常用函数

读写优化

```
inline int read()//快读
{
    int x=0,f=0;
    char ch=getchar();
    while(ch>'9'||ch<'0'){f|=(ch=='-');ch=getchar();}
    while(ch<='9'&&ch>='0'){x=(x<<1)+(x<<3)+
(ch^48);ch=getchar();}
    return f?-x:x;
}
inline void write(int x)
{
    char f[200];
    int cnt=0,tmp=x>0?x:-x;
    if(x<0)putchar('-');
    while(tmp>0)f[cnt++]=tmp%10+'0',tmp/=10;
    while(cnt>0)putchar(f[--cnt]);
}
```

头

```
#include<bits/stdc++.h>
#define close
std::ios::sync_with_stdio(false),cin.tie(0),cout.tie(0)
using namespace std;
typedef long long ll;
const ll MAXN = 3e5+7;
const ll mod =1e9+7;
const ll inf =0x3f3f3f3f;
const ll INF =0x3f3f3f3f3f3f3f3f;

int lowbit(int x){ return x&-x; }
int gcd(int x,int y){int k=0; if(x<y)
{k=x;x=y;y=k;}while(x%y!=0){k=x%y;x=y;y=k;}return y;}
ll _power(ll a,int b){ll ans=1,res=a;while(b){if(b&1)
ans=ans*res%mod;res=res*res%mod;b>>=1;}return ans%mod;}
```

欧拉筛

```
void init()
{
    memset(vis,false,sizeof(vis));
    vis[1]=vis[0]=true;
    for(int i=2;i<maxn;i++)
    {
        if(!vis[i])
            prime[cnt++]=i;
        for(int j=0;j<=cnt&&i*prime[j]<=maxn;j++)
        {
            vis[i*prime[j]]=true;
            if(i%prime[j]==0)break;
        }
    }
}
```

二维前缀和

$sum[i][j] = sum[i][j-1] + sum[i-1][j] - sum[i-1][j-1] + a[i][j]$

两个矩阵相加减去重复部分

求中间区间的和左上x1 y1 右下 x2 y2

`ans=sum[x2][y2]-sum[x1-1][y2]-sum[x2][y1-1]+sum[x1-1][y1-1]`

二维差分

`p[i][j]=a[i][j]-a[i-1][j]-a[i][j-1]+a[i-1][j-1]`

减要从另一个方向减

倍增

倍增求RMQ问题

```
for(int i=1;i<=20;i++){sz[i]=sz[i-1]*2}//预处理防止超时
for(int i=1;i<=n;i++){cin>>m[i][0]}//从第i个开始往后数1个数之
间最大值记为本身
for(int j=1;j<=log2(n);j++){
    for(int i=1;i+sz[j]-1<=n;i++){
        m[i][j]=max(m[i][j-1],md[i+sz[j-1][j-1]]);//计算方
程 最重要
    }
}
for(int i=0;i<m;i++){
    cin>>l>>r;//要双向过半覆盖整个区间 k=log(r-l+1)
    int k=len(l-r+1);
    int ans=max(m[l][k],m[r-2K][k]);
    cout<<ans<<"\n";

}
```

快速乘

```
inline ll qmul(ll x,ll y,ll p)
{
    ll z=(long double)x/p*y;
    ll res=(unsigned long long)x*y-(unsigned long
long)z*p;
    return (res+p)%p;
}
```

二分

三分查找

求一个导数单调的单峰数组 的极值部分

整数： (l,mid),(mid,mid+1),(mid+1,r) 当分不出三块 无法三分 此时暴力枚举

或者：mid1=l+(r-l)/3 mid2=r-(r-l)/3 去掉小于的那段 等于就去掉右边

浮点数三分 求给定函数最大值

```
double f(double){
    double res=0;
    //求函数过程 省略
}
double l,r,eps=1e-6;
```

```
cin>>n>>l>>r;
for(int i=0;i<=n;i++){
    while(abs(r-l)>eps){//浮点数的r! =l
        double mid1=l+(r-l)/3,mid2=r-(r-l)/3;
        // 也可以用 mid1=mid-eps,mid2=mid+eps;mid=(l+r)/2
        //速度快点由二分改进
        if(f(mid1)<f(mid2)){
            l=mid1;
        }
        else r=mid2;
    }
    printf("%.4f",r)//误差非常小 哪个都行;
}
```

