莫队

授课人：贾亮

莫队

教学内容：

1、掌握基础莫队算法及奇偶性排序优化

2、掌握回滚莫队

3、掌握单点修改莫队

4、掌握树上莫队

教学难点：

1、回滚莫队的区间移动过程

2、单点修改莫队时间轴的移动及排序方法

3、树上莫队欧拉序列的性质

一、概述

莫队算法是由IOer莫涛提出的算法，可以解决**静态离线区间询问问题**。**莫队算法仍然是暴力处理区间询问，只是改变了处理询问的顺序**（所以它是离线算法），**就使得所有询问总时间复杂度由$O(n^2)$变为$O(n\sqrt{n})$**（不包括嵌套其他数据结构花费的时间）。

莫队算法适用性极为广泛，在不方便实现区间信息合并时，我们就应该想到使用莫队。

同时将其加以扩展，便能轻松处理树上路径询问以及支持修改操作和回滚操作。

二、基础莫队

1. 问题描述（区间计数）
2. 暴力算法$\mathbf{O(N^2)}$
3. 基础莫队算法
4. 复杂性证明
5. 莫队优化——奇偶性排序
6. 莫队适用范围

1. 问题描述（区间计数）

给定一个大小为N(N<=100,000)的数组，数组中所有元素的大小<=N。你需要回答M(M<=100,000)个查询。每个查询的形式是L，R。你需要回答在范围[L，R]中至少重复3次的数字的个数。例如：数组为{ 1，2，3，1，1，2，1，2，3，1 }（索引从0开始）

查询：L = 0，R = 4。答案= 1。在范围[L，R]中的值 = { 1，2，3，1，1 }，只有1是至少重复3次的。

查询：L = 1， R = 8。答案= 2。在范围[L，R]中的值 = { 2，3，1，1，2，1，2，3 }， 1重复3遍并且2重复3次。至少重复3次的元素数目= 2。

2. 暴力算法$O(N^2)$

对于每一个查询，从L至R循环，统计元素出现频率，报告答案。考虑M = N的情况，以下程序在最坏的情况运行在$O(N^2)$。

设a[]数组用于保存给定的数组，c[]数组是用于统计个元素出现的次数的桶：

```
for each query:
    ans = 0
    c[] = 0
    for i in {ql..qr}:
        c[a[i]]++
        if c[a[i]] == 3:
            ans++
```

2. 暴力算法$O(N^2)$

上述算法总是从ql至qr循环，现在我们对上述算法稍作修改，从上一次查询的位置调整到当前的查询的位置。

增加函数Add用于处理多统计一个元素，增加一个函数Del用于处理减少一个元素的统计，当然算法仍然运行在$O(N^2)$

```
int c[N],a[N],ans,qans[N],n,m;
struct Query{ int l,r,id;}q[N];
inline void Add(int p){ if(++c[a[p]]==3)ans++;}
inline void Del(int p){ if(--c[a[p]]==2)ans--;}
```

2. 暴力算法$O(N^2)$

```
int c[N],a[N],ans,qans[N],n,m;
struct Query{ int l,r,id;}q[N];
inline void Add(int p){ if(++c[a[p]]==3)ans++;}
inline void Del(int p){ if(--c[a[p]]==2)ans--;}
        ...
        for(int i=1,L=0,R=-1;i<=m;++i){
                while(R<q[i].r)Add(++R);
                while(L>q[i].l)Add(--L);
                while(R>q[i].r)Del(R--);
                while(L<q[i].l)Del(L++);
                qans[q[i].id]=ans;
        }
        for(int i=1;i<=m;++i)
                printf("%d\n",qans[i]);
```

3. 基础莫队算法

每个查询都有L和R，我们称呼其为“起点”和“终点”。我们将给定的数组从左往右按照每连续$[\sqrt{N}]$个元素分块，最后一块可能小于$\sqrt{N}$，总共$\frac{N}{\sqrt{N}}=\sqrt{N}$个块。

