c++基础

1.c++ 基础部分

多组输入

```
while(cin>>a)
while(cin>>a,a!=-1)
```

结构体内函数

```
struct node{
    int a,b;
    int add(){//内部成员函数
        return a+b;
    }
}Member;
int main(){
    cin>>Member.a>>Member.b;//member为实例对象
    cout<<Member.add();
//把 struct 换成 class 变成类 不能访问私有属性 struct就够了
}
```

引用

```
int a=1;
int& goodname = a;//a变量有两个名字 可以用goodname来代替a 复制
地址
```

auto

自动选择类型

```
map<int,set<int> > mp;
map<int,set<int> > :: iterator it;
auto it = mp.begin();//上下两行等价 写起来方便
```

局部变量会屏蔽全局变量

自定义排序

```
sort(a,a+n,[](int a,int b){return a>b; });//匿名函数
struct node{//放在需要比较的结构体下面
    int l,r;
    friend bool operator < (const nose& a,const node& b){
        if(a.l!=b.l)
            return a.l<b.l;
        else
            return a.r<b.r;
    }
}
//自带比较函数
sort(a,a+n,greater<int>());
```

归并排序

```
void merge_sort(int l,int r){
    if(l==r) return ;
    int mid=l+r>>1;
    merge_sort(l,mid);
    merge_sort(mid+1,r);
    int i=l,j=mid+1,k=l;
    while(i<=mid&&j<=r){
        if(a[i]<=a[j])
            b[k++]=a[i++];
        else{
            b[k++]=a[j++];
            ans+=mid-i+1;//求逆序对
        }
    }
    while(i<=mid)  b[k++]=a[i++];
    while(j<=r) b[k++]=a[j++];
    for(int p=l;p<=r;p++) a[p]=b[p],b[p]=0;
}
```

数据生成

```
freopen("C:\\Users\\Youxinran\\Desktop\\111.txt","w",stdou
t);
freopen("C:\\Users\\Youxinran\\Desktop\\111.txt","r",stdin
);
```

2.stl

vector

```
vector<int> a[N]//构造N个vector
vector<int> a(n)//调用构造函数 构造了数据大小为n的vec
a.insert(a.begin()+k,val)//在下标为k的位置插入val
```

string

```
string sub=s.substr(k,n)//k表示从第几位开始 n表示几位 省略即全部
string s=itos(k)
int k=stoi(s)
```

queue

可以看首尾

```
queue<int> q;
q.pop(),q.push();
q.front(),q.back();
```

优先队列 priority

允许插队 即自动排序 最大放队首(o(logn))

```
priority_queue <int> q;
q.push(a);
q.pop();
q.top();
//改变顺序 小的放前面
priority_queue<int,vector<int>,greater<int> > q;
//需要重载时
priority_queue <node> //node为结构体 结构体中重载'<''
```

stack

只有一个口

```
stack<int> s;
s.top();//stack只有尾  不能看最先放进去的
```

pair

插入两个数据 用于嵌套

```
set<pair<int,int> > sz;
sz.insert(make_pair(a,b));//插入数据需要make_pair
auto i=sz.begin();
cout<<i.first<<i.second;//指针用-> ，可以在map中多重first 返回第一个值

piar<string,int> pp=make_pair(s,n);//单独使用pair
```

map/set

map里数据为pair类型 并且按第一个值升序

set就存一个值 也是升序

```
map<string,int> mp;
mp.insert(makepair(a,b));
mp[b]=b//上下都可以用
set<int,greater<int> > sz;//降序  如果要自定义同上
multimap<int,int> mm;
multiset<int> ms;//不会自动去重的
//二分查找
auto it=sz.lower_bound(a);//寻找第一个>=a的值
auto it=sz.upper_bound(a);//寻找第一个>a的值
unordered_set<int> us;//不排序的set
```

stl算法

```
count(a,a+n,x);
conut(a.begin(),a.end(),x);//统计容器中x的个数
fill(a.begin(),a.end(),x);//统一赋值
is_sorted(a.begin(),a.end(),cmp);//判断是否排列
unique(a.begin(),a.end());//去重
reverse(a.begin(),a.end());//反转容器内容
 int num[3]={1,2,3};
    do
    {
        cout<<num[0]<<" "<<num[1]<<" "<<num[2]<<endl;
    }while(next_permutation(num,num+3));  //使用函数就行全排列
```

小技巧

快速幂

原理

把次数进行分解 比如15->8+4+2+1 以倍数增加幂数

(任何一个整数都能拆分成很多个2的和)