二分答案

用二分来查找答案是否合法

最大值最小化 枚举答案 时间复杂度降低

视情况改变check

```
bool check(int mid){
    int pre=a[1];
    int cnt=0;
    for(int i=2;i<=n;i++){
        ;
    }
    if(cnt<=n-c)
        return true;//如果是太少了或者相等 要往右找
    else
        return false;
}
for(int i=1;i<=n;i++) cin>>a[i];
sort(a+1,a+n+1);
int l=1, r=a[n]-a[1];//r为最大最近距离
while(l<=r){
    int mid=l+r>>1;
    if(check(mid)){
        l=mid+1;//l一定mid是合法的下一个 r一定是mid不合法的上一
个
    }
    else r=mid-1;
}
cout<<r
```

浮点数二分 拿eps求精度 while(fabs(r-l)>eps)

三分 分不了三个区间 要小范围暴力

或者特判lr的边界条件

并查集

并查集

朴素模板

```
int f[MAXN],n,m;
void clean(){
    for(int i=1;i<=n;i++) f[i]=i;
}
int find(int x){
    if(x!=f[x]) f[x]=find(f[x]);  return f[x];
}
void add(int x,int y){
    int fx=find(x),fy=find(y);
    if(fx!=fy) f[fx]=fy;
}
//按秩合并 初始化rank=1
//让集合深度最小，减少遍历复杂度
void add(int x,int y){
    int fx=find(x),fy=find(y);
    if(fx!=fy){
        if(rank[i]>rank[j]) f[fy]=fx;
        else{
            if(rank[i]==rank[j]) rank[j]++;
            f[fx]=fy;
        }
    }
}
```

带权

```
void clean(){
    for(int i=1;i<=n;i++) f[i]=i;
}
int find(int x){
    if(x!=f[x]) {
        int t=f[x];
        f[x]=find(f[x]);
        d[x]=(d[x]+d[t])%3;
    }
    return f[x];
}
void add(int x,int y,int dd){
        int fx=find(x),fy=find(y);
    if(fx!=fy){
        f[fx]=fy;
        if(dd==2)
        d[fx]=(d[y]-d[x]+1+3)%3;
```

```
        else
        d[fx]=(d[y]-d[x]+3)%3;
    }
}
bool pd(int x,int y){
    if(d[x]==2&&d[y]==1) return true;
    if(d[x]==1&&d[y]==0) return true;
    if(d[x]==0&&d[y]==2) return true;
    return false;
}
```

搜索-拓扑

拓扑排序

bfs 检查每个点入度 为0即加入队列 并且把出度的点的入度-1

代码实现

```
vector<ll> ans;
while(sz.size()){
    ll now=sz.front();
    sz.pop();
    ans.push_back(now);
        for(auto i: Out[now]){
            In[i]--;
            if(In[i]==0) sz.push(i);
        }
}
//最后有剩余的点 说明无法拓扑
```

例题 D - Change Usernames (atcoder.jp)

本质上检查图有无形成环，有环则输出No 使用拓扑排序来检查是否能层级展开

图论基础

存图-链式前向星

链式前向星模板

```
int n,m;
struct Edge{
    int to; // 边的终点
```

```
    int v; // 边权
    int last; // 上一条边的下标
}edge[MAXN];  // 对于n个点的边，要开n(n-1)
int menu[MAXN]; // menu[i]: i为起点的最后一条边的下标
void Start(){
    for (int i = 1; i <= n; ++i)
        menu[i] = -1;
}
void Build(int s, int t, int v, int x){
    edge[x].to = t;
    edge[x].v = v;
    edge[x].last = menu[s];
    menu[s] = x;
}
//链式前向星板子
int h[MAXN], e[MAXN], ne[MAXN], idx;
void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}
//初始化 memset(h, -1, sizeof h); idx = 0;
//遍历 int i = h[u]; ~i; i = ne[i]
```

单源最短路

指定点

bellman-ford算法

1.除了起点 其他点初始化为inf

2.松弛操作对于每一条边v->u 取min(d[u],d[v]+w)

