FZOI

信息学

树链剖分

树 → 链

树链剖分

目的

- 维护**静态**树上路径信息。
- 将**树**剖分成若干条**链**。
- 每条链看做一个序列，
  树上的操作可以转换为序列上的区间操作，再使用线段树等解决。

盲目剖分

随机剖分

启发式剖分

在随机数据的情况下，三者的实际速度并没有太大差别，但如果数据有一定特征，那么前两个方法就会逊色不少。所以启发式剖分就是我们最常用的树链剖分方式。

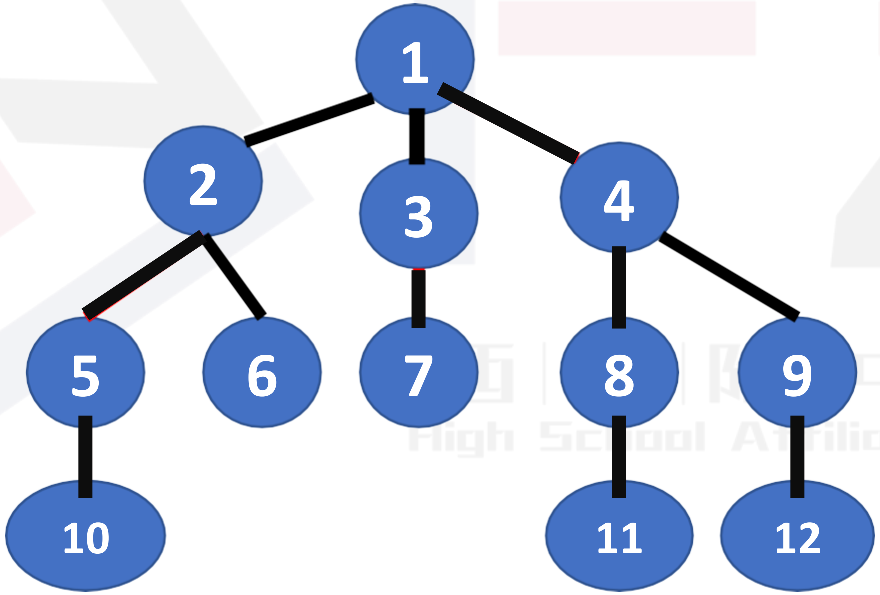
轻边与重边

定义size(X)为以X为根的子树的节点个数。

令V为U的儿子节点中size值最大的节点， 那么边(U,V)被称为重边，

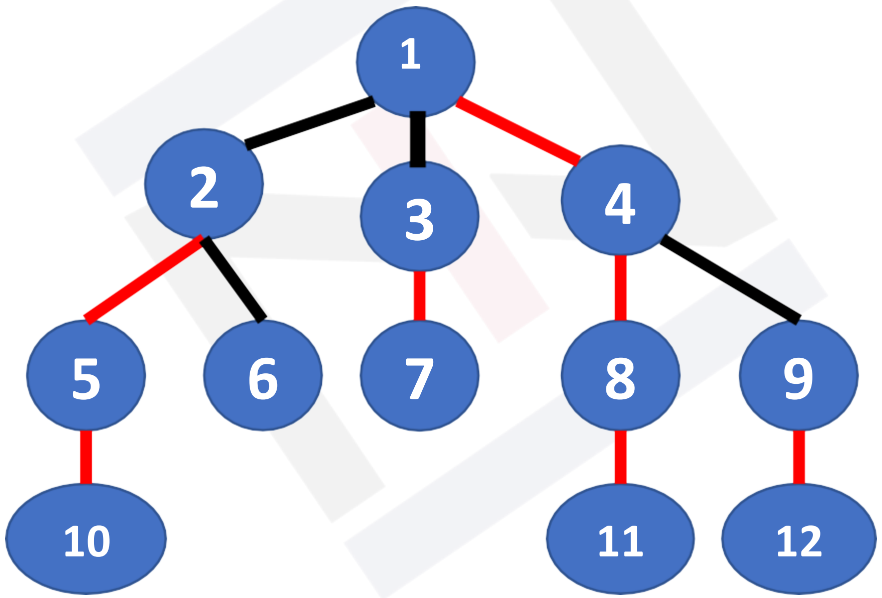
树中重边之外的边被称为轻边。

如果存在相同 的size[]，在相同的孩子里任选一个。



重路径（重链）

- 当某条路径全部由重边组成，称这条路径为重路径(重链)。
- 特殊的，一个点也算一条重路径。



