0]=1;1<<(i-1)<=n;i++) bin[i]=(bin[i-1]<<1);
            for(i=2,lg[1]=0;i<=n;i++) lg[i]=lg[i>>1]+1;
            for(i=1;i<=n;i++) mx[i][0]=mn[i][0]=v[i];
            for(j=1;1<<(j-1)<=n;j++)
            {
                for(i=1;i+bin[j]-1<=n;i++)
                {
                    mx[i][j]=max(mx[i][j-1],mx[i+bin[j-1]][j-1]);
                    mn[i][j]=min(mn[i][j-1],mn[i+bin[j-1]][j-1]);
                }
            }
        }
        type ask_max(int l,int r)
        {
            int t=lg[r-l+1];
            return max(mx[l][t],mx[r-bin[t]+1][t]);
        }
        type ask_min(int l,int r)
        {
            int t=lg[r-l+1];
            return min(mn[l][t],mn[r-bin[t]+1][t]);
        }
        #undef type
    }rmq;
    struct Segment_Tree
    {
        #define type int
        #define ls (id<<1)
        #define rs (id<<1|1)
        int n,ql,qr;
        type mn[MAX<<2],tag[MAX<<2],qv;
        void mdf(int id,type v){mn[id]+=v,tag[id]+=v;}
        void pushup(int id){mn[id]=min(mn[ls],mn[rs]);}
        void pushdown(int id)
        {
            if(!tag[id]) return;
            mdf(ls,tag[id]);
            mdf(rs,tag[id]);
            tag[id]=0;
        }
        void build(int l,int r,int id)
        {
```

```
        tag[id]=mn[id]=0;
        if(l==r) return;
        int mid=(l+r)>>1;
        build(l,mid,ls);
        build(mid+1,r,rs);
        pushup(id);
    }
    void update(int l,int r,int id)
    {
        if(l>=ql&&r<=qr){mdf(id,qv);return;}
        pushdown(id);
        int mid=(l+r)>>1;
        if(ql<=mid) update(l,mid,ls);
        if(qr>mid) update(mid+1,r,rs);
        pushup(id);
    }
    int query(int l,int r,int id)
    {
        if(l==r) return l;
        pushdown(id);
        int mid=(l+r)>>1;
        if(!mn[ls]) return query(l,mid,ls);
        else query(mid+1,r,rs);
    }
    void build(int _n){n=_n;build(1,n,1);}
    void upd(int l,int r,type v){ql=l;qr=r;qv=v;
        update(1,n,1);}
    type ask(int l,int r){ql=l;qr=r;return query
        (1,n,1);}
    #undef type
    #undef ls
    #undef rs
}tr;
bool check(int l,int r){return rmq.ask_max(l,r)-
    rmq.ask_min(l,r)==r-l;}
int st[MAX],st1[MAX],st2[MAX],top,top1,top2,m[
    MAX];
int tot,id[MAX],l[MAX],r[MAX],type[MAX];
VI mp[MAX];
void add_edge(int a,int b){mp[a].pb(b);}
int build(int n,int *a)
{
    int now,i,tmp;
    tr.build(n);
    rmq.work(n,a);
    for(i=0;i<=2*n;i++)
    {
        mp[i].clear();
        type[i]=0;
    }
    top=top1=top2=0;
    tot=0;
    for(i=1;i<=n;i++)
    {
        while(top1&&a[i]<=a[st1[top1]])
        {
            tr.upd(st1[top1-1]+1,st1[top1],a[st1[
                top1]]);
            top1--;
        }
        while(top2&&a[i]>=a[st2[top2]])
        {
            tr.upd(st2[top2-1]+1,st2[top2],-a[st2[
                top2]]);
            top2--;
        }
        tr.upd(st1[top1]+1,i,-a[i]);
        st1[++top1]=i;
        tr.upd(st2[top2]+1,i,a[i]);
        st2[++top2]=i;
        id[i]=++tot;
        l[tot]=r[tot]=i;
        tmp=tr.ask(1,n);
        now=tot;
        while(top&&l[st[top]]>=tmp)
        {
            if(type[st[top]]&&check(m[st[top]],i))
            {
                r[st[top]]=i;
                add_edge(st[top],now);
                now=st[top--];
            }
            else if(check(l[st[top]],i))
            {
                type[++tot]=1;
                l[tot]=l[st[top]];
                r[tot]=i;
                m[tot]=l[now];
                add_edge(tot,st[top--]);
                add_edge(tot,now);
                now=tot;
            }
            else
            {
                add_edge(++tot,now);
                do
                {
                    add_edge(tot,st[top--]);
                }while(top&&!check(l[st[top]],i));
                l[tot]=l[st[top]];
                r[tot]=i;
                add_edge(tot,st[top--]);
                now=tot;
            }
        }
        st[++top]=now;
        tr.upd(1,i,-1);
    }
```

```
        return st[1];
    }
    void work(int n,int *a)// a[1..n]
    {
        int rt=build(n,a);

    }
}pt;// MAX must *2
```

2.17 莫队算法

```
struct MO_Algorithm
{
    #define type int
    struct query_info{int l,r,id;type v;};
    vector<query_info> qst;
    int n,q,a[MAX];
    type ans[MAX],res;
    void init(int _n,int _q)
    {
        qst.clear();
        n=_n;
        q=_q;
    }
    void add_qst(int l,int r,int id,type v=0){qst.
        push_back({l,r,id,v});}
    void add(int x)
    {

    }
    void del(int x)
    {

    }
    void work()
    {
        int i,l,r,sq;
        sq=sqrt(q);
        sort(qst.begin(),qst.end(),[&](query_info a,
            query_info b){
            if(a.l/sq!=b.l/sq) return a.l/sq<b.l/sq;
            if((a.l/sq)&1) return a.r>b.r;
            else return a.r<b.r;
        });

        for(i=1;i<=q;i++) ans[i]=0;
        l=1;
        r=0;
        res=0;
        for(auto &q:qst)
        {
            while(l<q.l) del(l++);
            while(l>q.l) add(--l);
            while(r<q.r) add(++r);
            while(r>q.r) del(r--);
            ans[q.id]=res;
        }
    }
    #undef type
}mo;
/*
O(n*sqrt(q))
mo.init(n,q);
mo.add_qst(l,r,id);
mo.work();
*/
```

2.18 ODT

```
struct ODT
{
    #define type int
    #define init_val 0
    struct ODT_node
    {
        int l,r;
        mutable type v;
        inline bool operator<(const ODT_node &o)const
            {return l<o.l;}
    };
    int n;
    set<ODT_node> odt;
    typedef set<ODT_node>::iterator odt_iter;
    void init(int _n)
    {
        n=_n;
        odt.clear();
        odt.insert({1,n,init_val});
    }
    odt_iter find(int x){return --odt.upper_bound({x
        ,0,init_val});}
    odt_iter split(int x)
    {
        if(x>n) return odt.end();
        odt_iter it=find(x);
        if(it->l==x) return it;
        int l=it->l,r=it->r;
        type v=it->v;
        odt.erase(it);
        odt.insert({l,x-1,v});
        return odt.insert({x,r,v}).first;
    }
    void assign(int l,int r,type v)
    {
        odt_iter itr=split(r+1),itl=split(l);
        odt.erase(itl,itr);
```

```
        odt.insert({l,r,v});
    }
    #undef init_val
    #undef type
}odt;
```

2.19 分块

```
struct Block
{
    #define type int
    static const int N=MAX;
    static const int SZ=sqrt(N);
    static const int B=N/SZ+10;
    int n,bl[B],br[B],bid[N];
    type a[N],sum[B],tag[B];
    void build(int _n)
    {
        int i,j,id;
        n=_n;
        id=0;
        for(i=1;i<=n;i+=SZ)
        {
            bl[++id]=i;
            br[id]=min(n,i+SZ-1);
            tag[id]=0;
            for(j=bl[id];j<=br[id];j++)
            {
                bid[j]=id;

            }
        }
    }
    void pushdown(int id)
    {
        int i;
        if(tag[id])
        {

        }
    }
    void upd_point(int x,type v)
    {

    }
    void upd_block(int id,type v)
    {

    }
    void upd(int l,int r,type v)
    {
        int i;
        if(bid[l]==bid[r])
        {
            pushdown(bid[l]);
            for(i=l;i<=r;i++) upd_point(i,v);
            return;
        }
        pushdown(bid[l]);
        pushdown(bid[r]);
        for(i=l;i<=br[bid[l]];i++) upd_point(i,v);
        for(i=bl[bid[r]];i<=r;i++) upd_point(i,v);
        for(i=bid[l]+1;i<=bid[r]-1;i++) upd_block(i,v
            );
    }
    type res;
    void ask_point(int x)
    {

    }
    void ask_block(int id)
    {

    }
    type ask(int l,int r)
    {
        int i;
        res=0;
        if(bid[l]==bid[r])
        {
            pushdown(bid[l]);
            for(i=l;i<=r;i++) ask_point(i);
            return res;
        }
        pushdown(bid[l]);
        pushdown(bid[r]);
        for(i=l;i<=br[bid[l]];i++) ask_point(i);
        for(i=bl[bid[r]];i<=r;i++) ask_point(i);
        for(i=bid[l]+1;i<=bid[r]-1;i++) ask_block(i);
        return res;
    }
    #undef type
}blk;
```

3 树

3.1 LCA

3.1.1 倍增 LCA

```
struct LCA
{
    static const int N=MAX;
    static const int LOG=log2(N)+2;
    int fa[LOG][N],dep[N];
    vector<int> *mp;
```

```
    void dfs(int x,int pre)
    {
        int i;
        fa[0][x]=pre;
        for(i=1;i<LOG;i++) fa[i][x]=fa[i-1][fa[i-1][x
            ]];
        for(auto &to:mp[x])
        {
            if(to==pre) continue;
            dep[to]=dep[x]+1;
            dfs(to,x);
        }
    }
    void work(int root,vector<int> *_mp)
    {
        mp=_mp;
        for(int j=0;j<LOG;j++) fa[j][0]=0;
        dep[root]=0;
        dfs(root,0);
    }
    int go(int x,int d)
    {
        for(int i=0;i<LOG;i++)
        {
            if((d>>i)&1) x=fa[i][x];
        }
        return x;
    }
    int lca(int x,int y)
    {
        if(dep[x]<dep[y]) swap(x,y);
        x=go(x,dep[x]-dep[y]);
        if(x==y) return x;
        for(int i=LOG-1;~i;i--)
        {
            if(fa[i][x]!=fa[i][y])
            {
                x=fa[i][x];
                y=fa[i][y];
            }
        }
        return fa[0][x];
    }
}lca;
/*
O(nlogn)-O(logn)
lca.work(n,root,mp);
*/
```

3.1.2 RMQ 维护欧拉序求 LCA

```
struct LCA
{
    static const int N=MAX;
    static const int LOG2N=log2(2*N)+3;
    #define type int
    struct node{int to;type w;};
    type dis[N];
    int path[2*N],deep[2*N],first[N],len[N],tot,n;
    int dp[2*N][LOG2N];
    vector<node> mp[N];
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++)
        {
            dis[i]=len[i]=0;
            mp[i].clear();
        }
    }
    void add_edge(int a,int b,type w=1){mp[a].
        push_back({b,w});}
    void dfs(int x,int pre,int h)
    {
        int i;
        path[++tot]=x;
        first[x]=tot;
        deep[tot]=h;
        for(i=0;i<mp[x].size();i++)
        {
            int to=mp[x][i].to;
            if(to==pre) continue;
            dis[to]=dis[x]+mp[x][i].w;
            len[to]=len[x]+1;
            dfs(to,x,h+1);
            path[++tot]=x;
            deep[tot]=h;
        }
    }
    void ST(int n)
    {
        int i,j,x,y;
        for(i=1;i<=n;i++) dp[i][0]=i;
        for(j=1;(1<<j)<=n;j++)
        {
            for(i=1;i+(1<<j)-1<=n;i++)
            {
                x=dp[i][j-1];
                y=dp[i+(1<<(j-1))][j-1];
                dp[i][j]=deep[x]<deep[y]?x:y;
            }
        }
    }
    int query(int l,int r)
    {
        int len,x,y;
        len=__lg(r-l+1);
        x=dp[l][len];
```

```
        y=dp[r-(1<<len)+1][len];
        return deep[x]<deep[y]?x:y;
    }
    int lca(int x,int y)
    {
        int l,r,pos;
        l=first[x];
        r=first[y];
        if(l>r) swap(l,r);
        pos=query(l,r);
        return path[pos];
    }
    type get_dis(int a,int b){return dis[a]+dis[b
        ]-2*dis[lca(a,b)];}
    int get_len(int a,int b){return len[a]+len[b]-2*
        len[lca(a,b)];}
    void work(int rt)
    {
        tot=0;
        dfs(rt,0,0);
        ST(2*n-1);
    }
    int lca_root(int rt,int x,int y)
    {
        int fx,fy;
        fx=lca(x,rt);
        fy=lca(y,rt);
        if(fx==fy) return lca(x,y);
        else
        {
            if(get_len(fx,rt)<get_len(fy,rt)) return
                fx;
            else return fy;
        }
    }
    #undef type
}lca;
/*
O(n*log(n))-O(1)

lca.init(n);
lca.add_edge(a,b,w);
lca.work(rt);
*/
```

3.2 树链剖分

3.2.1 轻重链剖分

```
struct Heavy_Light_Decomposition
{
    #define type int
    struct edge{int a,b;type v;};
    vector<int> mp[MAX];
    vector<edge> e;
    int dep[MAX],fa[MAX],sz[MAX],son[MAX];
    int id[MAX],top[MAX],dfn[MAX],tot;
    int n,rt;
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++) mp[i].clear();
        e.clear();
        e.push_back({0,0,0});
    }
    void add_edge(int a,int b,type v=0)
    {
        e.push_back({a,b,v});
        mp[a].push_back(b);
        mp[b].push_back(a);
    }
    void dfs1(int x,int pre,int h)
    {
        int i,to;
        dep[x]=h;
        fa[x]=pre;
        sz[x]=1;
        for(i=0;i<mp[x].size();i++)
        {
            to=mp[x][i];
            if(to==pre) continue;
            dfs1(to,x,h+1);
            sz[x]+=sz[to];
            if(son[x]==-1||sz[to]>sz[son[x]]) son[x]=
                to;
        }
    }
    void dfs2(int x,int tp)
    {
        int i,to;
        dfn[x]=++tot;
        id[dfn[x]]=x;
        top[x]=tp;
        if(son[x]==-1) return;
        dfs2(son[x],tp);
        for(i=0;i<mp[x].size();i++)
        {
            to=mp[x][i];
            if(to!=son[x]&&to!=fa[x]) dfs2(to,to);
        }
    }
    void work(int _rt)
    {
        rt=_rt;
        for(int i=0;i<=n;i++) son[i]=-1;
        tot=0;
        dfs1(rt,0,0);
        dfs2(rt,rt);
```

```
    }
    int lca(int x,int y)
    {
        while(top[x]!=top[y])
        {
            if(dep[top[x]]<dep[top[y]]) swap(x,y);
            x=fa[top[x]];
        }
        if(dep[x]>dep[y]) swap(x,y);
        return x;
    }
    int find_yson_toward_x(int x,int y)
    {
        while(top[x]!=top[y])
        {
            if(fa[top[x]]==y) return top[x];
            x=fa[top[x]];
        }
        return son[y];
    }
    //node
    void init_node(type *v)
    {
        for(int i=1;i<=n;i++) tr.a[dfn[i]]=v[i];
        tr.build(n);
    }
    void upd_node(int x,int y,type v)
    {
        while(top[x]!=top[y])
        {
            if(dep[top[x]]<dep[top[y]]) swap(x,y);
            tr.upd(dfn[top[x]],dfn[x],v);
            x=fa[top[x]];
        }
        if(dep[x]>dep[y]) swap(x,y);
        tr.upd(dfn[x],dfn[y],v);
    }
    type ask_node(int x,int y)
    {
        type res=0;
        while(top[x]!=top[y])
        {
            if(dep[top[x]]<dep[top[y]]) swap(x,y);
            res+=tr.ask(dfn[top[x]],dfn[x]);
            x=fa[top[x]];
        }
        if(dep[x]>dep[y]) swap(x,y);
        res+=tr.ask(dfn[x],dfn[y]);
        return res;
    }
    //path
    void init_path()
    {
        tr.a[dfn[rt]]=0;
        for(int i=1;i<n;i++)
        {
            if(dep[e[i].a]<dep[e[i].b]) swap(e[i].a,e[i].b);
            tr.a[dfn[e[i].a]]=e[i].v;
        }
        tr.build(n);
    }
    void upd_edge(int id,type v)
    {
        if(dep[e[id].a]>dep[e[id].b]) tr.upd(dfn[e[id].a],dfn[e[id].a],v);
        else tr.upd(dfn[e[id].b],dfn[e[id].b],v);
    }
    void upd_path(int x,int y,type v)
    {
        while(top[x]!=top[y])
        {
            if(dep[top[x]]<dep[top[y]]) swap(x,y);
            tr.upd(dfn[top[x]],dfn[x],v);
            x=fa[top[x]];
        }
        if(dep[x]>dep[y]) swap(x,y);
        if(x!=y) tr.upd(dfn[x]+1,dfn[y],v);
    }
    type ask_path(int x,int y)
    {
        type res=0;
        while(top[x]!=top[y])
        {
            if(dep[top[x]]<dep[top[y]]) swap(x,y);
            res+=tr.ask(dfn[top[x]],dfn[x]);
            x=fa[top[x]];
        }
        if(dep[x]>dep[y]) swap(x,y);
        if(x!=y) res+=tr.ask(dfn[x]+1,dfn[y]);
        return res;
    }
    // sub tree
    void change_root(int x){rt=x;}
    void upd_subtree(int x,type v)
    {
        if(x==rt) tr.upd(1,n,v);
        if(dfn[rt]>=dfn[x]&&dfn[rt]<=dfn[x]+sz[x]-1)
        {
            x=find_yson_toward_x(rt,x);
            tr.upd(1,dfn[x]-1,v);
            tr.upd(dfn[x]+sz[x],n,v);
        }
        tr.upd(dfn[x],dfn[x]+sz[x]-1,v);
    }
    type ask_subtree(int x)
    {
        if(x==rt) return tr.ask(1,n).v;
```

