Little Pumpkin doesn't know FFT

内容目录

一、字符串

KMP 算法

```
const int N=1e6+10;

void get_next(const char*s ,int* nex){
   int len=strlen(s);
   nex[0]=-1;
   for(int i=1,j;i<len;i++){
      j=nex[i-1];
      while(j!=-1 && s[j+1]!=s[i]) j=nex[j];
      if(s[j+1]==s[i]) nex[i]=j+1;
      else nex[i]=-1;
   }
}

int nex[N];
//s1 中匹配 s2
int kmp(const char* s1,const char* s2){
   int l1=strlen(s1),l2=strlen(s2);
   get_next(s2,nex);
   int ans=0;
   for(int i=0,j=-1;i<l1;i++){
      while(j!=-1 && s2[j+1]!=s1[i]) j=nex[j];
      if(s2[j+1]==s1[i]) j++;
      if(j==l2-1) ans++,j=nex[j];
   }
   return ans;
```

```
}
```

Manacher 算法

```
const int N=1e6+10;

char s[N<<1],t[N];
int r[N<<1];

int manacher(const char* a){
   int id=0,mx=0,len=0;
   s[len++]='!',s[len++]='#';
   for(int i=0;a[i];i++) s[len++]=a[i],s[len++]='#';
   s[len]=0;

   int ans=0;
   for(int i=0;i<len;i++){
      if(mx>i) r[i]=min(r[2*id-i],mx-i);
      else r[i]=1;

      while(s[i-r[i]]==s[i+r[i]]) r[i]++;
      if(i+r[i]>mx) id=i,mx=i+r[i];
      ans=max(ans,r[i]-1);
   }
   return ans;
}
```

最小表示法

```
int getmin(char* s)
```

```
{
   int n = strlen(s);
   int i = 0,j = 1,k = 0;
   while(i < n && j < n && k < n)
   {
      int t = s[(i+k)%n]-s[(j+k)%n];
      if(t == 0) k++;
      else
      {
         if(t > 0) i += k+1;
         else j += k+1;
         if(i == j) j++;
         k = 0;
      }
   }
   return min(i,j);
}
```

后缀数组

```
const int maxn = 100005;
int rk[maxn],sa[maxn],height[maxn],w[maxn],cnt[maxn],res[maxn];
int n;

//sa[i]是排名为 i 的后缀在原串中的位置
//rank[i]是 i 位置的后缀排名
void getSa (int up)
{
   int *k = rk,*id = height,*r = res;
   for(int i = 0;i < up;i++) cnt[i] = 0;
```

```
   for(int i = 0;i < n;i++) cnt[k[i] = w[i]]++;
   for(int i = 0;i < up;i++) cnt[i+1] += cnt[i];
   for(int i = n - 1; i >= 0; i--) sa[--cnt[k[i]]] = i;
   for (int d = 1;d <= n;d <<= 1)
   {
      int p = 0;
      for(int i = n - d; i < n; i++) id[p++] = i;
      for(int i = 0;i < n;i++) if(sa[i] >= d) id[p++] = sa[i] - d;
      for(int i = 0;i < n;i++) r[i] = k[id[i]];
      for(int i = 0;i < up;i++) cnt[i] = 0;
      for(int i = 0;i < n;i++) cnt[r[i]]++;
      for(int i = 0;i < up;i++) cnt[i+1] += cnt[i];
      for(int i = n - 1; i >= 0; i--) sa[--cnt[r[i]]] = id[i];
      swap(k,r);
      p = 0;
      k[sa[0]] = p++;
      for(int i = 0;i < n-1;i++)
      {
         if(sa[i]+d < n && sa[i+1]+d <n &&r[sa[i]] == r[sa[i+1]]&&
r[sa[i]+d] == r[sa[i+1]+d])
            k[sa[i+1]] = p - 1;
         else k[sa[i+1]] = p++;
      }
      if(p >= n) return ;
      up = p;
   }
}
//height[i]是排名 i 的后缀与排名 i-1 的后缀的公共部分
void getHeight()
{
   for(int i = 0;i < n;i++) rk[sa[i]] = i;
   height[0] = 0;
```

```
   for(int i = 0,p = 0; i < n; i++)
   {
      if(rk[i] == 0) continue;
      int j = sa[rk[i]-1];
      while(i+p < n && j+p < n && w[i+p] == w[j+p]) p++;
      height[rk[i]] = p;
      p = max(0,p - 1);
   }
}
//s[] 是原串
void getSuffix(char s[])
{
   n = strlen(s);
   int up = 0;
   for(int i = 0; i < n; i++)
   {
      w[i] = s[i];
      up = max(up,w[i]);
   }
   getSa(up+1);
   getHeight();
}
```

AC 自动机

```
const int maxn = 500010;
const int up = 26;

int cnt = 0;
int root = 0;
```

```
struct node
{
   int num;
   int fail;
   int last;
   int nxt[up];
}ns[maxn];

int newnode()
{
   for(int i = 0;i < up;i++)
      ns[cnt].nxt[i] = -1;
   ns[cnt].fail = 0;
   ns[cnt].last = 0;
   ns[cnt].num = 0;
   return cnt++;
}

void Insert(char* str)
{
   int len = strlen(str);
   int p = root;
   for(int i = 0;i < len;i++)
   {
      int pos = str[i]-'a';
      if(ns[p].nxt[pos] == -1) ns[p].nxt[pos] = newnode();
      p = ns[p].nxt[pos];
   }
   ns[p].num++;
}

void getfail()
```

```
{
   queue<int> q;
   for(int i = 0;i < up;i++)
   {
      int x = ns[root].nxt[i];
      if(x == -1) ns[root].nxt[i] = root;
      else
      {
         ns[x].fail = ns[x].last = root;
         q.push(x);
      }
   }
   while(!q.empty())
   {
      int temp = q.front();
      q.pop();
      for(int i = 0;i < up;i++)
      {
         int x = ns[temp].nxt[i];
         if(x == -1)
            ns[temp].nxt[i] = ns[ns[temp].fail].nxt[i];
         else
         {
            ns[x].fail = ns[ns[temp].fail].nxt[i];
            q.push(x);
            if(ns[ns[x].fail].num > 0) ns[x].last = ns[x].fail;
            else ns[x].last = ns[ns[x].fail].last;
         }
      }
   }
}
```

```
int query(char *str)
{
   int len = strlen(str);
   int ans = 0;
   int p = root;
   int temp;
   for(int i = 0;i < len;i++)
   {
      int pos = str[i] - 'a';
      p = ns[p].nxt[pos];
      temp = p;
      while(temp != root)
      {
         if(ns[temp].num > 0)
         {
            ans += ns[temp].num;
            ns[temp].num = 0;
         }
         temp = ns[temp].last;
      }
   }
   return ans;
}

void init()
{
   cnt = 0;
   root = newnode();
}
```

后缀自动机

```
const int maxn = 250010;
const int up = 30;

struct state
{
   int len,pre;
   int right;
   int nxt[up];
}ns[maxn*2];

int cnt = 0;

int newnode(int l)
{
   ns[cnt].len = l;
   ns[cnt].pre = -1;
   mem(ns[cnt].nxt,-1);
   return cnt++;
}

int root = 0;
int last = 0;

void init()
```

```
{
   cnt = 0;
   last = 0;
   root = newnode(0);
}

void extend(int x)
{
   int p = last;
   int np = newnode(ns[p].len+1);
   while(p != -1 && ns[p].nxt[x] == -1)
      ns[p].nxt[x] = np,p = ns[p].pre;
   if(p == -1)
      ns[np].pre = root;
   else
   {
      int q = ns[p].nxt[x];
      if(ns[p].len+1 == ns[q].len)
      {
         ns[np].pre = q;
      }
      else
      {
         int clone = newnode(ns[p].len+1);
         for(int i = 0;i < up;i++)
            ns[clone].nxt[i] = ns[q].nxt[i];
         ns[clone].pre = ns[q].pre;
         for(;p != -1 && ns[p].nxt[x] == q;p = ns[p].pre)
```

```
            ns[p].nxt[x] = clone;
         ns[q].pre = ns[np].pre = clone;
         ns[clone].right = 0;
      }
   }
   last = np;
   ns[np].right = 1;
}

char a[maxn];

int c[maxn];
int sa[maxn*2];

int main()
{
   init();
   scanf("%s",a);
   int n = strlen(a);
   for(int i = 0;i < n;i++)
      extend(a[i]-'a');
   for(int i = 1;i < cnt;i++) c[ns[i].len]++;
   for(int i = 1;i <= n;i++) c[i] += c[i-1];
   for(int i = cnt-1;i > 0;i--) sa[c[ns[i].len]--] = i;
   for(int i = cnt-1;i > 0;i--)
   {
      ns[ns[sa[i]].pre].right += ns[sa[i]].right;
   }
```

```
    return 0;
}
```

回文树

```
const int maxn = 200010;
const int up = 30;

struct node
{
    int nxt[up];
    int fail;
    int num,cnt;
    int len;
}ns[maxn];

int last,len;

int cnt = 0;
int newnode(int l)
{
    mem(ns[cnt].nxt,0);
    ns[cnt].fail = -1;
    ns[cnt].num = ns[cnt].cnt = 0;
    ns[cnt].len = l;
    return cnt++;
}
```

```
void init()
{
   cnt = 0;
   newnode(0);
   newnode(-1);
   last = 0;
   len = 0;
   ns[0].fail = 1;
}

int getFail(int x)
{
   while(s1[len-ns[x].len-1] != s1[len]) x = ns[x].fail;
   return x;
}

void extend(int x)
{
   ++len;
   int cur = getFail(last);
   if(ns[cur].nxt[x] == 0)
   {
      int p = newnode(ns[cur].len+2);
      ns[p].fail = ns[getFail(ns[cur].fail)].nxt[x];
      ns[cur].nxt[x] = p;
      ns[p].num = ns[ns[p].fail].num+1;
   }
   last = ns[cur].nxt[x];
```

```
    ns[last].cnt++;
}
```

exkmp

```
const int maxn = 1000010;
int f[maxn],ex[maxn];

void getFail(char* t)
{
    int p = 0,mx = 0,n = strlen(t);
    f[0] = n;
    for(int i = 1;i < n;i++)
    {
        if(mx > i+f[i-p])
        {
            f[i] = f[i-p];
            continue;
        }
        f[i] = max(mx-i,0);
        while(t[i+f[i]] == t[f[i]]) f[i]++;
        if(i+f[i] > mx) mx = i+f[i],p = i;
    }
}

void ex_kmp(char* s,char* t)
{
    getFail(t);
```

```
    int p = 0,mx = 0,n = strlen(s);
    for(int i = 0;i < n;i++)
    {
        if(mx > i+f[i-p])
        {
            ex[i] = ex[i-p];
            continue;
        }
        ex[i] = max(mx-i,0);
        while((i+ex[i] < n) && s[i+ex[i]] == t[ex[i]]) ex[i]++;
        if(i+ex[i] > mx) mx = i+ex[i],p = i;
    }
}
```

二、网络流

最大流最小割

Dinic

//点不是非常多就用这个
/*****************************

有最小流量时可以这样添加边,新增源点 S，汇点 T
add_edge(u,v,c[i]-b[i]);
add_edge(S,v,b[i]);

```
add_edge(u,T,b[i]);

add_edge(S,s,INF);
add_edge(t,T,INF);
需先验证新源汇点没有连回旧源汇点前流量是不是 b[i]之和
不是则该上下界网络流问题无解

*****************************/

//O(E*V^2)
//实际应用很快

const int V=20005;
const int INF=1e9;

struct Edge{
   int to,cap,rev;
};
vector<Edge>G[V];
int iter[V],level[V];

void add_edge(int u,int v,int w){
   G[u].push_back((Edge){v,w,(int)G[v].size()});
   G[v].push_back((Edge){u,0,(int)G[u].size()-1});
}

queue<int>que;
void bfs(int s){
   mem(level,-1);
   level[s]=0;
   que.push(s);
   while(!que.empty()){
```

```
      int v=que.front();
      que.pop();
      for(int i=0;i<G[v].size();i++){
         Edge& e=G[v][i];
         if(level[e.to]<0 && e.cap>0) level[e.to]=level[v]
+1,que.push(e.to);
      }
   }
}

int dfs(int s,int t,int f){
   if(s==t) return f;
   //弧优化
   for(int& i=iter[s];i<(int)G[s].size();i++){
      Edge& e=G[s][i];
      if(level[s]<level[e.to] && e.cap>0){
         int d=dfs(e.to,t,min(f,e.cap));
         if(d>0){
            e.cap-=d,G[e.to][e.rev].cap+=d;
            return d;
         }
      }
   }
   return 0;
}

int max_flow(int s,int t){
   int flow=0;
   while(1){
      bfs(s);
      if(level[t]<0) return flow;
      mem(iter,0);
```

```
      int f;
      while(f=dfs(s,t,INF)) flow+=f;
   }
}
```

Ford-Fulkerson 算法

```
//O(FE)

const int V=1005;
const int INF=1e9;

struct Edge{
   int to,cap,rev;
};
vector<Edge>G[V];
bool vis[V];

void add_edge(int u,int v,int w){
   G[u].push_back((Edge){v,w,G[v].size()});
   G[v].push_back((Edge){u,0,G[u].size()-1});
}

int dfs(int s,int t,int f){
   if(s==t) return f;
   vis[s]=1;
   for(int i=0;i<G[s].size();i++){
      Edge& e=G[s][i];
      if(!vis[e.to] && e.cap>0){
         int d=dfs(e.to,t,min(f,e.cap));
```

```
            if(d>0){
               e.cap-=d,G[e.to][e.rev].cap+=d;
               return d;
            }
         }
      }
      return 0;
}

int max_flow(int s,int t){
      int flow=0;
      while(1){
         mem(vis,0);
         int f=dfs(s,t,INF);
         if(f==0) return flow;
         flow+=f;
      }
}
```

isap

```
//点太多用这个
//时间复杂度 O(??)，能跑 10^5 个点

const int V=1e5+10;
const int E=V<<2;
const int INF=1e9;

struct Edge{
   int to,nex,cap,flow;
```

```
}e[E];

int head[V],cnt;
void add_edge(int u,int v,int w,int rw=0){
   e[cnt].to=v,e[cnt].nex=head[u],e[cnt].cap=w,e[cnt].flow=0;
   head[u]=cnt++;
   e[cnt].to=u,e[cnt].nex=head[v],e[cnt].cap=rw,e[cnt].flow=0;
   head[v]=cnt++;
}

int level[V],cur[V],gap[V];
void bfs(int s,int t){
   queue<int>que;
   que.push(t);
   mem(level,-1),mem(gap,0);
   level[t]=0,gap[0]=1;
   while(!que.empty()){
      int u=que.front();
      que.pop();
      for(int i=head[u];~i;i=e[i].nex){
         int v=e[i].to;
         if(level[v]==-1){
            level[v]=level[u]+1;
            gap[level[v]]++;
            que.push(v);
         }
      }
   }
}

int st[V];
int isap(int s,int t,int vi){
```

```
int u=s,top=0,ans=0;
bfs(s,t);
memcpy(cur,head,sizeof(int)*vi);
while(level[s]<vi){
   if(u==t){
      int add=INF,id=0;
      for(int i=0;i<top;i++){
         int j=st[i];
         if(add>e[j].cap-e[j].flow){
            add=e[j].cap-e[j].flow;
            id=i;
         }
      }
      for(int i=0;i<top;i++){
         e[st[i]].flow+=add;
         e[st[i]^1].flow-=add;
      }
      ans+=add;
      top=id;
      u=e[st[top]^1].to;
      continue;
   }
   bool flag=0;
   for(int i=cur[u];~i;i=e[i].nex){
      int v=e[i].to;
      if(level[v]+1==level[u] && e[i].cap-e[i].flow){
         flag=1;
         cur[u]=i;
         break;
      }
   }
   if(flag){
```

```
            st[top++]=cur[u];
            u=e[cur[u]].to;
            continue;
        }

        int Min=V;
        for(int i=head[u];~i;i=e[i].nex){
            int v=e[i].to;
            if(e[i].cap-e[i].flow && Min>level[v]){
                Min=level[v];
                cur[u]=i;
            }
        }
        gap[level[u]]--;
        if(!gap[level[u]]) break;
        level[u]=Min+1;
        gap[level[u]]++;
        if(u!=s) u=e[st[--top]^1].to;
    }
    return ans;
}
```