更新顺序影响操作次数

```
for(int i=1;i<=n;i++){//判断负环 加到n次
    for(int u=1;u<=n;u++){
        for(j=0;j<adj[u].size();j++){
            int v=adj[u][v].first,w=adj[u][v].second;
            if(d[v]>d[u]+w){
               // if(i==n) return true//有负环
               //d[v]=d[u]+w;
            }
        }
    }
}
```

spfa-优化bellman

***清空vector！！！

1.建立队列 塞起点

2.取出队列头，扫描出边，如果能更新，节点入队。

3.重复操作知道为空

注：一个队列中只会存在一个数字

```
bool spfa(int st){
    memset(cnt,0,sizeof(cnt));
    for(int i=1;i<=n;i++) d[i]=inf;
    d[st]=0;
    queue<int> que;
    que.push(st);
    memset(vis,0,sizeof(vis));
    vis[st]=1;
    while(queue.size()){
        int u=que.front();que.pop();
        vis[u]=1;
        for(auto i:adj[u]){
            v=i.first;w=i.second;
            if(d[v]>d[u]+w){
                d[v]=d[u]+w;
                cnt[v]=cnt[u]+1;//判断阻断路径的条数
                if(cnt[v]>=n) return true;
                if(!vis[v]){
                    que.push(v);
                    vis[v];
                }
            }
        }
    }
    return false;
}
```

dijkstra(非负权)

1.除st外 记inf

2.找出一个未被标记的 d[u]最小的点 并更新所有的d 标记该点

3.重复2 直到全部标记

优先队列优化O(mlogm)

稠密图可以用 与边的个数有关

```
priority_queue<pair<int,int>,vector<pair<int,int>>,greater
<pair<int,int>>>  que;//小顶堆优化
que.push({0,st}) ;//要存d[u],u 不然会错
    while(que.size()){
```

```
        int u=que.top().second;que.pop();
        if(vis[u]) continue;
        vis[u]=1;
        for(auto i:adj[u]){//更新
            v=i.first;w=i.second;
            if(d[v]>d[u]+w){
                d[v]=d[u]+w;
                que.push({d[v],v});
            }
        }
    }
```

任意两点最短路

floyd算法(不允许负环)

设d[k,u,v]表示经过最多为k个点(编号不超过k),u到v的最短路

状态转移方程 d[k,u,v]=min(d[k-1,u,v],d[k-1,u,k]+d[k-1,k,v])

n^3

```
for(int k=1;k<=n;k++){
    for(int u=1;u<=n;u++){
        for(int v=1;v<=n;v++){
            d[u][v]=min(d[u][v],d[u][k]+d[k][v]);
        }
    }
}
```

特殊最短路

有向无环图->拓扑

边权均为1-bfs

边权0/1->双端队列

动态规划

动态规划

最长公共子序列

```
cin>>s1>>s2;
for(int i=1;i<=s1.length();i++){
    for(int j=1;j<=s2.length();j++){
        if(s[i-1]==s2[j-1] )dp[i][j]=dp[i-1][j-1]+1;
        else
            dp[i][j]=max(dp[i-1][j]),dp[i][j-1]);
    }
}
```

本质来说是一种贪心 从头遍历 来计算如何走会使结果最大

01背包

```
//w[i]为重量，v[i]为价值
for(int i=1;i<=n;i++){
    for(int j=1;j<=m;j++){
        if(j>=w[i])
            dp[i][j]=max(dp[i-1][j],dp[i-1][j-
w[i]]+v[i]);//选择取或者不取
        else
            dp[i][j]=dp[i-1][j];//没法取
    }
}
cout<<dp[n][m];
//滚动优化空间：  i-> i&1，n->n&1;
```

二维费用背包

```
void solve(){
    int n,m,k;cin>>n>>m>>k;
    m--;
    for(int i=1;i<=k;i++){
        cin>>v[i]>>w[i];
    }
    for(int l=1;l<=k;l++){
        for(int i=n;i>=v[l];i--){
            for(int j=m;j>=w[l];j--){
                dp[i][j]=max(dp[i][j],dp[i-v[l]][j-
w[l]]+1);
            }
        }
    }
    int maxs=dp[n][m],ans;
    for(int j=m;j>=0;j--){
        if(dp[n][j]==maxs) ans=j;
    //  cout<<j<<" : "<<dp[n][j]<<"\n";
    }
    cout<<maxs<<" "<<m+1-ans;
```

```
}
```