我们将查询按照“起点”所在块升序排列。如果某查询的“起点”落在第p块中，则该查询属于第p块。属于第p块的查询，我们按照“终点”由小到大依次处理。

最终的排序是怎样的？

所有询问按照“起点”所在块号为第一关键字，“终点”作为第二关键字升序排列。

3. 基础莫队算法

例如考虑如下的询问，假设我们会有3个大小为3的块（0-2,3-5,6-8）：

{0, 3} {4, 8} {1, 7} {2, 8} {7, 8} {4, 4} {1, 2}

让我们先根据“起点”所在块的编号升序排列它们

{0, 3} {1, 7} {2, 8} {1, 2} | {4, 8} {4, 4} | {7, 8}

现在我们按照“终点”升序排列

{1, 2} {0, 3} {1, 7} {2, 8} | {4, 4} {4, 8} | {7, 8}

3. 基础莫队算法

实现代码：

```
bool operator<(const Query &a, const Query &b){
        return a.l/bl^b.l/bl? a.l<b.l:a.r<b.r;
}

        ...
        bl=(int)sqrt(n);
        sort(q+1,q+m+1);
```

现在我们使用与上一节所述的暴力算法即可解决这个问题。上述算法是正确，因为我们没有做任何改变，只是重新排列了查询的顺序。显然，这是一个离线算法，且只允许有查询操作，不能有修改操作。

4. 复杂性证明

上述莫队算法，它只是一个重新排序。可怕的是它使得暴力的$O(N^2)$代码运行在$O(N\sqrt{N})$时间复杂度上。

先来讨论R的移动复杂度，属于每个块的查询R是递增的，移动量是$O(N)$的。共有$\sqrt{N}$个块，总共$O(N\sqrt{N})$。当然进入下一个块的查询，R会从最右端移动回来，复杂度仍是$O(N)$的，不影响总时间复杂度。

4. 复杂性证明

再来看看L的移动复杂度，对于每个块，所有查询的L落在同一个块中，前一个L 与当前的L移动是$O(\sqrt{N})$的，询问次数M==N，总时间复杂度$O(N\sqrt{N})$。当然进入下一个块的查询时，L从前一个块移动到下一个块，最多移动$2\sqrt{N}$次。$\sqrt{N}-1$次块间移动，共$O(N)$，同样不影响时间复杂度。

就这样，总时间复杂度为$O(N\sqrt{N})$。

5. 莫队优化——奇偶性排序

从上述复杂性证明可以看出，从一个快递查询进入下一块地查询时，R会从最右端移动回来，这增加了一倍的移动量。下面针对于这个问题，给出优化。

既是偶块的查询R升序排列，奇块的查询R降序排列，代码如下：

```
bool operator<(const Query &a, const Query &b){
  return a.l/bl^b.l/bl? a.l<b.l:a.l/bl&1? a.r>b.r:a.r<b.r;
}
```

经测试用时至少减少了1/4，这里不包含IO用时。

6. 莫队适用范围

如前所述，该算法是离线的，这意味着当题目要求查询强制在线时，我们不能使用莫队。这也意味着当有更新操作时也不能用这个算法。

不仅如此，还有一个重要的局限性：我们需要编写Add 和Del函数。会有很多的情况下，Add 是简单的，但Del不是。这样的一个例子就是我们想要求区间内最大值。当我们添加的元素，我们可以跟踪最大值。但当我们删除元素则不是那么容易了。当然我们可以维护一个multiset，用来查询最值。在这种情况下，添加和删除操作都是$O(log\ N)$，总复杂度$O(N\sqrt{N}\ log\ N)$。