最小费用流

```
/**************************************
添加费用为负值的边时，如果流量一定，可以这样添边消去负值
add_edge(v,u,c[i],-w[i]);
add_edge(s,v,INF,0);
add_edge(t,u,INF,0);
```

求最小流时流量应为原流量+负边权的容量和，而且最小费用加上所有负费用乘容量
即让负权边一开始满流
**********/

```
//加入了势函数和 dijkstra 优化，O(FElogV)

const int V=1005;
const int INF=1e9;

struct Edge{
   int to,cap,cost,rev;
};
vector<Edge>G[V];

//h[i]表示 s 到 i 的实际最短距离，dist 表示优化后的最短距离（正）
int h[V],dist[V];
int prev[V],pree[V];

void add_edge(int from,int to,int cap,int cost){
   G[from].push_back((Edge){to,cap,cost,G[to].size()});
   G[to].push_back((Edge){from,0,-cost,G[from].size()-1});
}

priority_queue<pii,vector<pii>,greater<pii> >que;
int min_cost_flow(int s,int t,int f){
   int res=0;
   mem(h,0);
   while(f){
      for(int i=0;i<V;i++) dist[i]=INF;
      dist[s]=0;
      que.push(pii(0,s));
```

```
        while(!que.empty()){
            pii p=que.top();
            que.pop();
            int v=p.second;
            if(dist[v]<p.first) continue;
            for(int i=0;i<G[v].size();i++){
                Edge& e=G[v][i];
                if(e.cap>0 && dist[e.to]>dist[v]+e.cost+h[v]-h[e.to]){
                    dist[e.to]=dist[v]+e.cost+h[v]-h[e.to];
                    prev[e.to]=v;
                    pree[e.to]=i;
                    que.push(pii(dist[e.to],e.to));
                }
            }
        }
        if(dist[t]==INF) return -1;
        for(int i=0;i<V;i++) h[i]+=dist[i];
        int d=f;
        for(int i=t;i!=s;i=prev[i]){
            d=min(d,G[prev[i]][pree[i]].cap);
        }
        f-=d;
        res+=d*h[t];
        for(int i=t;i!=s;i=prev[i]){
            Edge& e=G[prev[i]][pree[i]];
            e.cap-=d,G[i][e.rev].cap+=d;
        }
    }
    return res;
}
```

二分图

KM 算法

```
//求二分图最大权值匹配
//不要求完美匹配的话不存在的边边权为 0,否则边权为-INF
//O(V^3)

const int V=205;
const int INF=1e9;
int nx,ny;
int match[V];
int visx[V],visy[V],tx[V],ty[V];
int val[V][V];

bool dfs(int u){
   visx[u]=1;
   for(int i=0;i<ny;i++){
      if(!visy[i] && tx[u]+ty[i]==val[u][i]){
         visy[i]=1;
         if(match[i]==-1 || dfs(match[i])){
            match[i]=u;
            return 1;
         }
      }
   }
   return 0;
}

int KM(){
   mem(match,-1);
```

```
    mem(tx,0),mem(ty,0);
    for(int i=0;i<nx;i++){
        for(int j=0;j<ny;j++){
            tx[i]=max(tx[i],val[i][j]);
        }
    }
    for(int u=0;u<nx;u++){
        while(1){
            mem(visx,0),mem(visy,0);
            if(dfs(u)) break;
            int d=INF;
            for(int i=0;i<nx;i++){
                if(visx[i]){
                    for(int j=0;j<ny;j++){
                        if(!visy[j]) d=min(d,tx[i]+ty[j]-val[i][j]);
                    }
                }
            }
            //无解返回-1
            if(d==INF) return -1;
            for(int i=0;i<nx;i++) if(visx[i]) tx[i]-=d;
            for(int i=0;i<ny;i++) if(visy[i]) ty[i]+=d;
        }
    }
    int ans=0;
    for(int i=0;i<ny;i++){
        if(match[i]!=-1) ans+=val[match[i]][i];
    }
    return ans;
}
```

Hopcroft-Carp

```
//二分图匹配
//O(sqrt(V)*E)

const int INF=1e9;
const int V=40000+10;
const int E=V*5;

bool vis[V];
//mx,my 表示匹配，dx,dy 表示 bfs 序
int mx[V],my[V],dx[V],dy[V];
//nx,ny 分别表示二分图两种点的数目
int nx,ny;

struct Edge{
   int to,nex;
}e[E<<1];

int head[V],cnt;
void add_edge(int u,int v){
   e[cnt].to=v,e[cnt].nex=head[u],head[u]=cnt++;
}

int dis;
queue<int>que;
bool bfs(){
   while(!que.empty()) que.pop();
   mem(dx,-1);
   mem(dy,-1);
   dis=INF;
   for(int i=0;i<nx;i++){
```

```
      if(mx[i]<0) que.push(i),dx[i]=0;
   }
   while(!que.empty()){
      int u=que.front();
      que.pop();
      if(dx[u]>dis) break;
      for(int i=head[u];~i;i=e[i].nex){
         int v=e[i].to;
         if(dy[v]<0){
            dy[v]=dx[u]+1;
            if(my[v]<0) dis=dy[v];
            else{
               dx[my[v]]=dy[v]+1;
               que.push(my[v]);
            }
         }
      }
   }
   return dis!=INF;
}

bool dfs(int u){
   for(int i=head[u];~i;i=e[i].nex){
      int v=e[i].to;
      if(!vis[v] && dy[v]==dx[u]+1){
         vis[v]=1;
         if(my[v]!=-1 && dy[v]==dis) continue;
         if(my[v]==-1 || dfs(my[v])){
            my[v]=u,mx[u]=v;
            return 1;
         }
      }
```

```
    }
    return 0;
}

int bipartite_matching(){
    int ans=0;
    mem(mx,-1);
    mem(my,-1);
    while(bfs()){
        mem(vis,0);
        for(int i=0;i<nx;i++){
            if(mx[i]==-1 && dfs(i)) ans++;
        }
    }
    return ans;
}
```

匈牙利算法

```
//时间复杂度是O(VE)

const int V=105;
const int E=V*V;
bool vis[V];
int match[V];

struct Edge{
    int to,nex;
}e[E];

int head[N],cnt;
```

```
bool dfs(int u){
   vis[u]=1;
   for(int i=head[u];~i;i=e[i].nex){
      int v=e[i].to,w=match[v];
      if(w<0 || (!vis[w] && dfs(w))){
         match[u]=v;
         match[v]=u;
         return 1;
      }
   }
   return 0;
}


void add_edge(int u,int v){
   e[cnt].to=v,e[cnt].nex=head[u],head[u]=cnt++;
}

int bipartite_matching(){
   int ans=0;
   mem(match,-1);
   for(int i=0;i<V;i++){
      if(match[i]<0){
         mem(vis,0);
         if(dfs(i)) ans++;
      }
   }
   return ans;
}
```

三、数学

中国剩余定理

```
//互质版中国剩余定理

typedef long long ll;

ll quick_mod(ll base,ll n,ll mod){
   ll ans=1;
   base%=mod;
   while(n){
      if(n&1) (ans*=base)%=mod;
      (base*=base)%=mod;
      n>>=1;
   }
   return ans;
}

ll CRT(ll* mod,ll* a,int n){
   ll M=1,ans=0;
   for(int i=0;i<n;i++) M*=mod[i];

   for(int i=0;i<n;i++){
      ans+=(a[i]*(M/mod[i]))%M*quick_mod(M/mod[i],mod[i]-2,mod[i]);
      ans%=M;
   }
   return ans;
}
```

```
//非互质版,每次合并两个式子

ll exgcd(ll a,ll b,ll& x,ll& y){
   if(!b){
      x=1,y=0;
      return a;
   }
   ll d=exgcd(b,a%b,x,y);
   ll t=x;
   x=y,y=t-a/b*y;
   return d;
}

ll gcd(ll a,ll b){
   return !b?a:gcd(b,a%b);
}

ll inv(ll a,ll b){
   ll x,y;
   exgcd(a,b,x,y);
   x=(x%b+b)%b;
   return x;
}

bool process(ll a1,ll n1,ll a2,ll n2,ll& a3,ll& n3){
   ll d=gcd(n1,n2);
   if((a2-a1)%d) return 0;
   ll K=((a2-a1)/d*inv(n1/d,n2/d))%n2;
   if(K<0) K+=n2;
   n3=n1/d*n2;
```

```
   a3=((n1*K+a1)%n3+n3)%n3;
   return 1;
}

ll CRT(ll* a,ll* mod,int n){
   for(int i=0;i<n;i++) a[i]=(a[i]%mod[i]+mod[i])%mod[i];
   for(int i=1;i<n;i++){
      if(!process(a[0],mod[0],a[i],mod[i],a[0],mod[0])) return -1;
   }
   return a[0];
}
```

扩展欧几里德定理

```
ll exgcd(ll a,ll b,ll& x,ll& y){
   if(!b) { x=1,y=0;}
   else{
      exgcd(b,a%b,y,x);
      y-=x*(a/b);
   }
}

// ax+by=gcd(a,b) 找 x>0 && y>0 的解
pii cal(ll a,ll b){
   ll x,y;
   exgcd(a,b,x,y);
   //如果 a 和 b 不同号，在这加上 if(a*x+b*y!=-1) x=-x,y=-y;
   ll k=max((x+b-1)/b,(-y+a-1)/a)+1;
   return pii(x-k*b,y+k*a);
}
```

高斯消元

```
const int N=205;
const double eps=1e-7;

//O(N^3)
//若方程有唯一解，则第 i 行除了 a[i][i]和 a[i][n]外都是 0
//否则可根据物理事实推断
void gauss(double a[N][N],int n){
   for(int i=0;i<n;i++){
      int r=i;
      for(int j=i+1;j<n;j++){
         if(fabs(a[j][i])>fabs(a[r][i])) r=j;
      }
      if(fabs(a[r][i])<eps) continue;
      for(int j=0;j<=n;j++) swap(a[i][j],a[r][j]);
      for(int k=0;k<n;k++){
         if(k==i) continue;
         //不能反
         for(int j=n;j>=i;j--){
            a[k][j]-=a[k][i]/a[i][i]*a[i][j];
         }
      }
   }
}
```

自适应辛普森积分

```
//贼快
```

//自适应辛普森积分，调用 asr(a,b,eps)获得精度在 eps 范围内的 F(x)在(a,b)的积分

```
double F(double x){}

double simpson(double a,double b){
   double c=(a+b)/2;
   return (F(a)+F(b)+4*F(c))*(b-a)/6;
}

double asr(double a,double b,double eps,double A){
   double c=(a+b)/2;
   double L=simpson(a,c),R=simpson(c,b);
   if(fabs(L+R-A)<=15*eps) return L+R+(L+R-A)/15;
   return asr(a,c,eps/2,L)+asr(c,b,eps/2,R);
}

double asr(double a,double b,double eps){
   return asr(a,b,eps,simpson(a,b));
}
```

FFT

//求两个大整数的乘积
//圆周率 PI

```
const double PI = acos(-1.0);
```

```
//复数的结构体
struct Complex
{
   double a,b; //a+bi
   Complex(double _a=0.0,double _b=0.0){
      a=_a,b=_b;
   }
   //重载复数类的运算符
   Complex operator - (const Complex& temp){
      return Complex(a-temp.a,b-temp.b);
   }
   Complex operator + (const Complex& temp){
      return Complex(a+temp.a,b+temp.b);
   }
   Complex operator * (const Complex& temp){
      return Complex(a*temp.a-b*temp.b,a*temp.b+temp.a*b);
   }
   Complex operator / (const double& temp){
      return Complex(a/temp,b/temp);
   }
   Complex conj(){
      return Complex(a,-b);
   }
};

//进行 fft 前的预处理，将 i 与其倒置二进制对应的系数互换
void init(Complex* x,int n){
   for(int i=1,j=n/2;i<n-1;i++){
      if(i<j) swap(x[i],x[j]);
      int k=n/2;
      while(j>=k) j-=k,k/=2;
      j+=k;
```

```
    }
}


//FFT 主代码
//on==1 代表 DFT，on==-1 代表 IDFT 过程
void FFT(Complex* x,int n,int on){
    init(x,n);
    for(int i=2;i<=n;i<<=1){
        //e^(k*i)=cos(k)+i*sin(k),这里的 i 是复数基本单位，即 sqrt(-1)
        Complex wn(cos(-on*2*PI/i),sin(-on*2*PI/i));
        for(int j=0;j<n;j+=i){
            Complex w(1,0);
            for(int k=j;k<j+i/2;k++){
                Complex u=x[k],v=w*x[k+i/2];
                x[k]=u+v,x[k+i/2]=u-v;
                w=w*wn;
            }
        }
    }
    if(on==-1){
        for(int i=0;i<n;i++){
            x[i].a/=n;
        }
    }
}

const int N=4e5+10;
Complex x1[N],x2[N];
int ans[N];

void solve(char* s1,char* s2){
```

```
mem(x1,0),mem(x2,0);
int l1=strlen(s1),l2=strlen(s2),len=1;
while(len<2*max(l1,l2)) len<<=1;
for(int i=0;i<l1;i++) x1[i]=Complex(s1[l1-1-i]-'0',0);
for(int i=0;i<l2;i++) x2[i]=Complex(s2[l2-1-i]-'0',0);

for(int i=0;i<len;i++) x1[i]=x1[i]+Complex(0,1)*x2[i];

FFT(x1,len,1);

x2[0]=x1[0].conj();
for(int i=1;i<len;i++){
   x2[i]=x1[len-i].conj();
}
for(int i=0;i<len;i++){
   x1[i]=((x1[i]+x2[i])/2) * (Complex(0,1)*((x2[i]-x1[i])/2));
}
FFT(x1,len,-1);

mem(ans,0);
for(int i=0;i<len;i++){
   ans[i]=int(x1[i].a+0.5);
}
for(int i=0;i<len;i++){
   ans[i+1]+=ans[i]/10;
   ans[i]%=10;
}
while(ans[len]<=0 && len>=1) len--;
for(int i=len;i>=0;i--){
   putchar(ans[i]+'0');
}
puts("");
```

```
}

char a[N>>2],b[N>>2];
int main()
{
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
      //std::ios_base::sync_with_stdio(false);
   while(~scanf("%s%s",a,b)){
      solve(a,b);
   }
    return 0;
}
```