重路径：

① 1 4 8 11
② 9 12
③ 3 7
④ 2 5 10
⑤ 6

轻边(U,V)，size(V)<=size(U)/2。
从根到某一点的路径上，
不超过O(logN)条轻边，
不超过O(logN)条重路径。

我们考虑用数据结构来维护重路径上的查询，轻边直接查询

每个点在且只在一条重路径上

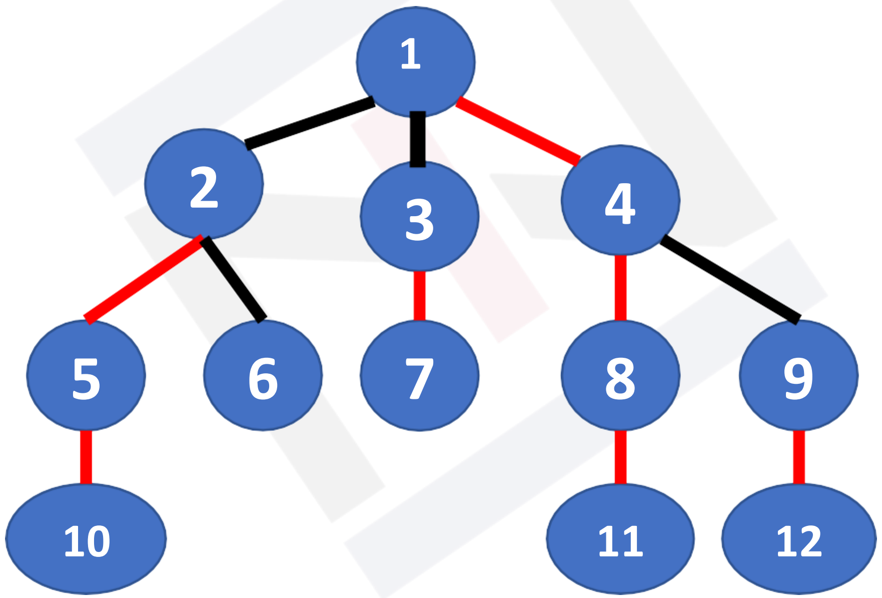
轻边两端点一定在某两条重路径上

因此，**路径上(u,v)的操作，可以看作是重链上的操作**

重路径（重链）

➢ 当某条路径全部由重边组成，称这条路径为重路径(重链)。

➢ 特殊的，一个点也算一条重路径。



重路径：

① 1 4 8 11
② 9 12
③ 3 7
④ 2 5 10
⑤ 6

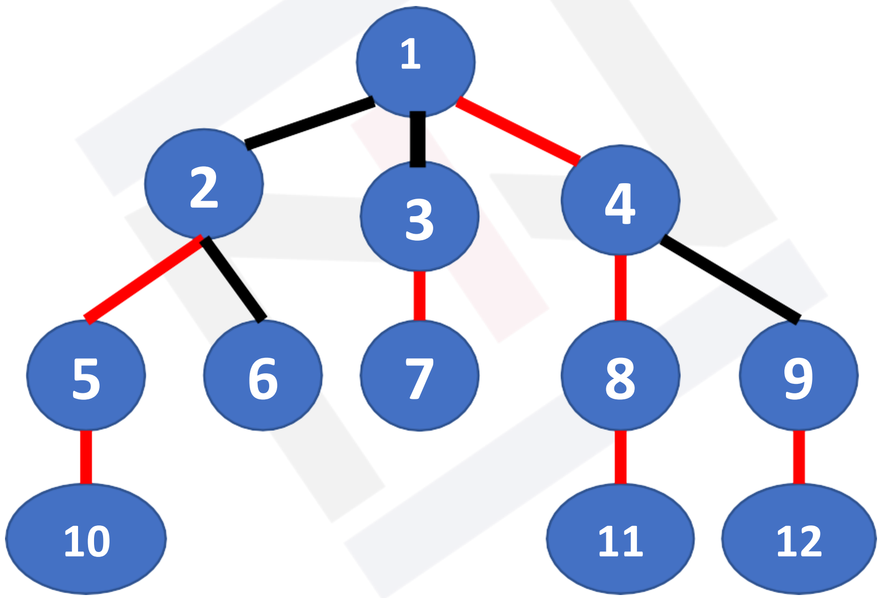
轻边(U,V)，size(V)<=size(U)/2。
从根到某一点的路径上，
不超过O(logN)条轻边，
不超过O(logN)条重路径。