复杂度为o(logn) 能过1e14

```
ll qpow(lla ,ll b,ll p){
    LL res=1;
    while(b){
        if(b&1) res=res*a%p;
        a=a*a%p;
        b>>=1;//除2
    }
    return res;
}
```

前缀和

某一段(L-R)数的和 即为sum[R]-sum[L-1]

乘法逆元

基本定义

对于两个数a,p,$gcd(a,p)$,则必定存在a*b=1(mod p) 此时b为a关于p的乘法逆元

费马小定理求逆元

当a,p互质 且p为质数 满足$a^p = a(modp)$

即 $a * a^(p-2) = 1(modp)$

扩展欧几里得求逆元

等价于求 ab = kp +1

即 ab-kp=1,求b

逆元应用

$a/bmodp = a * b^(p-2)$

下面为费马小定理对乘法逆元的代码

```
ll Powmod(ll a,int b){
    ll ret =1;
    while(b){
        if(b&1) ret*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return ret;//快速幂 直接调用Powmod(a,mod-2)
}
```

差分

引入：给定一个数组 有m次操作 每次给[L,R]加上一个数字 之后输出整个数组

定义数组 d d[i]=d[i]-d[i-1] d[1]=d[1];

差分数组的前缀和sum[i]=a[i] 还原成原数组 当对d[i]+1时 相当于对a[i]~a[n]都加了1

对上题解法 即在d[L]+x,在d[R+1]-x 完成了一次操作

做两次差分->加上等差数列

做三次差分->加上平方数

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

1

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

位运算

~表示取反码

取第k位置 `x&(1<<k-1)`

取出第k位 `x>>(k-1)&1`

取最低位的1 `lowbit(x)=x&(-x)` 树状数组中用

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";

}
```

二分

二分查找

时间复杂度为O(logn)

整数二分 如下

```
for(int i=1;i<=n;i++){
    cin>>a[i];
}
while(m--){
    int q;
    cin>>q;
    int l=1,r=n;
    while(l<=r){
        int mid=(l+r)/2;//找中间的位置
        if(a[mid]>=q) r=mid-1;//小于等于 就往左
        else l=mid+1;//一直往前 找到第一个小于等于q的数字
```

```
    }
   if(a[l]==q) cout<<l;
    else cout<<-1;
}
```

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;
}
```

交互类二分

```
int l=1,r=1e9;
while(l<=r){
int mid=l+r>>1,f;
cout<<mid<<"\n";
    cin>>f;
    if(f==0) break;
    else if(f==-1) l=mid+1;
    else r=mid-1;
}
```

交互类二分依旧可以用二分答案来写

例题 Problem - 1698D - Codeforces

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(a[i]-pre>=mid){
            pre = a[i];//枚举距离 如果距离大 就保留牛 更新下一个
尾
        }
        else
            cnt++;//如果距离小 就不更新尾部 相当于删去右边的点
    }
   // if(len-pre>=mid) ;
  //  else cnt++; //如果a[n]不是终点  要把最后一个去掉 不然会影
响答案
    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
```

二分函数

```
vector<int> a;
set<int> s;
int d=4;
auto it1=lower_bound(a.begin(),a.end(),d);//没找到返回end
pre(it1)//指的上一个
nex(it1)//下一个
```

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;
}
```

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)l
        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]
```

最短路存在条件

从u出发到v的路径中 不存在一个负环（环，且路过一圈路径变小）

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]=0;
        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

朴素O(n^2)

稀疏图可以用 与点的个数有关

```
for(int i=1;i<=m;i++){
    int u=0;
    for(int j=1;j<=n;j++){//手动找最小的点
        if(!vis[j]&&(u==0||d[j]<d[u])) u=j;
    }
    v[u]=1;//每个点只记录一遍
    for(auto j:adj[u]){//更新这个最小点的周围值
        d[v]=min(d[v],d[u]+w);
    }
}
```

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空间负责度高,最多500个点(int-512MB ll-1024MB)

使用滚动数组可以进行一定优化

```
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]);
        }
    }
}
```

floyd计算传递闭包

图论中的可达关系

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

总结

超过500 不用floyd

能用dij 就不用spfa 会被卡

对于负权图 可能会被卡掉

动态规划

动态规划

最长公共子序列

acm.hdu.edu.cn

```
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]);
            }
        }
    }
```

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
viod 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 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
```