```
        if(dfn[rt]>=dfn[x]&&dfn[rt]<=dfn[x]+sz[x]-1)
        {
            x=find_yson_toward_x(rt,x);
            return tr.merge(tr.ask(1,dfn[x]-1),
                            tr.ask(dfn[x]+sz[x],n)).v;
        }
        return tr.ask(dfn[x],dfn[x]+sz[x]-1).v;
    }
    #undef type
}hld;
/*
hld.init(n)
hld.add_edge(a,b,v=0); a <-> b
hld.work(rt);
hld.lca(a,b);
---- node ----
hld.init_node(type *v);
hld.upd_node(a,b,v);
hld.ask_node(a,b);
---- path ----
hld.init_path();
hld.upd_edge(id,v); id:1..n-1
hld.upd_path(a,b,v);
hld.ask_path(a,b);
---- subtree ----
hld.change_root(rt);
hld.upd_subtree(x,v);
hld.ask_subtree(x);
*/
```

3.3 树的重心

```
struct Tree_Centroid
{
    vector<int> *mp;
    int sz[MAX],mx[MAX],n;
    void dfs(int x,int fa)
    {
        sz[x]=1;
        mx[x]=0;
        for(auto &to:mp[x])
        {
            if(to==fa) continue;
            dfs(to,x);
            sz[x]+=sz[to];
            mx[x]=max(mx[x],sz[to]);
        }
        mx[x]=max(mx[x],n-sz[x]);
    }
    vector<int> get_tree_centroid(int _n,vector<int>
         *_mp,int root)
    {
        int i,mn;
        n=_n;
        mp=_mp;
        dfs(root,-1);
        vector<int> res;
        mn=n+1;
        for(i=1;i<=n;i++) mn=min(mn,mx[i]);
        for(i=1;i<=n;i++)
        {
            if(mx[i]==mn) res.push_back(i);
        }
        return res;
    }
}trct;
```

3.4 树上倍增

```
const int LOG=log2(MAX);
int fa[MAX][LOG+1];
void dfs(int x,int pre)
{
    fa[x][0]=pre;
    for(int i=1;i<=LOG;i++) fa[x][i]=fa[fa[x][i-1]][
        i-1];
    for(auto &to:mp[x])
    {
        if(to==pre) continue;
        dfs(to,x);
    }
}
int jump(int x,int target)
{
    for(int i=LOG;~i;i--)
    {
        if(fa[x][i]&&w[fa[x][i]]<=target) x=fa[x][i];
    }
    return x;
}
int main()
{
    memset(fa[0],0,sizeof fa[0]);
    return 0;
}
```

3.5 树 hash

```
struct Tree_Hash
{
    const ull mask=std::chrono::steady_clock::now().
        time_since_epoch().count();
    ull shift(ull x)
    {
      x^=mask;
      x^=x<<13;
```

```
    x^=x>>7;
    x^=x<<17;
    x^=mask;
    return x;
  }
  ull hash[MAX];
  void dfs(VI *mp,int x,int fa)
  {
      hash[x]=1;
      for(auto to:mp[x])
      {
          if(to==fa) continue;
          dfs(mp,to,x);
          hash[x]+=shift(hash[to]);
      }
  }
  void get_tree_hash(VI *mp,int root)
  {
      dfs(mp,root,0);
  }
}trha;
```

3.6 虚树

```
struct Virtual_Tree
{
  int st[MAX],top;
  int build_vtree(VI &a,VI vtree_mp[])// return root
  {
      int now_lca;
      sort(all(a),[&](int x,int y){return lca.dfn[x]<lca.dfn[y];});
      a.erase(unique(all(a)),a.end());
      assert(sz(a)>0);
      top=0;
      st[top++]=a[0];
      VI tmp;
      for(int i=1;i<sz(a);i++)
      {
          if(top==0)
          {
              st[top++]=a[i];
              continue;
          }
          now_lca=lca.lca(a[i],st[top-1]);
          while(top>1&&lca.dfn[st[top-2]]>=lca.dfn[now_lca])
          {
              vtree_mp[st[top-2]].pb(st[top-1]);
              top--;
          }
          if(now_lca!=st[top-1])
          {
              vtree_mp[now_lca].pb(st[top-1]);
              st[top-1]=now_lca;
              tmp.push_back(now_lca);
          }
          st[top++]=a[i];
      }
      while(top>1)
      {
          vtree_mp[st[top-2]].pb(st[top-1]);
          top--;
      }
      for(auto it:tmp) a.push_back(it);
      return st[0];
  }
  void clear_vtree(VI &a,VI vtree_mp[])
  {
      for(auto it:a) vtree_mp[it].clear();
  }
}vt; // need lca and dfn
```

3.7 树分治

3.7.1 点分治

```
struct tree_divide
{
  #define type int
  struct edge{int to;type w;};
  vector<edge> mp[MAX];
  bool vis[MAX];
  int sz[MAX],mx[MAX],sum,rt,n;
  void init(int _n)
  {
      int i;
      n=_n;
      for(i=0;i<=n;i++) mp[i].clear();
  }
  void add_edge(int x,int y,type w){mp[x].push_back({y,w});}
  void get_centroid(int x,int fa)
  {
      sz[x]=1;
      mx[x]=0;
      for(auto &it:mp[x])
      {
          int to=it.to;
          if(to==fa||vis[to]) continue;
          get_centroid(to,x);
          sz[x]+=sz[to];
          mx[x]=max(mx[x],sz[to]);
      }
      mx[x]=max(mx[x],sum-sz[x]);
      if(mx[x]<mx[rt]) rt=x;
```

```
    }
    void get_sz(int x,int fa)
    {
        sz[x]=1;
        for(auto &it:mp[x])
        {
            int to=it.to;
            if(to==fa||vis[to]) continue;
            get_sz(to,x);
            sz[x]+=sz[to];
        }
    }
    type d[MAX],del[MAX];
    int totd,totdel;
    void get_dis(int x,int fa,type h)
    {
        d[++totd]=h;
        for(auto &it:mp[x])
        {
            int to=it.to;
            if(to==fa||vis[to]) continue;
            get_dis(to,x,h+it.w);
        }
    }
    void cal(int x)
    {
        int i,j;
        totdel=0;
        for(auto &it:mp[x])
        {
            int to=it.to;
            if(vis[to]) continue;
            totd=0;
            get_dis(to,x,it.w);
            // do something
            for(i=1;i<=totd;i++) del[++totdel]=d[i];
        }
        for(i=1;i<=totdel;i++)
        {
            // del
        }
    }
    void divide(int x)
    {
        vis[x]=1;
        cal(x);
        for(auto &it:mp[x])
        {
            int to=it.to;
            if(vis[to]) continue;
            get_sz(to,-1);
            rt=0;
            mx[rt]=sum=sz[to];
            get_centroid(to,-1);
            divide(rt);
        }
    }
    void work()
    {
        int i;
        for(i=0;i<=n;i++) vis[i]=0;
        rt=0;
        mx[rt]=sum=n;
        get_centroid(1,-1);
        // init ans
        divide(rt);
    }
    #undef type
}tdv;
/*
tdv.init(n);
tdv.add_edge(x,y,w) // x -> y
tdv.work();
*/
```

3.8 Link-Cut-Tree

```
struct Link_Cut_Tree
{
    #define type int
    const type inf=INF;
    struct node
    {
        int ch[2],fa,sz,rev,tag;
        type v,sum;
    }t[MAX];
    int tot,root,st[MAX];
    void maintain(int id,type v)
    {
        t[id].v=v;
        t[id].sum=v;
    }
    void pushup(int id)
    {
        int ls=t[id].ch[0];
        int rs=t[id].ch[1];
        t[id].sz=t[ls].sz+t[rs].sz+1;
        t[id].sum=t[ls].sum+t[rs].sum+t[id].v;
    }
    void pushdown(int id)
    {
        int ls=t[id].ch[0];
        int rs=t[id].ch[1];
        if(t[id].tag)
        {
            if(ls)
            {
```

```
            maintain(ls,t[id].v);
            t[ls].tag=1;
        }
        if(rs)
        {
            maintain(rs,t[id].v);
            t[rs].tag=1;
        }
        t[id].tag=0;
    }
    if(t[id].rev)
    {
        t[ls].rev^=1;
        t[rs].rev^=1;
        swap(t[ls].ch[0],t[ls].ch[1]);
        swap(t[rs].ch[0],t[rs].ch[1]);
        t[id].rev=0;
    }
}
int newnode(type v,int fa)
{
    int id=++tot;
    memset(t[id].ch,0,sizeof t[id].ch);
    t[id].fa=fa;
    t[id].sz=1;
    t[id].tag=t[id].rev=0;
    maintain(id,v);
    return id;
}
int is_splay_root(int x)
{
    int fa=t[x].fa;
    return t[fa].ch[0]!=x&&t[fa].ch[1]!=x;
}
void rotate(int x)
{
    int y,z,k;
    y=t[x].fa;
    z=t[y].fa;
    k=(x==t[y].ch[1]);
    if(!is_splay_root(y)) t[z].ch[y==t[z].ch[1]]=
        x;
    t[y].ch[k]=t[x].ch[k^1];
    if(t[x].ch[k^1]) t[t[x].ch[k^1]].fa=y;
    t[x].ch[k^1]=y;
    t[y].fa=x;
    t[x].fa=z;
    pushup(y);
}
void splay(int x)
{
    int y,z,top;
    top=0;
    y=x;
    st[++top]=y;
    while(!is_splay_root(y))
    {
        y=t[y].fa;
        st[++top]=y;
    }
    while(top>0) pushdown(st[top--]);
    while(!is_splay_root(x))
    {
        y=t[x].fa;
        z=t[y].fa;
        if(!is_splay_root(y))
        {
            if((t[z].ch[0]==y)^(t[y].ch[0]==x))
                rotate(x);
            else rotate(y);
        }
        rotate(x);
    }
    pushup(x);
}
void init_null_node()
{
    memset(t[0].ch,0,sizeof t[0].ch);
    t[0].sz=t[0].fa=0;
    t[0].v=t[0].sum=0;
}
void init(int n,type *v)
{
    int i;
    tot=0;
    init_null_node();
    for(i=1;i<=n;i++) newnode(v[i],0);
}
void init(int n)
{
    int i;
    tot=0;
    init_null_node();
    for(i=1;i<=n;i++) newnode(0,0);
}
int access(int x)
{
    int fa=0;
    while(x)
    {
        splay(x);
        t[x].ch[1]=fa;
        pushup(x);
        fa=x;
        x=t[x].fa;
    }
    return fa;
}
```

```
    int findroot(int x)
    {
        access(x);
        splay(x);
        pushdown(x);
        while(t[x].ch[0])
        {
            x=t[x].ch[0];
            pushdown(x);
        }
        splay(x);
        return x;
    }
    void makeroot(int x)
    {
        x=access(x);
        swap(t[x].ch[0],t[x].ch[1]);
        t[x].rev^=1;
    }
    int split(int x,int y)
    {
        makeroot(x);
        access(y);
        splay(y);
        return y;
    }
    void link(int x,int y)
    {
        makeroot(x);
        splay(x);
        if(findroot(y)!=x) t[x].fa=y;
    }
    void cut(int x,int y)
    {
        makeroot(x);
        if(findroot(y)==x&&t[y].fa==x&&t[y].ch[0]==0)
        {
            t[y].fa=t[x].ch[1]=0;
            pushup(x);
        }
    }
    int is_connect(int x,int y)
    {
        makeroot(x);
        return findroot(y)==x;
    }
    void upd_node(int x,type v)
    {
        splay(x);
        maintain(x,v);
    }
    #undef type
}lct;
/*
lct.init(n);
lct.init(n,*v); v[1..n]
*/
```

4 图论

4.1 链式前向星

```
struct Fast_Vector
{
    static const int N=;
    static const int M=;
    Fast_Vector &operator[](const int _x){this->x=_x
        ;return *this;}
    int head[N],tot,x;
    struct node{int to,nex;} e[M];
    void clear(int n)
    {
        for(int i=0;i<=n;i++) head[i]=0;
        tot=0;
    }
    void push_back(const int &to)
    {
        tot++;
        e[tot].to=to;
        e[tot].nex=head[x];
        head[x]=tot;
    }
};
//for(i=mp.head[x];i;i=mp.e[i].nex) to=mp.e[i].to;
```

4.2 最短路

4.2.1 dijkstra

```
struct Dijkstra
{
    #define type int
    const type inf=INF;
    struct node
    {
        int id;
        type v;
        friend bool operator <(node a,node b){return
            a.v>b.v;}
    };
    static const int N=MAX;
    vector<node> mp[N];
    type dis[N];
    int n,vis[N];
    void init(int _n)
    {
        n=_n;
```

```
    for(int i=0;i<=n;i++) mp[i].clear();
}
void add_edge(int x,int y,type v){mp[x].
    push_back({y,v});}
void work(int s)
{
    int i,to;
    type w;
    priority_queue<node> q;
    for(i=0;i<=n;i++)
    {
        dis[i]=inf;
        vis[i]=0;
    }
    dis[s]=0;
    q.push({s,type(0)});
    while(!q.empty())
    {
        node t=q.top();
        q.pop();
        if(vis[t.id]) continue;
        vis[t.id]=1;// this node has already been
            extended
        for(auto &it:mp[t.id])
        {
            to=it.id;
            w=it.v;
            if(dis[to]>dis[t.id]+w)
            {
                dis[to]=dis[t.id]+w;
                if(!vis[to]) q.push({to,dis[to]});
            }
        }
    }
}
#undef type
}dij;
```

4.2.2 spfa

```
struct SPFA
{
    #define type int
    static const type inf=INF;
    static const int N=;
    vector<pair<int,type> > mp[N];
    type dis[N];
    int n,vis[N],cnt[N];
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++) mp[i].clear();
    }
    void add_edge(int x,int y,type v){mp[x].
        push_back({y,v});}
    bool work(int s)
    {
        int i,x,to;
        type w;
        queue<int> q;
        for(i=0;i<=n;i++)
        {
            dis[i]=inf;
            vis[i]=cnt[i]=0;
        }
        dis[s]=0;
        vis[s]=1;
        q.push(s);
        while(!q.empty())
        {
            x=q.front();
            q.pop();
            vis[x]=0;
            for(auto &it:mp[x])
            {
                to=it.first;
                w=it.second;
                if(dis[to]>dis[x]+w)
                {
                    dis[to]=dis[x]+w;
                    cnt[to]=cnt[x]+1;
                    if(cnt[to]>n)
                    {
                        // cnt is edge counts of
                            current short path
                        // if cnt >= (sum of node), the
                            graph exists negative ring
                        return false;
                    }
                    if(!vis[to])
                    {
                        q.push(to);
                        vis[to]=1;
                    }
                }
            }
        }
        return true;
    }
    #undef type
}spfa;
```

4.2.3 floyd 求最小环

```
struct Floyd
{
    #define type int
```

```
    const type inf=INF;
    static const int N=;
    int n;
    type mp[N][N],dis[N][N];
    type min_circle_3;// len(circle)>=3
    type min_circle; // len(circle)>=1
    void init(int _n)
    {
        int i,j;
        n=_n;
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                mp[i][j]=dis[i][j]=inf;
            }
        }
    }
    void add_edge(int x,int y,type w)
    {
        w=min(mp[x][y],w);
        mp[x][y]=dis[x][y]=w;
    }
    void work()
    {
        int i,j,k;
        min_circle_3=inf;
        for(k=1;k<=n;k++)
        {
            for(i=1;i<k;i++)
            {
                if(mp[i][k]==inf) continue;
                for(j=1;j<k;j++)
                {
                    if(i==j||mp[k][j]==inf||dis[j][i]==
                        inf) continue;
                    min_circle_3=min(min_circle_3,mp[i
                        ][k]+mp[k][j]+dis[j][i]);
                }
            }
            for(i=1;i<=n;i++)
            {
                if(dis[i][k]==inf) continue;
                for(j=1;j<=n;j++)
                {
                    if(dis[k][j]==inf) continue;
                    dis[i][j]=min(dis[i][j],dis[i][k]+
                        dis[k][j]);
                }
            }
        }
        min_circle=inf;
        for(i=1;i<=n;i++) min_circle=min(min_circle,
            dis[i][i]);
    }
    #undef type
}flyd;
/*
flyd.init(n);
flyd.add_edge(x,y,w); x,y [1..n] x->y
flyd.work();
*/
```

4.2.4 Johnson

```
struct Johnson
{
    #define type int
    #define inf INF
    #define PTI pair<type,int>
    static const int N=3000+10;
    vector<pair<int,type> > mp[N];
    type dis[N],h[N];
    int n,vis[N],cnt[N];
    bool spfa_flag;
    void init(int _n)
    {
        n=_n;
        spfa_flag=false;
        for(int i=0;i<=n;i++) mp[i].clear();
    }
    void add_edge(int x,int y,type v){mp[x].pb(MP(y,
        v));}
    bool spfa(int s)
    {
        int i,x,to;
        type w;
        queue<int> q;
        for(i=0;i<=n;i++)
        {
            h[i]=inf;
            vis[i]=cnt[i]=0;
        }
        h[s]=0;
        vis[s]=1;
        q.push(s);
        while(!q.empty())
        {
            x=q.front();
            q.pop();
            vis[x]=0;
            for(auto it:mp[x])
            {
                to=it.fi;
                w=it.se;
                if(h[to]>h[x]+w)
                {
                    h[to]=h[x]+w;
```