需要放完的完全背包

物品无限

```
for(int i=0;i<1005;i++) dp[i]=inf;dp[0]=0;//初始化
for(int i=1;i<=n;i++){
    for(int j=0;j<=m;j++){
        if(dp[j]!=inf)
            dp[j+w[i]]=min(dp[j+w[i]],dp[j]+v[i]);
        //检查的已经放了的物品数量 往后推
        //比如dp[0]=0，就是现在剩0格的时候 含有0的物品
        //放完第一个 假如第一个重量为w，则dp[w]=v;
        //当j遍历到w时 会再放物品 满足完全背包
    }
}
cout<<dp[m];
```

物品只有一个

```
    for(int j=0;j<=m;j++)
    ->
    for(int j=m;j>=v[i];j--)//这样就只能放一个
```

多重背包

多重背包需要拆分 利用 所有数都能被2^n数字相加表示的性质

进行二进制拆分

拆成1，2，4，8，16....(不大与n的数)个

```
int w[MAXN],v[MAXN],cnt=1,dp[6030];
void res(int a,int b,int t){
    int k=1;
    while(t>=k){
         w[cnt]=a*k,v[cnt++]=b*k;
        t-=k;
        k*=2;
    }
    if(t)
    w[cnt]=a*t,v[cnt++]=b*t;
}
void solve(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        int a,b,t;
```

```
        cin>>a>>b>>t;
        res(a,b,t);
    }
    for(int i=1;i<cnt;i++){
        for(int j=m;j>=w[i];j--){
            dp[j]=max(dp[j],dp[j-w[i]]+v[i]);
        }
    }
    cout<<dp[m];
}
```

背包路径记录

有条件背包（树上背包）

```
int v[MAXN],w[MAXN];//v表示价值，w表示重量
int dp[MAXN][MAXN]; //dp[i][j]表示第i个为子树，容量为j的最大v
void dfs(int u,int fa){
    for(int i=w[u];i<=m;i++) dp[u][i]=v[u];//
    for(auto i:adj[u]){
        dfs(i,u);
        for(int j=m;j>=w[u];j--){//小于w[i]放不下
            for(int k=0;k<=j-w[u];k++){//不能分配很多
                dp[u][j]=max(dp[u][j],dp[u][j-k]+dp[i]
[k]);
            }
        }
    }
}
void solve(){
    int root;cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>w[i]>>v[i];
        int u;cin>>u;
        if(u==-1) root=i;
        else{
            adj[u].push_back(i);
        }
    }
    dfs(root,-1);
    cout<<dp[root][m];
}
```

区间dp

基础：线性dp

合并石子问题，枚举区间长度与摆放位置

```
    int n;cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        a[i]+=a[i-1];//处理成前缀和
    }
    for(int len=2;len<=n;len++){//枚举区间长度
                for(int i=1;i+len-1<=n;i++){//枚举区间开始位
置
                dp[i][i+len-1]=inf;
                    for(int k=i;k<=i+len-2;k++){//枚举断开
处理的位置
                        dp[i][i+len-1]=min(dp[i][i+len-
1],dp[i][k]+dp[k+1][i+len-1]+a[i+len-1]-a[i-1]);

                    }
                }


    }
    cout<<dp[1][n];
```

环形区间dp

看成一条两倍长的链子自由选择n个进行dp

```
for(int i=1;i<=n;i++){
        cin>>b[i];
    //  a[i]+=a[i-1];//处理成前缀和
    }
    for(int i=n+1;i<=2*n;i++){
        b[i]=b[i-n];
    }
    for(int i=1;i<=n*2;i++){
        a[i]=a[i-1]+b[i];
        dp[i][i]=0,f[i][i]=0;
    }
    for(int len=2;len<=n;len++){//枚举区间长度
        for(int i=1;i+len-1<=2*n;i++){//枚举区间开始位置
            dp[i][i+len-1]=inf;
            f[i][i+len-1]=inf*-1;
            for(int k=i;k<=i+len-2;k++){//枚举断开处理的位置
                dp[i][i+len-1]=min(dp[i][i+len-1],dp[i]
[k]+dp[k+1][i+len-1]+a[i+len-1]-a[i-1]);
                f[i][i+len-1]=max(f[i][i+len-1],f[i]
[k]+f[k+1][i+len-1]+a[i+len-1]-a[i-1]);
            }
        }
    }
```