线段树基础

线段树基础

树要开4*N

建树

```
int a[N];
struct node{
    int l,r;
    int sum;//也可以是max min等等
}seg[N<<2];
void up(int id){
    seg[id].sum=seg[id<<1].sum+seg[id<<1|1].sum;
    //类似状态转移方程 可以更改
}
void build(int id,int l,int r){//建立节点编号为1，维护区间是l-
r
    seg[id].l=l;
    seg[id].r=r;
    if(l==r){
        seg[id].sum=a[l];
        return;
    }
    int mid=l+r>>1;
    build(id<<1,l,mid);
    build(id<<1|1,mid+1,r);
    up(id);//等处理结束后更新节点信息
}
int main(){
    build(1,1,n);
}
```

线段树单点修改

```
void change(int id,int pos,int val){
    int mid=seg[id].l+seg[id].r>>1;
    if(seg[id].l==seg[id].r){
        seg[id].sum+=val; //添加上val的值
        return;
    }
    if(pos<=mid){//说明需要修改的点在左儿子中
        change(id<<1,pos,val);
    }
    else{//反之在右儿子中
        change(id<<1|1,pos,val);
    }
    up(id);//回溯处理
}
```

区间询问

假设 1现在搜索到的区间 被目标包含 全部都要

2不相交 就退出

3否则 递归两个儿子

```
//基础写法
int query(int id,int ql,int qr){
    int l=seg[id].l;
    int r=seg[id].r;
    if(ql>r||qr<l) return 0;//缺点 如果维护的是复杂的信息
    //无法保证返回值对信息无影响
    if(ql<=l&&r<=qr) return seg[id].sum;
    return query(id<<1,ql,qr)+query(id<<1|1,ql,qr);
}
//常用写法
int query(int id,int ql,int qr){
    int l=seg[id].l;
    int r=seg[id].r;
    if(ql<=l&&r<=qr) return seg[id].sum;
    int mid=l+r>>1;
    if(qr<=mid) return query(id<<1,ql,qr);//只在左儿子
    else if(ql>mid) return query(id<<1|1,ql,qr);//只在右儿
子
    else return query(id<<1,ql,qr)+query(id<<1|1,ql,qr);

}
```

复杂信息合并

```
struct Info {
    int
};
Info operator + (const Info &a,const Info &b){
Info c;
return c
}
Info query()
Info val;
```

区间修改

lazy-tag 先+上数字 在记录标记 如果要继续下走 要把标记下放

朴素标记代码-维护区间和为例

```
struct node{
    int l,r;
    int sum,lazy;
}seg[N<<2];
void settag(int id,int tag){
    seg[id].sum+=(seg[id].r-seg[id].l+1)*tag;
    se[id].lazy+=tag;
}
```

```
void down(int id){
    if(seg[id].lazy==0)return;
    settag(id<<1,szg[id].lazy);
    settag(id<<1|1,szg[id].lazy);
    seg[id].lazy=0;
}
void modify (int id,int ql,int qr,int val){
    int l=seg[id],l;
    int r=seg[id].r;
    if(ql>r||qr<l) return;
    if(ql<=l&&r<=qr) {
    settag(id,val);
    return;
    }
    down(id);
    int mid=l+r>>1;
    modify(id<<1,ql,qr,val);
    modify(id<<1|1,ql,qr,val);
    up(id);//更新父节点
}
int query(int id,int ql,int qr){
    int l=seg[id].l;
    int r=seg[id].r;
    if(ql<=l&&r<=qr) return seg[id].sum;
    //要先下放标记
    down(id);
    int mid=l+r>>1;
    if(qr<=mid) return query(id<<1,ql,qr);//只在左儿子
    else if(ql>mid) return query(id<<1|1,ql,qr);//只在右儿
子
    else return query(id<<1,ql,qr)+query(id<<1|1,ql,qr);

}
modify(1,ql,qr,val);
//从头开始 可以把总的父节点更新了
```

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;
        }
    }
}
```

字符串处理

Hash

双模hash

H(S[l,r])=F(r)-F(l-r)*base(r-1+1)次

```
unsign long long hsh[N],base[N]
typedef pair <int,int> pii;
const int mod1 = 1e9 + 7;
const int mod2 = 1e9 + 9;
```

```
vector<pii> pw;
pii base;
pii operator+(const pii& a, const pii& b) {
    int c1 = a.fi + b.fi, c2 = a.se + b.se;
    if (c1 >= mod1) c1 -= mod1;
    if (c2 >= mod2) c2 -= mod2;
    return { c1, c2 };
}
pii operator-(const pii& a, const pii& b) {
    int c1 = a.fi - b.fi, c2 = a.se - b.se;
    if (c1 < 0) c1 += mod1;
    if (c2 < 0) c2 += mod2;
    return { c1, c2 };
}