```
                cnt[to]=cnt[x]+1;
                if(cnt[to]>n)
                {
                    // cnt is edge counts of current short path
                    // if cnt >= (sum of node), the graph exists negative ring
                    return false;
                }
                if(!vis[to])
                {
                    q.push(to);
                    vis[to]=1;
                }
            }
        }
    }
    spfa_flag=true;
    return true;
}
void dij(int s)
{
    assert(spfa_flag);
    int i,to;
    type w;
    priority_queue<PTI ,vector<PTI>,greater<PTI> > q;
    for(i=0;i<=n;i++)
    {
        dis[i]=inf;
        vis[i]=0;
    }
    dis[s]=0;
    q.push(MP(type(0),s));
    while(!q.empty())
    {
        PTI t=q.top();
        q.pop();
        if(vis[t.se]) continue;
        vis[t.se]=1;
        for(auto it:mp[t.se])
        {
            to=it.fi;
            w=it.se+h[t.se]-h[to];
            if(dis[to]>dis[t.se]+w)
            {
                dis[to]=dis[t.se]+w;
                if(!vis[to]) q.push(MP(dis[to],to));
            }
        }
    }
    for(i=0;i<=n;i++)
    {
        if(dis[i]==inf) continue;
        dis[i]-=h[s]-h[i];
    }
}
#undef type
#undef inf
#undef PTI
}js;
```

4.2.5 同余最短路

```
struct Dijkstra
{
    #define type ll
    const type inf=LLINF;
    struct node
    {
        int id;
        type v;
        friend bool operator <(node a,node b){return a.v>b.v;}
    };
    static const int N=MAX;
    vector<node> mp[N];
    type dis[N];
    int n,vis[N];
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++) mp[i].clear();
    }
    void add_edge(int x,int y,type v){ mp[x].push_back({y,v});}
    void work(int s)
    {
        int i,to;
        type w;
        priority_queue<node> q;
        for(i=0;i<=n;i++)
        {
            dis[i]=inf;
            vis[i]=0;
        }
        dis[s]=0;
        q.push({s,type(0)});
        while(!q.empty())
        {
            node t=q.top();
            q.pop();
            if(vis[t.id]) continue;
            vis[t.id]=1;// this node has already been extended
            for(auto &it:mp[t.id])
            {
```

```
            to=it.id;
            w=it.v;
            if(dis[to]>dis[t.id]+w)
            {
                dis[to]=dis[t.id]+w;
                if(!vis[to]) q.push({to,dis[to]});
            }
        }
    }
    #undef type
}dij;
//□□[l,r] b □□□□sum{ai*xi}=b, xi>=0
ll congruent_short_path(vector<int> &a,ll l,ll r)
{
    int n,i,j;
    ll res;
    n=a.size();
    sort(a.begin(),a.end());
    a.resize(unique(a.begin(),a.end())-a.begin());
    dij.init(a[0]);
    for(i=0;i<a[0];i++)
    {
        for(j=1;j<n;j++)
        {
            dij.add_edge(i,(i+a[j])%a[0],a[j]);
        }
    }
    dij.work(0);
    res=0;
    for(i=0;i<a[0];i++)
    {
        if(dij.dis[i]<=r) res+=(r-dij.dis[i])/a[0]+1;
        if(dij.dis[i]<=l-1) res-=((l-1)-dij.dis[i])/a
            [0]+1;
    }
    return res;
}
```

4.3 最小生成树

4.3.1 kruskal

```
struct Disjoint_Set_Union
{
    int pre[MAX],sz[MAX];
    void init(int n)
    {
        int i;
        for(i=1;i<=n;i++)
        {
            pre[i]=i;
            sz[i]=1;
        }
    }
    int find(int x)
    {
        if(pre[x]!=x) pre[x]=find(pre[x]);
        return pre[x];
    }
    bool merge(int x,int y,bool dir=false)
    {
        x=find(x);
        y=find(y);
        if(x==y) return 0;
        if(!dir && sz[x]>sz[y]) swap(x,y);
        pre[x]=y; // x -> y
        sz[y]+=sz[x];
        return 1;
    }
}dsu;
struct Kruskal
{
    #define type int
    #define inf INF
    struct edge{int x,y;type w;};
    vector<edge> e;
    void init(){e.clear();}
    void add_edge(int a,int b,type w){e.push_back({a
        ,b,w});}
    type work(int n)
    {
        int i,cnt;
        type res=0;
        dsu.init(n);
        sort(e.begin(),e.end(),[&](edge x,edge y){
            return x.w<y.w;
        });
        for(auto &it:e)
        {
            if(dsu.merge(it.x,it.y)) res+=it.w;
        }
        cnt=0;
        for(i=1;i<=n;i++) cnt+=dsu.find(i)==i;
        if(cnt!=1) return inf; // no connect
        return res;
    }
    #undef type
    #undef inf
}krsk;
```

4.3.2 kruskal 重构树

```
struct Disjoint_Set_Union
{
    int pre[MAX],sz[MAX];
    void init(int n)
    {
```

```
        int i;
        for(i=1;i<=n;i++)
        {
            pre[i]=i;
            sz[i]=1;
        }
    }
    int find(int x)
    {
        if(pre[x]!=x) pre[x]=find(pre[x]);
        return pre[x];
    }
    bool merge(int x,int y,bool dir=false)
    {
        x=find(x);
        y=find(y);
        if(x==y) return 0;
        if(!dir && sz[x]>sz[y]) swap(x,y);
        pre[x]=y; // x -> y
        sz[y]+=sz[x];
        return 1;
    }
}dsu;
struct Kruskal_Tree
{
    #define type int
    struct edge{int x,y;type w;};
    vector<edge> e;
    void init(){e.clear();}
    void add_edge(int x,int y,type w){e.push_back({x
        ,y,w});}
    int build_kruskal_tree(int n,vector<int> *mp,
        type *w)
    {
        int rt,x,y,i;
        for(i=1;i<=2*n-1;i++)
        {
            mp[i].clear();
            w[i]=0;
        }
        dsu.init(2*n-1);
        sort(e.begin(),e.end(),[&](edge x,edge y){
            return x.w<y.w;
        });
        rt=n;
        for(auto &it:e)
        {
            x=dsu.find(it.x);
            y=dsu.find(it.y);
            if(x==y) continue;
            rt++;
            w[rt]=it.w;
            mp[rt].push_back(x);
            mp[rt].push_back(y);
            dsu.merge(x,rt);
            dsu.merge(y,rt);
        }
        return rt;
    }
    #undef type
}krsk;
```

4.3.3 prim

```
struct Prim
{
    #define type int
    const type inf=INF;
    struct node{int id;type w;};
    static const int N=MAX;
    vector<node> mp[N];
    type dis[N];
    int n,vis[N];
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++) mp[i].clear();
    }
    void add_edge(int x,int y,type w)
    {
        mp[x].push_back({y,w});
        mp[y].push_back({x,w});
    }
    type work()
    {
        int i,x,cnt;
        type res,mn;
        for(i=0;i<=n;i++)
        {
            dis[i]=inf;
            vis[i]=0;
        }
        x=1;
        for(auto &to:mp[x]) dis[to.id]=min(dis[to.id
            ],to.w);
        res=0;
        cnt=0;
        while(1)
        {
            mn=inf;
            vis[x]=1;
            cnt++;
            for(i=1;i<=n;i++)
            {
                if(!vis[i]&&dis[i]<mn)
                {
                    mn=dis[i];
                    x=i;
```

```
            }
        }
        if(mn==inf) break;
        res+=mn;
        for(auto &to:mp[x]) dis[to.id]=min(dis[to
            .id],to.w);
    }
    if(cnt<n) return inf; // no connect
    return res;
}
#undef type
}prim;
/*
O(n^2+m)

prim.init(n);
prim.add_edge(x,y,w); x<->y
prim.work();
*/
```

4.4 二分图匹配

4.4.1 二分图最大匹配

```
/*□□C 牨 ǵĵ□□□
µ,□□□þ□□□□=□□Cǵı□□□
• ¾¶,²,=□□□¶¥µn□□□þ□□□□-□□□□
b¼¯□□C=□• ¾¶,²,=□□□¶¥µn□□□þ□□□□-
*/
struct Bipartite_Matching
{
    static const int N=;
    static const int M=;
    int n,m;
    vector<int> mp[N];
    int flag[N],s[N],link[M],used[M];
    void init(int _n,int _m)
    {
        n=_n;
        m=_m;
        for(int i=0;i<=n;i++) mp[i].clear();
    }
    void add_edge(int a,int b){mp[a].push_back(b);}
    bool dfs(int x,int timetag)
    {
        int i,to;
        flag[x]=1;
        for(i=0;i<mp[x].size();i++)
        {
            to=mp[x][i];
            if(used[to]==timetag) continue;
            used[to]=timetag;
            if(link[to]==-1||dfs(link[to],timetag))
            {
                link[to]=x;
                s[x]=to;
                return 1;
            }
        }
        return 0;
    }
    int max_match()
    {
        int i,res;
        memset(link,-1,sizeof link);
        memset(s,-1,sizeof s);
        memset(used,0,sizeof used);
        res=0;
        for(i=1;i<=n;i++)
        {
            if(mp[i].size()==0) continue;
            if(dfs(i,i)) res++;
        }
        return res;
    }
    int min_cover(vector<int> &x,vector<int> &y)
    {
        int i,res;
        res=max_match();
        memset(flag,0,sizeof flag);
        memset(used,0,sizeof used);
        x.clear();
        y.clear();
        for(i=1;i<=n;i++)
        {
            if(s[i]==-1) dfs(i);
        }
        for(i=1;i<=n;i++)
        {
            if(!flag[i]) x.push_back(i);
        }
        for(i=1;i<=m;i++)
        {
            if(used[i]) y.push_back(i);
        }
        return res;
    }
}bpm;
/*
O(n*m) : n node,m edge
bpm.init(n,m);
bpm.add_edge(a,b); a:left,1-n b:right,1-m
*/
```

4.4.2 二分图最大权完美匹配

```
struct Kuhn_Munkres
{
```

```
#define type int
const type inf=INF;
static const int N=;
int n,mx[N],my[N],pre[N];
type slk[N],lx[N],ly[N],w[N][N];
bool vx[N],vy[N];
void init(int _n)
{
    n=_n;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            /*
            - □□□□□□□□□þ□□, □□□□Ę=0
            - □□□□□□□□□þ□□²», □□□□Ę=-inf,□□□□□□□
              4□þ□□□'□□□□□□□µ,□□□□□□□□□
               • ¢»µò»¬¾
            */
            w[i][j]=0;
        }
    }
}
void add_edge(int x,int y,type val)
{
    // □L□ĘÏ□Þ□□□□þ□□¬°±¸º
    w[x][y]=val;
}
void match(int y){while(y) swap(y,mx[my[y]=pre[y
    ]]);}
void bfs(int x)
{
    int i,y;
    type d;
    for(i=1;i<=n;i++)
    {
        vx[i]=vy[i]=pre[i]=0;
        slk[i]=inf;
    }
    queue<int> q;
    q.push(x);
    vx[x]=1;
    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;
    }
}
type 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);
    res=0;
    for(i=1;i<=n;i++)
    {
        // ²»
        res+=w[i][mx[i]];
    }
    return res;
}
#undef type
}km;
/*
O(n^3)
```

```
km.init(n);
km.add_edge(a,b,val); a,b: 1~n
*/
```

4.5 最大流

4.5.1 dinic

```
struct Dinic
{
    #define type int
    const type inf=INF;
    static const int N=;
    struct node
    {
        int from,to;
        type cap,flow;
        node(int u,int v,type c,type f):from(u),to(v)
            ,cap(c),flow(f){}
    };
    int n,s,t;
    vector<node> edge;
    vector<int> mp[N];
    int vis[N],dist[N],id[N];
    void init(int _n)
    {
        n=_n;
        edge.clear();
        for(int i=0;i<=n;i++)
        {
            mp[i].clear();
            id[i]=dist[i]=vis[i]=0;
        }
    }
    void add_edge(int from,int to,type cap)
    {
        edge.push_back(node(from,to,cap,0));
        edge.push_back(node(to,from,0,0));
        int m=edge.size();
        mp[from].push_back(m-2);
        mp[to].push_back(m-1);
    }
    bool bfs()
    {
        int i,x;
        memset(vis,0,sizeof vis);
        queue<int>q;
        q.push(s);
        dist[s]=0;
        vis[s]=1;
        while(!q.empty())
        {
            x=q.front();
            q.pop();
            for(i=0;i<mp[x].size();i++)
            {
                node &e=edge[mp[x][i]];
                if(!vis[e.to]&&e.cap>e.flow)
                {
                    vis[e.to]=1;
                    dist[e.to]=dist[x]+1;
                    q.push(e.to);
                }
            }
        }
        return vis[t];
    }
    type dfs(int x,type a)
    {
        if(x==t||!a) return a;
        type flow=0,f;
        for(int &i=id[x];i<mp[x].size();i++)
        {
            node &e=edge[mp[x][i]];
            if(dist[x]+1==dist[e.to]&&(f=dfs(e.to,min
                (a,e.cap-e.flow)))>0)
            {
                e.flow+=f;
                edge[mp[x][i]^1].flow-=f;
                flow+=f;
                a-=f;
                if(!a) break;
            }
        }
        return flow;
    }
    type max_flow(int _s,int _t)
    {
        s=_s;
        t=_t;
        type res=0;
        while(bfs())
        {
            for(int i=0;i<=n;i++) id[i]=0;
            res+=dfs(s,inf);
        }
        return res;
    }
    #undef type
}dc;
/*
O(n^2*m)
bipartite graph: O(m*sqrt(n))

dc.init(n);
dc.add_edge(a,b,cap); a,b: 1~n
*/
```

4.5.2 有源汇上下界网络流

```
struct Dinic
{
    #define type int
    const type inf=INF;
    static const int N=;
    struct node
    {
        int from,to;
        type cap,flow;
        node(int u,int v,type c,type f):from(u),to(v)
            ,cap(c),flow(f){}
    };
    int n,s,t,s1,t1;
    vector<node> edge;
    vector<int> mp[N];
    int vis[N],dist[N],id[N];
    type in[N],out[N];
    void init(int _n)
    {
        s1=_n+1;
        t1=s1+1;
        n=t1;
        assert(n<N);
        edge.clear();
        for(int i=0;i<=n;i++)
        {
            mp[i].clear();
            id[i]=dist[i]=0;
            in[i]=out[i]=0;
        }
    }
    void add_edge(int from,int to,type lcap,type
        rcap)
    {
        edge.push_back(node(from,to,rcap-lcap,0));
        edge.push_back(node(to,from,0,0));
        in[to]+=lcap;
        out[from]+=lcap;
        int m=edge.size();
        mp[from].push_back(m-2);
        mp[to].push_back(m-1);
    }
    bool bfs()
    {
        int i,x;
        for(int i=0;i<=n;i++) vis[i]=0;
        queue<int>q;
        q.push(s);
        dist[s]=0;
        vis[s]=1;
        while(!q.empty())
        {
            x=q.front();
            q.pop();
            for(i=0;i<mp[x].size();i++)
            {
                node &e=edge[mp[x][i]];
                if(!vis[e.to]&&e.cap>e.flow)
                {
                    vis[e.to]=1;
                    dist[e.to]=dist[x]+1;
                    q.push(e.to);
                }
            }
        }
        return vis[t];
    }
    type dfs(int x,type a)
    {
        if(x==t||!a) return a;
        type flow=0,f;
        for(int &i=id[x];i<mp[x].size();i++)
        {
            node &e=edge[mp[x][i]];
            if(dist[x]+1==dist[e.to]&&(f=dfs(e.to,min
                (a,e.cap-e.flow)))>0)
            {
                e.flow+=f;
                edge[mp[x][i]^1].flow-=f;
                flow+=f;
                a-=f;
                if(!a) break;
            }
        }
        return flow;
    }
    type dinic(int _s,int _t)
    {
        int i;
        s=_s;
        t=_t;
        type res=0;
        while(bfs())
        {
            for(i=0;i<=n;i++) id[i]=0;
            res+=dfs(s,inf);
        }
        return res;
    }
    type min_flow(int _s,int _t)
    {
        int i;
        s=s1;
        t=t1;
        for(i=0;i<s1;i++)
        {
```

```
        if(in[i]>out[i]) add_edge(s,i,0,in[i]-out[i]);
        else if(in[i]<out[i]) add_edge(i,t,0,out[i]-in[i]);
    }
    add_edge(_t,_s,0,inf);
    dinic(s,t);
    for(i=0;i<mp[s].size();i++)
    {
        node &e=edge[mp[s][i]];
        if(e.cap-e.flow!=0) return -1;
    }
    s=_s;
    t=_t;
    type res;
    for(i=0;i<mp[t].size();i++)
    {
        node &e=edge[mp[t][i]];
        if(e.to==s)
        {
            res=e.flow;
            e.flow=edge[mp[t][i]^1].flow=0;
            e.cap=edge[mp[t][i]^1].cap=0;
        }
    }
    res-=dinic(t,s);
    return res;
}
type max_flow(int _s,int _t)
{
    int i;
    s=s1;
    t=t1;
    for(i=0;i<s1;i++)
    {
        if(in[i]>out[i]) add_edge(s,i,0,in[i]-out[i]);
        else if(in[i]<out[i]) add_edge(i,t,0,out[i]-in[i]);
    }
    add_edge(_t,_s,0,inf);
    dinic(s,t);
    for(i=0;i<mp[s].size();i++)
    {
        node &e=edge[mp[s][i]];
        if(e.cap-e.flow!=0) return -1;
    }
    s=_s;
    t=_t;
    type res;
    for(i=0;i<mp[t].size();i++)
    {
        node &e=edge[mp[t][i]];
        if(e.to==s)
        {
            res=e.flow;
            e.flow=edge[mp[t][i]^1].flow=0;
            e.cap=edge[mp[t][i]^1].cap=0;
        }
    }
    res+=dinic(s,t);
    return res;
}
#undef type
}dc;
/*
dc.init(n);
dc.add_edge(a,b,lcap,rcap); a,b: 1~n
dc.min_flow(s,t);
dc.max_flow(s,t);
*/
```

4.6 费用流

4.6.1 spfa 费用流

```
struct MCMF
{
    #define type int
    #define inf INF
    static const int N=;
    struct node
    {
        int from,to;
        type cap,flow,cost;
        node(){}
        node(int u,int v,type c,type f,type co):from(u),to(v),cap(c),flow(f),cost(co){}
    };
    int n,s,t;
    vector<node> edge;
    vector<int> mp[N];
    int vis[N],id[N];
    type d[N],a[N];
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++) mp[i].clear();
        edge.clear();
    }
    void add_edge(int from,int to,type cap,type cost=0)
    {
        edge.pb(node(from,to,cap,0,cost));
        edge.pb(node(to,from,0,0,-cost));
        int m=edge.size();
        mp[from].pb(m-2);
        mp[to].pb(m-1);
```