例如对路径（6，12）操作：

看作对重路径6、2、1->4、9->12进行操作

重路径（重链）

- 当某条路径全部由重边组成，称这条路径为重路径(重链)。
- 特殊的，一个点也算一条重路径。



重路径：

① 1 4 8 11
② 9 12
③ 3 7
④ 2 5 10
⑤ 6

轻边(U,V)，size(V)<=size(U)/2。
从根到某一点的路径上，
不超过O(logN)条轻边，
不超过O(logN)条重路径。

将重路径看做一个完整的序列：

1 4 8 11 9 12 3 7 2 5 10 6

如果将重路径看作一个完整的序列，路径的操作就是序列的区间操作，使用**线段树**维护。

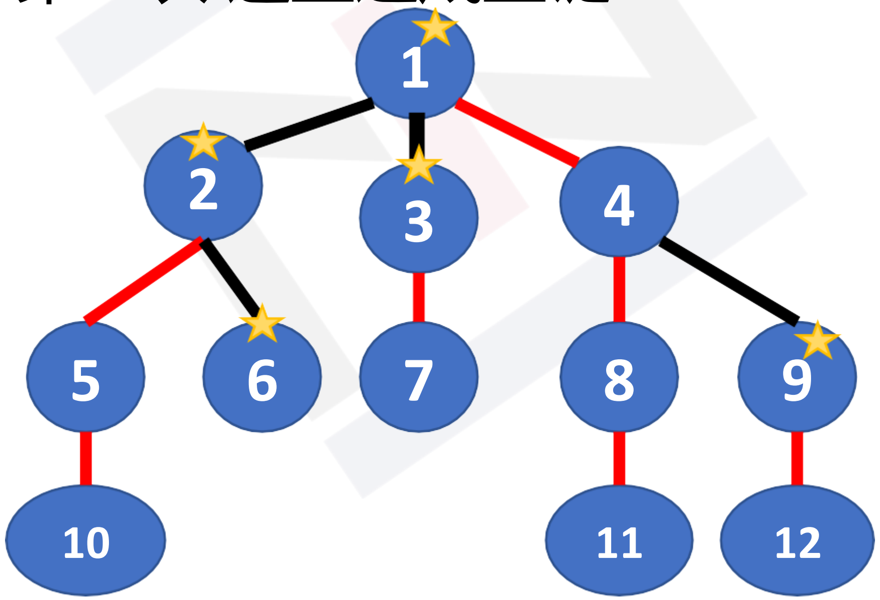
实现

重链剖分的过程为

2次DFS

第一次:找重边（重儿子)

第二次:连重边成重链



father[x]	x在树中的父亲
deep[x]	x在树中的深度
size[x]	x的子树节点总数
son[x]	x的重儿子
top[x]	x所在重链的顶部节点(深度最小的节点) ☆标记的就是该重链节点的top[]
id[x]	x在线段树中序列的下标
rev[x]	线段树中序列下标为x对应的树上节点编号，rev[id[x]]=x

➢ 第一次DFS求解：

father[x]
deep[x]
size[x]
son[x]

```
void DFS1(int u,int dad)
{
    size[u]=1;  //本身结点数为1
    father[u]=dad;
    deep[u]=deep[dad]+1;
    for(int i=head[u];i!=-1;i=nxt[i])
    {
        int v=to[i];
        if(v==dad) continue;
        DFS1(v,u);
        size[u]+=size[v];                //计算孩子节点总数
        if(size[v]>size[son[u]]) son[u]=v; //找重儿子
    }
}
```