数位dp

使用dfs实现

```
int dfs(int pos,int pre_num,int flag){
    if(pos<=0) return 1;
    if(!flag&&dp[pos][pre_number]!=-1) return dp[pos]
[pre_num];
    if(flag) max_number=upper_bound[pos];
    else max_number=9;
    ll ret=0;
    for(int i=0;i<=max_number;i++){
        //转移方程
        ret+=dfs(pos-1,i,flag(i==max_number));
    }
    if(!flag) dp[pos][pre_number]=ret;
    return ret;
}
//memset(dp,-1,dizeof(dp));
// dfs(位数最大,0,1);
```

LCA基础

并查集解离线LCA

```
vector<int> G[MAXN];
vector<pair<int,int> > Q[MAXN];
void clean(){
    for(int i=1;i<=n;i++) f[i]=i;
}
int find(int x) {
    if(f[x]!=x){
        f[x]=find(f[x]);
    }
    return f[x];
}
void add(int x,int y){
    int fx=find(x),fy=find(y);
    if(fx!=fy){
        f[fx]=fy;
    }
}
void dfs(int u,int fa){
    vis[u]=1;
    for(auto &v:G[u]){
        if(v==fa) continue;
```

```
        dfs(v,u);
        add(u,v);
    }
    for(auto& it:Q[u]){
        int v=it.second,id=it.first;
        if(vis[v]) ans[id]=find(v);
    }
}
```

倍增解在线LCA

```
vector<int> G[MAXN];
vector<pair<int,int> > Q[MAXN];
int par[MAXN][20],dep[MAXN];
//par[u][i]代表点u的祖先中 深度为max(1,dep[u]-2^i)是谁
void dfs(int u,int fa){
    dep[u]=dep[fa]+1;
    par[u][0]=fa;
    for(int i=1;i<20;++i){
        par[u][i]=par[par[u][i-1]][i-1];
    }
    for(auto &v:G[u]){
        if(v==fa) continue;
        dfs(v,u);
    }
}
int getLCA(int u,int v){
    if(dep[u]<dep[v]) swap(u,v);
    for(int i=19;i>=0;--i){
        if(dep[par[u][i]]>=dep[v]) u=par[u][i];
    }
    if(u==v) return u;
    for(int i=19;i>=0;i--){
        if(par[u][i]!=par[v][i]){
            u=par[u][i];
            v=par[v][i];
        }
    }
    return par[u][0];
}
```

树状数组基础

树状数组

基本模板

```
int t[MAXN];
```

```
int lowbit(int x){
    return x&-x;
}
ll getsum(int x){//求1-x
    ll sum=0;
    while(x){
        sum+=t[x];
        x-=lowbit(x);
    }
    return sum;
}
void addv(int x,int val){//单点修改
    while(x<=n){
        t[x]+=val;
        x+=lowbit(x);
    }
}
//求区间[l,r]=getsum(r)-getsum(l-1);
```

树状数组求和-基础一阶差分

```
addv(c,e);addv(d+1,-1*e);//在后面再减去一位
```

单点查询即为前缀和 区间查询需要维护一个b[i]*i的数组

```

ll getsum(int x){//求1-x
    ll sum=0,k=x;
    while(x){
        sum+=(k+1)*t[x]-b[x];//减法公式
        x-=lowbit(x);
    }
    return sum;
}
void addb(int x,int val){
    ll k=x;
    while(x<=n){
        b[x]+=val*k;//每次要*x
        x+=lowbit(x);
    }
}
        addv(c,e),addb(c,e);
        addv(d+1,-1*e),addb(d+1,(e*-1));
//区间加要写两遍
//求区间[l,r]=getsum(r)-getsum(l-1)
```

树状数组的二阶差分

在某一个点l加上x后，需要在结束r+1点减去(l-r+1)*x

再在r+2点加上(l-r)*d来抵消

```
addv(l+1,d),addv(r+1,(r-l+1)*d*-1),addv(r+2,(r-l)*d);
```

树状数组求逆序对

目标：求出前面有几个数比当前的数要小，剩下的即为逆序对数量

```
for(int i=1;i<=n;i++){
    cin>>a;
    addv(a,1);
    ans+=i-getsum(tmp)//求出第i个位置前逆序对数量 累加即可
}
//如果数字大 需要离散化
```