```
    }
    bool spfa(type& flow,type& cost)
    {
        for(int i=0;i<=n;i++)
        {
            d[i]=inf;
            vis[i]=0;
        }
        d[s]=0;vis[s]=1;id[s]=0;a[s]=inf;
        queue<int> q;
        q.push(s);
        while(!q.empty())
        {
            int x=q.front();
            q.pop();
            vis[x]=0;
            for(int i=0;i<mp[x].size();i++)
            {
                node& e=edge[mp[x][i]];
                int to=e.to;
                if(e.cap>e.flow&&d[to]>d[x]+e.cost)
                {
                    d[to]=d[x]+e.cost;
                    a[to]=min(a[x],e.cap-e.flow);
                    id[to]=mp[x][i];
                    if(!vis[to])
                    {
                        vis[to]=1;
                        q.push(to);
                    }
                }
            }
        }
        if(d[t]==inf) return false;
        flow+=a[t];
        cost+=a[t]*d[t];
        int x=t;
        while(x!=s)
        {
            edge[id[x]].flow+=a[t];
            edge[id[x]^1].flow-=a[t];
            x=edge[id[x]].from;
        }
        return true;
    }
    pair<type,type> mincost_maxflow(int _s,int _t)
    {
        type flow=0,cost=0;
        s=_s;
        t=_t;
        while(spfa(flow,cost));
        return MP(cost,flow);
    }
    #undef type
    #undef inf
}mcmf;
/*
mcmf.init(n);
mcmf.add_edge(a,b,cap,cost); a,b: 1~n
*/
```

4.6.2 dijkstra 费用流 (dij 求 h)

```
struct MCMF_dij
{
    #define type int
    #define inf INF
    #define PTI pair<type,int>
    static const int N=;
    struct node
    {
        int from,to;
        type flow,cost;
        node(){}
        node(int u,int v,type f,type co):from(u),to(v),flow(f),cost(co){}
    };
    int n,s,t,id[N];
    vector<node> edge;
    vector<int> mp[N];
    type dis[N],h[N];
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++) mp[i].clear();
        edge.clear();
    }
    void add_edge(int from,int to,type cap,type cost=0)
    {
        edge.push_back(node(from,to,cap,cost));
        edge.push_back(node(to,from,0,-cost));
        int m=edge.size();
        mp[from].push_back(m-2);
        mp[to].push_back(m-1);
    }
    bool dij()
    {
        int i,x,to;
        type cost,now_cost;
        for(i=0;i<=n;i++) dis[i]=inf;
        dis[s]=0;id[s]=0;
        priority_queue<PTI ,vector<PTI>,greater<PTI> > q;
        q.push({type(0),s});
        while(!q.empty())
        {
            PTI tmp=q.top();
```

```
            q.pop();
            cost=tmp.first;
            x=tmp.second;
            if(cost>dis[x]) continue;
            for(i=0;i<mp[x].size();i++)
            {
                node& e=edge[mp[x][i]];
                to=e.to;
                type now_cost=e.cost+h[x]-h[to];
                if(e.flow>0&&dis[to]>dis[x]+now_cost)
                {
                    dis[to]=dis[x]+now_cost;
                    q.push({dis[to],to});
                    e.from=x;
                    id[to]=mp[x][i];
                }
            }
        }
        return dis[t]!=inf;
    }
    pair<type,type> mincost_maxflow(int _s,int _t)
    {
        int i;
        type flow=0,cost=0;
        for(int i=0;i<=n;i++) h[i]=0;
        s=_s;
        t=_t;
        while(dij())
        {
            for(i=0;i<=n;i++) h[i]+=dis[i];
            type new_flow=inf;
            for(i=t;i!=s;i=edge[id[i]].from)
            {
                new_flow=min(new_flow,edge[id[i]].flow
                    );
            }
            for(i=t;i!=s;i=edge[id[i]].from)
            {
                edge[id[i]].flow-=new_flow;
                edge[id[i]^1].flow+=new_flow;
            }
            flow+=new_flow;
            cost+=new_flow*h[t];
        }
        return {cost,flow};
    }
    #undef type
    #undef inf
    #undef PTI
}mcmf; //upper: O(nmlog(nm) + max_flow*mlogm)
```

4.6.3 dijkstra 费用流 (spfa 求 h)

```
struct MCMF_dij
{
    #define type int
    #define inf INF
    #define PTI pair<type,int>
    static const int N=;
    struct node
    {
        int from,to;
        type flow,cost;
        node(){}
        node(int u,int v,type f,type co):from(u),to(v
            ),flow(f),cost(co){}
    };
    int n,s,t,id[N],vis[N];
    vector<node> edge;
    vector<int> mp[N];
    type dis[N],h[N];
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++) mp[i].clear();
        edge.clear();
    }
    void add_edge(int from,int to,type cap,type cost
        =0)
    {
        edge.push_back(node(from,to,cap,cost));
        edge.push_back(node(to,from,0,-cost));
        int m=edge.size();
        mp[from].push_back(m-2);
        mp[to].push_back(m-1);
    }
    void spfa()
    {
        int i,x,to;
        for(i=0;i<=n;i++)
        {
            h[i]=inf;
            vis[i]=0;
        }
        queue<int> q;
        q.push(s);
        h[s]=0;
        vis[s]=1;
        while(!q.empty())
        {
            x=q.front();
            q.pop();
            vis[x]=0;
            for(i=0;i<mp[x].size();i++)
            {
                node &e=edge[mp[x][i]];
                to=e.to;
```

```
            if(e.flow>0&&h[to]>h[x]+e.cost)
            {
                h[to]=h[x]+e.cost;
                if(!vis[to])
                {
                    vis[to]=1;
                    q.push(to);
                }
            }
        }
    }
}
bool dij()
{
    int i,x,to;
    type cost,now_cost;
    for(i=0;i<=n;i++) dis[i]=inf;
    dis[s]=0;id[s]=0;
    priority_queue<PTI ,vector<PTI>,greater<PTI>
        > q;
    q.push({type(0),s});
    while(!q.empty())
    {
        PTI tmp=q.top();
        q.pop();
        cost=tmp.first;
        x=tmp.second;
        if(cost>dis[x]) continue;
        for(i=0;i<mp[x].size();i++)
        {
            node& e=edge[mp[x][i]];
            to=e.to;
            type now_cost=e.cost+h[x]-h[to];
            if(e.flow>0&&dis[to]>dis[x]+now_cost)
            {
                dis[to]=dis[x]+now_cost;
                q.push({dis[to],to});
                e.from=x;
                id[to]=mp[x][i];
            }
        }
    }
    return dis[t]!=inf;
}
pair<type,type> mincost_maxflow(int _s,int _t)
{
    int i;
    type flow=0,cost=0;
    s=_s;
    t=_t;
    spfa();
    while(dij())
    {
        for(i=0;i<=n;i++) h[i]+=dis[i];
        type new_flow=inf;
        for(i=t;i!=s;i=edge[id[i]].from)
        {
            new_flow=min(new_flow,edge[id[i]].flow
                );
        }
        for(i=t;i!=s;i=edge[id[i]].from)
        {
            edge[id[i]].flow-=new_flow;
            edge[id[i]^1].flow+=new_flow;
        }
        flow+=new_flow;
        cost+=new_flow*h[t];
    }
    return {cost,flow};
}
#undef type
#undef inf
#undef PTI
}mcmf; // upper: O(nm + max_flow*mlogm)
```

4.7 连通性

4.7.1 强连通分量

```
struct Strongly_Connected_Components
{
    int scc_cnt,tot;
    int low[MAX],dfn[MAX],col[MAX],sz[MAX];
    int st[MAX],top,flag[MAX];
    vector<int> *mp;
    void dfs(int x)
    {
        int tmp;
        st[top++]=x;
        flag[x]=1;
        low[x]=dfn[x]=++tot;
        for(auto &to:mp[x])
        {
            if(!dfn[to])
            {
                dfs(to);
                low[x]=min(low[x],low[to]);
            }
            else if(flag[to]) low[x]=min(low[x],dfn[
                to]);
        }
        if(low[x]==dfn[x])
        {
            scc_cnt++;
            do
            {
                tmp=st[--top];
                flag[tmp]=0;
```

```
            col[tmp]=scc_cnt;
            sz[scc_cnt]++;
        }while(tmp!=x);
    }
}
void work(int n,vector<int> *_mp)
{
    int i;
    mp=_mp;
    for(i=1;i<=n;i++) col[i]=sz[i]=flag[i]=0;
    scc_cnt=top=tot=0;
    for(i=1;i<=n;i++)
    {
        if(col[i]) continue;
        dfs(i);
    }
}
void rebuild(int n,vector<int> *g)
{
    int i;
    for(i=1;i<=n;i++) g[i].clear();
    for(i=1;i<=n;i++)
    {
        for(auto &to:mp[i])
        {
            if(col[i]==col[to]) continue;
            g[col[i]].push_back(col[to]);
        }
    }
}
}scc;
/*
scc.work(n,mp);
*/
```

4.7.2 边双连通分量

```
struct Edge_Biconnected_Component
{
    int bcc_cnt,tot;
    int low[MAX],dfn[MAX],col[MAX];
    int st[MAX],top;
    vector<int> mp[MAX];
    vector<pair<int,int>> bridge;
    void dfs(int x,int fa)
    {
        int i,tmp,k;
        st[top++]=x;
        low[x]=dfn[x]=++tot;
        k=0;
        for(auto &to:mp[x])
        {
            if(to==fa&&!k)
            {
                k++;
                continue;
            }
            if(!dfn[to])
            {
                dfs(to,x);
                low[x]=min(low[x],low[to]);
                if(low[to]>dfn[x]) bridge.push_back({x
                    ,to});
            }
            else low[x]=min(low[x],dfn[to]);
        }
        if(low[x]==dfn[x])
        {
            bcc_cnt++;
            do
            {
                tmp=st[--top];
                col[tmp]=bcc_cnt;
            }while(tmp!=x);
        }
    }
    void work(int n,vector<int> *_mp)
    {
        int i;
        for(i=1;i<=n;i++)
        {
            low[i]=dfn[i]=0;
            mp[i]=_mp[i];
        }
        bcc_cnt=top=tot=0;
        bridge.clear();
        for(i=1;i<=n;i++)
        {
            if(!dfn[i]) dfs(i,i);
        }
    }
    void rebuild(int n,vector<int> *g)
    {
        int i;
        for(i=1;i<=n;i++) g[i].clear();
        for(i=1;i<=n;i++)
        {
            for(auto &to:mp[i])
            {
                if(col[i]==col[to]) continue;
                g[col[i]].push_back(col[to]);
            }
        }
    }
}bcc;
/*
bcc.work(n,mp);
*/
```

4.7.3 点双连通分量

```
struct Node_Biconnected_Component
{
    int low[MAX],dfn[MAX],tot;
    vector<int> *mp,cut_node;
    bool cut_vis[MAX];
    void dfs(int x,int fa)
    {
        int i,cnt;
        low[x]=dfn[x]=++tot;
        cnt=0;
        for(auto &to:mp[x])
        {
            if(!dfn[to])
            {
                cnt++;
                dfs(to,x);
                low[x]=min(low[x],low[to]);
                if(x!=fa && low[to]>=dfn[x]) cut_vis[x
                    ]=1;
            }
            else if(to!=fa) low[x]=min(low[x],dfn[to
                ]);
        }
        if(x==fa && cnt>1) cut_vis[x]=1;
        if(cut_vis[x]) cut_node.push_back(x);
    }
    void work(int n,vector<int> *_mp)
    {
        int i;
        mp=_mp;
        cut_node.clear();
        for(i=0;i<=n;i++) low[i]=dfn[i]=cut_vis[i]=0;
        tot=0;
        for(i=1;i<=n;i++)
        {
            if(!dfn[i]) dfs(i,i);
        }
    }
}bcc;
/*
bcc.work(n,mp);
*/
```

4.7.4 圆方树

```
struct Round_Square_Tree
{
    int tot,id,bcc_cnt;
    int low[MAX],dfn[MAX];
    int st[MAX],top;
    vector<int> *mp,*g;
    void dfs(int x,int fa)
    {
        int i,tmp,k;
        st[top++]=x;
        low[x]=dfn[x]=++tot;
        for(auto &to:mp[x])
        {
            if(!dfn[to])
            {
                dfs(to,x);
                low[x]=min(low[x],low[to]);
                if(low[to]==dfn[x])
                {
                    id++;
                    while(top)
                    {
                        tmp=st[--top];
                        g[tmp].push_back(id);
                        g[id].push_back(tmp);
                        if(tmp==to) break;
                    }
                    g[x].push_back(id);
                    g[id].push_back(x);
                }
            }
            else low[x]=min(low[x],dfn[to]);
        }
    }
    int build(int n,vector<int> *_mp,vector<int> *_g
        )
    {
        int i;
        mp=_mp;
        g=_g;
        for(i=0;i<=n;i++) low[i]=dfn[i]=0;
        for(i=0;i<=(n<<1);i++) g[i].clear();
        top=tot=0;
        id=n;
        for(i=1;i<=n;i++)
        {
            if(!dfn[i]) dfs(i,i);
        }
        bcc_cnt=id-n;
        return id;
    }
}rst;
/*
vector<int> mp[MAX],g[MAX<<1];
tot=rst.build(n,mp,g);
*/
```

4.8 团

4.8.1 最大团

```
struct Maximum_Clique
{
    static const int N=;
    vector<int> sol; // vertex of maximum clique
    int mp[N][N/30+1],s[N][N/30+1];
    int n,ans,dp[N];
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++)
        {
            dp[i]=0;
            mem(mp[i],0);
        }
    }
    void add_edge(int a,int b) //0~n-1
    {
        if(a>b) swap(a,b);
        if(a==b) return;
        mp[a][b/32]|=(1<<(b%32));
    }
    bool dfs(int x,int k)
    {
        int c=0,d=0;
        for(int i=0;i<(n+31)/32;i++)
        {
            s[k][i]=mp[x][i];
            if(k!=1) s[k][i]&=s[k-1][i];
            c+=__builtin_popcount(s[k][i]);
        }
        if(c==0)
        {
            if(k>ans)
            {
                ans=k;
                sol.clear();
                sol.pb(x);
                return 1;
            }
            return 0;
        }
        for(int i=0;i<(n+31)/32;i++)
        {
            for(int a=s[k][i];a;d++)
            {
                if(k+(c-d)<=ans) return 0;
                int lb=a&(-a),lg=0;
                a^=lb;
                while(lb!=1)
                {
                    lb=(unsigned int)(lb)>>1;
                    lg++;
                }
                int u=i*32+lg;
                if(k+dp[u]<=ans) return 0;
                if(dfs(u,k+1))
                {
                    sol.pb(x);
                    return 1;
                }
            }
        }
        return 0;
    }
    int maximum_clique()
    {
        ans=0;
        for(int i=n-1;i>=0;i--)
        {
            dfs(i,1);
            dp[i]=ans;
        }
        return ans;
    }
}mcp;
/*
undirected graph
mcp.init(n);
mcp.add_edge(a,b); a,b: 0~n-1
*/
```

4.8.2 极大团计数

```
struct Bron_Kerbosch
{
    static const int N=;
    bitset<N> MASK,ZERO,mp[N];
    int n,cnt_clique;
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++) mp[i].reset();
        ZERO.reset();
        MASK=ZERO;
        MASK.flip();
    }
    void add_edge(int a,int b) //0~n-1 , undir
    {
        if(a==b) return;
        mp[a][b]=mp[b][a]=1;
    }
    void dfs(bitset<N> now,bitset<N> some,bitset<N>
        none)
    {
        if(some.none()&&none.none())//one maximal
            clique
        {
```

```
            cnt_clique++;
            return;
        }
        bitset<N> r=some;
        bool fi=1;
        for(int i=0;i<n;i++)
        {
            if(!r[i]) continue;
            if(fi)
            {
                fi=0;
                r&=mp[i]^MASK;
            }
            now[i]=1;
            dfs(now,some&mp[i],none&mp[i]);
            now[i]=0;
            some[i]=0;
            none[i]=1;
        }
    }
    int count_maximal_clique()
    {
        cnt_clique=0;
        bitset<N> now;
        dfs(now,MASK,ZERO);
        return cnt_clique;
    }
}bk;
/*
undirected graph
bk.init(n);
bk.add_edge(a,b); a,b: 0~n-1
*/
```

4.9 拓扑排序

```
vector<int> topsort(vector<int> &node,vector<int>
    mp[],int *in)
{
    queue<int> q;
    for(auto &it:node)
    {
        if(!in[it]) q.push(it);
    }
    vector<int> toplist;
    while(!q.empty())
    {
        int x=q.front();
        q.pop();
        toplist.push_back(x);
        for(auto &to:mp[x])
        {
            in[to]--;
            if(!in[to]) q.push(to);
        }
    }
    return toplist;
}
```

4.10 2-sat

4.10.1 2-sat 输出任意解

```
//判断是否有解 输出任意一组解O(n+m)
int scc,top,tot;
vector<int> mp[MAX];
int low[MAX],dfn[MAX],belong[MAX];
int stk[MAX],flag[MAX];
int pos[MAX],degree[MAX],ans[MAX],outflag[MAX],cnt;
vector<int> dag[MAX];
void init(int n)
{
    int i;
    for(i=0;i<2*n;i++)
    {
        mp[i].clear();
        dag[i].clear();
        low[i]=0;
        dfn[i]=0;
        stk[i]=0;
        flag[i]=0;
        degree[i]=0;
        outflag[i]=0;
    }
    scc=top=tot=0;
}
void tarjan(int x)
{
    int to,i,temp;
    stk[top++]=x;
    flag[x]=1;
    low[x]=dfn[x]=++tot;
    for(i=0;i<mp[x].size();i++)
    {
        to=mp[x][i];
        if(!dfn[to])
        {
            tarjan(to);
            low[x]=min(low[x],low[to]);
        }
        else if(flag[to]) low[x]=min(low[x],dfn[to]);
    }
    if(low[x]==dfn[x])
    {
        scc++;
        do
        {
```