大数进制转换

```
const int N=6e5+10;
Complex x1[N],x2[N];
int ans[N];
char ret[N];

string solve(const string& s1,const string& s2){

   int l1=s1.size(),l2=s2.size(),len=1;
   while(len<2*max(l1,l2)) len<<=1;


   for(int i=0;i<l1;i++) x1[i]=Complex(s1[l1-1-i]-'0',0);
   for(int i=l1;i<len;i++) x1[i]=Complex(0,0);
```

```
for(int i=0;i<l2;i++) x2[i]=Complex(s2[l2-1-i]-'0',0);
for(int i=l2;i<len;i++) x2[i]=Complex(0,0);

for(int i=0;i<len;i++) x1[i]=x1[i]+Complex(0,1)*x2[i];

FFT(x1,len,1);

x2[0]=x1[0].conj();
for(int i=1;i<len;i++) x2[i]=x1[len-i].conj();

for(int i=0;i<len;i++){
   x1[i]=((x1[i]+x2[i])/2) * (Complex(0,1)*((x2[i]-x1[i])/2));
}
FFT(x1,len,-1);

for(int i=0;i<len*2;i++) ans[i]=0;
for(int i=0;i<len;i++) ans[i]=int(x1[i].a+0.5);
for(int i=0;i<len;i++){
   ans[i+1]+=ans[i]/10;
   ans[i]%=10;
}
while(ans[len]<=0 && len>=1) len--;

int cnt=0;
for(int i=len;i>=0;i--){
   //putchar(ans[i]+'0');
   ret[cnt++]=ans[i]+'0';
}
ret[cnt]=0;

return string(ret);
```

```
}

string change(int n){
   char s[10]={};
   int cnt=0;
   while(n) s[cnt++]=n%10+'0',n/=10;
   for(int i=0;i<cnt/2;i++) swap(s[i],s[cnt-1-i]);
   s[cnt]=0;
   return string(s);
}

int tmpa[N],tmpb[N];
char strans[N];

int getInt(const char& ch){
    if(ch<='9') return ch-'0';
    else return ch-'A'+10;
}

string add(const string& a,const string& b){

    int lena=(int)a.size();
    int lenb=(int)b.size();

    for(int i=0;i<lena;i++) tmpa[i]=getInt(a[lena-i-1]);
    for(int i=0;i<lenb;i++) tmpb[i]=getInt(b[lenb-i-1]);

    int len=max(lena,lenb)+5;
    for(int i=lena;i<=len;i++) tmpa[i]=0;
    for(int i=lenb;i<=len;i++) tmpb[i]=0;

    for(int i=0;i<=len;i++) ans[i]=0;
```

```
    for(int i=0;i<len;i++){
        ans[i]+=tmpa[i]+tmpb[i];
        ans[i+1]+=ans[i]/10;
        ans[i]%=10;
    }

    while(ans[len]==0 && len>=1) len--;

    for(int i=0;i<=len;i++) strans[i]=ans[len-i]+'0';
    strans[len+1]=0;

 //   cerr<<a<<' '<<b<<' '<<string(strans)<<endl;

    return string(strans);
}

string pw[18];

void init(){
    pw[0]="36";
    for(int i=1;i<18;i++) pw[i]=solve(pw[i-1],pw[i-1]);
}

string quick_pow(int n){
    string ans="1";
    for(int i=0;i<18;i++){
        if((n>>i)&1) ans=solve(ans,pw[i]);
    }
    return ans;
}
```

```
char s[N];

string fuck(int l,int r){
    if(l==r){
        return change(getInt(s[l]));
    }
    int m=l+(r-l)/3;
    string L=solve(fuck(l,m),quick_pow(r-m));
    string R=fuck(m+1,r);
   // cerr<<l<<' '<<r<<' '<<add(L,R)<<endl;
    return add(L,R);
}
```

NTT

```
//P=c*2^k+1,g 为其原根

const ll P=(479LL<<21)+1;
const ll g=3;
ll wn[20];

ll quick_pow(ll base,ll n,ll mod){
   ll ans=1;
   while(n){
      if(n&1) (ans*=base)%=mod;
      (base*=base)%=mod;
      n>>=1;
   }
   return ans;
}
```

```
void init(ll* x,int n){
   for(int i=1,j=n/2;i<n-1;i++){
      if(i<j) swap(x[i],x[j]);
      int k=n/2;
      while(j>=k) j-=k,k/=2;
      j+=k;
   }
}


void NTT(ll* x,int n,int on){
   init(x,n);
   int id=0;
   for(int i=0;i<n;i++) if(x[i]>=P) x[i]%=P;
   for(int i=2;i<=n;i<<=1){
      id++;
      for(int j=0;j<n;j+=i){
         ll w=1;
         for(int k=j;k<j+i/2;k++){
            ll u=x[k],v=w*x[k+i/2]%P;
            x[k]=u+v,x[k+i/2]=u-v+P;
            if(x[k]>=P) x[k]-=P;
            if(x[k+i/2]>=P) x[k]-=P;
            w=w*wn[id]%P;
         }
      }
   }
   if(on==-1){
      reverse(x+1,x+n);
      ll inv=quick_pow(n,P-2,P);
```

```
      for(int i=0;i<n;i++){
         x[i]=x[i]*inv%P;
      }
   }
}

void Init(){
   wn[0]=1;
   for(int i=1;i<20;i++){
      wn[i]=quick_pow(g,(P-1)/(1<<i),P);
   }
}
```

NTT+CRT

```
//////////////////
//二模数 NTT,CRT 合并

ll quick_pow(ll base,ll n,ll mod){
   ll ans=1;
   while(n){
      if(n&1) (ans*=base)%=mod;
      (base*=base)%=mod;
      n>>=1;
   }
   return ans;
}

ll mul(ll a,ll b,ll M){
   ll ans=0;
```

```
    while(a){
        if(a&1) (ans+=b)%=M;
        (b<<=1)%=M;
        a>>=1;
    }
    return ans;
}

struct NTT{
    ll P,g;
    ll wn[20];

    void Init(ll mod){
        P=mod,wn[0]=1,g=3;
        for(int i=1;i<20;i++) wn[i]=quick_pow(g,(P-1)/(1<<i),P);
    }
    void init(ll* x,int n){
        for(int i=1,j=n/2;i<n-1;i++){
            if(i<j) swap(x[i],x[j]);
            int k=n/2;
            while(j>=k) j-=k,k/=2;
            j+=k;
        }
    }

    void FFT(ll* x,int n,int on){
        init(x,n);
        for(int i=0;i<n;i++) if(x[i]>=P) x[i]%=P;
        int id=0;
        for(int i=2;i<=n;i<<=1){
            id++;
            for(int j=0;j<n;j+=i){
```

```
            ll w=1;
            for(int k=j;k<j+i/2;k++){
               ll u=x[k],v=w*x[k+i/2]%P;
               x[k]=u+v,x[k+i/2]=u-v+P;
               if(x[k]>=P) x[k]-=P;
               if(x[k+i/2]>=P) x[k+i/2]-=P;
               w=w*wn[id]%P;
            }
         }
      }
      if(on==-1){
         reverse(x+1,x+n);
         ll inv=quick_pow(n,P-2,P);
         for(int i=0;i<n;i++){
            x[i]=x[i]*inv%P;
         }
      }
   }

}solver[2];

ll mod[2]={(479LL<<21)+1, (331LL<<20)+1},M;
ll inv0,inv1,p=100003;

const int N=4e5+10;
ll f[20][N];
ll a[N],b[N],c[N],d[N],x[N];

void solve(int l,int r,int dep){

   if(l==r){
      f[dep][0]=1,f[dep][1]=x[l];
```

```
   return ;
}

int m=(l+r)>>1;
solve(l,m,dep+1);
for(int i=0;i<=m-l+1;i++) f[dep][i]=f[dep+1][i];
solve(m+1,r,dep+1);

int n=1;
while(n<=(r-l+1)) n<<=1;

for(int i=m-l+2;i<n;i++) f[dep][i]=0;
for(int i=r-m+1;i<n;i++) f[dep+1][i]=0;

for(int i=0;i<n;i++) a[i]=c[i]=f[dep][i],b[i]=d[i]=f[dep+1][i];

solver[0].FFT(a,n,1);
solver[0].FFT(b,n,1);
for(int i=0;i<n;i++) a[i]=a[i]*b[i]%mod[0];
solver[0].FFT(a,n,-1);

solver[1].FFT(c,n,1);
solver[1].FFT(d,n,1);
for(int i=0;i<n;i++) c[i]=c[i]*d[i]%mod[1];
solver[1].FFT(c,n,-1);

for(int i=0;i<=r-l+1;i++){
   if(a[i]<0) a[i]+=mod[0];
   if(c[i]<0) c[i]+=mod[1];

   f[dep][i]=(mul(a[i],inv0,M)+mul(c[i],inv1,M))%M%p;
}
```

```
    /*
    cout<<l<<' '<<r<<endl;

    for(int i=0;i<=r-l+1;i++) cout<<f[dep][i]<<' ';
    cout<<endl;
    */
}


int main(){
     //freopen("in.txt","r",stdin);
     //freopen("out.txt","w",stdout);

    solver[0].Init(mod[0]);
    solver[1].Init(mod[1]);
    M=mod[0]*mod[1];

    inv0=quick_pow(mod[1],mod[0]-2,mod[0])*mod[1]%M;
    inv1=quick_pow(mod[0],mod[1]-2,mod[1])*mod[0]%M;

    int n,q;
    scanf("%d%d",&n,&q);
    for(int i=1;i<=n;i++) scanf("%lld",x+i),x[i]%=p;

    solve(1,n,0);
    for(int i=0;i<=n;i++) if(f[0][i]<0) f[0][i]+=p;

    while(q--){
       int k;
       scanf("%d",&k);
       printf("%lld\n",f[0][k]);
```

```
    }

     return 0;
}



////////////////////////////////
//三模数 NTT

ll quick_pow(ll base,ll n,ll mod){
    base%=mod;
    ll ans=1;
    while(n){
        if(n&1) (ans*=base)%=mod;
        (base*=base)%=mod;
        n>>=1;
    }
    return ans;
}

ll mul(ll a,ll b,ll M){
    ll ans=0;
    while(a){
        if(a&1) (ans+=b)%=M;
        (b<<=1)%=M;
        a>>=1;
    }
    return ans;
}

void init(ll* x,int n){
    for(int i=1,j=n/2;i<n-1;i++){
```

```
        if(i<j) swap(x[i],x[j]);
        int k=n/2;
        while(j>=k) j-=k,k/=2;
        j+=k;
    }
}

struct NTT{
    ll P,g;
    ll wn[20];

    void Init(ll mod){
        P=mod,wn[0]=1,g=3;
        for(int i=1;i<20;i++) wn[i]=quick_pow(g,(P-1)/(1<<i),P);
    }

    void FFT(ll* x,int n,int on){
        init(x,n);
        int id=0;
        for(int i=2;i<=n;i<<=1){
            id++;
            for(int j=0;j<n;j+=i){
                ll w=1;
                for(int k=j;k<j+i/2;k++){
                    ll u=x[k]%P,v=w*(x[k+i/2]%P)%P;
                    x[k]=(u+v)%P,x[k+i/2]=(u-v+P)%P;
                    w=w*wn[id]%P;
                }
            }
        }
        if(on==-1){
            reverse(x+1,x+n);
```

```
            ll inv=quick_pow(n,P-2,P);
            for(int i=0;i<n;i++){
               x[i]=(x[i]%P)*inv%P;
            }
         }
      }

}solver[3];

ll mod[3]={(479LL<<21)+1,(331LL<<20)+1,(119LL<<23)+1};
ll inv0,inv1,M,p=1e9+7;

const int N=4e5+10;
ll x[N],y[N];

ll a[6][N];

int main(){

   for(int i=0;i<3;i++) solver[i].Init(mod[i]);
   M=mod[0]*mod[1];
   inv0=quick_pow(mod[1],mod[0]-2,mod[0])*mod[1]%M;
   inv1=quick_pow(mod[0],mod[1]-2,mod[1])*mod[0]%M;

   int n,k;
   scanf("%d%d",&n,&k);

   for(int i=1;i<=n;i++) scanf("%lld",x+i);
   x[0]=0;

   y[0]=1;
   for(int i=1;i<=n-1;i++){
```

```
   y[i]=y[i-1]*(k-1+i)%p*quick_pow(i,p-2,p)%p;
}

for(int i=0;i<=n;i++){
   a[0][i]=a[2][i]=a[4][i]=x[i];
   a[1][i]=a[3][i]=a[5][i]=y[i];
}

int len=1;
while(len<=n+1) len<<=1;
len<<=1;

solver[0].FFT(a[0],len,1);
solver[0].FFT(a[1],len,1);
for(int i=0;i<len;i++) a[0][i]=a[0][i]*a[1][i]%mod[0];
solver[0].FFT(a[0],len,-1);

solver[1].FFT(a[2],len,1);
solver[1].FFT(a[3],len,1);
for(int i=0;i<len;i++) a[2][i]=a[2][i]*a[3][i]%mod[1];
solver[1].FFT(a[2],len,-1);

solver[2].FFT(a[4],len,1);
solver[2].FFT(a[5],len,1);
for(int i=0;i<len;i++) a[4][i]=a[4][i]*a[5][i]%mod[2];
solver[2].FFT(a[4],len,-1);

for(int i=1;i<=n;i++){
   if(a[0][i]<0) a[0][i]+=mod[0];
   if(a[2][i]<0) a[2][i]+=mod[1];

   ll A=(mul(a[0][i],inv0,M)+mul(a[2][i],inv1,M))%M;
```

```
        ll K=(a[4][i]-A)%mod[2]*quick_pow(M,mod[2]-2,mod[2])%mod[2];
        if(K<0) K+=mod[2];

        ll ans=(K%p)*(M%p)+A;
        ((ans%=p)+=p)%=p;
        printf("%lld\n",ans);
    }

    return 0;
}
```

快速沃舍尔变换

```
//c[n]=sum(a[i]*b[j]) where (i bit_operator j == n)

void FWT(ll* a,int n){
    for(int i=2;i<=n;i<<=1){
        int w=i>>1;
        for(int j=0;j<n;j+=i){
            for(int d=0;d<w;d++){
                ll u=a[j+d],v=a[j+d+w];
                a[j+d]=u+v;a[j+d+w]=u-v;
                //xor: a[j+d]=u+v,a[j+d+w]=u-v
                //and: a[j+d]=u+v
                //or : a[j+d+w]=u+v
            }
        }
    }
}
```

```
void UFWT(ll* a,int n){
   for(int i=2;i<=n;i<<=1){
      int w=i>>1;
      for(int j=0;j<n;j+=i){
         for(int d=0;d<w;d++){
            ll u=a[j+d],v=a[j+d+w];
            a[j+d]=(u+v)/2;a[j+d+w]=(u-v)/2;
            //xor: a[j+d]=(u+v)/2,a[j+d+w]=(u-v)/2
            //and: a[j+d]=u-v
            //or : a[j+d+w]=v-u
         }
      }
   }
}
```

多项式求逆

```
ll tmp[N];
//多项式求逆元
void getInv(ll* a,ll* b,int n){
   if(n==1){
      //此处 a[0]*b[0]%P=1
      b[0]=1;
      return ;
   }
   getInv(a,b,n>>1);
   int k=n<<1;
   for(int i=0;i<n;i++) tmp[i]=a[i];
   for(int i=n;i<k;i++) tmp[i]=b[i]=0;
   NTT(tmp,k,1);
```

```
   NTT(b,k,1);
   for(int i=0;i<k;i++){
      b[i]=b[i]*(2-tmp[i]*b[i]%P)%P;
      if(b[i]<0) b[i]+=P;
   }
   NTT(b,k,-1);
   for(int i=n;i<k;i++) b[i]=0;
}
```