树状数组求区间最大值

```
int lowbit(int x){
    return x&-x;
}
void updata(int x)//单点修改
{
    //使用方法 : 直接改a[x],然后update(x);
    int lx, i;
    while (x <= n)
    {
        h[x] = a[x];
        lx = lowbit(x);
        for (i=1; i<lx; i<<=1)
            h[x] = max(h[x], h[x-i]);
        x += lowbit(x);
    }
}
int query(int x, int y)//区间查询
{
    int ans = 0;
    while (y >= x)
    {
        ans = max(a[y], ans);
        y --;
        for (; y-lowbit(y) >= x; y -= lowbit(y))
            ans = max(h[ans);
    }
    return ans;
}
```

数论

基本符号

a|b(b能被a整除)

扩展欧几里得算法

求

$$ax+by=gcd(a,b)$$

```
ll exgcd(ll m,ll &x,ll n,ll &y) //Extend Euclid
{
    ll x1,y1,x0,y0;
    x0=1;y0=0;
    x1=0;y1=1;
    ll r=(m%n+n)%n;
    ll q=(m-r)/n;
    x=0;y=1;
    while(r)
    {
        x=x0-q*x1;
        y=y0-q*y1;
        x0=x1;
        y0=y1;
        x1=x;y1=y;
        m=n;n=r;r=m%n;
        q=(m-r)/n;
    }
    return n;
}
int main()
{
    ll a,b,x,y,c;
    cin>>a>>b>>c;
    int A=a,B=b,C=c;
    ll M=exgcd(a,x,b,y);
    if(c%M)
        cout<<"Impossible"<<endl;
    else
    {
        cout<<"a="<<A<<" b="<<B<<" c="<<C<<endl;
        x=x*C/M;
        x=(x%(b/M)+b/M)%(b/M);
        cout<<"x="<<x;
        y=(C-x*A)/B;
        cout<<"  y="<<y<<endl;
        cout<<A*x<<"+"<<B*y<<"="<<C<<endl;
```

```
    }
    return 0;
}
```

分解质因数

```
vector<ll> sz;
for(int i=2;i<=n/i;i++){
if(n%i==0){
sz.push_back(i);
while(n%i==0) n/=i;
}
}
```

预处理

```
void getprime(){
    for(int i=2;i<N;i++){
        low[1]=1;
        if(!low[i]){
            for(int j=i;j<N;j+=i){
               low[j]=i;
            }
        }
    }
}
```

欧拉筛模板

```
bool isprime[MAXN];
int prime[MAXN];
void euler()
{
    memset(isprime, true, sizeof(isprime)); // 先全部标记为
素数
    isprime[1] = false;
    for(int i = 2; i <= n; ++i)
    {
        if(isprime[i]) prime[++cnt] = i;
        for(int j = 1; j <= cnt && i * prime[j] <= n; ++j)
        {
            isprime[i * prime[j]] = false;
            if(i % prime[j] == 0) break;
        }
    }
}
```

逆元求组合数

```
ll inv[MAXN],fac[MAXN];
```

```
inline int Inv(int x) {
    int res=1;
    int p=mod-2;
    while(p){
        if(p&1) res=ll(res)*x%mod;
        p>>=1;
        x=ll(x)*x%mod;
    }
    return res;
}
inline int C(int n,int k){
    if(n<0||k<0||k>n) return 0;
    return  ll(fac[n]*inv[k]%mod)*inv[n-k]%mod;
}
void init(){
    fac[0]=inv[0]=1;
    for(int i=1;i<MAXN;i++){
        fac[i]=ll(fac[i-1])*i%mod;
        inv[i]=Inv(fac[i]);
    }
}
```