pii operator*(const pii& a, const pii& b) {
    return { 1LL * a.fi * b.fi % mod1, 1LL * a.se * b.se %
mod2 };
}
for(int i=1;i<=n;i++){
    pw[i]=pw[i-1]*base;
    s[i]=s[i-1]*base+mp(str[i],str[i]);
}
void init_strhash(int lim = 0) {//预处理
    pw = vector<pii>(lim + 1);
    base = { 29 % mod1, 29 % mod2 };
    pw[0] = { 1, 1 };
    for (int i = 1; i <= lim; i++) pw[i] = pw[i - 1] *
base;
}
struct str_hash {
    vector<pii> v;
    str_hash(){}
    void init(const strinf &s){//转化数字
        char ch=s[j-1];
        v[j]=v[j-1]*base+make_pair(ch,ch);
    }
    //v存储的是前缀和
    pii get(int L,int R){//处理L-R的hash 默认1-n
        return v[R]-(v[L-1]*pw[R-L+1]);
    }
}
int main(){
    vector<pii> ans;
    int n;cin>>n;
    init_strhash(n);//处理出n个base次方
    vector<sting> vs(n);
    for(int i=0;i<n;i++) cin>>vs[i];
    for(int i=0;i<n;i++){
        str_hash hs;
        hs.init(vs[i]);//处理出每位的hash
```

```
        ans.push_bask(hs.get(1,vs[i].size()));//需要整个字符串hash
    }
    //接下来是题目部分
}
```

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;
}
```

树形dp

求子树大小

先展开后处理 子节点改变父节点

```
int sum[MAXN];
void dfs_sum(int u,int fa){
    sum[u]=1;
    for(auto it:adj[u]){
        if(it==fa) continue;
        dfs_sum(it,u);
        sum[u]+=sum[it];
    }
}
```

求子树最值

同上

```
void dfs_min(int u,int fa){
     mins[u]=inf;
     for(auto it:adj[u]){
        if(it==fa) continue;
        dfs_min(it,u);
        mins[u]=min(mins[u],mins[it]);
     }
     mins[u]=min(mins[u],u);
}
```

求树的最长路径

计算向下的路径和向上的路径 要求两个路径 防止是一条路 不能加

```
ll dp[2][200005];
vector<pair<int,int>> adj[200005];
ll ans;
void dfs(int u,int fa){
    for(auto [v,w]:adj[u]){
        if(v==fa) continue;
        dfs(v,u);
        if(dp[0][v]+w>dp[0][u]){
            dp[1][u]=dp[0][u],dp[0][u]=dp[0][v]+w;
        }
        else if(dp[0][v]+w>dp[1][u]){
            dp[1][u]=dp[0][v]+w;
        }
    }
    ans=max(ans,dp[0][u]+dp[1][u]);
}
```

树的中心

中心就是某个点，到树上的其他点的最长距离最小

```
vector<pair<int,int> > adj[MAXN];
ll up1[MAXN],up2[MAXN],down[MAXN];
void dfs_down(int u,int fa){
    up1[u]=0,up2[u]=0;
    for(auto [v,w]:adj[u]){//往下寻找下面的最大值
        if(v==fa) continue;
        dfs_down(v,u);
        if(up1[v]+w>=up1[u]){//更新最大的
            up2[u]=up1[u];
            up1[u]=up1[v]+w;
        }
        else if(up1[v]+w>=up2[u]){//更新第二大的
            up2[u]=up1[v]+w;
        }
    }
}
```

```
void dfs_up(int u,int fa){//往上寻找上面（子树外）的最大值
//用父节点的值更新子节点
    for(auto [v,w]:adj[u]){
        if(v==fa) continue;
        if(w+up1[v]==up1[u]){//最大的直线在一条树上
            //用次长子树更新
            down[v]=max(up2[u]+w,down[u]+w);//取次长的子树+w与u最长的路径+w的最大值

        }
        else {
            down[v]=max(up1[u]+w,down[u]+w);
        }
        dfs_up(v,u);
    }
}

/*for(int i=1;i<=n;i++) {
        ans=min(ans,max(up1[i],down[i]));
    }*/
```

树的重心

某一个点，最大子树大小最小

1-最大子树大小不大于整个树的一半

2-一棵树最多有两个重心，且必定相邻 次数树的节点个数为偶数个

3-树的所有点到重心的距离和是最小的

4-在原树上增加或者减少一个叶子，重心不变或者多出一个

```
void dfs(int u,int fa){
    sz[u]=1,mss[u]=0;
    for(auto v:adj[u]){
        dfs(v,u);
        mss[u]=max(mss[u],sz[v]);
        sz[u]+=sz[v];
    }
    mss[u]=max(mss[u],n-sz[u]);
    if(mss[u]<=n/2) ans.push_back(u);
}
```

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