三、回滚莫队

真的遇到Add简单，而Del难以实现的问题，该怎么处理呢？我们先来看一道题目：

1. 问题描述（历史研究）

JOI教授为了通过古代IOI国留下的日记来研究古代IOI国的生活，开始着手调查日记中记载的事件。

日记中记录了连续N天发生的时间，每天发生一件。

事件有种类之分。第i天($1 \leq i \leq N$)发生的事件的种类用一个整数 X_i 表示，X_i 越大，事件的规模就越大。

JOI教授决定用如下的方法分析这些日记：

1. 选择日记中连续的一些天作为分析的时间段。
2. 事件种类t的重要度为t*(这段时间内重要度为t的事件数)。
3. 计算出所有事件种类的重要度，输出其中的最大值。

现在你被要求制作一个帮助教授分析的程序，每次给出分析的区间，你需要输出重要度的最大值。

数据范围：$1 \leq N \leq 2 \times 10^5$；$1 \leq Q \leq 10^5$；$1 \leq X_i \leq 10^9$；

2. 分析

暴力算法就不说了，时间复杂度 $O(N^2)$ ，TLE。

考虑分块算法：

处理方式与区间众数类似，首先预处理f[i][j]，表示i块至j块的重要度最大值，时间复杂度 $O(N\sqrt{N})$。

对于整块的查询，答案从f 数组中获取；

对于散块的查询，我们可以先预处理vector数组，记录不同事件出现的位置，这样散块中事件对答案的影响就可以二分查找判断了。时间复杂度$O(N\sqrt{N}\ log\ N)$。

这样还是会TLE，考虑优化散块的查找，去掉二分查找：（设待查找 $[l,r]$ 区间）

我们再预处理g[i][j]，表示前i个块数字j出现的次数。

这样我们可以在$2\sqrt{N}$次内获得 $[1,l-1]$ 和 $[1,r]$ 桶的状态。也可以在$2\sqrt{N}$次内判断散块中事件对答案的影响。时间复杂度$O(N\sqrt{N})$。

2. 分析

考虑莫队算法：

这道题目Add函数，显然只需要将事件对应的桶加1，顺便维护一下答案即可；而Del函数将事件对应的桶减1可以，但答案就很难 $O(1)$ 时间获得。

我们再回顾一下没有奇偶性优化的莫队算法：

1. “终点”的移动，属于某一块的查询，r 是不断递增的不会有调用Del的情况。而进入下一个块的时候，r 会从最右边移动回来需要Del，但这不需要更新答案，只要将桶清空，下一个块的查询我们再重新处理。

2. “起点”的移动，属于某一块的查询，l 总是在块内左右移动，这必然需要用到Del函数。同样这也是可以避免的，我们可以让该块内**每个查询的 l 都从该块右端点外移动至查询位置**，当然移动前需要记录当前答案，方便下个查询的回滚。“起点”每个查询的移动同样是 $O(\sqrt{N})$ 次，复杂度没有增加。

2. 分析

我们来看一下示例，已知共16日事件类型如下，第2块内的查询有[4,10]及[6,13]，我们来看看具体的移动过程：

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
事件		1	2	1	1	2	1	1	3	1	2	1	1	1	3	3	2
								R	L								

ans: 0 backans: 0 [4,10]: [6,13]:

2. 分析

我们来看一下示例，已知共16日事件类型如下，第2块内的查询有[4,10]及[6,13]，我们来看看具体的移动过程：

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
事件		1	2	1	1	2	1	1	3	1	2	1	1	1	3	3	2
									L		R						

ans: 3　　backans: 3　　[4,10]:　　[6,13]:

2. 分析

我们来看一下示例，已知共16日事件类型如下，第2块内的查询有[4,10]及[6,13]，我们来看看具体的移动过程：

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
事件		1	2	1	1	2	1	1	3	1	2	1	1	1	3	3	2
					L						R						

ans: 4 backans: 3 [4,10]: 4 [6,13]:

2. 分析

我们来看一下示例，已知共16日事件类型如下，第2块内的查询有[4,10]及[6,13]，我们来看看具体的移动过程：

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
事件		1	2	1	1	2	1	1	3	1	2	1	1	1	3	3	2
回滚									L		R						

ans: 3 backans: 3 [4,10]: 4 [6,13]:

2. 分析

我们来看一下示例，已知共16日事件类型如下，第2块内的查询有[4,10]及[6,13]，我们来看看具体的移动过程：

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
事件		1	2	1	1	2	1	1	3	1	2	1	1	1	3	3	2
									L					R			

ans: 4 backans: 4 [4,10]: 4 [6,13]:

2. 分析

我们来看一下示例，已知共16日事件类型如下，第2块内的查询有[4,10]及[6,13]，我们来看看具体的移动过程：

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
事件		1	2	1	1	2	1	1	3	1	2	1	1	1	3	3	2
							L							R			

ans: 5 backans: 4 [4,10]: 4 [6,13]: 5

2. 分析

我们来看一下示例，已知共16日事件类型如下，第2块内的查询有[4,10]及[6,13]，我们来看看具体的移动过程：

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
事件		1	2	1	1	2	1	1	3	1	2	1	1	1	3	3	2
回滚									L					R			

ans: 4 backans: 4 [4,10]: 4 [6,13]: 5

2. 分析

3. “起点”和“终点”在同一个块内，从上面的例子我们可以看到，他没有办法解决这种问题。我们可以提前将这种询问全部暴力处理掉。每个询问时间复杂度$O(\sqrt{N})$，不会增加总时间复杂度。

3. 实现

定义所需变量、数组及函数

```
int c[N],a[N],b[N],bl,n,m,d;
LL ans,backans,qans[N];
struct Query{int l,r,id;}q[N];
bool operator<(const Query &a, const Query &b){
      return a.l/bl^b.l/bl? a.l<b.l:a.r<b.r;
}
inline void Add(int pos){
      ans=max((LL)(++c[a[pos]])*b[a[pos]],ans);
}
```

3. 实现

在主过程中首先离散化：

```
n=Read(); m=Read();
for(int i=1;i<=n;++i)a[i]=b[i]=Read();
sort(b+1,b+n+1);
d=unique(b+1,b+n+1)-b-1;
for(int i=1;i<=n;++i)
  a[i]=lower_bound(b+1,b+d+1,a[i])-b;
```

3. 实现

读入询问，并处理块内询问：

```
bl=(int)sqrt(n);
for(int i=1;i<=m;++i){
    q[i].l=Read(); q[i].r=Read(); q[i].id=i;
    if(q[i].l/bl==q[i].r/bl){
        for(int j=q[i].l;j<=q[i].r;++j) Add(j);
        qans[q[i].id]=ans; ans=0;
        for(int j=q[i].r;j>=q[i].l;--j) --c[a[j]];
        q[i].l=q[i].r=n+bl;
    }
}
```

核心代码，带回滚的莫队：

```
sort(q+1,q+m+1);
for(int i=1,L=bl,R=bl-1,bi=0;q[i].l<=n&&i<=m;++i){
    if(bi^q[i].l/bl){
        bi=q[i].l/bl; L=bi*bl+bl; R=L-1; backans=ans=0;
        memset(c,0,sizeof(c));
    }
    while(R<q[i].r)Add(++R);
    backans=ans;
    while(L>q[i].l)Add(--L);
    qans[q[i].id]=ans;
    while(L<bi*bl+bl)c[a[L++]]--;
    ans=backans;
}
```

四、带单点修改的莫队

1. 问题描述（数颜色）
2. 分析
3. 单点修改莫队的排序
4. 修改操作的更新函数
5. 主过程中的移动操作
6. 分块大小及复杂度说明

1. 问题描述（数颜色）

墨墨购买了一套N支彩色画笔（其中有些颜色可能相同），1~N摆成一排，你需要回答墨墨的提问。墨墨会像你发布如下指令：

1、 Q L R代表询问你从第L支画笔到第R支画笔中共有几种不同颜色的画笔。

2、 R P Col 把第P支画笔替换为颜色Col。

N≤50000，M≤50000；输入数据中整数均大于等于1且不超过10^6。

2. 分析

前面说过，莫队算法是离线算法，不支持修改，强制在线莫队无能为力。但是对于某些允许离线的带修改区间查询来说，莫队还是可以试试的。做法就是用莫队求得每个区间的答案后再修正修改操作的影响，变为带修莫队。