```
            temp=stk[--top];
            flag[temp]=0;
            belong[temp]=scc;
        }while(temp!=x);
    }
}
void add(int x,int y)
{
    mp[x].pb(y);
}
void topsort(int n)
{
    int i,t;
    queue<int> q;
    cnt=0;
    for(i=1;i<=scc;i++)
    {
        if(degree[i]==0) q.push(i);
        outflag[i]=0;
    }
    while(!q.empty())
    {
        t=q.front();
        q.pop();
        if(outflag[t]==0)
        {
            outflag[t]=1;
            outflag[pos[t]]=2;
        }
        for(i=0;i<sz(dag[t]);i++)
        {
            int to=dag[t][i];
            degree[to]--;
            if(degree[to]==0) q.push(to);
        }
    }
}
void builddag(int n)
{
    int i,j,to;
    for(i=0;i<2*n;i++)
    {
        for(j=0;j<sz(mp[i]);j++)
        {
            to=mp[i][j];
            if(belong[i]!=belong[to])
            {
                degree[belong[i]]++;
                dag[belong[to]].pb(belong[i]);
            }
        }
    }
}
void twosat(int n)
{
    int i;
    for(i=0;i<2*n;i++)
    {
        if(!dfn[i]) tarjan(i);
    }
    for(i=0;i<n;i++)
    {
        if(belong[2*i]==belong[2*i+1])//无解
        {
            puts("NO");
            return;
        }
        pos[belong[2*i]]=belong[2*i+1];
        pos[belong[2*i+1]]=belong[2*i];
    }
    builddag(n);
    topsort(n);
    cnt=0;
    for(i=0;i<2*n;i++)
    {
        if(outflag[belong[i]]==1) ans[cnt++]=i+1;
    }
    for(i=0;i<cnt;i++)
    {
        printf("%d\n",ans[i]);
    }
}
```

4.10.2 2-sat 字典序最小解

```
//判断是否有解 输出字典序最小的解O(n*m)
vector<int> mp[MAX];
bool flag[MAX];
int cnt,s[MAX];
void init(int n)
{
    int i;
    for(i=0;i<2*n;i++)
    {
        mp[i].clear();
    }
    mem(flag,0);
}
bool dfs(int x)
{
    int i;
    if(flag[x^1]) return 0;
    if(flag[x]) return 1;
    s[cnt++]=x;
    flag[x]=1;
    for(i=0;i<sz(mp[x]);i++)
    {
        if(!dfs(mp[x][i])) return 0;
```

```
    }
    return 1;
}
void twosat(int n)
{
    int i;
    for(i=0;i<2*n;i++)
    {
        if(!flag[i]&&!flag[i^1])
        {
            cnt=0;
            if(!dfs(i))
            {
                while(cnt) flag[s[--cnt]]=0;
                if(!dfs(i^1))//无解
                {
                    puts("NO");
                    return;
                }
            }
        }
    }
    for(i=0;i<2*n;i+=2)
    {
        if(flag[i]) printf("%d\n",i+1);
        else printf("%d\n",i+2);
    }
}
```

4.11 支配树

```
struct Dominator_Tree
{
    int n,tot,dfn[MAX],best[MAX],semi[MAX],idom[MAX],id[MAX],fa[MAX];
    vector<int> nex[MAX],pre[MAX],tmp[MAX],son[MAX];
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=n;i++)
        {
            nex[i].clear();
            pre[i].clear();
            tmp[i].clear();
            son[i].clear();
            dfn[i]=0;
            idom[i]=semi[i]=best[i]=fa[i]=i;
        }
    }
    void add_edge(int x,int y)
    {
        nex[x].pb(y);
        pre[y].pb(x);
    }
    int ckmin(int x,int y){return dfn[semi[x]]<dfn[semi[y]]?x:y;}
    int getfa(int k)
    {
        if(k==fa[k]) return k;
        int ret=getfa(fa[k]);
        best[k]=ckmin(best[fa[k]],best[k]);
        return fa[k]=ret;
    }
    void dfs(int x)
    {
        dfn[x]=++tot;
        id[tot]=x;
        for(auto &to:nex[x])
        {
            if(dfn[to]) continue;
            dfs(to);
            son[x].pb(to);
        }
    }
    void tarjan(vector<int> *mp)
    {
        int i,j,k;
        for(i=tot;i;i--)
        {
            k=id[i];
            for(auto &to:pre[k])
            {
                if(!dfn[to]) continue;
                if(dfn[to]<dfn[k])
                {
                    if(dfn[to]<dfn[semi[k]]) semi[k]=to;
                }
                else
                {
                    getfa(to);
                    semi[k]=semi[ckmin(best[to],k)];
                }
            }
            if(k!=semi[k]) tmp[semi[k]].pb(k);
            for(auto &to:tmp[k])
            {
                getfa(to);
                if(semi[best[to]]==k) idom[to]=k;
                else idom[to]=best[to];
            }
            for(auto &to:son[k]) fa[to]=k;
        }
        for(i=2;i<=tot;i++)
        {
            k=id[i];
```

```
        if(idom[k]!=semi[k]) idom[k]=idom[idom[k]];
        if (k!=idom[k])
        {
            mp[idom[k]].push_back(k); //add edge

        }
    }
}
void work(int rt,vector<int> *mp)
{
    for(int i=0;i<=n;i++) mp[i].clear();
    tot=0;
    dfs(rt);
    tarjan(mp);
}
}dt;
/*
dt.init(n);
dt.add_edge(a,b); // DAG
dt.work(rt,mp);
*/
```

4.12 最小斯坦纳树

```
struct Minimum_Steiner_Tree
{
#define type int
const type inf=INF;
static const int N=;
static const int K=;
struct node
{
    int id;
    type v;
    friend bool operator <(node a,node b){return a.v>b.v;}
};
vector<node> mp[N];
type dp[(1<<K)+3][N];
int n,vis[N];
void init(int _n)
{
    n=_n;
    for(int i=1;i<=n;i++) mp[i].clear();
}
void add_edge(int x,int y,type v){ mp[x].push_back({y,v});}
void dijkstra(int s)
{
    int i,to;
    type w;
    priority_queue<node> q;
    for(i=1;i<=n;i++)
    {
        vis[i]=0;
        if(dp[s][i]!=inf) q.push({i,dp[s][i]});
    }
    while(!q.empty())
    {
        node t=q.top();
        q.pop();
        if(vis[t.id]) continue;
        vis[t.id]=1;
        for(auto &it:mp[t.id])
        {
            to=it.id;
            w=it.v;
            if(dp[s][to]>dp[s][t.id]+w)
            {
                dp[s][to]=dp[s][t.id]+w;
                if(!vis[to]) q.push({to,dp[s][to]});
            }
        }
    }
}
type work(vector<int> key_node)
{
    int s,t,i,k;
    type res;
    k=key_node.size();
    for(s=0;s<(1<<k);s++)
    {
        for(i=1;i<=n;i++) dp[s][i]=inf;
    }
    for(i=0;i<k;i++) dp[(1<<i)][key_node[i]]=0;
    for(s=0;s<(1<<k);s++)
    {
        for(t=s&(s-1);t;t=s&(t-1))
        {
            if(t<(s^t)) break;
            for(i=1;i<=n;i++)
            {
                dp[s][i]=min(dp[s][i],dp[t][i]+dp[s^t][i]);
            }
        }
        dijkstra(s);
    }
    res=inf;
    for(i=1;i<=n;i++) res=min(res,dp[(1<<k)-1][i]);
    return res;
}
#undef type
}stn;
```

```
/*
minimum spanning tree including all k key_node
O(n*3^k + m*log(m)*2^k)

stn.init(n);
stn.add_edge(a,b,w);
stn.work(key_node);
*/
```

5 数论

5.1 素数筛

5.1.1 埃筛

```
//x is a prime if prime[x]==x(x>=2)
int p[MAX],tot,prime[MAX];
void init(int n)
{
    int i,j;
    tot=0;
    mem(prime,0);
    prime[1]=1;
    for(i=2;i<=n;i++)
    {
        if(prime[i]) continue;
        p[tot++]=i;
        for(j=i;j<=n;j+=i)
        {
            if(!prime[j]) prime[j]=i;
        }
    }
}
```

5.1.2 线性筛

```
//x is a prime if prime[x]==x(x>=2)
int p[MAX],tot,prime[MAX];
void init_prime(int n)
{
    int i,j;
    tot=0;
    memset(prime,0,sizeof prime);
    prime[1]=1;
    for(i=2;i<=n;i++)
    {
        if(!prime[i]) prime[i]=p[tot++]=i;
        for(j=0;j<tot&&p[j]*i<=n;j++)
        {
            prime[i*p[j]]=p[j];
            if(i%p[j]==0) break;
        }
    }
}
```

5.1.3 区间筛

```
//O(r-l+1)
const int N=2e7+10;
ll p[N],tot;
bool vis[N],prime[N];
void init(ll l,ll r)
{
    ll i,j,sq=sqrt(r+0.5);
    tot=0;
    for(i=0;i<=sq;i++) vis[i]=1;
    for(i=l;i<=r;i++) prime[i-l]=1;
    if(l==0) prime[0]=prime[1]=0;
    if(l==1) prime[0]=0;
    for(i=2;i<=sq;i++)
    {
        if(!vis[i]) continue;
        for(j=i+i;j<=sq;j+=i) vis[j]=0;
        for(j=max(2LL,(l+i-1)/i)*i;j<=r;j+=i) prime[j
            -l]=0;
    }
    for(i=l;i<=r;i++)
    {
        if(prime[i-l]) p[tot++]=i;
    }
}
```

5.2 逆元

5.2.1 exgcd 求逆元

```
ll inv_exgcd(ll a,ll p)
{
    ll g,x,y;
    g=exgcd(a,p,x,y);
    return g==1?(x%p+p)%p:-1;
}
/*
gcd(a,p)!=1, no solution, return -1
*/
```

5.2.2 线性预处理

```
ll inv[MAX];
void init_inv(int n,ll p)
{
    inv[1]=1;
    for(int i=2;i<=n;i++) inv[i]=(p-p/i)*inv[p%i]%p;
}
```

5.3 扩展欧几里得

5.3.1 exgcd

```
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    ll g,tmp;
    g=exgcd(b,a%b,x,y);
    tmp=x;
    x=y;
    y=tmp-a/b*y;
    return g;
}
/*
xa+yb=gcd(a,b)

get x,y
x1=x+b/gcd(a,b)*t
y1=y-a/gcd(a,b)*t
t is any integer
*/
```

5.3.2 ax+by=c

```
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    ll g,tmp;
    g=exgcd(b,a%b,x,y);
    tmp=x;
    x=y;
    y=tmp-a/b*y;
    return g;
}
ll linear_equation(ll a,ll b,ll c,ll &x,ll &y)
{

    ll g,dx,dy,t;
    g=exgcd(a,b,x,y);
    if(a==0&&b==0)
    {
        if(c) return -1;
        x=y=0;
        return g;
    }
    if(c%g) return -1; //no solution
    dx=b/g,dy=-a/g;
    x*=c/g,y*=c/g;
    t=(x%dx-x)/dx;
    if(x+dx*t<=0) t++;
    x+=dx*t,y+=dy*t;
    return g;
}
/*
xa+yb=c

get x,y and x>0 is min
have solution: c%gcd(a,b)==0
x=x0+dx*t
y=y0-dy*t
*/
```

5.4 中国剩余定理

5.4.1 CRT

```
ll qmul(ll a,ll b,ll p)
{
    ll res=0;
    while(b>0)
    {
        if(b&1) res=(res+a)%p;
        a=(a+a)%p;
        b>>=1;
    }
    return res;
}
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    ll g,tmp;
    g=exgcd(b,a%b,x,y);
    tmp=x;
    x=y;
    y=tmp-a/b*y;
    return g;
}
ll inv_exgcd(ll a,ll p)
{
    ll g,x,y;
    g=exgcd(a,p,x,y);
    return g==1?(x%p+p)%p:-1;
```

```
}
ll CRT(int n,ll *a,ll *m)
{
    int i;
    ll p,mi,res,invmi;
    p=1;
    res=0;
    for(i=1;i<=n;i++) a[i]=(a[i]%m[i]+m[i])%m[i];
    for(i=1;i<=n;i++) p*=m[i];
    for(i=1;i<=n;i++)
    {
        mi=p/m[i];
        invmi=inv_exgcd(mi,m[i]);
        res=(res+qmul(a[i]*mi,invmi,p))%p;
    }
    return res;
}
/*
x = a_i (mod m_i) a:1..n, m:1..n
m_i must coprime

CRT return: x = res + k*p, k is interge
*/
```

5.4.2 exCRT

```
ll qmul(ll a,ll b,ll p)
{
    ll res=0;
    while(b>0)
    {
        if(b&1) res=(res+a)%p;
        a=(a+a)%p;
        b>>=1;
    }
    return res;
}
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    ll g,tmp;
    g=exgcd(b,a%b,x,y);
    tmp=x;
    x=y;
    y=tmp-a/b*y;
    return g;
}
ll linear_equation(ll a,ll b,ll c,ll &x,ll &y)
{

    ll g,dx,dy,t;
    g=exgcd(a,b,x,y);
    if(a==0&&b==0)
    {
        if(c) return -1;
        x=y=0;
        return g;
    }
    if(c%g) return -1; //no solution
    dx=b/g;
    x=qmul(x,c/g,dx);
    if(x<0) x+=dx;
    return g;
}
ll exCRT(int n,ll *a,ll *m)
{
    int i;
    ll prep,p,res,g,x,y;
    for(i=1;i<=n;i++) a[i]=(a[i]%m[i]+m[i])%m[i];
    p=m[1];
    res=a[1];
    for(i=2;i<=n;i++)
    {
        /*
        m[i-1]*x-m[i]*y=a[i]-a[i-1]
        m[i-1]=p, a[i-1]=res
        -> p*x-m[i]*y=a[i]-res
        */
        g=linear_equation(p,m[i],(a[i]-res)%m[i]+m[i
            ],x,y);
        if(g==-1) return -1;
        prep=p;
        p=p/g*m[i];
        res=(res+qmul(prep,x,p))%p;
    }
    return res;
}
/*
x = a_i (mod m_i) a:1..n, m:1..n

exCRT return: x = res + k*p, k is interge
*/
```

5.5 欧拉函数

<=n 且与 n 互质的数的和：n*phi[n]/2

5.5.1 直接求

```
//O(sqrt(n))
int get_phi(int n)
```

```
{
    int ans,i;
    ans=n;
    for(i=2;i*i<=n;i++)
    {
        if(n%i==0)
        {
            ans=ans-ans/i;
            while(n%i==0) n/=i;
        }
    }
    if(n>1) ans=ans-ans/n;
    return ans;
}
```

5.5.2 线性筛

```
int p[MAX],phi[MAX],tot,prime[MAX];
void init_phi(int n)
{
    int i,j;
    tot=0;
    memset(prime,0,sizeof prime);
    prime[1]=phi[1]=1;
    for(i=2;i<=n;i++)
    {
        if(!prime[i])
        {
            prime[i]=p[tot++]=i;
            phi[i]=i-1;
        }
        for(j=0;i*p[j]<=n;j++)
        {
            prime[i*p[j]]=p[j];
            if(i%p[j]==0)
            {
                phi[i*p[j]]=phi[i]*p[j];
                break;
            }
            else phi[i*p[j]]=phi[i]*(p[j]-1);
        }
    }
}
```

5.6 莫比乌斯函数

```
int mo[MAX],prime[MAX],tot;
bool flag[MAX];
void initmo(int n)
{
    int i,j;
    mem(flag,0);
    mem(mo,0);
    tot=0;
    mo[1]=1;
    for(i=2;i<=n;i++)
    {
        if(!flag[i])
        {
            prime[tot++]=i;
            mo[i]=-1;
        }
        for(j=0;j<tot&&prime[j]*i<=n;j++)
        {
            flag[i*prime[j]]=1;
            if(i%prime[j]==0)
            {
                mo[prime[j]*i]=0;
                break;
            }
            mo[prime[j]*i]=-mo[i];
        }
    }
}
```

5.7 Berlekamp-Massey

```
//Berlekamp-Massey
typedef vector<int> VI;
namespace linear_seq
{
    #define rep(i,a,n) for (int i=a;i<n;i++)
    #define SZ(x) ((int)(x).size())
    const ll mod=1e9+7;
    ll powmod(ll a,ll b){ll res=1;a%=mod; assert(b
        >=0); for(;b;b>>=1){if(b&1)res=res*a%mod;a=a
        *a%mod;}return res;}
    const int N=10010;
    ll res[N],base[N],_c[N],_md[N];
    vector<int> Md;
    void mul(ll *a,ll *b,int k)
    {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i
            +j]+a[i]*b[j])%mod;
        for (int i=k+k-1;i>=k;i--) if (_c[i])
            rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[
                j]]-_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    int solve(ll n,VI a,VI b){
        ll ans=0,pnt=0;
        int k=SZ(a);
        assert(SZ(a)==SZ(b));
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
```

```
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) res[i]=base[i]=0;
        res[0]=1;
        while ((1ll<<pnt)<=n) pnt++;
        for (int p=pnt;p>=0;p--) {
            mul(res,res,k);
            if ((n>>p)&1) {
                for (int i=k-1;i>=0;i--) res[i+1]=res[
                    i];res[0]=0;
                rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j
                    ]]-res[k]*_md[Md[j]])%mod;
            }
        }
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
    VI BM(VI s){
        VI C(1,1),B(1,1);
        int L=0,m=1,b=1;
        rep(n,0,SZ(s)){
            ll d=0;
            rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
            if(d==0) ++m;
            else if(2*L<=n){
                VI T=C;
                ll c=mod-d*powmod(b,mod-2)%mod;//□□□
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%
                    mod;
                L=n+1-L; B=T; b=d; m=1;
            } else {
                ll c=mod-d*powmod(b,mod-2)%mod;//□□□
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%
                    mod;
                ++m;
            }
        }
        return C;
    }
    int gao(VI a,ll n)
    {
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)
            ));
    }
};//linear_seq::gao(VI{},n-1)
```

5.8 exBSGS