Meissel_Lehmer 算法

```
//O(N^(2/3))求 1-N 素数个数
//二分求第 N 个素数

typedef long long ll;

const int N=105;
const int M=10005;
const int P=1e6+10;

ll dp[N][M];
bool vis[P];
int prime[P],num[P];

void init(){
   int t=0;
```

```
    vis[0]=vis[1]=1;
    for(int i=2;i<P;i++){
        if(!vis[i]){
            prime[t++]=i;
            for(int j=2*i;j<P;j+=i) vis[j]=1;
        }
    }

    for(int i=1;i<P;i++) num[i]=num[i-1]+(!vis[i]);

    for(int i=0;i<N;i++){
        for(int j=0;j<M;j++){
            if(!i) dp[i][j]=j;
            else dp[i][j]=dp[i-1][j]-dp[i-1][j/prime[i-1]];
        }
    }
}

ll phi(ll m,int n){
    if(n==0) return m;
    if(prime[n-1]>=m) return 1;
    if(m<M && n<N) return dp[n][m];
    return phi(m,n-1)-phi(m/prime[n-1],n-1);
}

ll Meissel_Lehmer(ll n){
    if(n<M) return num[n];

    int a=(int)sqrt(n+0.5),b=(int)cbrt(n+0.5);
    a=num[a],b=num[b];
    ll ans=phi(n,b)+b-1;
    for(int i=b+1;i<=a;i++){
```

```
      ans-=Meissel_Lehmer(n/prime[i-1]);
   }
   ans+=((ll)a*(a-1))/2 - ((ll)b*(b-1)/2);
   return ans;
}

bool small[P<<4];
int solve(ll a,ll b){
   mem(small,0);
   int k=(int)sqrt(b+0.5);
   for(int i=2;i<=k;i++){
      if(!vis[i]){
         for(ll j=(ll)i*max(2LL,(a+i-1)/i);j<=b;j+=i) small[j-a]=1;
      }
   }
   int ans=0;
   for(ll i=a;i<=b;i++) ans+=(!small[i-a]);
   if(a==1) ans--;
   return ans;
}

ll f(double n){
   return (ll)(n*(log(n)+log(log(n))-1) + n*(log(log(n))-2)/log(n));
}

int main()
{
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
      init();
   int n;
   cin>>n;
```

```
    if(n<=1000000){
        ll l=1,r=20000000;
        while(l<r){
            int m=(l+r)>>1;
            if(Meissel_Lehmer(m)<n) l=m+1;
            else r=m;
        }
        printf("%d\n",l);
    }
    else{
        ll tmp=f(double(n));
        ll l=tmp-4000000,r=tmp+4000000;
        ll now=Meissel_Lehmer(l-1);
        ll l1=l,r1=r;
        while(l1<r1){
            ll m=(l1+r1)>>1;
            if(solve(l,m)+now<n) l1=m+1;
            else r1=m;
        }
        printf("%lld\n",l1);
    }
     return 0;
}
```

原根

//设 m 是正整数，a 是整数，若 a 模 m 的阶等于 φ(m)，则称 a 为模 m 的一个原根。（其中 φ(m)表示 m 的欧拉函数）

```
const int N=100;
int divv[N],t;
int p;
void divide(int x){
   t=0;
   for(int i=2;i<=x;i++){
      if(x%i==0){
         divv[t++]=i;
         while(x%i==0) x/=i;
      }
   }
   if(x!=1) divv[t++]=x;
}

ll qp(int x,int n){
   ll ans=1,base=x;
   while(n){
      if(n&1) (ans*=base)%=(p+1);
      (base*=base)%=(p+1);
      n>>=1;
   }
   return ans;
}

int main(){
   scanf("%d",&p);
   p--;
   divide(p);
   for(int i=2;;i++){
      int ok=1;
      for(int j=0;j<t;j++){
```

```
            if(qp(i,p/divv[j])==1){
                ok=0;
                break;
            }
        }
        if(ok){
            printf("%d\n",i);
            break;
        }
    }
     return 0;
}
```

线性基

//消元求异或

```
const int N=61;

void gauss(vector<ll>& y,vector<ll>& ans){
    sort(y.begin(),y.end());
    ans.clear();

    vector<ll>x;
    int sz=(int)y.size();

    if(!sz) return ;

    x.push_back(y[0]);
    for(int i=1;i<sz;i++) if(y[i]!=y[i-1]) x.push_back(y[i]);

    reverse(x.begin(),x.end());

    sz=(int)x.size();
    int id=N-1;
    for(int i=0;i<sz && id>=0;id--){
```

```
        int j=i;
        while(j<sz && !((x[j]>>id)&1)) j++;
        if(j==sz) continue;

        swap(x[i],x[j]);
        for(int k=0;k<sz;k++){
            if(k==i) continue;
            if((x[k]>>id)&1) x[k]^=x[i];
        }
        i++;
    }

    for(int i=0;i<sz;i++){
        if(!x[i]) return ;
        ans.push_back(x[i]);
    }
}
```

Miller_Rabin&rho

```
ll gcd(ll a,ll b){
   return !b?a:gcd(b,a%b);
}

void add(ll& a,ll b,ll mod){
   a+=b;
   if(a>=mod) a-=mod;
}

ll mul(ll a,ll b,ll mod){
   ll ans=0;
   while(b){
      if(b&1) add(ans,a,mod);
      add(a,a,mod);
```

```
        b>>=1;
    }
    return ans;
}

ll quick_pow(ll a,ll n,ll mod){
    a%=mod;
    ll ans=1;
    while(n){
        if(n&1) ans=mul(ans,a,mod);
        a=mul(a,a,mod);
        n>>=1;
    }
    return ans;
}

//[0,n)
ll getRandom(ll n){
    return ((1ULL*rand())+((1ULL*rand())<<8)+((1ULL*rand())<<16))%n;
}

//返回 1 说明是合数,返回 0 说明可能是素数
bool check(ll n){

    ll a=getRandom(n-2)+2,d=n-1;
    int s=0;
    while(!(d&1)) d>>=1,s++;
```

```
    ll tmp=quick_pow(a,d,n);
    if(tmp==1) return 0;

    for(int i=1;i<=s;i++){
        if((tmp+1)%n==0) return 0;
        tmp=mul(tmp,tmp,n);
    }
    return 1;
}

bool miller_rabin(ll n){
    if(n<=1) return 0;
    if(n==2) return 1;
    if(!(n&1)) return 0;

    int t=40;
    while(t--){
        if(check(n)) return 0;
    }
    return 1;
}

void resolve(ll n,map<ll,int>& ans){

    if(n<=1) return;
    if(miller_rabin(n)){
        ans[n]++;
        return ;
```

```
    }

    while(1){
        ll A = getRandom(n), x = getRandom(n) ,y = x;
        ll k=2;
        int cnt=0;
        while(++cnt<=65536){
            x = (mul(x,x,n)+A)%n;
            ll g=gcd(abs(x-y),n);
            if(g != n && g != 1){
                resolve(g,ans);
                resolve(n/g,ans);
                return ;
            }
            if(cnt==k) y=x,k<<=1;
        }
    }
}
```

四、数据结构与离线算法

莫队算法

//使用莫队算法时如果单次修改复杂度是O(logN)或者更差，可考虑离线优化常数

```
const int N=20005;
```

```
struct Query{
   int l,r,block,id;
   bool operator < (const Query& tmp)const{
      if(block!=tmp.block) return block<tmp.block;
      return r<tmp.r;
   }
}q[N];

struct BIT{
   ll bit[N];
   int n;
   void init(int _n){
      n=_n;
      mem(bit,0);
   }

   void add(int i,int x){
      while(i<=n){
         bit[i]+=x;
         i+=i&-i;
      }
   }

   ll getsum(int i){
      ll ans=0;
      while(i){
         ans+=bit[i];
         i-=i&-i;
      }
      return ans;
   }
```

```
}cnt,sum;

ll ans[N],s[N],curans;
int curl,curr,Max,a[N];

void query(int l,int r){
   if(r<curr){
      for(int i=curr;i>r;i--){
         int Rank=cnt.getsum(a[i]);
         curans-=Rank*a[i]+sum.getsum(Max)-sum.getsum(a[i]);
         cnt.add(a[i],-1);
         sum.add(a[i],-a[i]);
      }
   }
   else{
      for(int i=curr+1;i<=r;i++){
         int Rank=cnt.getsum(a[i])+1;
         curans+=Rank*a[i]+sum.getsum(Max)-sum.getsum(a[i]);
         cnt.add(a[i],1);
         sum.add(a[i],a[i]);
      }
   }
   curr=r;

   if(l<curl){
      for(int i=curl-1;i>=l;i--){
         int Rank=cnt.getsum(a[i])+1;
         curans+=Rank*a[i]+sum.getsum(Max)-sum.getsum(a[i]);
         cnt.add(a[i],1);
         sum.add(a[i],a[i]);
      }
   }
```

```
    else{
        for(int i=curl;i<l;i++){
            int Rank=cnt.getsum(a[i]);
            curans-=Rank*a[i]+sum.getsum(Max)-sum.getsum(a[i]);
            cnt.add(a[i],-1);
            sum.add(a[i],-a[i]);
        }
    }
    curl=l;
}

void solve(){
    int n,m;
    scanf("%d%d",&n,&m);
    int len=(int)sqrt(n);
    Max=0;
    for(int i=1;i<=n;i++){
        scanf("%d",a+i);
        s[i]=s[i-1]+a[i];
        Max=max(Max,a[i]);
    }
    cnt.init(Max);
    sum.init(Max);
    for(int i=1;i<=m;i++){
        scanf("%d%d",&q[i].l,&q[i].r);
        q[i].block=(q[i].l-1)/len+1;
        q[i].id=i;
        ans[i]=(q[i].r-q[i].l+2)*(s[q[i].r]-s[q[i].l-1]);
    }
    sort(q+1,q+m+1);
    curl=1,curr=0;
    curans=0;
```

```
   for(int i=1;i<=m;i++){
      query(q[i].l,q[i].r);
      ans[q[i].id]-=curans;
   }
   for(int i=1;i<=m;i++){
      printf("%lld\n",ans[i]);
   }
}

int main(){
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
      int T;
   scanf("%d",&T);
   while(T--) solve();
    return 0;
}
```

线段树+扫描线求矩阵并面积

```
const int N=1e5;

double Hash[N<<1];

struct Line{
   int flag;
   double l,r,y;
   Line(){}
```

```
    Line(int flag,double l,double r,double y){
        this->flag=flag;
        this->l=l;
        this->r=r;
        this->y=y;
    }
    bool operator < (const Line& tmp)const{
        return y<tmp.y;
    }
}line[N<<1];

struct Node{
    int flag;
    double sum;
}t[N<<3];

void build(int root,int l,int r){
    t[root].flag=0;
    t[root].sum=0.0;
    if(l<r){
        int m=(l+r)>>1;
        build(lson,l,m);
        build(rson,m+1,r);
    }
}

void pushup(int root,int l,int r){
    if(t[root].flag) t[root].sum=Hash[r+1]-Hash[l];
    else if(l==r) t[root].sum=0;
    else t[root].sum=t[lson].sum+t[rson].sum;
}
```

```
void update(int root,int flag,int a,int b,int l,int r){
   if(a==l && b==r){
      t[root].flag+=flag;
      pushup(root,l,r);
      return ;
   }
   int m=(l+r)>>1;
   if(b<=m) update(lson,flag,a,b,l,m);
   else if(a>m) update(rson,flag,a,b,m+1,r);
   else{
      update(lson,flag,a,m,l,m);
      update(rson,flag,m+1,b,m+1,r);
   }
   pushup(root,l,r);
}

int n,m;
int getid(double x){
   return lower_bound(Hash+1,Hash+m+1,x)-Hash;
}

bool solve(int cas){
   if(scanf("%d",&n) && !n) return 0;

   for(int i=1;i<=n;i++){
      double x1,y1,x2,y2;
      scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
      Hash[i]=x1,Hash[i+n]=x2;
      line[i]=Line(1,x1,x2,y1);
      line[i+n]=Line(-1,x1,x2,y2);
   }
   sort(Hash+1,Hash+2*n+1);
```

```
   m=unique(Hash+1,Hash+2*n+1)-Hash-1;
   sort(line,line+2*n+1);

   build(1,1,m-1);
   double ans=0;
   for(int i=1;i<=2*n;i++){
      if(i>1) ans+=t[1].sum*(line[i].y-line[i-1].y);
      update(1,line[i].flag,getid(line[i].l),getid(line[i].r)-1,1,m-
1);
   }

   printf("Test case #%d\nTotal explored area: %.2lf\n\n",cas,ans);
   return 1;
}

int main(){
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
   int cas=1;
      while(solve(cas++));
    return 0;
}
```

主席树

```
//主席树求区间不同数个数，此处可省略建树阶段
//可求区间第 k 大

const int N=30005*20;

int a[N],b[N];
```

```
int t[N],lson[N],rson[N],num[N];
int cnt;

int build(int l,int r){
   int root=cnt++;
   if(l<r){
      int m=(l+r)>>1;
      lson[root]=build(l,m);
      rson[root]=build(m+1,r);
   }
   return root;
}

void copy(int i,int j){
   lson[i]=lson[j];
   rson[i]=rson[j];
   num[i]=num[j];
}

int update(int root,int pos,int val,int l,int r){
   int newroot=cnt++;
   copy(newroot,root);
   if(l==r) num[newroot]+=val;
   else{
      int m=(l+r)>>1;
      if(pos<=m){
         lson[newroot]=update(lson[root],pos,val,l,m);
         rson[newroot]=rson[root];
         num[newroot]=num[lson[newroot]]+num[rson[newroot]];
      }
      else{
         lson[newroot]=lson[root];
```

```
            rson[newroot]=update(rson[root],pos,val,m+1,r);
            num[newroot]=num[lson[newroot]]+num[rson[newroot]];
        }
    }
    return newroot;
}

int query(int root,int pos,int l,int r){
    if(l==r) return num[root];
    int m=(l+r)>>1;
    if(pos<=m) return query(lson[root],pos,l,m)+num[rson[root]];
    else return query(rson[root],pos,m+1,r);
}

int pre[N];

int main(){
     //freopen("in.txt","r",stdin);
     //freopen("out.txt","w",stdout);
       int n;
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%d",a+i),b[i]=a[i];
    sort(b+1,b+n+1);
    t[0]=build(1,n);
    for(int i=1;i<=n;i++){
        int pos=lower_bound(b+1,b+n+1,a[i])-b;
        if(pre[pos]){
            int tmproot=update(t[i-1],pre[pos],-1,1,n);
            t[i]=update(tmproot,i,1,1,n);
        }
        else{
            t[i]=update(t[i-1],i,1,1,n);
```

```
      }
      pre[pos]=i;
   }
   int q;
   scanf("%d",&q);
   while(q--){
      int l,r;
      scanf("%d%d",&l,&r);
      printf("%d\n",query(t[r],l,1,n));
   }
    return 0;
}
```

//主席树求区间第 k 大

```
const int N=100010;
int t[N],lson[N*20],rson[N*20],num[N*20],cnt;

int update(int root,int pos,int val,int l,int r){
   int newroot=cnt++;
   if(l==r) num[newroot]+=val;
   else{
      int m=(l+r)>>1;
      if(pos<=m){
         lson[newroot]=update(lson[root],pos,val,l,m);
         rson[newroot]=rson[root];
         num[newroot]=num[lson[newroot]]+num[rson[newroot]];
      }
      else{
         lson[newroot]=lson[root];
```

```
        rson[newroot]=update(rson[root],pos,val,m+1,r);
        num[newroot]=num[lson[newroot]]+num[rson[newroot]];
      }
   }
   return newroot;
}

int query(int lt,int rt,int k,int l,int r){
   if(l==r) return l;
   int m=(l+r)>>1;
   if(num[rson[rt]]-num[rson[lt]]>=k) return
query(rson[lt],rson[rt],k,m+1,r);
   else return query(lson[lt],lson[rt],k-(num[rson[rt]]-
num[rson[lt]]),l,m);
}

int a[N],b[N];

int main(){
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    int n;
   scanf("%d",&n);
   for(int i=1;i<=n;i++) scanf("%d",a+i),b[i]=a[i];
   sort(b+1,b+n+1);
   t[0]=cnt++;
   for(int i=1;i<=n;i++){
      int pos=lower_bound(b+1,b+n+1,a[i])-b;
      t[i]=update(t[i-1],pos,1,1,n);
   }
   int q;
   scanf("%d",&q);
```

```
   while(q--){
      int l,r,k;
      scanf("%d%d%d",&l,&r,&k);
      l++,r++;
      printf("%d\n",b[query(t[l-1],t[r],k,1,n)]);
   }
    return 0;
}
```