做法是把**修改操作**编号，称为**"时间戳"**，**从1开始计数**，而**查询操作的时间戳沿用之前最近的修改操作的时间戳**。**跑主算法时定义当前时间戳为t（起始t=0）**，对于每个查询操作，如果当前时间戳相对太大了，说明已进行的修改操作比要求的多，就把多改的改回来，反之往后改。只有当前区间和查询区间左右端点、时间戳均重合时，此时的答案才是本次查询的最终答案。

2. 分析

我们看看询问类 Query 的修改，还要增加修改类：

```
struct Query{int l,r,id, t ;}q[N];
struct Change{int p,c;}ch[N];
```

这样，当前区间的移动方向从四个（[l-1,r]、[l+1,r]、[l,r-1]、[l,r+1]）变成了六个（[l-1,r,t]、[l+1,r,t]、[l,r-1,t]、[l,r+1,t]、**[l,r,t-1]、[l,r,t+1]**），代码并没有增加多少。

3. 单点修改莫队的排序

在原先莫队排序的基础上，增加一个关键字时间戳，当然奇偶性优化仍然有一些效果，这里就不加了。

所有询问按照“起点”所在块号，作为第1关键字，“终点”所在块号，作为第2关键字，时间戳作为第3关键字，由小到大排序。

```
bool operator<(const Query &a, const Query &b){
    return a.l/bl^b.l/bl? a.l<b.l:
                a.r/bl^b.r/bl? a.r<b.r: a.t<b.t;
}
```

4. 修改操作的更新函数

```
inline void Add(int p){if(!c[a[p]]++)ans++;}
inline void Del(int p){if(!--c[a[p]])ans--;}
inline void Update(int t, int l, int r){
        if(l<=ch[t].p && ch[t].p<=r)Del(ch[t].p);
        swap(ch[t].c, a[ch[t].p]);
        if(l<=ch[t].p && ch[t].p<=r)Add(ch[t].p);
}
```

5. 主过程中移动操作，增加了两步

```
for(char c;(c=getchar()),m--;){
        if(c=='Q'){
                q[Q].l=Read(); q[Q].r=Read();
                q[Q].t=T; q[Q].id=Q; ++Q;
        }
        if(c=='R')
                ch[++T].p=Read(), ch[T].c=Read();
}
bl=int(pow(n,2.0/3));
sort(q,q+Q);
```

5. 主过程中移动操作，增加了两步

```
for(int i=0,L=1,R=0,t=0;i<Q;++i){
        while(R<q[i].r)Add(++R);
        while(L>q[i].l)Add(--L);
        while(R>q[i].r)Del(R--);
        while(L<q[i].l)Del(L++);
        while(t<q[i].t)Update(++t,L,R);
        while(t>q[i].t)Update(t--,L,R);
        qans[q[i].id]=ans;
}
for(int i=0;i<Q;++i)
        printf("%d\n",qans[i]);
```

6. 分块大小及复杂度说明

（用 B 表示分块大小，c 表示修改个数，q 表示询问个数，l 块表示以 $\frac{l}{B}$ 分的块，r 块表示以 $\frac{r}{B}$ 分的块）

1. 对于时间指针 now：对于每个 r 块，最坏情况下会移动 c，共有 $\left(\frac{n}{B}\right)^2$ 个 r 块，所以总移动次数为 $\frac{cn^2}{B^2}$。