```
//a^□x¡b (mod c)
ll exBSGS(ll a,ll b,ll c)
{
    ll i,g,d,num,now,sq,t,x,y;
    if(c==1) return b?-1:(a!=1);
    if(b==1) return a?0:-1;
    if(a%c==0) return b?-1:1;
    num=0;
    d=1;
    while((g=__gcd(a,c))>1)
    {
        if(b%g) return -1;
        num++;
        b/=g;
        c/=g;
        d=(d*a/g)%c;
        if(d==b) return num;
    }
    mp.clear();
    sq=ceil(sqrt(c));
    t=1;
    for(i=0;i<sq;i++)
    {
        if(!mp.count(t)) mp[t]=i;
        else mp[t]=min(mp[t],i);
        t=t*a%c;
    }
    for(i=0;i<sq;i++)
    {
        exgcd(d,c,x,y);
        x=(x*b%c+c)%c;
        if(mp.count(x)) return i*sq+mp[x]+num;
        d=d*t%c;
    }
    return -1;
}
```

5.9 Miller_Rabin+Pollard_rho

```
struct Montgomery
{
    #define i64 int64_t
    #define u64 uint64_t
    #define u128 __uint128_t
    u64 mod,R;
    void set_mod(u64 p)
    {
        mod=R=p;
        for(int i=0;i<5;i++) R*=2-mod*R;
    }
    u64 mul_add(u64 a,u64 b,u64 c)
    {
        u128 d=u128(a)*b;
```

```
        u64 e=c+mod+(d>>64);
        u64 f=u64(d)*R;
        u64 g=(u128(f)*mod)>>64;
        return e-g;
    }
    u64 mul(u64 a,u64 b) {return mul_add(a,b,0);}
    #undef i64
    #undef u64
    #undef u128
};
// m.set_mod(p);
ll qpow(ll a,ll b,ll p)
{
    ll res=1;
    while(b>0)
    {
        if(b&1) res=(__int128)res*a%p;
        a=(__int128)a*a%p;
        b>>=1;
    }
    return res;
}
bool miller_rabin(ll n,const initializer_list<ll>&
    as)
{
    int e=__builtin_ctzll(n-1);
    return all_of(as.begin(),as.end(),[&](ll it)
    {
        if(n<=it) return true;
        ll z=qpow(it,(n-1)>>e,n); // it^(t*2^e)
        if(z==1||z==n-1) return true;
        for(int i=0;i<e-1;i++)
        {
            z=__int128(z)*z%n;
            if(z==1) return false;
            if(z==n-1) return true;
        }
        return false;
    });
}
bool is_prime(ll n)
{
    if(n<=2) return n==2;
    if(!(n&1)) return false;
    if(n<4759123141LL) return miller_rabin(n
        ,{2,7,61});
    return miller_rabin(n
        ,{2,325,9375,28178,450775,9780504,1795265022})
        ;
}
ll pollard_rho(ll n)
{
    Montgomery m;
    m.set_mod(n);
    if(!(n&1)) return 2;
    int i,j,k;
    ll q1,q2,g1,g2,w1,w2,x1,x2,y1,y2,z1,z2,g,c,x,z;
    constexpr ll c1=1;
    constexpr ll c2=2;
    constexpr ll M=512;
    w1=1,w2=2;
retry:
    z1=w1,z2=w2;
    for(i=M;;i<<=1)
    {
        x1=z1+n,x2=z2+n;
        for(j=0;j<i;j+=M)
        {
            y1=z1,y2=z2;
            q1=1,q2=2;
            for(k=0;k<M;k++)
            {
                z1=m.mul_add(z1,z1,c1);
                z2=m.mul_add(z2,z2,c2);
                q1=m.mul(q1,(x1-z1));
                q2=m.mul(q2,(x2-z2));
            }
            g=__gcd((ll)m.mul(q1,q2),n);
            if(g==1) continue;
            if(g!=n) return g;
            g1=__gcd(q1,n);
            g2=__gcd(q2,n);
            if(g1!=1) c=c1,x=x1,z=y1,g=g1;
            else c=c2,x=x2,z=y2,g=g2;
            if(g==n)
            {
                do{
                    z=m.mul_add(z,z,c);
                    g=__gcd(n,x-z);
                }while(g==1);
            }
            if(g!=n) return g;
            w1+=2,w2+=2;
            goto retry;
        }
    }
}
void find_fac(ll n,vector<ll>& res)
{
    if(n<=1) return;
    if(is_prime(n)) {res.push_back(n); return;}
    ll p=pollard_rho(n);
    find_fac(p,res);
    find_fac(n/p,res);
}
vector<ll> factorize(ll n)
{
    vector<ll> res;
```

```
    find_fac(n,res);
    return res;
}
```

5.10 第二类 Stirling 数

```
//dp[i][j表示]个元素划分到个不可区分的非空盒子里的方案数。ik
ll dp[MAX][MAX];
void init()
{
    ll i,j;
    mem(dp,0);
    dp[1][1]=1;
    for(i=2;i<MAX;i++)
    {
        for(j=1;j<=i;j++)
        {
            dp[i][j]=(dp[i-1][j-1]+j*dp[i-1][j])%mod;
        }
    }
}
```

5.11 原根

原根性质

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

指标法则

1. $I(a*b)\ I(a)+I(b)(mod p-1)$
2. $I(a^k)\ k*I(a)(mod p-1)$

```
int p[MAX],tot,prime[MAX];
void init(int n)
{
    int i,j;
    tot=0;
    mem(prime,0);
    prime[1]=1;
    for(i=2;i<=n;i++)
    {
        if(!prime[i]) prime[i]=p[tot++]=i;
        for(j=0;j<tot&&p[j]*i<=n;j++)
        {
            prime[i*p[j]]=p[j];
            if(i%p[j]==0) break;
        }
    }
}
ll pow2(ll a,ll b,ll p)
{
    ll res=1;
    while(b)
    {
        if(b&1) res=res*a%p;
        a=a*a%p;
        b>>=1;
    }
    return res;
}
int tp[MAX];
int find_root(int x)//求素数原根
{
    if(x==2) return 1;
    int f,phi=x-1;
    tp[0]=0;
    for(int i=0;phi&&i<tot;i++)
    {
        if(phi%p[i]==0)
        {
            tp[++tp[0]]=p[i];
            while(phi%p[i]==0) phi/=p[i];
        }
    }
    if(phi!=1) tp[++tp[0]]=phi;
    phi=x-1;
    for(int g=2;g<=x-1;g++)
    {
        f=1;
        for(int i=1;i<=tp[0];i++)
        {
            if(pow2(g,phi/tp[i],x)==1)
            {
                f=0;
                break;
            }
        }
        if(f) return g;
    }
    return 0;
}
int I[MAX];
void get_I(int p)//求指标表
{
    int g,now;
    g=find_root(p);
    now=1;
    for(int i=1;i<p;i++)
    {
```

```
        now=now*g%p;
        I[now]=i;
    }
}
```

5.12 二次剩余

```
//O(log^2)
struct Tonelli_Shanks
{
    ll mul2(ll a,ll b,ll p)
    {
        ll res=0;
        a%=p;
        while(b)
        {
            if(b&1)
            {
                res+=a;
                if(res>=p) res-=p;
            }
            a+=a;
            if(a>=p) a-=p;
            b>>=1;
        }
        return res;
    }
    ll pow2(ll a,ll b,ll p)
    {
        ll res=1;
        while(b)
        {
            if(b&1) res=mul2(res,a,p);
            a=mul2(a,a,p);
            b>>=1;
        }
        return res;
    }
    ll sqrt(ll n,ll p)
    {
        if(p==2) return (n&1)?1:-1;
        if(pow2(n,p>>1,p)!=1) return -1;
        if(p&2) return pow2(n,(p+1)>>2,p);
        ll q,z,c,r,t,tmp,s,i,m;
        s=__builtin_ctzll(p^1);
        q=p>>s;
        z=2;
        for(;pow2(z,p>>1,p)==1;z++);
        c=pow2(z,q,p);
        r=pow2(n,(q+1)>>1,p);
        t=pow2(n,q,p);
        for(m=s;t!=1;)
        {
            for(i=0,tmp=t;tmp!=1;i++) tmp=tmp*tmp%p;
            for(;i<--m;) c=c*c%p;
            r=r*c%p;
            c=c*c%p;
            t=t*c%p;
        }
        return r;
    }
}ts;
```

6 组合数学

6.1 组合数

6.1.1 公式预处理

```
ll qpow(ll a,ll b)
{
    ll res=1;
    while(b>0)
    {
        if(b&1) res=res*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return res;
}
ll inv(ll x){return qpow(x,mod-2);}
int fac[MAX],invfac[MAX];
void init_comb(int n)
{
    assert(n<MAX);
    fac[0]=1;
    for(int i=1;i<=n;i++) fac[i]=1ll*fac[i-1]*i%mod;
    invfac[n]=inv(fac[n]);
    for(int i=n-1;~i;i--) invfac[i]=1ll*invfac[i
        +1]*(i+1)%mod;
}
ll C(int n,int m)
{
    if(m>n||m<0||n<0) return 0;
    return 1ll*fac[n]*invfac[m]%mod*invfac[n-m]%mod;
}
ll A(int n,int m)
{
    if(m>n||m<0||n<0) return 0;
    return 1ll*fac[n]*invfac[n-m]%mod;
}
```

6.1.2 杨辉三角递推

```
int comb[MAX][105];
void init_comb(int n,int m)
```

```
{
    int i,j;
    comb[0][0]=1;
    for(i=1;i<=n;i++)
    {
        comb[i][0]=1;
        for(j=1;j<=i;j++)
        {
            comb[i][j]=comb[i-1][j]+comb[i-1][j-1];
            if(comb[i][j]>=mod) comb[i][j]-=mod;
        }
    }
}
ll C(int n,int m)
{
    if(m>n||m<0||n<0) return 0;
    return comb[n][m];
}
```

6.2 卢卡斯定理

6.2.1 Lucas

```
ll Lucas(ll n,ll m,int p)
{
    if(m==0) return 1;
    return C(n%p,m%p)*Lucas(n/p,m/p,p)%p;
}
/*
p must be a prime number
init(p-1);
*/
```

6.2.2 exLucas

```
namespace exLucas
{
    ll qpow(ll a,ll b,ll p)
    {
        ll res=1;
        while(b>0)
        {
            if(b&1) res=res*a%p;
            a=a*a%p;
            b>>=1;
        }
        return res;
    }
    ll qmul(ll a,ll b,ll p)
    {
        ll res=0;
        while(b>0)
        {
            if(b&1) res=(res+a)%p;
            a=(a+a)%p;
            b>>=1;
        }
        return res;
    }
    ll exgcd(ll a,ll b,ll &x,ll &y)
    {
        if(b==0)
        {
            x=1;
            y=0;
            return a;
        }
        ll g,tmp;
        g=exgcd(b,a%b,x,y);
        tmp=x;
        x=y;
        y=tmp-a/b*y;
        return g;
    }
    ll inv(ll a,ll p)
    {
        ll g,x,y,res;
        g=exgcd(a,p,x,y);
        res=(g==1?(x+p)%p:-1);
        assert(res!=-1);
        return res;
    }
    ll CRT(vector<ll> a,vector<ll> m)
    {
        int i,n;
        ll p,mi,res,invmi;
        n=a.size();
        p=1;
        res=0;
        for(i=0;i<n;i++) a[i]=(a[i]%m[i]+m[i])%m[i];
        for(i=0;i<n;i++) p*=m[i];
        for(i=0;i<n;i++)
        {
            mi=p/m[i];
            invmi=inv(mi,m[i]);
            res=(res+qmul(a[i]*mi,invmi,p))%p;
        }
        return res;
    }
    map<ll,pair<vector<ll>,vector<ll>>> mp;
    map<pair<ll,ll>,vector<ll>> fac;
    void init(vector<ll> mod_list)
    {
        ll i,j,p;
        mp.clear();
        fac.clear();
        for(auto &mod_i:mod_list)
```

```
    {
        p=mod_i;
        vector<ll> a,b;
        for(i=2;i*i<=p;i++)
        {
            if(p%i) continue;
            b.push_back(1LL);
            while(p%i==0) b[b.size()-1]*=i,p/=i;
            a.push_back(i);
        }
        if(p>1) a.push_back(p),b.push_back(p);
        mp[mod_i]={a,b};
        for(i=0;i<a.size();i++)
        {
            if(fac.count({a[i],b[i]})) continue;
            vector<ll> fac_tmp(b[i]+1);
            fac_tmp[0]=1;
            for(j=1;j<=b[i];j++)
            {
                if(j%a[i]) fac_tmp[j]=fac_tmp[j-1]*
                    j%b[i];
                else fac_tmp[j]=fac_tmp[j-1];
            }
            fac[{a[i],b[i]}]=fac_tmp;
        }
    }
}
ll cal_fac(ll n,ll x,ll p)
{
    if(!n) return 1LL;
    ll res=1;
    assert(fac.count({x,p}));
    res=res*fac[{x,p}][p-1]%p;
    res=qpow(res,n/p,p);
    res=res*fac[{x,p}][n%p]%p;
    return res*cal_fac(n/x,x,p)%p;
}
ll comb(ll n,ll m,ll x,ll p)
{
    if(m>n) return 0;
    ll i,cnt;
    cnt=0;
    for(i=n;i;i/=x) cnt+=i/x;
    for(i=m;i;i/=x) cnt-=i/x;
    for(i=n-m;i;i/=x) cnt-=i/x;
    return qpow(x,cnt,p)* \
          cal_fac(n,x,p)%p* \
          inv(cal_fac(m,x,p),p)%p* \
          inv(cal_fac(n-m,x,p),p)%p;
}
ll C(ll n,ll m,ll p)
{
    if(m>n||m<0||n<0) return 0;
    ll i,res;
    vector<ll> a,b,resa;
    assert(mp.count(p));
    a=mp[p].first;
    b=mp[p].second;
    for(i=0;i<a.size();i++) resa.push_back(comb(n
        ,m,a[i],b[i]));
    res=CRT(resa,b);
    assert(res!=-1);
    return res;
}
};
/*
mod=p1^k1 * p2^k2 * ...
init: O(sqrt(mod)+sum{p^k})
C(n,m,mod): O(log)

exLucas::init({mod}); //init mod list
exLucas::C(n,m,mod);
*/
```

6.3 卡特兰数公式

```
C(2*n,n)-C(2*n,n-1)
```

7 多项式

7.1 FFT

```
namespace FFT
{
    #define rep(i,a,b) for(int i=(a);i<=(b);i++)
    const double pi=acos(-1);
    const int maxn=(1<<19)+10;
    struct cp
    {
        double a,b;
        cp(){}
        cp(double _x,double _y){a=_x,b=_y;}
        cp operator +(const cp &o)const{return (cp){a
            +o.a,b+o.b};}
        cp operator -(const cp &o)const{return (cp){a
            -o.a,b-o.b};}
        cp operator *(const cp &o)const{return (cp){a
            *o.a-b*o.b,b*o.a+a*o.b};}
        cp operator *(const double &o)const{return (
            cp){a*o,b*o};}
        cp operator !()const{return (cp){a,-b};}
    }x[maxn],y[maxn],z[maxn],w[maxn];
    void fft(cp x[],int k,int v)
    {
        int i,j,l;
        for(i=0,j=0;i<k;i++)
```

```
        {
            if(i>j)swap(x[i],x[j]);
            for(l=k>>1;(j^=l)<l;l>>=1);
        }
        w[0]=(cp){1,0};
        for(i=2;i<=k;i<<=1)
        {
            cp g=(cp){cos(2*pi/i),(v?-1:1)*sin(2*pi/i
                )};
            for(j=(i>>1);j>=0;j-=2)w[j]=w[j>>1];
            for(j=1;j<i>>1;j+=2)w[j]=w[j-1]*g;
            for(j=0;j<k;j+=i)
            {
                cp *a=x+j,*b=a+(i>>1);
                for(l=0;l<i>>1;l++)
                {
                    cp o=b[l]*w[l];
                    b[l]=a[l]-o;
                    a[l]=a[l]+o;
                }
            }
        }
        if(v)for(i=0;i<k;i++)x[i]=(cp){x[i].a/k,x[i].
            b/k};
    }

    // a=b*c
    // b[0..l1]
    // c[0..l2]
    void mul(int *a,int *b,int *c,int l1,int l2)
    {
        if(l1<128&&l2<128)
        {
            rep(i,0,l1+l2)a[i]=0;
            rep(i,0,l1)rep(j,0,l2)a[i+j]+=b[i]*c[j];
            return;
        }
        int K;
        for(K=1;K<=l1+l2;K<<=1);
        rep(i,0,l1)x[i]=cp(b[i],0);
        rep(i,0,l2)y[i]=cp(c[i],0);
        rep(i,l1+1,K)x[i]=cp(0,0);
        rep(i,l2+1,K)y[i]=cp(0,0);
        fft(x,K,0);fft(y,K,0);
        rep(i,0,K)z[i]=x[i]*y[i];
        fft(z,K,1);
        rep(i,0,l1+l2)a[i]=(ll)(z[i].a+0.5);
    }
};
```

7.2 NTT

```
namespace NTT
{
    const int g=3;
    const int p=998244353;
    int wn[35];
    int qpow(int a,int b)
    {
        int res=1;
        while(b>0)
        {
            if(b&1) res=1ll*res*a%p;
            a=1ll*a*a%p;
            b>>=1;
        }
        return res;
    }
    void getwn()
    {
        assert(p==mod);
        for(int i=0;i<25;i++) wn[i]=qpow(g,(p-1)/(1LL
            <<i));
    }
    void ntt(vector<int> &a,int len,int f)
    {
        int i,j=0,t,k,w,id;
        for(i=1;i<len-1;i++)
        {
            for(t=len;j^=t>>=1,~j&t;);
            if(i<j) swap(a[i],a[j]);
        }
        for(i=1,id=1;i<len;i<<=1,id++)
        {
            t=i<<1;
            for(j=0;j<len;j+=t)
            {
                for(k=0,w=1;k<i;k++,w=1ll*w*wn[id]%p)
                {
                    int x=a[j+k],y=1ll*w*a[j+k+i]%p;
                    a[j+k]=x+y;
                    if(a[j+k]>=p) a[j+k]-=p;
                    a[j+k+i]=x-y;
                    if(a[j+k+i]<0) a[j+k+i]+=p;
                }
            }
        }
        if(f)
        {
            for(i=1,j=len-1;i<j;i++,j--) swap(a[i],a[
                j]);
            int inv=qpow(len,p-2);
            for(i=0;i<len;i++) a[i]=1ll*a[i]*inv%p;
        }
    }
    vector<int> qpow(vector<int> a,int b)//limt: sz(
        a)*b is small
```