可持久化字典树

//区间异或最值查询

```
const int N=5e4+10;

int t[N];
int ch[N*32][2],val[N*32];
int cnt;

void init(){
   mem(ch,0);
   mem(val,0);
   cnt=1;
}

int add(int root,int x){
   int newroot=cnt++,ret=newroot;
   for(int i=30;i>=0;i--){
      ch[newroot][0]=ch[root][0];
```

```
      ch[newroot][1]=ch[root][1];
      int now=(x>>i)&1;
      root=ch[root][now];
      ch[newroot][now]=cnt++;
      newroot=ch[newroot][now];
      val[newroot]=val[root]+1;
   }
   return ret;
}

int query(int lt,int rt,int x){
   int ans=0;
   for(int i=30;i>=0;i--){
      int now=(x>>i)&1;
      if(val[ch[rt][now^1]]-val[ch[lt][now^1]]){
         ans|=(1<<i);
         rt=ch[rt][now^1];
         lt=ch[lt][now^1];
      }
      else{
         rt=ch[rt][now];
         lt=ch[lt][now];
      }
   }
   return ans;
}
```

整体二分

//k 大数查询

```
const int N=1e5+10;

struct Query{
    int x,y,c,id,ty;
    ll sum,k;
}Q[N],Q1[N],Q2[N];

ll val[N<<2],tag[N<<2];

void pushdown(int root,int l,int r){
    if(!tag[root]) return ;
    int c=tag[root],m=(l+r)>>1;
    tag[root]=0;
    tag[lson]+=c,tag[rson]+=c;
    val[lson]+=(m-l+1)*c;
    val[rson]+=(r-m)*c;
}

void update(int root,int l,int r,int a,int b,int v){
    if(a==l && b==r){
        val[root]+=(r-l+1)*v;
        tag[root]+=v;
        return ;
    }
    pushdown(root,l,r);
    int m=(l+r)>>1;
    if(b<=m) update(lson,l,m,a,b,v);
    else if(a>m) update(rson,m+1,r,a,b,v);
    else{
        update(lson,l,m,a,m,v);
        update(rson,m+1,r,m+1,b,v);
    }
    val[root]=val[lson]+val[rson];
}

ll query(int root,int l,int r,int a,int b){
    if(l==a && r==b) return val[root];
    pushdown(root,l,r);
    int m=(l+r)>>1;
    if(b<=m) return query(lson,l,m,a,b);
    else if(a>m) return query(rson,m+1,r,a,b);
    else return query(lson,l,m,a,m)+query(rson,m+1,r,m+1,b);
}

ll tmp[N*2];
int ans[N];
int n;
```

```
void fuck(int a,int b,int l,int r){
    if(a>b) return ;
    if(l==r){
        for(int i=a;i<=b;i++){
            if(Q[i].ty==2){
                ans[Q[i].id]=l;
            }
        }
        return ;
    }

    int m=(l+r)>>1;
    for(int i=a;i<=b;i++){
        if(Q[i].ty==1 && Q[i].c<=m){
            update(1,1,n,Q[i].x,Q[i].y,1);
        }
        else if(Q[i].ty==2){
            tmp[i]=query(1,1,n,Q[i].x,Q[i].y);
        }
    }

    for(int i=a;i<=b;i++){
        if(Q[i].ty==1 && Q[i].c<=m){
            update(1,1,n,Q[i].x,Q[i].y,-1);
        }
    }

    int t1=0,t2=0;
    for(int i=a;i<=b;i++){

        if(Q[i].ty==1){
            if(Q[i].c<=m) Q1[++t1]=Q[i];
            else Q2[++t2]=Q[i];
        }
        else{
            if(Q[i].sum+tmp[i]>=Q[i].k) Q1[++t1]=Q[i];
            else Q[i].sum+=tmp[i],Q2[++t2]=Q[i];
        }
    }

    for(int i=a;i<=a+t1-1;i++) Q[i]=Q1[i-a+1];
    for(int i=a+t1;i<=b;i++) Q[i]=Q2[i-(a+t1)+1];
    fuck(a,a+t1-1,l,m);
    fuck(a+t1,b,m+1,r);
}

void solve(){
    int m;
```

```
    scanf("%d%d",&n,&m);

    int cnt=0;
    for(int i=1;i<=m;i++){
        int ty,a,b,c;
        scanf("%d%d%d%d",&ty,&a,&b,&c);
        if(ty==1){
            Q[i].ty=1,Q[i].x=a,Q[i].y=b,Q[i].c=c;
            update(1,1,n,a,b,1);
        }
        else{
            Q[i].ty=2,Q[i].x=a,Q[i].y=b;
            Q[i].k=query(1,1,n,a,b)-c+1;
            Q[i].id=++cnt,Q[i].sum=0;
        }
    }
    mem(val,0),mem(tag,0);
    fuck(1,m,0,n);
    for(int i=1;i<=cnt;i++) printf("%d\n",ans[i]);
}

int main(){
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);

    solve();
    return 0;
}
```

CDQ 分治

//用于求三维偏序，一维排序，二维分治，三维线段树（树状数组）维护
//求动态逆序对

```
const int N=1e5+10;

struct Query{
    int pos,val,id;
    Query(){}
    Query(int pos,int val,int id){
        this->pos=pos;
        this->val=val;
```

```
        this->id=id;
    }
}Q[N];

struct Bit{
    int bit[N];

    void reset(){
        mem(bit,0);
    }

    Bit(){
        reset();
    }

    void add(int i,int d){
        while(i<N){
            bit[i]+=d;
            i+=i&-i;
        }
    }

    void clear(int i){
        while(i<N){
            bit[i]=0;
            i+=i&-i;
        }
    }

    int sum(int i){
        int ans=0;
        while(i){
            ans+=bit[i];
            i-=i&-i;
        }
        return ans;
    }
}bit;

int anss[N],ansb[N];
Query tmp[N];
void cdq(int l,int r){
    if(l>=r) return ;
    int m=(l+r)>>1;
    cdq(l,m);
    cdq(m+1,r);

    int t1=l,t2=m+1,t=0;
```

```
    while(t1<=m && t2<=r){
        if(Q[t1].pos<Q[t2].pos){
            bit.add(Q[t1].val,1);
            tmp[++t]=Q[t1++];
        }
        else{
            int id=Q[t2].id;
            int tv=bit.sum(Q[t2].val);
            anss[id]+=tv;
            ansb[id]+=bit.sum(N-1)-tv;
            tmp[++t]=Q[t2++];
        }
    }
    while(t1<=m) tmp[++t]=Q[t1++];

    while(t2<=r){
        int id=Q[t2].id;
        int tv=bit.sum(Q[t2].val);
        anss[id]+=tv;
        ansb[id]+=bit.sum(N-1)-tv;
        tmp[++t]=Q[t2++];
    }

    for(int i=l;i<=m;i++) bit.clear(Q[i].val);
    for(int i=l;i<=r;i++) Q[i]=tmp[i-l+1];
}

bool vis[N];
int a[N],b[N],c[N],pos[N];
void solve(){
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",a+i),pos[a[i]]=i;
    for(int i=1;i<=m;i++){
        int x;
        scanf("%d",&x);
        Q[n-i+1]=Query(pos[x],x,n-i+1);
        c[n-i+1]=x,vis[x]=1;
    }
    int t=0;
    for(int i=1;i<=n;i++){
        if(!vis[i]){
            ++t;
            Q[t]=Query(pos[i],i,t),c[t]=i;
        }
    }
    cdq(1,n);
```

```
    vector<ll>ans;
    ll cur=0;
    bit.reset();
    for(int i=1;i<=n;i++){
        cur+=ansb[i]+bit.sum(c[i])-anss[i];
        bit.add(c[i],1);
        if(i>=n-m+1) ans.push_back(cur);
    }
    for(int i=(int)ans.size()-1;i>=0;i--) printf("%lld\n",ans[i]);
}

int main(){

    solve();
    return 0;
}
```

//求三维偏序

```
const int N=1e5+10;


struct Query{

   int a,b,id;

   Query(){}

   Query(int a,int b,int id){

      this->a=a;

      this->b=b;

      this->id=id;

   }

   bool operator < (const Query& tmp)const{
```

```
      return a<tmp.a;
   }
};

int val[N<<2];
void update(int root,int l,int r,int pos,int v){
   if(l==r) {val[root]=max(val[root],v);return ;}
   int m=(l+r)>>1;
   if(pos<=m) update(lson,l,m,pos,v);
   else update(rson,m+1,r,pos,v);
   val[root]=max(val[lson],val[rson]);
}

void set(int root,int l,int r,int pos,int v){
   if(l==r) {val[root]=v;return ;}
   int m=(l+r)>>1;
   if(pos<=m) set(lson,l,m,pos,v);
   else set(rson,m+1,r,pos,v);
   val[root]=max(val[lson],val[rson]);
}

int query(int root,int l,int r,int a,int b){
   if(a>b) return 0;
   if(l==a && r==b) return val[root];
```

```
    int m=(l+r)>>1;
    if(b<=m) return query(lson,l,m,a,b);
    else if(a>m) return query(rson,m+1,r,a,b);
    else return max(query(lson,l,m,a,m),query(rson,m+1,r,m+1,b));
}

int dp[N],H[N],n;

Query Q[N];
Query tmp[N];
int pos[N];

void cdq(int l,int r){
    if(l==r) return ;
    int m=(l+r)>>1;
    cdq(l,m);

    for(int i=m+1;i<=r;i++) tmp[i]=Q[i];
    sort(Q+m+1,Q+r+1);

    int t1=l,t2=m+1;
    while(t1<=m && t2<=r){
        if(Q[t1].a<Q[t2].a){
            update(1,1,n,pos[Q[t1].id],dp[Q[t1].id]);
```

```
         ++t1;
      }
      else{
         dp[Q[t2].id]=max(dp[Q[t2].id],query(1,1,n,1,pos[Q[t2].id]-
1)+1);
         ++t2;
      }
   }
   while(t2<=r){
      dp[Q[t2].id]=max(dp[Q[t2].id],query(1,1,n,1,pos[Q[t2].id]-
1)+1);
      ++t2;
   }
   for(int i=l;i<=m;i++) set(1,1,n,pos[Q[i].id],0);
   for(int i=m+1;i<=r;i++) Q[i]=tmp[i];
   cdq(m+1,r);
   t1=l,t2=m+1;
   int t=0;
   while(t1<=m && t2<=r){
      if(Q[t1].a<Q[t2].a) tmp[++t]=Q[t1++];
         else tmp[++t]=Q[t2++];
   }
   while(t1<=m) tmp[++t]=Q[t1++];
   while(t2<=r) tmp[++t]=Q[t2++];
```

```
    for(int i=l;i<=r;i++) Q[i]=tmp[i-l+1];
}

void solve(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        int a,b;
        scanf("%d%d",&a,&b);
        Q[i]=Query(a,b,i);
        H[i]=b,dp[i]=1;
    }
    sort(H+1,H+n+1);
    for(int i=1;i<=n;i++) pos[Q[i].id]=lower_bound(H+1,H+n+1,Q[i].b)-
H;

    cdq(1,n);
    int ans=0;
    for(int i=1;i<=n;i++){
        ans=max(ans,dp[i]);
        //cout<<dp[i]<<endl;
    }
    printf("%d\n",ans);
}
```

```
int main(){
    solve();
    return 0;
}
```

点分治

```
typedef pair<int,int>pii;
const int N=2e4+10;
const int INF=1e9;

struct Edge{
   int to,cost;
   Edge(){}
   Edge(int to,int cost):to(to),cost(cost){}
};
vector<Edge>G[N];
int subtree_size[N];
bool centroid[N];

void compute_subtree_size(int u,int f){
   subtree_size[u]=1;
   for(int i=0;i<(int)G[u].size();i++){
      int v=G[u][i].to;
```

```
      if(v==f || centroid[v]) continue;
      compute_subtree_size(v,u);
      subtree_size[u]+=subtree_size[v];
   }
}

pii search_centroid(int u,int f,int tot){
   pii ret=pii(INF,-1);
   int tmp=0;
   for(int i=0;i<(int)G[u].size();i++){
      int v=G[u][i].to;
      if(v==f || centroid[v]) continue;
      ret=min(ret,search_centroid(v,u,tot));
      tmp=max(tmp,subtree_size[v]);
   }
   tmp=max(tmp,tot-subtree_size[u]);
   return min(ret,pii(tmp,u));
}

void get_paths(int u,int f,int d,vector<int>&ds){
   ds[d%3]++;
   for(int i=0;i<(int)G[u].size();i++){
      int v=G[u][i].to;
      if(v==f || centroid[v]) continue;
```

```
      get_paths(v,u,d+G[u][i].cost,ds);
   }
}

int ans=0;

void dfs(int u){
   compute_subtree_size(u,-1);
   u=search_centroid(u,-1,subtree_size[u]).second;
   centroid[u]=1;

   for(int i=0;i<(int)G[u].size();i++){
      int v=G[u][i].to;
      if(centroid[v]) continue;
      dfs(v);
   }

   vector<int>tot(3,0);
   tot[0]=1;
   for(int i=0;i<(int)G[u].size();i++){
      int v=G[u][i].to;
      if(centroid[v]) continue;
      vector<int>ds(3,0);
      get_paths(v,u,G[u][i].cost,ds);
```

```
        for(int j=0;j<3;j++) ans+=ds[j]*tot[(3-j)%3];
        for(int j=0;j<3;j++) tot[j]+=ds[j];
    }
    //不能少
    centroid[u]=0;
}


void solve(){
    int n;
    scanf("%d",&n);
    for(int i=1;i<n;i++){
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        w%=3;
        G[u].push_back(Edge(v,w));
        G[v].push_back(Edge(u,w));
    }
    dfs(1);
    ans=ans*2+n;
    int g=__gcd(ans,n*n);
    printf("%d/%d\n",ans/g,n*n/g);
}
```

```
int main(){
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);

   solve();
    return 0;
}
```