2. 对于左端点指针 l：l 块内移动每次最多 B，换 l 块每次最多 $2B$，所以总移动次数为 $O(qB)$。

3. 对于右端点指针 r：r 块内移动每次最多 B，换 r 块每次最多 $2B$，所有 l 块内移动次数之和为 $O(qB)$；换 l 块时最多移动 n，总的换 l 块时移动次数为 $O(\frac{n^2}{B})$；所以总的移动次数为$O(qB+\frac{n^2}{B})$。

6. 分块大小及复杂度说明

所以：总移动次数为 $O(\frac{cn^2}{B^2}+qB+\frac{n^2}{B})$，由于一般的题目都不会告诉你修改和询问分别的个数，所以统一用 m 表示，即$O(\frac{mn^2}{B^2}+mB+\frac{n^2}{B})$

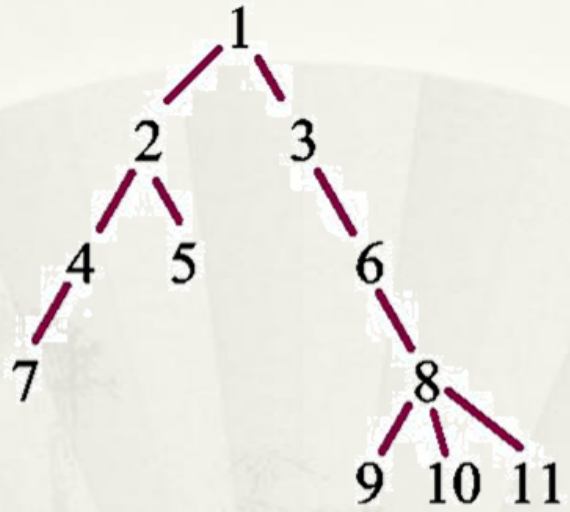
那么 B 取多少呢？当 $n=m$ 的话，就可以得到总移动次数为 $O(\frac{n^3}{B^2}+mB+\frac{n^2}{B})$，那么 $B=n^{\frac{2}{3}}$ 时取最小值$O(n^{\frac{5}{3}})$。

五、树上莫队

前面我们所使用的莫队都是在一维的序列上进行。那么树上两点间简单路径的统计类问题能否用莫队来处理呢？这只需要将树转变为序列。

同学们或许已经想到树链刨分，它可以将树上两点间简单路径转换为 $\log N$ 个区间，当然时间复杂度也会增加 $\log N$。这不失为一个较好的解决办法。有没有不增加时间复杂度的方法呢？

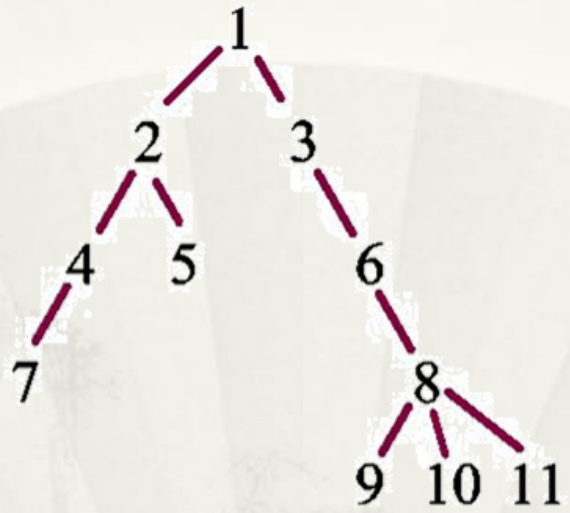
1. 欧拉序列



欧拉序：1 2 4 7 7 4 5 5 2 3 6 8 9 9 10 10 11 11 8 6 3 1

这样的欧拉序列，又称为括号序，是DFS序的一种。它可以在DFS每进入和退出一个节点时输出得到，同时打上时间戳，方便我们将树上两点间简单路径问题，转换为一个区间问题。