```
    {
        int len,i,l1;
        l1=a.size();
        for(len=1;len<(l1+1)*b-1;len<<=1);
        a.resize(len,0);
        ntt(a,len,0);
        vector<int> res(len);
        for(i=0;i<len;i++) res[i]=pow2(a[i],b);
        ntt(res,len,1);
        res.resize((l1+1)*b-1);
        return res;
    }
    vector<int> mul(vector<int> a,vector<int> b)
    {
        int len,i,l1,l2;
        l1=a.size();
        l2=b.size();
        for(len=1;len<l1+l2;len<<=1);
        a.resize(len,0);
        b.resize(len,0);
        ntt(a,len,0);ntt(b,len,0);
        vector<int> res(len);
        for(i=0;i<len;i++) res[i]=1ll*a[i]*b[i]%p;
        ntt(res,len,1);
        res.resize(l1+l2-1);
        return res;
    }

    //get kth
    vector<int> merge_generating_functions(vector<
        vector<int>> &dp,int k)
    {
        int i,j;
        priority_queue<pair<int,int>> q;
        for(i=0;i<dp.size();i++) q.push({-dp[i].size
            (),i});
        while(q.size()>1)
        {
            i=q.top().second;
            q.pop();
            j=q.top().second;
            q.pop();
            dp[i]=mul(dp[i],dp[j]);
            if(dp[i].size()>k) dp[i].resize(k+1);
            q.push({-dp[i].size(),i});
        }
        return dp[q.top().second];
    }
};//NTT::getwn();
```

7.3 FWT

```
namespace FWT
{
    const int p=998244353;
    const ll inv2=(p+1)/2;
    void fwt(vector<int> &a,int n,int f,int v)
    {
        for(int d=1;d<n;d<<=1)
        {
            for(int m=d<<1,i=0;i<n;i+=m)
            {
                for(int j=0;j<d;j++)
                {
                    ll x=a[i+j],y=a[i+j+d];
                    if(v==1)
                    {
                        if(f==1) a[i+j]=(x+y)%p,a[i+j+d
                            ]=(x-y+p)%p;//xor
                        else if(f==2) a[i+j]=(x+y)%p;//
                            and
                        else if(f==3) a[i+j+d]=(x+y)%p;
                            //or
                    }
                    else
                    {
                        if(f==1) a[i+j]=(x+y)*inv2%p,a[
                            i+j+d]=(x-y+p)%p*inv2%p;//
                            xor
                        else if(f==2) a[i+j]=(x-y+p)%p;
                            //and
                        else if(f==3) a[i+j+d]=(y-x+p)%
                            p;//or
                    }
                }
            }
        }
    }

    vector<int> XOR(vector<int> a,vector<int> b)
    {
        int n,len;
        n=a.size();
        for(len=1;len<n;len<<=1);
        a.resize(len,0);
        b.resize(len,0);
        fwt(a,len,1,1);
        fwt(b,len,1,1);
        for(int i=0;i<len;i++) a[i]=1ll*a[i]*b[i]%p;
        fwt(a,len,1,-1);
        return a;
    }
    vector<int> AND(vector<int> a,vector<int> b)
    {
        int n,len;
        n=a.size();
        for(len=1;len<n;len<<=1);
```

```
        a.resize(len,0);
        b.resize(len,0);
        fwt(a,len,2,1);
        fwt(b,len,2,1);
        for(int i=0;i<len;i++) a[i]=1ll*a[i]*b[i]%p;
        fwt(a,len,2,-1);
        return a;
    }
    vector<int> OR(vector<int> a,vector<int> b)
    {
        int n,len;
        n=a.size();
        for(len=1;len<n;len<<=1);
        a.resize(len,0);
        b.resize(len,0);
        fwt(a,len,3,1);
        fwt(b,len,3,1);
        for(int i=0;i<len;i++) a[i]=1ll*a[i]*b[i]%p;
        fwt(a,len,3,-1);
        return a;
    }
}
```

7.4 拉格朗日插值

```
namespace polysum {
    #define rep(i,a,n) for (int i=a;i<n;i++)
    #define per(i,a,n) for (int i=n-1;i>=a;i--)
    const int D=101000;
    ll a[D],tmp[D],f[D],g[D],p[D],p1[D],p2[D],b[D],h[D][2],C[D];
    ll powmod(ll a,ll b){ll res=1;a%=mod;assert(b>=0);for(;b;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
    ll calcn(int d,ll *a,ll n) { // a[0].. a[d] a[n]
        if (n<=d) return a[n];
        p1[0]=p2[0]=1;
        rep(i,0,d+1) {
            ll t=(n-i+mod)%mod;
            p1[i+1]=p1[i]*t%mod;
        }
        rep(i,0,d+1) {
            ll t=(n-d+i+mod)%mod;
            p2[i+1]=p2[i]*t%mod;
        }
        ll ans=0;
        rep(i,0,d+1) {
            ll t=g[i]*g[d-i]%mod*p1[i]%mod*p2[d-i]%mod*a[i]%mod;
            if ((d-i)&1) ans=(ans-t+mod)%mod;
            else ans=(ans+t)%mod;
        }
        return ans;
    }
    void init(int M) {
        f[0]=f[1]=g[0]=g[1]=1;
        rep(i,2,M+5) f[i]=f[i-1]*i%mod;
        g[M+4]=powmod(f[M+4],mod-2);
        per(i,1,M+4) g[i]=g[i+1]*(i+1)%mod;
    }
    ll polysum(ll n,ll *a,ll m) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]
        rep(i,0,m+1) tmp[i]=a[i];
        tmp[m+1]=calcn(m,tmp,m+1);
        rep(i,1,m+2) tmp[i]=(tmp[i-1]+tmp[i])%mod;
        return calcn(m+1,tmp,n-1);
    }
    ll qpolysum(ll R,ll n,ll *a,ll m) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
        if (R==1) return polysum(n,a,m);
        a[m+1]=calcn(m,a,m+1);
        ll r=powmod(R,mod-2),p3=0,p4=0,c,ans;
        h[0][0]=0;h[0][1]=1;
        rep(i,1,m+2) {
            h[i][0]=(h[i-1][0]+a[i-1])*r%mod;
            h[i][1]=h[i-1][1]*r%mod;
        }
        rep(i,0,m+2) {
            ll t=g[i]*g[m+1-i]%mod;
            if (i&1) p3=((p3-h[i][0]*t)%mod+mod)%mod,p4=((p4-h[i][1]*t)%mod+mod)%mod;
            else p3=(p3+h[i][0]*t)%mod,p4=(p4+h[i][1]*t)%mod;
        }
        c=powmod(p4,mod-2)*(mod-p3)%mod;
        rep(i,0,m+2) h[i][0]=(h[i][0]+h[i][1]*c)%mod;
        rep(i,0,m+2) C[i]=h[i][0];
        ans=(calcn(m,C,n)*powmod(R,n)-c)%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
} // polysum::init();
```

8 矩阵

8.1 矩阵类

```
const ll mod=1e9+7;
struct Matrix
{
    static const int N=;
    int n;
    ll c[N][N];
    Matrix(){}
    Matrix(int _n,ll v=0)
    {
```

```
        int i,j;
        n=_n;
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                c[i][j]=v;
            }
        }
    }
    void init_identity_matrix() {for(int i=0;i<n;i
        ++) c[i][i]=1;}
    Matrix operator *(const Matrix &b)const
    {
        int i,j,k;
        Matrix res(n);
        for(k=0;k<n;k++)
        {
            for(i=0;i<n;i++)
            {
                if(!c[i][k]) continue;
                for(j=0;j<n;j++)
                {
                    res.c[i][j]+=c[i][k]*b.c[k][j];
                    if(res.c[i][j]>=mod) res.c[i][j]%=
                        mod;
                }
            }
        }
        return res;
    }
};
Matrix matqpow(Matrix a,ll b)
{
    Matrix res(a.n);
    res.init_identity_matrix();
    while(b)
    {
        if(b&1) res=res*a;
        a=a*a;
        b>>=1;
    }
    return res;
}
```

8.2 高斯消元

8.2.1 浮点数方程组

```
struct Gauss
{
    const double eps=1e-7;
    double mp[][];
    int gauss(int n,int m)
    {
        int i,j,k,pos,r;
        double tmp;
        r=0;
        for(k=1;k<=m;k++)
        {
            pos=r+1;
            if(pos>n) return -1; // no solution
            for(i=pos+1;i<=n;i++)
            {
                if(fabs(mp[i][k])>fabs(mp[pos][k]))
                    pos=i;
            }
            if(fabs(mp[pos][k])<eps) continue;
            r++;
            swap(mp[pos],mp[r]);
            tmp=mp[r][k];
            for(j=k;j<=m+1;j++) mp[r][j]/=tmp;
            for(i=r+1;i<=n;i++)
            {
                tmp=mp[i][k];
                for(j=k;j<=m+1;j++)
                {
                    mp[i][j]-=mp[r][j]*tmp;
                }
            }
        }
        return r;
    }
    int work(int n,int m,double *res)
    {
        int i,j,cnt;
        cnt=gauss(n,m);
        if(cnt==-1) return -1;
        for(i=cnt+1;i<=n;i++)
        {
            if(fabs(mp[i][m+1])>eps)
            {
                // no solution
                return -1;
            }
        }
        if(cnt<m)
        {
            // multi solution
            return 1;
        }
        res[m]=mp[m][m+1];
        for(i=m-1;i>=1;i--)
        {
            res[i]=mp[i][m+1];
            for(j=i+1;j<=m;j++)
            {
                res[i]-=mp[i][j]*res[j];
```

```
            }
        }
        return 0;
    }
}gs;
/*
(mp[1][1]*x1) + (mp[1][2]*x2) + ... + (mp[1][m]*xm) = mp[1][m+1]
(mp[2][1]*x1) + (mp[1][2]*x2) + ... + (mp[2][m]*xm) = mp[2][m+1]
...
(mp[n][1]*x1) + (mp[n][2]*x2) + ... + (mp[n][m]*xm) = mp[n][m+1]

x x x x | x
0 x x x | x
0 0 x x | x
0 0 0 x | x

O(n*m^2) m<=n

gs.work(n,m,res); mp[1..n][1..m+1], res[1..m]

-1: no solution
0 : one solution
1 : multi solution
*/
```

8.2.2 异或方程组

```
struct Gauss
{
    bitset<> mp[];
    int gauss_jordan(int n,int m)
    {
        int i,j,k,pos,r;
        r=0;
        for(k=1;k<=m;k++)
        {
            pos=r+1;
            if(pos>n) return -1; // no solution
            while(pos<n&&!mp[pos][k]) pos++;
            if(!mp[pos][k]) continue;
            r++;
            swap(mp[pos],mp[r]);
            for(i=1;i<=n;i++)
            {
                if(i!=r&&mp[i][k]) mp[i]^=mp[r];
            }
        }
        return r;
    }
    int work(int n,int m,int *res)
    {
        int i,j,cnt;
        cnt=gauss_jordan(n,m);
        if(cnt==-1) return -1;
        for(i=cnt+1;i<=n;i++)
        {
            if(mp[i][m+1])
            {
                // no solution
                return -1;
            }
        }
        if(cnt<m)
        {
            // multi solution
            return 1;
        }
        for(i=1;i<=m;i++) res[i]=mp[i][m+1];
        return 0;
    }
}gs;
/*
(mp[1][1]*x1) xor (mp[1][2]*x2) xor ... xor (mp[1][m]*xm) = mp[1][m+1]
(mp[2][1]*x1) xor (mp[1][2]*x2) xor ... xor (mp[2][m]*xm) = mp[2][m+1]
...
(mp[n][1]*x1) xor (mp[n][2]*x2) xor ... xor (mp[n][m]*xm) = mp[n][m+1]

a:0/1 x:0/1

x 0 0 0 | x
0 x 0 0 | x
0 0 x 0 | x
0 0 0 x | x


O((n*m^2)/64)

gs.work(n,m,res); mp[1..n][1..m+1], res[1..m]

-1: no solution
0 : one solution
1 : multi solution
*/
```

8.2.3 同余方程组

```
namespace Gauss
{
    int p;
    ll mp[905][905],sol[905];
    void set_mod(int _p)
    {
```

```
    p=_p;
    mem(mp,0);
    mem(sol,0);
}
ll pow2(ll a,ll b)
{
    ll res=1;
    while(b)
    {
        if(b&1) res=res*a%p;
        a=a*a%p;
        b>>=1;
    }
    return res;
}
ll inv(ll x){return pow2(x,p-2);}
ll lcm(ll a,ll b){return a/__gcd(a,b)*b;}
int gauss(int n,int m)
{
    int r,c,id,i,j;
    ll tmp,ta,tb;
    r=c=0;
    while(r<n&&c<m)
    {
        id=r;
        for(i=r+1;i<n;i++)
        {
            if(abs(mp[i][c])>abs(mp[id][c])) id=i;
        }
        if(id!=r)
        {
            for(i=0;i<=m;i++) swap(mp[r][i],mp[id
                ][i]);
        }
        if(abs(mp[r][c])!=0)
        {
            for(i=r+1;i<n;i++)
            {
                if(abs(mp[i][c])==0) continue;
                tmp=lcm(abs(mp[i][c]),abs(mp[r][c])
                    );
                ta=tmp/abs(mp[i][c]);
                tb=tmp/abs(mp[r][c]);
                if(mp[i][c]*mp[r][c]<0) tb=-tb;
                for(j=c;j<=m;j++)
                {
                    mp[i][j]=(mp[i][j]*ta-mp[r][j]*
                        tb)%p;
                    if(mp[i][j]<0) mp[i][j]+=p;
                }
            }
            r++;
        }
        c++;
    }
    for(i=r;i<n;i++)
    {
        if(mp[i][m]!=0) return -1;//no solution
    }
   // if(r<m) return m-r;//multi solution
    for(i=m-1;~i;i--)
    {
        tmp=mp[i][m];
        for(j=i+1;j<m;j++)
        {
            if(mp[i][j]==0) continue;
            tmp=(tmp-mp[i][j]*sol[j])%p;
            if(tmp<0) tmp+=p;
        }
        sol[i]=tmp*inv(mp[i][i])%p;
    }
    return 0;
}
}
using namespace Gauss;
//set_mod();
```

8.3 线性基

8.3.1 线性基

```
struct Base
{
    #define type ll
    #define mx 60
    type d[mx+3];
    void init()
    {
        memset(d,0,sizeof(d));
    }
    bool insert(type x)
    {
        for(int i=mx;~i;i--)
        {
            if(!(x&(1LL<<i))) continue;
            if(!d[i])
            {
                d[i]=x;
                break;
            }
            x^=d[i];
        }
        return x>0;
    }
    type ask_max()
    {
        type res=0;
        for(int i=mx;~i;i--)
```

```
        {
            if((res^d[i])>res) res^=d[i];
        }
        return res;
    }
    void merge(Base x)
    {
        for(int i=mx;~i;i--)
        {
            if(x.d[i]) insert(x.d[i]);
        }
    }
    #undef type
}bs;
```

8.3.2 带删除线性基

```
struct Base
{
    #define type ll
    #define mx 60
    type d[mx+3];
    int p[mx+3],cnt;
    void init()
    {
        memset(d,0,sizeof(d));
        cnt=0;
    }
    bool insert(type x,int pos=0)
    {
        int i;
        for(i=mx;~i;i--)
        {
            if(!(x&(1LL<<i))) continue;
            if(!d[i])
            {
                cnt++;
                d[i]=x;
                p[i]=pos;
                break;
            }
            if(p[i]<pos)
            {
                swap(d[i],x);
                swap(p[i],pos);
            }
            x^=d[i];
        }
        return x>0;
    }
    type query_max(int pos=-1)
    {
        int i;
        type res=0;
        for(i=mx;~i;i--)
        {
            if(p[i]>=pos)
            {
                if((res^d[i])>res) res^=d[i];
            }
        }
        return res;
    }
    type query_min(int pos=-1)
    {
        for(int i=0;i<=mx;i++)
        {
            if(d[i]&&p[i]>=pos) return d[i];
        }
        return 0;
    }
    void merge(Base x)
    {
        if(cnt<x.cnt)
        {
            swap(cnt,x.cnt);
            swap(d,x.d);
            swap(p,x.p);
        }
        for(int i=mx;~i;i--)
        {
            if(x.d[i]) insert(x.d[i]);
        }
    }
    //kth min
    //first use rebuild()
    type tp[mx+3];
    void rebuild()
    {
        int i,j;
        cnt=0;
        for(i=mx;~i;i--)
        {
            for(j=i-1;~j;j--)
            {
                if(d[i]&(1LL<<j)) d[i]^=d[j];
            }
        }
        for(i=0;i<=mx;i++)
        {
            if(d[i]) tp[cnt++]=d[i];
        }
    }
    type kth(type k)
    {
        type res=0;
        if(k>=(1LL<<cnt)) return -1;
        for(int i=mx;~i;i--)
```

```
        {
            if(k&(1LL<<i)) res^=tp[i];
        }
        return res;
    }
};
```

9 博弈

9.1 sg 函数

```
int sg[MAX],a[MAX],n;
int dfs(int x)
{
    if(sg[x]!=-1) return sg[x];
    int i,j,flag[105]={0};
    for(i=1;i<=n;i++)
    {
        if(x>=a[i])
        {
            dfs(x-a[i]);
            flag[sg[x-a[i]]]=1;
        }
    }
    for(i=0;;i++)
    {
        if(!flag[i])
        {
            j=i;
            break;
        }
    }
    return sg[x]=j;
}
```

9.2 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 ai 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 ai 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 mod (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

10 dp

10.1 LIS