字符串处理

Hash

```
typedef unsigned long long ULL;
ULL h[N], p[N]; // h[k]存储字符串前k个字母的哈希值, p[k]存储
P^k mod 2^64

// 初始化
p[0] = 1;
for (int i = 1; i <= n; i ++ )
{
    h[i] = h[i - 1] * P + str[i];
    p[i] = p[i - 1] * P;
}

// 计算子串 str[l ~ r] 的哈希值
ULL get(int l, int r)
{
    return h[r] - h[l - 1] * p[r - l + 1];
}
ull get_hash(char s[]){
```

```
    ull res=0;
    int len=strlen(s);
    over(i,0,len-1){
        res=res*p+(ull)s[i];
    }
    return res;
}
```

kmp

border-周期

周期与border互相转化 n-周期=其中一个边界

求简单的边界 hash验证复杂度O(n)常数很大

传递性

s的border的border 也是s的border

kmp 求所有前缀最大border

kmp预处理

```
//下标从1开始
for(int i=2,j=0;i<=m;i++){
    while(j&&str2[i]!=str2[j+1] )j=ne[j];
    if(str2[i]==str2[j+1]) j++;
    ne[i]=j;
}
//写成函数
int ne[MAXN];
string str2;
void getnext(int m){
    int j=0;
    ne[0]=0;
    for(int i=1;i<m;i++){
        while(j&&str2[i]!=str2[j]) j=ne[j-1];
        //不匹配就继续检测上一位 如果一直不匹配 j会=0;
        if(str2[i]==str2[j]) j++;
        //匹配就将这时候的j+1
        ne[i]=j;
    }
}
```

kmp寻找匹配位置

```
int kmp_p(int n,int m){
    int i,j=0;
    int p=-1;
    getnext(m);//预处理ne
    for(i=0;i<n;i++){
        while(j&&str2[j]!=str[i]) j=ne[j-1];
        if(str2[j]==str[i]) j++;
        if(j==m){
            p=i-m+1;
            break;
        }
    }
    return p;
}
```

kmp寻找匹配次数

```
int kmp(int n,int m){
    int i,j=0,res=0;
    getnext(m);
    for(i=0;i<n;i++){
        while(j&&str2[j]!=str[i]) j=ne[j-1];
        if(str2[j]==str[i]) j++;
        if(j==m) res++;
    }
    return res;
}
```

树上问题

LCA+路径记录

记录一个val数组 `val[i][j]`表示 从i点向上跳j步经过最小边权

```
int par[MAXN][20],dep[MAXN];
//par[u][i]代表点u的祖先中 深度为max(1,dep[u]-2^i)是谁
void dfs(int u,int fa){
    dep[u]=dep[fa]+1;
    par[u][0]=fa;
    for(int i=1;i<20;++i){
        par[u][i]=par[par[u][i-1]][i-1];
    }
    for(auto &v:G[u]){
        if(v==fa) continue;
        dfs(v,u);
    }
}
void getLCA(int u,int v){
```

```
    //int ans=1<<30;
    if(dep[u]<dep[v]) swap(u,v);
    int d=dep[v]-dep[u];
    for(int i=19;i>=0;++i)if(d&(1<<j){
        //ans
        if(dep[par[u][i]]>=dep[v]) u=par[u][i];
    }
    if(u==v) return u;
    for(int i=19;i>=0;i--){
        if(par[u][i]!=par[v][i]){
            u=par[u][i];
            v=par[v][i];
        }
    }
    return par[u][0];
}
```

树上差分

树上的前缀和就是子树和

点差分

$cnt[a]++, cnt[b]++, cnt[lca(a,b)]--, cnt[fa(lca(a,b))]--$

边差分

把边分给儿子节点，根节点没有边

$cnt[a]++, cnt[b]++, cnt[lca(a,b)]-=2$

只进行一次减 减完，不存在点差分的中点问题

```
Point_index[N];//储存点标记
void dfs_pre(int x,int fa){
    sum[x]+=dif[x];
    for(auto i:G[x]){
        if(i==fa) continue;
        dfs(i,x);
        sum[x]+=sum[i];
    }
    ans[Point_index]=sum[x];
}
```

dfs序

为了即时求字数和 把树拍扁 让每一颗子树都是连续

根据dfs的顺序展开

```
int L[N],R[N],tot;
//
void dfs(int x,int fa){
    L[x]=++tot;
    for(auto u:G[x]){
        if(u=fa) continue;
        dfs(x,u);
    }
    R[x]=tot;
}
```

求 u v总和

$v->根+u->根-2*lca(u,v)->根+lca(u,v)$

改变一个点 影响整个子树 维护点到根经过的点权和

修改时 把整个子树L[i]~R[i]修改

把dfs序放树状数组就可以