1. 欧拉序列

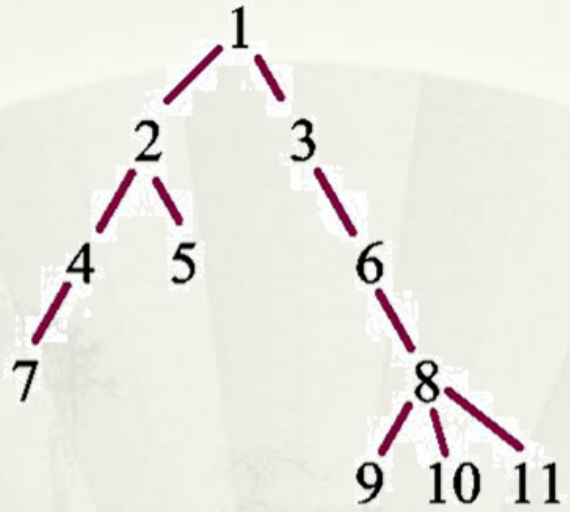


欧拉序：1 2 4 7 7 4 5 5 2 3 6 8 9 9 10 10 11 11 8 6 3 1

我们定义数组first[i]和last[i]，分别表示i点进入和退出时的时间戳。

性质1：欧拉序上两个相同编号x之间的所有编号都出现两次，且都位于x子树上。

1. 欧拉序列

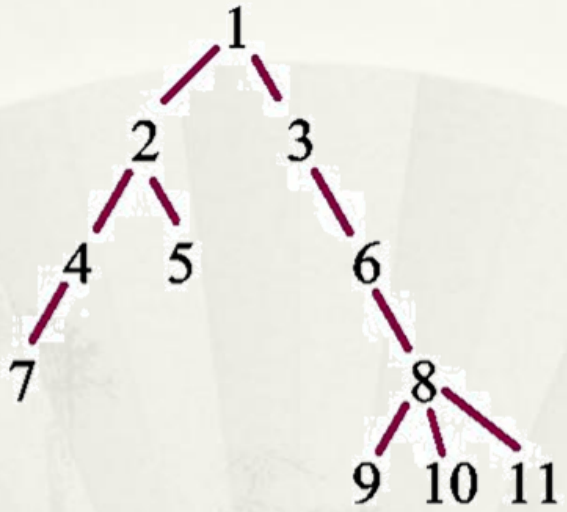


欧拉序：1 2 4 7 7 4 5 5 2 3 6 8 9 9 10 10 11 11 8 6 3 1

我们定义数组first[i]和last[i]，分别表示i点进入和退出时的时间戳。

性质2：lca(x,y)不等于x或y，设first[x]≤first[y]（否则交换x、y），x和y两点在树上简单路径上的点，既是[last[x],first[y]]中出现一次的点，除lca(x,y)外。

1. 欧拉序列



欧拉序：1 2 4 7 7 4 5 5 2 3 6 8 9 9 10 10 11 11 8 6 3 1

我们定义数组first[i]和last[i]，分别表示i点进入和退出时的时间戳。

性质3：lca(x,y)等于x或y，设lca(x,y)==x（否则交换x、y），x和y两点在树上简单路径上的点，既是[first[x],first[y]]中出现一次的点。

2. 树上莫队

有了上一节的性质，我们可以方便的将树上两点简单路径问题转换为序列区间问题，直接使用莫队即可。值得注意的是，**由于无需考虑的点会出现两次**，需要**定义标记数组，没访问就加，访问过就删，每次操作把标记异或1**即可。

对于**统计**树上两点x和y简单**路径**上**信息**的问题。由于边上的信息一般会保存在边下方节点中，所以统计信息时，**不能加入lca(x,y)**中的信息。那么对于性质2的情况直接统计[last[x],first[y]]即可，而对于性质3的情况区间需要改为[first[x]+1,first[y]]。

对于**统计**树上两点x和y简单路径上**节点信息**的问题。对于性质2的情况在统计[last[x],first[y]]后，**还需统计lca(x,y)**中的信息，而对于性质3的情况直接统计[first[x],first[y]]，即可。

谢　谢