```
struct LIS
{
    #define type int
    const type inf=INF;
    vector<int> work(vector<type> a,bool strict)
    {
        int i,pos,len,n;
        n=a.size();
        vector<type> b(n,inf);
        vector<int> tmp(n),res;
        for(i=0;i<n;i++)
        {
            // strict: lower_bound
            //not strict: upper_bound
            if(strict) pos=lower_bound(b.begin(),b.
                end(),a[i])-b.begin();
            else pos=upper_bound(b.begin(),b.end(),a[
                i])-b.begin();
            b[pos]=a[i];
            tmp[i]=pos;
        }
        len=lower_bound(b.begin(),b.end(),inf)-b.
            begin();
        for(i=n-1;~i;i--)
        {
            if(!len) break;
            if(tmp[i]+1==len)
            {
                len--;
                res.push_back(i);
            }
        }
```

```
        return res;
    }
    #undef type
}lis;
```

10.2 LPS

```
int dp[105][105];
void LPS(char *s,int n) // s[1..n]
{
    int i,len,l,r;
    memset(dp,0,sizeof dp);
    for(i=1;i<=n;i++) dp[i][i]=1;
    for(len=2;len<=n;len++)
    {
        for(l=1;l+len-1<=n;l++)
        {
            r=l+len-1;
            if(s[l]==s[r]) dp[l][r]=dp[l+1][r-1]+2;
            else dp[l][r]=max(dp[l+1][r],dp[l][r-1]);
        }
    }
}
```

10.3 数位 dp

```
const int DIG=20+2;
ll dp[DIG][2];
ll gao(ll x)
{
    const int base=10;
    int p[DIG],tot=0;
    if(x==-1) return 0;
    while(1)
    {
        p[tot++]=x%base;
        x/=base;
        if(!x) break;
    }
    function<ll(int,int,int,int)> dfs=[&](int pos,
        int lead,int sta,int limt)->ll
    {
        if(pos==-1) return ;
        if(!limt&&!lead&&dp[pos][sta]!=-1) return dp[
            pos][sta];
        ll res=0;
        for(int i=(limt?p[pos]:base-1);~i;i--)
        {

            res+=dfs(pos-1,lead&&i==0&&pos,,limt&&i==
                p[pos]);
        }
        if(!limt&&!lead) dp[pos][sta]=res;
        return res;
    };
    return dfs(tot-1,1,0,1);
}
```

11 杂项

11.1 FastIO

```
namespace fastIO{
    #define BUF_SIZE 100000
    #define OUT_SIZE 100000
    #define ll long long
    //fread->read
    bool IOerror=0;
// inline char nc(){char ch=getchar();if(ch==-1)
    IOerror=1;return ch;}
    inline char nc(){
        static char buf[BUF_SIZE],*p1=buf+BUF_SIZE,*
            pend=buf+BUF_SIZE;
        if(p1==pend){
            p1=buf;pend=buf+fread(buf,1,BUF_SIZE,
                stdin);
            if(pend==p1){IOerror=1;return -1;}
        }
        return *p1++;
    }
    inline bool blank(char ch){return ch==' '||ch=='
        \n'||ch=='\r'||ch=='\t';}
    template<class T> inline bool read(T &x){
        bool sign=0;char ch=nc();x=0;
        for(;blank(ch);ch=nc());
        if(IOerror)return false;
        if(ch=='-')sign=1,ch=nc();
        for(;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
        if(sign)x=-x;
        return true;
    }
    inline bool read(double &x){
        bool sign=0;char ch=nc();x=0;
        for(;blank(ch);ch=nc());
        if(IOerror)return false;
        if(ch=='-')sign=1,ch=nc();
        for(;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
        if(ch=='.'){
            double tmp=1; ch=nc();
            for(;ch>='0'&&ch<='9';ch=nc())tmp/=10.0,x
                +=tmp*(ch-'0');
        }
        if(sign)x=-x;
        return true;
    }
    inline bool read(char *s){
```

```
        char ch=nc();
        for(;blank(ch);ch=nc());
        if(IOerror)return false;
        for(;!blank(ch)&&!IOerror;ch=nc())*s++=ch;
        *s=0;
        return true;
    }
    inline bool read(char &c){
        for(c=nc();blank(c);c=nc());
        if(IOerror){c=-1;return false;}
        return true;
    }
    template<class T,class... U>bool read(T& h,U&...
        t){return read(h)&&read(t...);}
    //fwrite->print
    struct Ostream_fwrite{
        char *buf,*p1,*pend;
        Ostream_fwrite(){buf=new char[BUF_SIZE];p1=
            buf;pend=buf+BUF_SIZE;}
//  void out(char ch){putchar(ch);}
        void out(char ch){if(p1==pend){fwrite(buf,1,
            BUF_SIZE,stdout);p1=buf;}*p1++=ch;}
        template<class T>void print(T x){
            static char s[33],*s1;s1=s;
            if(!x)*s1++='0';if(x<0)out('-'),x=-x;
            while(x)*s1++=x%10+'0',x/=10;
            while(s1--!=s)out(*s1);
        }
        void print(double x,int y){
            static ll mul[]=
            {1,10,100,1000,10000,100000,1000000,10000000,100000000,1000000000,

            10000000000LL,100000000000LL
                ,1000000000000LL,10000000000000LL,
            100000000000000LL,1000000000000000LL
                ,10000000000000000LL
                ,100000000000000000LL};
            if(x<-1e-12)out('-'),x=-x;
            ll x2=(ll)floor(x);if(!y&&x-x2>=0.5)++x2;
                x-=x2;x*=mul[y];
            ll x3=(ll)floor(x);if(y&&x-x3>=0.5)++x3;
                print(x2);
            if(y>0){out('.');for(size_t i=1;i<y&&x3*
                mul[i]<mul[y];out('0'),++i);print(x3)
                ;}
        }
        void print(char *s){while(*s)out(*s++);}
        void print(const char *s){while(*s)out(*s++)
            ;}
        void flush(){if(p1!=buf){fwrite(buf,1,p1-buf,
            stdout);p1=buf;}}
        ~Ostream_fwrite(){flush();}
    }Ostream;
    template<class T>void print(T x){Ostream.print(x
        );}
    inline void print(char x){Ostream.out(x);}
    inline void print(char *s){Ostream.print(s);}
    inline void print(string s){Ostream.print(s.
        c_str());}
    inline void print(const char *s){Ostream.print(s
        );}
    inline void print(double x,int y){Ostream.print(
        x,y);}
    template<class T,class... U>void print(const T&
        h,const U&... t){print(h);print(t...);}
    void println(){print('\n');}
    template<class T,class... U>void println(const T
        & h,const U&... t){print(h);println(t...);}
    inline void flush(){Ostream.flush();}
    #undef ll
    #undef OUT_SIZE
    #undef BUF_SIZE
};
using namespace fastIO;
```

11.2 O(1) 快速乘

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

11.3 快速模

```
struct FAST_MOD // Montgomery modular
    multiplication
{
    #define mod_byte 64 // 32 or 64

#if mod_byte==32
    #define itype int32_t
    #define utype uint32_t
    #define utype2 uint64_t
#else
    #define itype int64_t
    #define utype uint64_t
    #define utype2 __uint128_t
#endif

    static utype mod,r,r2;
    utype x;
    static utype init(utype x){return reduce(utype2(
        x)*r2);}
    static utype get_mod(){return mod;}
    static void set_mod(utype m)
```

```
    {
        mod=r=m;
        assert((mod&1)&&m>2);
        for(int i=0;i<5;i++) r*=2-r*m;
        r2=-utype2(m)%m;
    }
    static utype reduce(utype2 x)
    {
        utype y=utype(x>>mod_byte)-utype((utype2(
            utype(x)*r)*mod)>>mod_byte);
        return itype(y)<0?y+mod:y;
    }
    FAST_MOD():x(0){}
    FAST_MOD(utype n):x(init(n)){}
    FAST_MOD operator+(const FAST_MOD &a)const {
        return FAST_MOD(*this)+=a;}
    FAST_MOD operator-(const FAST_MOD &a)const {
        return FAST_MOD(*this)-=a;}
    FAST_MOD operator*(const FAST_MOD &a)const {
        return FAST_MOD(*this)*=a;}
    FAST_MOD& operator+=(const FAST_MOD &a)
    {
        x+=a.x-mod;
        if(itype(x)<0) x+=mod;
        return *this;
    }
    FAST_MOD& operator-=(const FAST_MOD &a)
    {
        x=x>=a.x?x-a.x:mod-a.x+x;
        return *this;
    }
    FAST_MOD& operator*=(const FAST_MOD &a)
    {
        x=reduce(utype2(x)*a.x);
        return *this;
    }
    FAST_MOD& operator++()
    {
        x++;
        if(x==mod) x=0;
        return *this;
    }
    FAST_MOD& operator++(int)
    {
        x++;
        if(x==mod) x=0;
        return *this;
    }
    FAST_MOD& operator--()
    {
        if(x==0) x=get_mod();
        x--;
        return *this;
    }
    FAST_MOD& operator--(int)
    {
        if(x==0) x=get_mod();
        x--;
        return *this;
    }
// friend bool operator<(const FAST_MOD& a,const
    FAST_MOD& b){return reduce(a.x)<reduce(b.x);}
// friend bool operator>(const FAST_MOD& a,const
    FAST_MOD& b){return reduce(a.x)>reduce(b.x);}
    friend bool operator==(const FAST_MOD& a,const
        FAST_MOD& b){return a.x==b.x;}
    friend bool operator!=(const FAST_MOD& a,const
        FAST_MOD& b){return a.x!=b.x;}
    friend ostream &operator<<(ostream &os, const
        FAST_MOD &a){return os<<reduce(a.x);}
    utype get_val()const {return reduce(x);}
};
utype FAST_MOD::mod,FAST_MOD::r,FAST_MOD::r2;
#undef itype
#undef utype
#undef utype2
typedef FAST_MOD mint;
/*
mint::set_mod(p);
*/
```

11.4 xor_sum(1,n)

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

11.5 约瑟夫环 kth

```
ll kth(ll n,ll m,ll k)
{
    if(m==1) return k;
    ll res=(m-1)%(n-k+1);
    for(ll i=n-k+2,stp=0;i<=n;i+=stp,res+=stp*m)
    {
        if(res+m>=i)
        {
            res=(res+m)%i;
            i++;
            stp=0;
        }
        else
        {
            stp=(i-res-2)/(m-1);
```

```
            if(i+stp>n)
            {
                res+=(n-(i-1))*m;
                break;
            }
        }
    }
    return res+1;
}
```

11.6 判断星期几

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

11.7 整数三分

```
while(l<r)
{
    lm=l+(r-l)/3;
    rm=r-(r-l)/3;
    tl=cal(lm);
    tr=cal(rm);
    // tl>tr:min tl<tr:max
    if(tl>tr) l=lm+1;
    else r=rm-1;
}
```

11.8 有根树与 prufer 序列的转换

```
//TODO
```

11.9 网格整数点正方形个数

```
struct node
{
    int x,y;
    void input(){scanf("%d%d",&x,&y);}
}p[511];
int main()
{
    int n,i,j,ans;
    while(~scanf("%d",&n))
    {
        map<pair<int,int>,int> mp;
        for(i=0;i<n;i++)
        {
            p[i].input();
            mp[MP(p[i].x,p[i].y)]=1;
        }
        ans=0;
        for(i=0;i<n;i++)
        {
            for(j=i+1;j<n;j++)
            {
                int a,b,c,d,e,f,g,h;
                a=p[i].x;
                b=p[i].y;
                c=p[j].x;
                d=p[j].y;
                e=a+b+c-d;
                f=-a+b+c+d;
                g=a-b+c+d;
                h=a+b-c+d;
                if(abs(e%2)+abs(f%2)+abs(g%2)+abs(h%2)
                    ==0)
                {
                    if(mp[MP(e/2,f/2)]&&mp[MP(g/2,h/2)
                        ]) ans++;
                }
            }
        }
        printf("%d\n",ans/2);
    }
    return 0;
}
```

11.10 模拟退火

```
/*简单版
1. 模拟退火求费马点->复杂版

2.求矩形区域内一点到各点距离之和最短时间复杂度

 cnt*c1*c2*n

*/
int sgn(double x)
{
    if(fabs(x)<eps) return 0;
    else return x>0?1:-1;
}
struct Point
{
    double x,y;
    Point(){}
    Point(double a,double b)
```

```
    {
        x=a;
        y=b;
    }
    void input()
    {
        scanf("%lf%lf",&x,&y);
    }
};
typedef Point Vector;
Vector operator -(Vector a,Vector b){return Vector(a.x-b.x,a.y-b.y);}
double dot(Vector a,Vector b){return a.x*b.x+a.y*b.y;}
double dist(Point a,Point b){return sqrt(dot(a-b,a-b));}
double lx,ly;//矩形区域(0,0)-(lx,ly)
int check(double x,double y)
{
    if(sgn(x)<0||sgn(y)<0||sgn(x-lx)>0||sgn(y-ly)>0)
        return 1;
    return 0;
}
double Rand(double r,double l)
{
    return(rand()%((int)(l-r)*1000))/(1000.0+r);
}
double getres(Point t,Point *p,int n)//求距离之和
{
    double res=0;
    for(int i=0;i<n;i++)
    {
        res+=dist(t,p[i]);
    }
    return res;
}
pair<Point,double> SA(Point *p,int n)//模拟退火
{
    srand(time(0));//重置随机种子
    const double k=0.85;//退火常数
    const int c1=30;//随机取点的个数
    const int c2=50;//退火次数
    Point q[c1+10];//随机取点
    double dis[c1+10];//每个点的计算结果
    int i,j;
    for(i=1;i<=c1;i++)
    {
        q[i]=Point(Rand(0,lx),Rand(0,ly));
        dis[i]=getres(q[i],p,n);
    }
    double tmax=max(lx,ly);
    double tmin=1e-3;
// int cnt计算外层循环次数=0;//
    while(tmax>tmin)
    {
        for(i=1;i<=c1;i++)
        {
            for(j=1;j<=c2;j++)
            {
                double ang=Rand(0,2*PI);
                Point z;
                z.x=q[i].x+cos(ang)*tmax;
                z.y=q[i].y+sin(ang)*tmax;
                if(check(z.x,z.y)) continue;
                double temp=getres(z,p,n);
                if(temp<dis[i])
                {
                    dis[i]=temp;
                    q[i]=z;
                }
            }
        }
//  cnt++;
        tmax*=k;
    }
// cout<<cnt*c1*c2*n<<endl时间复杂度;//
    int pos=1;
    for(i=2;i<=c1;i++)
    {
        if(dis[i]<dis[pos])
        {
            pos=i;
        }
    }
    pair<Point,double> res;
    res=make_pair(q[pos],dis[pos]);
    return res;
}
```

11.11 斐波那契 01 串的第 k 个字符

```
int fib[MAX],tot;
void init_fib(ll limt)
{
    assert(limt>=0);
    tot=1;
    fib[0]=fib[1]=1;
    for(int i=2;;i++)
    {
        if(fib[i-1]>limt-fib[i-2])
        {
            tot=i-1;
            break;
        }
        fib[i]=fib[i-1]+fib[i-2];
    }
}
```

```
//S(0)="0", S(1)="1", S(i)=S(i-1)+S(i-2)
//FibString: S(i) [i>=1]
int get_kth_FibString(ll k)
{
    int i;
    for(i=tot;i>=2;i--)
    {
        if(k>fib[i]) k-=fib[i];
    }
    return k==1;
}
int is_fib(ll x)
{
    for(int i=1;i<=tot;i++)
    {
        if(x==fib[i]) return 1;
    }
    return 0;
}
```

11.12 光速幂

```
struct light_speed_pow
{
    #define type int
    int n,sq;
    type res[MAX][2],val;
    void init(int _n,type _val)
    {
        n=_n;
        val=_val;
        sq=sqrt(n)+1;
        res[0][0]=res[0][1]=1;
        for(int i=1;i<=sq;i++) res[i][0]=1ll*res[i
            -1][0]*val%mod;
        for(int i=1;i<=sq;i++) res[i][1]=1ll*res[i
            -1][1]*res[sq][0]%mod;
    }
    type qpow(int exp)
    {
        if(exp<=sq) return res[exp][0];
        return 1ll*res[exp/sq][1]*res[exp-exp/sq*sq
            ][0]%mod;
    }
    #undef type
}lsp;
/*
val^exp
O(sqrt max_exp)-O(1)
lsp.init(max_exp,val);
*/
```

11.13 倍增求等比数列和

```
ll cal(ll a0,ll q,ll n,ll p)
{
    int i;
    ll tmp[33],sum[33],res,now;
    tmp[0]=0;
    tmp[1]=q;
    for(i=2;i<=30;i++) tmp[i]=tmp[i-1]*tmp[i-1]%p;
    sum[0]=1;
    for(i=1;i<=30;i++) sum[i]=sum[i-1]*(1+tmp[i])%p;
    res=0;
    now=1;
    for(i=30;~i;i--)
    {
        if((n>>i)&1)
        {
            res=(res+now*sum[i])%p;
            now=(now*tmp[i+1])%p;
        }
    }
    return a0*res%p;
}
```

11.14 矩阵旋转 90 度

```
int n,m,a[MAX][MAX],tmp[MAX][MAX];
void rotate90()
{
    int i,j;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=m;j++)
        {
            tmp[i][j]=a[i][j];
        }
    }
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=m;j++)
        {
            a[j][i]=tmp[n-i+1][j];
        }
    }
    swap(n,m);
}
```

12 附录

12.1 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

12.2 线性基求交

```
LBasis intersection(const LBasis &a, const LBasis &
    b){
   LBasis ans, c = b, d = b;
   ans.init();
   for (int i = 0; i <= 32; i++){
      ll x = a.d[i];
      if(!x)continue;
      int j = i;
      ll T = 0;
      for(; j >= 0; --j){
         if((x >> j) & 1)
            if(c.d[j]) {x ^= c.d[j]; T ^= d.d[j];}
            else break;
      }
      if(!x) ans.d[i] = T;
      else {c.d[j] = x; d.d[j] = T;}
   }
   return ans;
}
```