动态点分治

```
//树上所有点要么 0 要么 1，修改某个点的状态，询问 0 对中最远的距离。
//初始情况所有点均为 0

typedef pair<int,int>pii;

const int N=1e5+10;
const int INF=1e9;

vector<int>G[N];

struct Heap{
```

```
priority_queue<int>que,tmp;


void push(int x){
   que.push(x);
}
void del(int x){
   tmp.push(x);
}
int sz(){
   return (int)(que.size()-tmp.size());
}
bool empty(){
   return sz()==0;
}
int top(){
   while(!que.empty() && !tmp.empty() && que.top()==tmp.top())
      que.pop(),tmp.pop();
   return que.top();
}
void pop(){
   while(!que.empty() && !tmp.empty() && que.top()==tmp.top())
      que.pop(),tmp.pop();
   que.pop();
}
```

```
   int sum_of_top2(){
      int tt=top();
      pop();
      int ans=tt+top();
      push(tt);
      return ans;
   }
}a[N],b[N],c;
//a[i]维护以 i 为重心的分离子树到重心的最长链
//b[i]维护以 i 为重心的子树到父重心的最长链
//c 维护所有重心的答案（最大+次大）
//a 是 b 的 top 集，c 由 a 算出

bool centroid[N];
int subtree_size[N];

void compute_subtree(int u,int f){
   subtree_size[u]=1;
   for(int i=0;i<(int)G[u].size();i++){
      int v=G[u][i];
      if(v==f || centroid[v]) continue;
      compute_subtree(v,u);
      subtree_size[u]+=subtree_size[v];
   }
```

```
}

pii search_centroid(int u,int f,int tot){
   pii ret=pii(INF,-1);
   int tmp=0;
   for(int i=0;i<(int)G[u].size();i++){
      int v=G[u][i];
      if(v==f || centroid[v]) continue;
      ret=min(search_centroid(v,u,tot),ret);
      tmp=max(tmp,subtree_size[v]);
   }
   tmp=max(tmp,tot-subtree_size[u]);
   ret=min(ret,pii(tmp,u));
   return ret;
}

void get_paths(int u,int f,int fcent,Heap& h){
   h.push(dis(u,fcent));
   for(int i=0;i<(int)G[u].size();i++){
      int v=G[u][i];
      if(v==f || centroid[v]) continue;
      get_paths(v,u,fcent,h);
   }
}
```

```
int fa[N];
void solve_sub(int u,int f){
   fa[u]=f;
   centroid[u]=1;

   a[u].push(0);
   for(int i=0;i<(int)G[u].size();i++){
      int v=G[u][i];
      if(centroid[v]) continue;
      compute_subtree(v,-1);
      int root=search_centroid(v,-1,subtree_size[v]).second;
      solve_sub(root,u);
      a[u].push(b[root].top());
   }

   if(a[u].sz()>=2) c.push(a[u].sum_of_top2());

   if(f>=1){
      b[u].push(dis(u,f));
      for(int i=0;i<(int)G[u].size();i++){
         int v=G[u][i];
         if(centroid[v]) continue;
         get_paths(v,u,f,b[u]);
```

```
      }
   }

   centroid[u]=0;
}

bool flag[N];
void update(int u){
   int s=u;
   if(a[u].sz()>=2) c.del(a[u].sum_of_top2());

   //删除
   if(!flag[u]){
      a[u].del(0);
      if(a[u].sz()>=2) c.push(a[u].sum_of_top2());
      while(fa[u]>=1){
         if(a[fa[u]].sz()>=2) c.del(a[fa[u]].sum_of_top2());
         a[fa[u]].del(b[u].top());
         b[u].del(dis(s,fa[u]));
         if(!b[u].empty()) a[fa[u]].push(b[u].top());
         if(a[fa[u]].sz()>=2) c.push(a[fa[u]].sum_of_top2());
         u=fa[u];
      }
   }
```

```
    else{
        a[u].push(0);
        if(a[u].sz()>=2) c.push(a[u].sum_of_top2());
        while(fa[u]>=1){
            if(a[fa[u]].sz()>=2) c.del(a[fa[u]].sum_of_top2());
            if(!b[u].empty()) a[fa[u]].del(b[u].top());
            b[u].push(dis(s,fa[u]));
            a[fa[u]].push(b[u].top());
            if(a[fa[u]].sz()>=2) c.push(a[fa[u]].sum_of_top2());
            u=fa[u];
        }
    }
}

void solve(){
    int n;
    scanf("%d",&n);
    for(int i=1;i<n;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        add_edge(u,v);
    }

    dfs(1,0,0);
```

```
compute_subtree(1,-1);
int root=search_centroid(1,-1,subtree_size[1]).second;
solve_sub(root,-1);

for(int i=1;i<=n;i++) flag[i]=0;
int tot=n;
char op[3];
int Q;
scanf("%d",&Q);
while(Q--){
   scanf("%s",op);
   if(op[0]=='C'){
      int u;
      scanf("%d",&u);
      update(u);
      flag[u]^=1;
      if(flag[u]) tot--;
      else tot++;
   }
   else{
      if(tot==0) printf("-1\n");
      else if(tot==1) printf("0\n");
      else printf("%d\n",c.top());
   }
```

```
    }
}
```

动态点分治+线段树

```
typedef pair<int,int>pii;
const int N=1e5+10;
const int K=70;
const int INF=1e9;

struct Edge{
    int to,nex;
}e[N<<1];

int head[N],cnt=1;

void add_edge(int u,int v){
    e[cnt].to=v,e[cnt].nex=head[u],head[u]=cnt++;
}

int dep[N],dp[20][N];
void dfs(int u,int f,int d){
    dep[u]=d,dp[0][u]=f;
```

```
    for(int i=1;i<20;i++) dp[i][u]=dp[i-1][dp[i-1][u]];
    for(int i=head[u];i;i=e[i].nex){
        int v=e[i].to;
        if(v==f) continue;
        dfs(v,u,d+1);
    }
}

int LCA(int u,int v){
    if(dep[u]<dep[v]) swap(u,v);
    int tmp=dep[u]-dep[v];
    for(int i=0;i<20;i++){
        if((tmp>>i)&1) u=dp[i][u];
    }
    if(u==v) return u;
    for(int i=19;i>=0;i--){
        if(dp[i][u]!=dp[i][v]) u=dp[i][u],v=dp[i][v];
    }
    return dp[0][u];
}

int dis(int u,int v){
    int lca=LCA(u,v);
    return abs(dep[lca]-dep[u])+abs(dep[lca]-dep[v]);
```

```
}

int w[N],n;
int val[N*K],lson[N*K],rson[N*K],cou=0;

void update(int& root,int l,int r,int pos,int v){
    if(!root) root=++cou;
    val[root]+=v;
    if(l==r) return ;
    int m=(l+r)>>1;
    if(pos<=m) update(lson[root],l,m,pos,v);
    else update(rson[root],m+1,r,pos,v);
}

int query(int root,int l,int r,int a,int b){
    if(!root) return 0;
    if(a>b) return 0;
    if(l==a && r==b) return val[root];
    int m=(l+r)>>1;
    if(b<=m) return query(lson[root],l,m,a,b);
    else if(a>m) return query(rson[root],m+1,r,a,b);
    else return query(lson[root],l,m,a,m)
+query(rson[root],m+1,r,m+1,b);
}
```

```
bool centroid[N];
int subtree_size[N];

void compute_subtree(int u,int f){
    subtree_size[u]=1;
    for(int i=head[u];i;i=e[i].nex){
        int v=e[i].to;
        if(v==f || centroid[v]) continue;
        compute_subtree(v,u);
        subtree_size[u]+=subtree_size[v];
    }
}

pii search_centroid(int u,int f,int tot){
    pii ret=pii(INF,-1);
    int tmp=0;
    for(int i=head[u];i;i=e[i].nex){
        int v=e[i].to;
        if(v==f || centroid[v]) continue;
        ret=min(ret,search_centroid(v,u,tot));
        tmp=max(tmp,subtree_size[v]);
    }
    tmp=max(tmp,tot-subtree_size[u]);
```

```
    return min(ret,pii(tmp,u));
}

int t[2][N];
void get_paths(int rt,int flag,int u,int d,int f){
    update(t[flag][rt],0,n,d,w[u]);
    for(int i=head[u];i;i=e[i].nex){
        int v=e[i].to;
        if(v==f || centroid[v]) continue;
        get_paths(rt,flag,v,d+1,u);
    }
}

int fa[N];
void solve_sub(int u,int f){
    fa[u]=f;
    centroid[u]=1;

    get_paths(u,0,u,0,0);
    for(int i=head[u];i;i=e[i].nex){
        int v=e[i].to;
        if(centroid[v]) continue;
        compute_subtree(v,-1);
        int root=search_centroid(v,-1,subtree_size[v]).second;
```

```
        get_paths(root,1,v,1,u);
        solve_sub(root,u);
    }

    centroid[u]=0;
}

int getans(int u,int k){
    int ans=query(t[0][u],0,n,0,k);
    int s=u;
    while(fa[u]>=1){
        int tmpd=k-dis(s,fa[u]);
        ans+=query(t[0][fa[u]],0,n,0,tmpd);
        ans-=query(t[1][u],0,n,0,tmpd);
        u=fa[u];
    }
    return ans;
}

void update(int u,int v){
    int s=u;
    update(t[0][u],0,n,0,v);
    while(fa[u]>=1){
        int tmpd=dis(s,fa[u]);
```

```
        update(t[0][fa[u]],0,n,tmpd,v);
        update(t[1][u],0,n,tmpd,v);
        u=fa[u];
    }
}

void scan(int& x){
    x=0;
    char ch=getchar();
    while(ch==' ' || ch=='\n') ch=getchar();
    while(ch>='0' && ch<='9') x=10*x+ch-'0',ch=getchar();
}

void print(int x){
    if(x>9) print(x/10);
    putchar(x%10+'0');
}

void solve(){

   int m;
    scan(n),scan(m);

    for(int i=1;i<=n;i++) scan(w[i]),head[i]=t[0][i]=t[1][i]=0;
```

```
    for(int i=1;i<n;i++){
        int u,v;
        scan(u),scan(v);
        add_edge(u,v);
        add_edge(v,u);
    }

    dfs(1,0,0);
    compute_subtree(1,-1);
    int root=search_centroid(1,-1,n).second;
    solve_sub(root,-1);

}
```

树套树

```
//区间内不同数个数带修改
//树状数组套主席树

const int N=20005;

int val[N*200],lson[N*200],rson[N*200];
int cnt;

void update(int root,int& newroot,int l,int r,int pos,int v){
```

```
   newroot=++cnt;
   val[newroot]=val[root]+v;
   if(l==r) return ;
   lson[newroot]=lson[root];
   rson[newroot]=rson[root];
   int m=(l+r)>>1;
   if(pos<=m) update(lson[root],lson[newroot],l,m,pos,v);
   else update(rson[root],rson[newroot],m+1,r,pos,v);
}

//   >=a
int query(int root,int l,int r,int a){
   if(l==r) return val[root];
   int m=(l+r)>>1;
   if(m>=a) return query(lson[root],l,m,a)+val[rson[root]];
   else return query(rson[root],m+1,r,a);
}

int t[N];
int n;
void add(int i,int pos,int v){
   while(i<=n+1){
      update(t[i],t[i],1,n+1,pos,v);
      i+=i&-i;
   }
}

int sum(int i,int pos){
```

```
    int ans=0;
    while(i){
        ans+=query(t[i],1,n+1,pos);
        i-=i&-i;
    }
    return ans;
}

struct Query{
    int type,l,r;
}Q[N];

int nex[N],a[N],H[N],vis[N];
set<int>st[N];

void solve(){
    int m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        scanf("%d",a+i);
        H[i]=a[i];
    }
    int hn=n;
    for(int i=1;i<=m;i++){
        char s[3];
        scanf("%s%d%d",s,&Q[i].l,&Q[i].r);
        if(s[0]=='Q') Q[i].type=1;
        else Q[i].type=2,H[++hn]=Q[i].r;
```

```
}
sort(H+1,H+hn+1);

for(int i=1;i<=n;i++){
   a[i]=lower_bound(H+1,H+hn+1,a[i])-H;
   if(vis[a[i]]) nex[vis[a[i]]]=i;
   vis[a[i]]=i;
   st[a[i]].insert(i);
}
for(int i=1;i<=n;i++) if(!nex[i]) nex[i]=n+1;
for(int i=1;i<=n;i++){
   add(i,nex[i],1);
}

for(int i=1;i<=m;i++){
   if(Q[i].type==1){
      printf("%d\n",sum(Q[i].r,Q[i].r+1)-sum(Q[i].l-1,Q[i].r+1));
   }
   else{
      int col=lower_bound(H+1,H+hn+1,Q[i].r)-H;
      set<int>::iterator it = st[a[Q[i].l]].lower_bound(Q[i].l);
      add(Q[i].l,nex[Q[i].l],-1);
      int tmp=nex[Q[i].l];
      if(it!=st[a[Q[i].l]].begin()){
         --it;
         add(*it,Q[i].l,-1);
         add(*it,tmp,1);
         nex[*it]=tmp;
```

```
}
st[a[Q[i].l]].erase(Q[i].l);

it=st[col].lower_bound(Q[i].l);
if(it==st[col].end()){
   if(it==st[col].begin()){
      nex[Q[i].l]=n+1;
      add(Q[i].l,n+1,1);
   }
   else{
      --it;
      add(*it,n+1,-1);
      add(*it,Q[i].l,1);
      add(Q[i].l,n+1,1);
      nex[*it]=Q[i].l;
      nex[Q[i].l]=n+1;
   }
}
else{
   if(it==st[col].begin()){
      add(Q[i].l,*it,1);
      nex[Q[i].l]=*it;
   }
   else{
      int last=*it;
      int pre=*(--it);
      add(pre,last,-1);
      add(pre,Q[i].l,1);
```

```
                add(Q[i].l,last,1);
                nex[pre]=Q[i].l;
                nex[Q[i].l]=last;
            }
        }
        st[col].insert(Q[i].l);
        a[Q[i].l]=col;
    }
  }
}

int main(){
   solve();
   return 0;
}



//k 大数查询
//线段树套线段树


const int N=1e5+10;

ll val[N*200];
int lson[N*200],rson[N*200],t[N*200],tag[N*200];
int cnt;

void pushdown(int root,int l,int r){
    if(!tag[root]) return ;
    int c=tag[root],m=(l+r)>>1;
    tag[root]=0;
    if(!lson[root]) lson[root]=++cnt;
    if(!rson[root]) rson[root]=++cnt;
```

```
    val[lson[root]]+=1LL*(m-l+1)*c;
    val[rson[root]]+=1LL*(r-m)*c;
    tag[lson[root]]+=c,tag[rson[root]]+=c;
}

void update1(int& root,int l,int r,int a,int b){
    if(!root) root=++cnt;
    if(l==a && r==b){
        val[root]+=b-a+1;
        tag[root]++;
        return ;
    }
    pushdown(root,l,r);
    int m=(l+r)>>1;
    if(b<=m) update1(lson[root],l,m,a,b);
    else if(a>m) update1(rson[root],m+1,r,a,b);
    else{
        update1(lson[root],l,m,a,m);
        update1(rson[root],m+1,r,m+1,b);
    }
    val[root]=val[lson[root]]+val[rson[root]];
}

ll query1(int root,int l,int r,int a,int b){
    if(!root) return 0;
    if(l==a && r==b) return val[root];
    pushdown(root,l,r);
    int m=(l+r)>>1;
    if(b<=m) return query1(lson[root],l,m,a,b);
    else if(a>m) return query1(rson[root],m+1,r,a,b);
    else return query1(lson[root],l,m,a,m)
+query1(rson[root],m+1,r,m+1,b);
}

int n;
void update2(int& root,int l,int r,int pos,int a,int b){
    if(!root) root=++cnt;
    update1(t[root],1,n,a,b);
    if(l==r) return ;

    int m=(l+r)>>1;
    if(pos<=m) update2(lson[root],l,m,pos,a,b);
    else update2(rson[root],m+1,r,pos,a,b);
}

int query(int root,int l,int r,int a,int b,int k){
    if(l==r) return l;
    int m=(l+r)>>1;
```

```
    ll tmp=query1(t[rson[root]],1,n,a,b);
    if(tmp>=k) return query(rson[root],m+1,r,a,b,k);
    else return query(lson[root],l,m,a,b,k-tmp);
}

void solve(){
    int m;
    scanf("%d%d",&n,&m);
    int root=0;
    while(m--){
        int ty,a,b,c;
        scanf("%d%d%d%d",&ty,&a,&b,&c);
        if(ty==1) update2(root,0,n,c,a,b);
        else{
            printf("%d\n",query(root,0,n,a,b,c));
        }
    }
}

int main(){
    solve();
    return 0;
}
```

