[总结] 四毛子算法

21/09/22 07:29 1241 0 1 3066 6:07 ~ 10:13

RMQ 笛卡尔树 数据结构 倍增

对于原序列的一个区间 $[L,R]$，在笛卡尔树中找到其对应位置，它们的 LCA 就是求解的答案。

2. LCA 问题的处理

在 Enler 序上解决，也就是欧拉序，也就是在欧拉序上的一个新的 RMQ 问题。

而在欧拉序上的 RMQ 求解的就不是 $value$ 的 RMQ 问题了，而是找到区间 $[L,R]$ **深度最小的点。**

四毛子算法的主要思想就基于此：**原来的不规则 RMQ 问题转化为了** ± 1 **RMQ 问题**，也就是说，相邻两个点最多变化 1，且一定变化。

3. ± 1 RMQ 问题求解

设 t 为 Enler 序列的长度，取块长 $b=\lceil\frac{\log_2 t}{2}\rceil$。对欧拉序列进行分块，使用 ST 表处理大块间的信息维护，复杂度为 $\mathrm{O}(\frac{t}{b}\log t)=\mathrm{O}(n)$。

对于那些边角块的处理，需要通过预处理达到 $\mathrm{O}(1)$ 的复杂度，由于差分数组总共只有 2^{b-1} 种，所以预处理的复杂度可以达到 $\mathrm{O}(b2^b)$，不超过 $\mathrm{O}(n)$。

具体来说，可以把每一个序列形成的规则折线状压起来，如果第 i 位是 1 表示 $i->i+1$ 折线下降了 1，反之表示序列上升了 1。

最后用常规思路处理边角块问题即可。

4. 细节

对于原序列上的区间 $[L,R]$，dfn_L 不一定小于 dfn_R。

```
const int maxn = 2e5 + 10;
const int maxb = 17;
struct node{
    int val,de,dfn;
    int son[2];
}t[maxn];
int stk[maxn],top,rt;
int n,m;
void build(){
    for(int i=1;i<=n;i++){
        while(top && t[i].val>t[stk[top]].val)t[i].son[0]=stk[top--];
        if(top)t[stk[top]].son[1]=i;
        stk[++top]=i;
    }
    rt=stk[1];//the top of the right segment
}
```

```
int idx;
node a[maxn];
void dfs(int u){
    t[u].dfn=++idx;a[idx]=t[u];
    for(int i=0;i<2;i++){
        if(t[u].son[i]){
            t[t[u].son[i]].de=t[u].de+1;
            dfs(t[u].son[i]);
            a[++idx]=t[u];
        }
    }
}
int n1,blocks[maxn],L[maxn],R[maxn];
node f[maxn][maxb+1];
node min(node a,node b){
    return (a.de<b.de)?a:b;
}
void ST_init(){
    n1=(int)(ceil(log2(idx)/2));
    for(int i=1;i<=idx;i++){
        blocks[i]=(i-1)/n1+1;
        if(!L[blocks[i]])L[blocks[i]]=i;
        R[blocks[i]]=i;
    }
    for(int i=1;i<=blocks[idx];i++){
        f[i][0]=a[L[i]];
        for(int j=L[i]+1;j<=R[i];j++)f[i][0]=min(f[i][0],a[j]);
    }
    int tmp=(int)log2(idx+0.5);
    for(int j=1;j<=tmp;j++){
        for(int i=1;i<=blocks[idx];i++){
            if(i+(1<<j-1)-1<=idx)
                f[i][j]=min(f[i][j-1],f[i+(1<<j-1)][j-1]);
        }
    }
}
int Pos[(1<<maxb)+10],Dif[maxn];
void small_init(){
    for(int i=1;i<=blocks[idx];i++){
        for(int j=L[i]+1;j<=R[i];j++){
            if(a[j].de<a[j-1].de)Dif[i]|=(1<<j-1-L[i]);
        }
    }
    for(int s=0;s<(1<<n1-1);s++){//state:b-1 bits
        int mx=0,v=0;
        for(int i=1;i<n1;i++){
            v+=(s>>i-1&1)?(-1):1;
            if(mx>v)mx=v,Pos[s]=i;
        }
    }
}
node ST_query(int l,int r){
    int c=log2(r-l+1+0.5);
    return min(f[l][c],f[r-(1<<c)+1][c]);
}
node small_query(int l,int r){
    int p=blocks[l],S=(Dif[p]>>(l-L[p])) & ((1<<r-l)-1);
    //cerr<<l<<" "<<r<<" "<<Pos[S]<<endl;//
    return a[Pos[S]+l];
}
node query(int l,int r){
    if(l>r)return query(r,l);
    if(blocks[l]==blocks[r])return small_query(l,r);
    else {
        node s=min(small_query(l,R[blocks[l]]),small_query(L[blocks[r]],r));
        if(blocks[r]-blocks[l]>1)s=min(s,ST_query(blocks[l]+1,blocks[r]-1));
        return s;
    }
}
#define read() read<int>()
int main(){
    n=read();m=read();
    for(int i=1;i<=n;i++)t[i].val=read();
    build();dfs(rt);
    ST_init();small_init();
    while(m--){
        int l=read(),r=read();
        printf("%d\n",query(t[l].dfn,t[r].dfn).val);
    }
    return 0;
}
```

__EOF__