五、图论

强联通分量分解

//线性时间分解强联通分量，可用于缩点
//有向图中，沿正向边 dfs 后再沿反向边 dfs，沿反向边 dfs 能到达的点在同一个环（即强联通）内。

```
const int V=4e5+10;

vector<int>G[V];
vector<int>rG[V];
```

```
vector<int>vs,cmp[V];
bool vis[V];
int id[V];

void add_edge(int u,int v){
   G[u].push_back(v);
   rG[v].push_back(u);
}

void dfs(int u){
   vis[u]=1;
   for(int i=0;i<G[u].size();i++){
      if(!vis[G[u][i]]) dfs(G[u][i]);
   }
   vs.push_back(u);
}

void rdfs(int u,int k){
   vis[u]=1;
   cmp[k].push_back(u);
   id[u]=k;
   for(int i=0;i<rG[u].size();i++){
      if(!vis[rG[u][i]]) rdfs(rG[u][i],k);
   }
}

//返回强联通分量个数
int scc(int n){
   mem(vis,0);
   vs.clear();
   for(int i=1;i<=n;i++)
      if(!vis[i]) dfs(i);
```

```
   mem(vis,0);
   int id=0;
   for(int i=vs.size()-1;i>=0;i--){
      if(!vis[vs[i]]) rdfs(vs[i],++id);
   }
   return id;
}
```

树链剖分
//每个点到根节点经过最多 log(n)条链

```
const int N=1e5+10;
vector<int>G[N];
int fa[N],dep[N],son[N],sz[N],top[N],pos[N];
int cnt;

void add_edge(int u,int v){
   G[u].push_back(v);
   G[v].push_back(u);
}

void init(int n){
   for(int i=1;i<=n;i++) G[i].clear();
   mem(son,-1),cnt=0;
```

```
}

void dfs(int u,int f,int d){
   fa[u]=f,dep[u]=d,sz[u]=1;
   for(int i=0;i<G[u].size();i++){
      int v=G[u][i];
      if(v==fa[u]) continue;
      dfs(v,u,d+1);
      sz[u]+=sz[v];
      if(son[u]==-1 || sz[v]>sz[son[u]]) son[u]=v;
   }
}
void dfs2(int u,int t){
   top[u]=t,pos[u]=++cnt;
   if(son[u]==-1) return ;
   dfs2(son[u],t);
   for(int i=0;i<G[u].size();i++){
      int v=G[u][i];
      if(v!=son[u] && v!=fa[u]) dfs2(v,v);
   }
}

void process(int u,int v){
   while(top[u]!=top[v]){
```

```
        if(dep[top[u]]<dep[top[v]]) swap(u,v);
        //solve(pos[top[u]],pos[u]);
        u=fa[top[u]];
    }
    if(dep[u]<dep[v]) swap(u,v);
    //solve(pos[v],pos[u]);

}
```

前向星

添加函数：

1.void del(int u,int i)

(1)输入：u-入点，i-边号

```
struct EdgeSet{
        int ec;
        int head[V];
        int v[E],c[E],nex[E],pre[E];
        void init(){
            ec=0;
            mem(head,-1);
        }
        void add(int u,int _v,int _c){
            v[ec]=_v;
            c[ec]=_c;
            nex[ec]=head[u];
            pre[ec]=-1;
            if (~head[u]) pre[head[u]]=ec;
```

```
        head[u]=ec;
        ec++;
    }
    void del(int u,int i){
        if (~pre[i]) nex[pre[i]] = nex[i];
        if (~nex[i]) pre[nex[i]] = pre[i];
        if (head[u]==i) head[u]=nex[i];
    }
}es;
```

动态倍增求 **LCA**

常数：点数 V，点数的对数 N
函数：
1.void add(int u,int fa)
(1) 时间：O(logn)
2.int getLca(int a,int b)
(1)时间：O(logn)
注意：
1.虚根为-1，若 a,b 中最小值小于-1，返回大者；等于-1，返回-1。
*/

```
struct Lca{
    int dep[V],f[V][N];
    void init(){ mem(f,-1);}
    void add(int u,int fa){
        if (~fa) dep[u] = dep[fa] + 1;
        else dep[u] = 0;
        f[u][0] = fa;
        int n=0;
        while (~f[u][n]){
```

```
            n++;
            f[u][n] = f[ f[u][n-1] ][n-1];
        }
    }
    int getLca(int a,int b){
        if (a>b) swap(a,b);
        if (a<-1) return b;
        else if (a==-1) return -1;

        if (dep[a]<dep[b]) swap(a,b);
        int d=dep[a]-dep[b],n=0;
        while (d){
            if (d & 1) a=f[a][n];
            n++;
            d>>=1;
        }
        n=N-1;
        while (n>=0){
         if (f[a][n] != f[b][n]) {
             a = f[a][n];
             b = f[b][n];
         }
         n--;
       }
       if (a == b)return a;
       else return f[a][0];
    }
}lca;
```

拓扑排序

时间：O(n)

函数：

1. void solve(int n,const EdgeSet &es)

(1) 输入：n-点数，es-边集

(2) 输出：数据成员 que[V]

```
struct topu{
    int head,tail;
    int indu[V],que[V];
    void init(){
        head=tail=-1;
        mem(indu,0);
    }
    void solve(int n,const EdgeSet &es){
        for (int u=0; u<n; u++){
            if (!indu[u]) {
                que[++tail]=u;
            }
        }
        while(head < tail){
            int u=que[++head];
            for (int i=es.head[u]; ~i; i=es.nex[i]){
                int v=es.v[i];
                indu[v]--;
                if (!indu[v]) {
                    que[++tail]=v;
                }
            }
        }
    }
}topu;
```

spfa 单源最短路

常数：点数 N
时间：O(m*2)
空间：O(n)
预处理：读入边集
下标：不限
函数：
1.void solve(int s,const EdgeSet &es)
(1)输入：s-源点，es-前向星边集
(2)输出：数据成员 d[u]为 u 到 s 的最距离，无法到达时为 0

```
struct Spfa{
    int d[N];//到源点的距离， - 1 表示无穷远
    int head,tail;//队列首尾
    int que[N];//枚举队列
    bool inQue[N];//是否在队列中
    void push(int u){
        if (!inQue[u]){
            tail++;
            tail %= N;
            que[tail]=u;
            inQue[u]=true;
        }
    }
    int fpop(){
        head++;
        head %= N;
        int u = que[head];
        inQue[u] = false;
        return u;
```

```
    }
    void init(){
        mem(d,-1);
        mem(inQue, 0);
        head=tail=0;
    }

    void solve(int s,const EdgeSet &es){
        init();
        d[s]=0;
        push(s);
        while (head!=tail){
            int u=fpop();
            //cerr<<u<<endl;
            for (int i=es.head[u]; ~i; i=es.nex[i]){
                int v=es.v[i];
                int c=es.c[i];
                if (d[v]==-1 || d[v]>d[u]+c){
                    d[v]=d[u]+c;
                    push(v);
                }
            }
        }
    }
}spfa;
```

A*算法求 k 短路（不同路相同值算多个）

常数：点数 N

时间：O(n*k*log(n*k)+m*2)

空间：O(n*k*log(n*k))

预处理：读入正向边集，构造逆向边集，跑逆向边的 spfa
下标：不限
函数：

```
1.int solve(int s,int t,int k,const EdgeSet &es)
(1)输入：s-源点，t-汇点，k-第 k 短路，es-前向星边集
(2)输出：俩点间的 k 短距离
*/
struct KpA{
    struct An{
        int u,g,h;
        An(){}
        An(int u,int g,int h):u(u),g(g),h(h){}
        bool operator<(const An &ot)const{
            return g+h > ot.g+ot.h;
        }
    };
    priority_queue<An> que;
    int cnt[N];

    int solve(int s,int t,int k,int *d,const EdgeSet &es){
        mem(cnt,0);
        if (d[s]==-1) return -1;
        if (s==t) k++;
        que.push( An( s, 0, d[s]));
        while (!que.empty()){
            An an=que.top();
            que.pop();
            int u=an.u;
            cnt[u]++;
            if (cnt[t]==k) return an.g+an.h;
            if (cnt[u]>k) continue;
            for (int i=es.head[u]; ~i; i=es.nex[i]){
```

```
                int v=es.v[i];
                int c=es.c[i];
                que.push( An( v, an.g+c, d[v]));
            }
        }
        return -1;
    }
}kpa;
```

dijkstra 算法求单源 k 短路及其数量

常数：点数 N，k 短 K
时间：O((k*n)^2 + m*k)
空间：O((k*n))
预处理：读入边集
下标：0
函数：
1.void solve(const int n,const int s,const int k,const EdgeSet es)
(1)输入：略
(2)输出：数据成员 d[u][k]表示源点到 u 的第 k 短路，cnt[u][k]表示其个数
备注：
1.可用单调队列优化：时间(m*k*log(m*k)) 空间(m*k*log(m*k))

```
struct KpDij{
    int d[N][K];
    int cnt[N][K];
    bool vis[N][K];

    void init(const int n,const int k){
        for (int i=0; i<n; i++) {
            for (int ki=0; ki<k; ki++){
```

```
                d[i][ki]=INF;
            }
        }
        mem( cnt, 0);
        mem( vis, 0);
    }
    void moveBack(int u,int l,int r){
        for (r--; r>l; r--){
            d[u][r]=d[u][r-1];
            cnt[u][r]=cnt[u][r-1];
        }
    }
    void solve(const int n,const int s,const int k,const EdgeSet &e
s){
        init(n,k);

        d[s][0]=0;
        cnt[s][0]=1;

        for (int i=1; i<k*n; i++){
            int mv=INF,mj,mk;

            for (int j=0; j<n; j++){
                for (int kj=0; kj<k; kj++){
                    if (!vis[j][kj] && d[j][kj] <mv){
                        mv = d[j][kj];
                        mj = j;
                        mk = kj;
                    }
                }
            }
            vis[mj][mk]=true;
```

```
            for (int j=es.head[mj]; ~j; j=es.nex[j]){
                int v=es.v[j];
                int c=es.c[j];

                for (int kj=0; kj<k; kj++){
                    int tmp=d[mj][mk]+c - d[v][kj];
                    if (tmp<0){
                        moveBack(v, kj, k);
                        d[v][kj] += tmp;
                        cnt[v][kj] = cnt[mj][mk];
                        break;
                    }else if (tmp==0){
                        cnt[v][kj] += cnt[mj][mk];
                        break;
                    }
                }
            }
        }
    }
}kpd;
```

floyd+矩阵快幂 求任意俩点恰好走 **k** 条路径的最短路

常数：点数 N
时间：O(n^3 * log(k))
空间：O(n^2)
预处理：输入初始矩阵的 n 和边值
下标：0
函数：
1.Matax pow(int k)

```
(1)输入：k-次方
(2)输出：恰好走 k 条路径的最短距离矩阵
*/
struct Matrx{
    int n;
    int d[N][N];
    Matrx(int n):n(n){
        mem(d,0x3f);
    }
    Matrx operator*(const Matrx& ot)const{
        Matrx ret(n);
        for (int i=0; i<n; i++)
            for (int j=0; j<n; j++)
                for (int k=0; k<n; k++){
                    ret.d[i][j]=min(ret.d[i][j],d[i][k]+ot.d[k][j]);
                }
        return ret;
    }

    Matrx Pow(int k){
        k--;
        Matrx ret=(*this),tmp=(*this);
        while (k){
            if (k&1) ret = ret*tmp;
            tmp = tmp*tmp;
            k>>=1;
        }
        return ret;
    }
};
```

```
/*
简介：2 分查找 + spfa 判断负权环 求最优比例环
原理分析：
1.二分枚举 k 然后通过 spfa 判断是否有负权环
2.有负权环<=>有负权简单环
*/
[ 代码略 ]
```

kruskal 算法

常数：点数 N
预处理：调用 init()函数
函数：
1.int solve(int m,Edge *e,EdgeSet &es)
(1)输入：m-源边数，e-源边数组，es-生成树边集
(2)输出：ret-最小生成树的值，数据成员 ec-生成树的边数，es-生成树的边集
注意：
1.可以将 es 相关直接去掉

```
*/
struct Kruskal{
    int ec;
    int zu[V];
    void init(){
        for (int i=0; i<V; i++) zu[i]=i;
    }
    int getZu(int x){
        if (zu[x] != x) zu[x] = getZu(zu[x]);
        return zu[x];
    }
```

```
    ll solve(int m,Edge *e,EdgeSet &es){
        ec=0;
        ll ret = 0;
        sort(e,e+m);
        for (int i=0; i<m; i++){
            int a = e[i].a;
            int b = e[i].b;
            int c = e[i].c;
            int za = getZu(a);
            int zb = getZu(b);

            if (za != zb){
                zu[za] = zb;
                es.add(a,b,c);
                es.add(b,a,c);
                ret += c;
                ec++;
            }
        }
        return ret;
    }
}kruskal;
```

有度数限制的最小生成树

常数：点数 V
时间：O(n*k)
空间：O(n)
预处理：
1.kruskal 跑除了树根外的最小生成树，添加到 es

2.kruskal 在 1 的基础上跑包树根的最小生成树，添加到 es,k -= 此时加上的边
3.若 k<0 不存在解，若 k==0 直接输出，若 k>0 跑本算法
函数：
1.int solve(int root,int k,int m,Edge *e,EdgeSet *_es)
(1)输入：root-树根，k-度数限制，m-和根连的边数，e-和根连的边，_es-树上边集的指针
(2)输出：加上额外边导致最小生成树值的变化量
*/

```
struct LimitMst{
    int mx[V];
    pai pi[V];
    int ei[V];
    EdgeSet *es;

    void dfs(int u,int fa){
        for (int i=es->head[u]; ~i; i=es->nex[i]){
            int v=es->v[i];
            int c=es->c[i];
            if (v!=fa){
                if (mx[u] > c) {
                    mx[v] = mx[u];
                    ei[v] = ei[u];
                    pi[v] = pi[u];
                }else{
                    mx[v] = c;
                    ei[v] = i;
                    pi[v] = pai(u,v);
                }
                dfs(v,u);
            }
        }
```

```
    }

    int solve(int root,int k,int m,Edge *e,EdgeSet * _es){
        es=_es;
        ll ret=0;
        mx[root]=0;
        dfs(root,-1);
        for (int i=1; i<=k; i++){
            int minV=1,minP,minC;
            for (int j=0; j<m; j++){
                int v=e[j].a;
                int c=e[j].c;
                int tmp = c - mx[v];
                if (tmp <  minV){
                    minV = tmp;
                    minP = v;
                    minC = c;
                }
            }

            if (minV <= 0){
                ret += minV;
                mx[minP] = minC;
                es->del(pi[minP].first,ei[minP]);
                es->del(pi[minP].second,ei[minP]^1);
                dfs(minP,root);
            }else break;
        }
        return ret;
    }
}lmt;
```

/*
简介：通过跑俩次 kruskal 判断最小生成树是否唯一
思路：
1. 跑一遍 kruskal 后标记树上点
2. 将边重排，权值相同但被标记的边放后面
3. 再跑一次 kruskal，若树上边出现未被标记的边，则不唯一，反之唯一
*/
代码略

/*
简介：最大边与最小边差值最大的生成树
思路：
1. 枚举最小边
2. 固定最小边后的生成树的最大边>=最小生成树的最大边
3. 分别跑最小生成树
*/
代码略

最小树形图（有向图的最小生成树）

常数：点数 V

时间：O(n*m)

空间：O(n)

预处理：无

函数：

1. int solve(int root,int n,int m,Edge *e)

(1) 输入：root-根节点，n-点数，m-边数，e-边数组

(2) 输出：最小树形图的值

```
struct DirectMst{
    double in[V];
    int pre[V],id[V],vis[V];

    double solve(int root,int n,int m,Edge *e){
```

```
double ret=0;
while (true) {
    /*第一步，找每个点的最小入边*/
    for (int i=0; i<n; i++) {
        in[i]=inf;
        pre[i]=-1;
    }
    for (int i=0; i<m; i++){
        int u=e[i].u;
        int v=e[i].v;
        double c=e[i].c;
        if (u!=v && v!=root && c<in[v]){
            in[v]=c;
            pre[v]=u;
        }
    }

    /*第二步，当存在根外的孤点时，不存在树形图*/
    for (int i=0; i<n; i++){
        if (i!=root && pre[i]==-1) return -1;
    }

    int cnt=0;
    mem(id, -1);
    mem(vis, -1);

    /*第三步，找环*/
    for (int i=0; i<n; i++){
        if (i==root) continue;
        ret+=in[i];
        int u=i;
        while (u!=root && vis[u]!=i && id[u]==-1){
```

```
                vis[u]=i;
                u=pre[u];
                //printf("%d - %d\n",i,u);
            }

            if (u!=root && id[u]==-1){//重新标号
                //printf("ls %d\n",i);
                for (int v=pre[u]; v!=u; v=pre[v]){
                    id[v]=cnt;
                }
                id[u]=cnt++;
            }
        }
        //printf("ps %d\n",cnt);
        if (cnt==0) break;
        for (int i=0; i<n; i++){//重新标号
            if (id[i]==-1) id[i]=cnt++;
        }
        /*第四步，更新其它点到环的距离*/
        for (int i=0; i<m; i++){
            double c=in[e[i].v];
            e[i].u=id[e[i].u];
            e[i].v=id[e[i].v];
            if (e[i].u!=e[i].v){
                e[i].c -= c;
            }
        }
        n=cnt;
        root=id[root];
    }
    return ret;
}
```

```
}dmst;
```

支配树

时间：O(n)
空间：O(n)
函数：

```
1. void solve(int n,int *que,const EdgeSet &es,Lca &lca)
(1) 输入：n-点数，que-拓扑序的反序列，es-点集，lca-lca 对象
(2) 输出：cnt[V]-支配树某子树的大小，es_res-支配树边集
*/
struct ControlTree{
    int cnt[V];
    EdgeSet es_res;

    void init(){
        mem(cnt,0);
        es_res.init();
    }

    void dfs(int u){
        cnt[u]=1;
        for (int i=es_res.head[u]; ~i; i=es_res.nex[i]){
            int v=es_res.v[i];
            dfs(v);
            cnt[u] += cnt[v];
        }
    }

    void solve(int n,int *que,const EdgeSet &es,Lca &lca){
```

```
        init();
        lca.init();
        for (int i=n-1; i>=0; i--){
            int u=que[i];
            //printf("que[%d] = %d\n",i+1,u+1);
            int fa=-2;
            for (int j=es.head[u]; ~j; j=es.nex[j]){
                int v=es.v[j];
                fa = lca.getLca(fa,v);
                //printf("ts %d %d\n",v+1,fa+1);
            }
            if (fa==-2) fa=-1;
            lca.add(u,fa);
            es_res.add(fa+1,u+1);
        }
        dfs(0);
        for (int i=0; i<n; i++) cnt[i]=cnt[i+1];
    }
}ctr;
```

六、其它

数位 DP 模板

//数位 DP 求完美数个数（完美数：能被十进制表示的各个非零数整除）

```
#include <stdio.h>
#include <iostream>
```

```
#include <string.h>

using namespace std;
typedef long long ll;
#define mem(a,b) memset(a,b,sizeof(a))

const int M=2525;

ll dp[20][M][60];
int id[M],num[20];

void init(){
   int cur=0;
   for(int i=1;i<=2520;i++){
      if(2520%i==0) id[i]=++cur;
   }
}

int gcd(int a,int b){
   return !b?a:gcd(b,a%b);
}

ll DP(int len,int flag,int sum,int lcm){
   if(len==0) return (sum%lcm==0);
```

```
   if(flag && dp[len][sum][id[lcm]]!=-1) return dp[len][sum]
[id[lcm]];

   int x=flag?9:num[len];

   ll ans=0;
   for(int i=0;i<=x;i++){
      if(i) ans+=DP(len-1,flag||(i<x),(sum*10+i)
%2520,lcm*i/gcd(lcm,i));
      else ans+=DP(len-1,flag||(i<x),(sum*10+i)%2520,lcm);
   }
   if(flag) dp[len][sum][id[lcm]]=ans;
   return ans;
}

ll getans(ll x){
   int len=0;
   while(x){
      num[++len]=x%10;
      x/=10;
   }
   return DP(len,0,0,1);
}

int main(){
```

```
    init();
    mem(dp,-1);
    int T;
    scanf("%d",&T);
    while(T--){
        ll l,r;
        scanf("%lld%lld",&l,&r);
        printf("%lld\n",getans(r)-getans(l-1));
    }
}
```

哈希表

```
struct HashTable{

    //最大存 K 条数据的哈希表
    const static int MOD=65537;
    const static int K=1e5+10;

    int head[MOD],val[K],nex[K],to[K],cnt;

    void clear(){
        for(int i=1;i<cnt;i++){
            head[to[i]%MOD]=0;
            to[i]=0,nex[i]=0,val[i]=0;
```

```
    }
    cnt=1;
}

HashTable(){
    clear();
}

void Insert(ull x){
    int left=x%MOD;
    for(int i=head[left];i;i=nex[i]){
        if(to[i]==x){
            val[i]++;
            return ;
        }
    }
    to[cnt]=x,nex[cnt]=head[left],val[cnt]=1,head[left]=cnt++;
}

void Del(ull x){
    int left=x%MOD;
    for(int i=head[left];i;i=nex[i]){
        if(to[i]==x){
            val[i]--;
            return ;
        }
    }
}
```

```
    int query(ull x){
        int left=x%MOD;
        for(int i=head[left];i;i=nex[i]){
            if(to[i]==x){
                return val[i];
            }
        }
        return 0;
    }
};
```

等差数列异或和

```
//求等差数列前 n 项异或和
//a 是首项，b 是公差，c 是除数，n 是项数
//枚举 c 为 1<<i
ll solve(ll a,ll b,ll c,ll n){
    if(!n) return 0;
    return n*(a/c)+n*(n-1)/2*(b/c)+solve((a+b*n)%c,c,b%c,(a%c+(b
%c)*n)/c);
}
```

单纯形法

```
struct Simplex{

    const static int maxN=1050;
```

```
const static int maxM=55;
double INF=1e100;
double eps=1e-7;

int m,n;
//求最大值的线性规划
//0~m-1 式是约束,0~n-1 是变量,m 式是目标函数,n 是常数项.
//每个式子都是 a0x0+a1x1+...+an-1xn-1+an>=0
int B[maxM],N[maxN];
double a[maxM][maxN];

void init(){
   mem(a,0);
}

void pivot(int r,int c){
   swap(B[r],N[c]);

   a[r][c]=1.0/a[r][c];
   for(int j=0;j<=n;j++) if(j!=c) a[r][j]*=a[r][c];

   for(int i=0;i<=m;i++){
      if(i==r) continue;
      for(int j=0;j<=n;j++){
         if(j==c) continue;
         a[i][j]-=a[i][c]*a[r][j];
      }
      a[i][c]=-a[i][c]*a[r][c];
   }
}

bool feasible(){
```

```
   while(1){
      int r,c;
      double p=0;
      for(int i=0;i<m;i++){
         if(a[i][n]<p) p=a[r=i][n];
      }
      if(p>-eps) return 1;

      p=0;
      for(int i=0;i<n;i++){
         if(a[r][i]<p) p=a[r][c=i];
      }

      if(p>-eps) return 0;

      /*
      p=a[r][n]/a[r][c];
      for(int i=0;i<m;i++){
         if(a[i][c]>eps){
            double t=a[i][n]/a[i][c];
            if(t<p) p=t,r=i;
         }
      }
      */
      pivot(r,c);
   }
}


int solve(int n,int m,double x[maxN],double& ret){
   this->n=n;
   this->m=m;
```

```
    for(int i=0;i<n;i++) N[i]=i;
    for(int i=0;i<m;i++) B[i]=n+i;

    if(!feasible()) return 0;

    while(1){
        int r,c;
        double p=0;
        for(int i=0;i<n;i++){
            if(-a[m][i]>p) p=-a[m][c=i];
        }
        if(p<eps){
            for(int i=0;i<n;i++) if(N[i]<n) x[N[i]]=0;
            for(int i=0;i<m;i++) if(B[i]<n) x[B[i]]=a[i][n];
            ret=a[m][n];
            return 1;
        }

        p=INF;
        for(int i=0;i<m;i++){
            if(a[i][c]>eps){
                double t=a[i][n]/a[i][c];
                if(t<p) p=t,r=i;
            }
        }
        if(p==INF) return -1;

        pivot(r,c);
    }
}
```

```
};
```

手动加栈

```
#pragma comment(linker, "/STACK:1024000000,1024000000")
```

输入输出优化

（1）一般优化版本

```
template<typename T>
void scan(T& x){
   x=0;
   char ch=getchar();
   while(ch==' ' || ch=='\n') ch=getchar();
   do{
      x=10*x+ch-'0';
      ch=getchar();
   }while(ch>='0' && ch<='9');
}

template<typename T>
void print(T x){
   if(x<=9) putchar(x+'0');
   else{
      print(x/10);
      putchar((x%10)+'0');
```

```
    }
}
```

（2）超级快版本

```
const int BufferSize=4e7;
char *head,*tail;
char buffer[BufferSize],ch;
bool flag;

char getch(){
// return getchar();
    if(head==tail){
        int len=fread(buffer,1,BufferSize,stdin);
        tail=(head=buffer)+len;
        flag=!flag;
    }
    return *head++;
}

void scan(int& x){
    x=0;
    ch=getch();
    while(ch==' ' || ch=='\n') ch=getch();
    while(ch>='0' && ch<='9') x=10*x+ch-'0',ch=getch();
}
```

大数类

```
// 输入：scan，=。
// 输出：print，getNum。
// 运算：+，* ，- ，/ ，% ，> 。
// 不支持负数。

class BigNum
{
   public:
      const static int MaxL=50000;
      const static ll Mod=10000;
      const static int Dlen=4;

      int num[MaxL/Dlen+1],len;

      void changeStoNum(const char *);

   public:
      BigNum() { len=1; memset(num,0,sizeof(num)); }
      BigNum(int);
      BigNum(const char *s) { changeStoNum(s); }

      BigNum operator + (const BigNum &) const;
      BigNum operator - (const BigNum &) const;
      BigNum operator * (const BigNum &) const;
      BigNum operator / (const int) const;
      bool operator > (const BigNum &a) const;
      int operator % (const int m) const;

      bool scan();
      void print();
```

```
};

BigNum::BigNum(int x)
{
   memset(num,0,sizeof(num));
   len=0;

   do
   {
      num[len++]=x-(x/Mod)*Mod;
      x/=Mod;

   }while(x);
}

void BigNum::changeStoNum(const char *x)
{
   int L=strlen(x);
   int temp,tp;
   int p=0;

   memset(num,0,sizeof(num));
   len=(L-1)/Dlen+1;

   for(int i=L-1;i>=0;i-=Dlen)
   {
      temp=0;
      tp=i-Dlen+1;

      if(tp<0)
         tp=0;
```

```
      for(int j=tp;j<=i;++j)
         temp=(temp<<3)+(temp<<1)+x[j]-'0';

      num[p++]=temp;
   }

   while(num[len-1]==0 && len>1)
      --len;
}

bool BigNum::scan()
{
   char x[MaxL];

   if(scanf("%s",x)==-1)
      return 0;

   changeStoNum(x);
}

void BigNum::print()
{
   printf("%d",num[len-1]);
   for(int i=len-2;i>=0;--i)
      printf("%04d",num[i]);
}

BigNum BigNum::operator + (const BigNum &b) const
{
   BigNum ret(*this);

   int L=max(b.len,len);
```

```
   for(int i=0;i<L;++i)
   {
      ret.num[i]+=b.num[i];

      if(ret.num[i]>=Mod)
      {
         ++ret.num[i+1];
         ret.num[i]-=Mod;
      }
   }

   ret.len=L;

   if(ret.num[L])
      ++ret.len;

   return ret;
}

BigNum BigNum::operator - (const BigNum &b) const     // 不支持负数，一定要是 a>b。
{
   int L=this->len;
   int p;
   BigNum ret(*this);

   for(int i=0;i<L;++i)
   {
      if(ret.num[i]<b.num[i])
      {
         p=i+1;
```

```
            while(!ret.num[p])
               ++p;

            --ret.num[p--];

            while(p>i)
               ret.num[p--]=Mod-1;

            ret.num[i]+=Mod-b.num[i];
         }
         else
            ret.num[i]-=b.num[i];
      }

      while(!ret.num[ret.len-1] && ret.len>1)
         --ret.len;

      return ret;
}

BigNum BigNum::operator * (const BigNum &b) const
{
      BigNum ret;
      int temp;

      ret.len=len+b.len;

      for(int i=0;i<len;++i)
         for(int j=0;j<b.len;++j)
         {
            ret.num[i+j]+=num[i]*b.num[j];
```

```
            if(ret.num[i+j]>=Mod)
            {
                temp=ret.num[i+j]/Mod;
                ret.num[i+j+1]+=temp;
                ret.num[i+j]-=temp*Mod;
            }
        }

    while(ret.num[ret.len-1]==0 && ret.len>1)
        --ret.len;

    return ret;
}

BigNum BigNum::operator / (const int b) const
{
    BigNum ret;
    int down=0;

    for(int i=len-1;i>=0;--i)
    {
        ret.num[i]=(num[i]+down*Mod)/b;
        down=num[i]+down*Mod-ret.num[i]*b;
    }

    ret.len=len;

    while(ret.num[ret.len-1]==0 && ret.len>1)
        --ret.len;

    return ret;
```

```
}

int BigNum::operator % (const int b) const
{
   int ret=0;

   for(int i=len-1;i>=0;--i)
      ret=((ret*Mod)%b+num[i])%b;

   return ret;
}

bool BigNum::operator > (const BigNum &b) const
{
   if(len!=b.len)
      return len>b.len;

   for(int i=len-1;i>=0;--i)
      if(num[i]!=b.num[i])
         return num[i]>b.num[i];

   return 0;
}
```

Java 大数类

```
//BigInteger 在 math 库
import java.math.*;
import java.util.*;

public class Main{
```

```
    public static void main(String []args){

        Scanner in = new Scanner(System.in);

        //EOF
        while(in.hasNext()){}

        BigInteger zero = BigInteger.valueOf(0);

        BigInteger a=in.nextBigInteger();
        BigInteger b=in.nextBigInteger();
        BigInteger c=in.nextBigInteger();
        int d=in.nextInt();

        a.add(b);
        a.subtract(b);
        a.multiply(b);
        a.divide(b);
        a.mod(b);
        a.compareTo(b);
        a.negate();
        //a^(-1)
        a.modInverse(b);
        //a^b%c;
        a.modPow(b,c);
        a.pow(d);
    }
}
```

VIM配置

```
set nu
set shortmess=atI
syntax on
set cursorline
set shiftwidth=4
set tabstop=4
set autoindent
set mouse=a
set nobackup
map<C-A> ggvG
map<C-C> "+y


map <F2> :call SetTitle()<CR>
func! SetTitle()
    exec "w"
    if &filetype is 'cpp'|| &filetype is 'c' || &filetype is 'cc'
      let l = 0
      let l = l + 1 | call setline(l,' ')
   endif
endfunc


"快捷键<F5>编译运行
map <F5> :call Compile()<CR>
func! Compile()
```

```
    exec "w"
    if &filetype is 'cpp' || &filetype is 'c'
        exec "!g++ % -o %<"
        exec "!./%<"
    endif
    if &filetype is 'java'
        exec "!javac %"
        exec "!java %<"
    endif
    if &filetype is 'python'
        exec "!python %"
    endif
endfunc
```