第二次DFS

- 第二次DFS,让同一条重路径上的点按顺序排在连续的一段位置，同一子树在连续的一段位置（DFS序）：

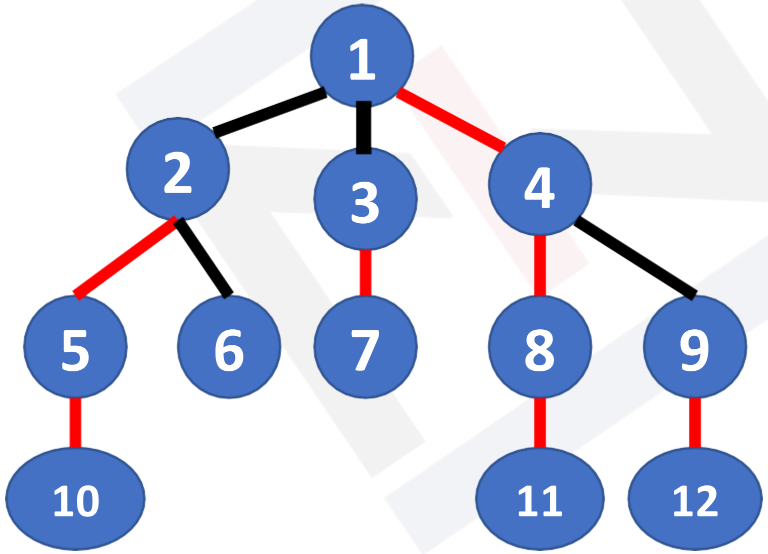
top[x]
id[x]
rev[x]

以根节点为起点，沿重边向下拓展，拉成重链。不在当前重链上的节点，都以该节点为起点向下重新拉一条重链。

```
//根节点无法在DFS2赋值，在主函数中赋值
DFS1(1,0);
id[1]=++t; //初始化根节点
top[1]=1;
rev[1]=1;
DFS2(1,0);
```

```
void DFS2(int u,int dad)
{
   if(son[u])              //优先选择重儿子
   {
      int v=son[u];
      id[v]=++t;           //dfs序列
      top[v]=top[u];
      rev[t]=v;
      DFS2(v,u);
   }
   for(int i=head[u];i!=-1;i=nxt[i])
   {
      int v=to[i];
      if(!top[v])
      {
         id[v]=++t;
         top[v]=v;
         rev[t]=v;
         DFS2(v,u);
      }
   }
}
```

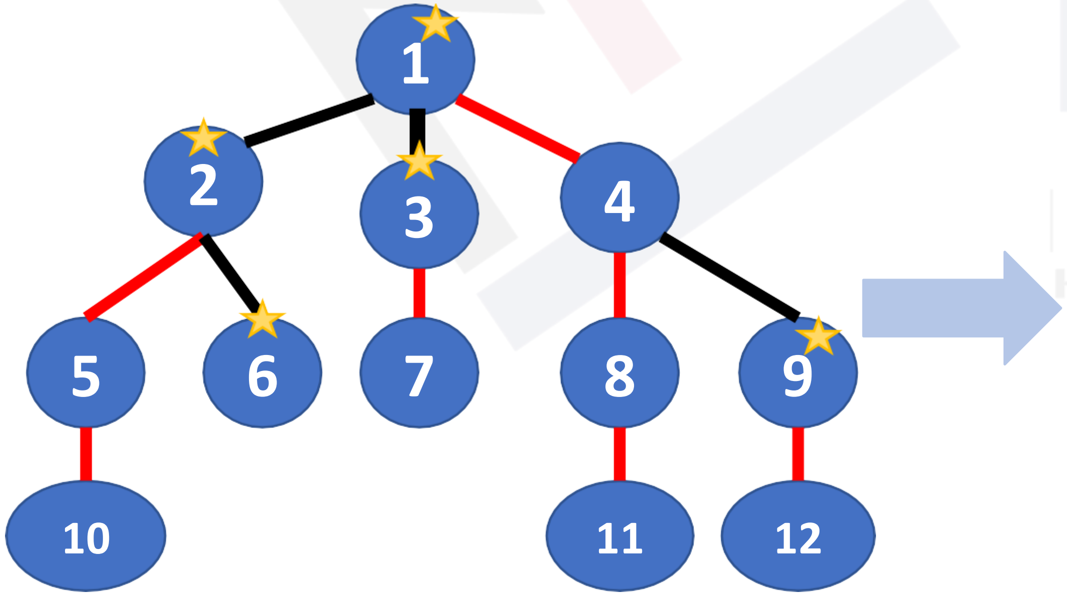
➢树通过剖分成多条重链，将每条重链看作一个序列，多条重链首尾相连看作一个完整的序列：



线段树维护序列下标id	1	2	3	4	5	6	7	8	9	10	11	12
树上节点编号rev	1	4	8	11	9	12	3	7	2	5	10	6

维护

- **单点修改**：将节点x增加s。
- 修改线段树维护序列下标 id[x]+=s，线段树单点修改。
- 时间复杂度：O(logN)



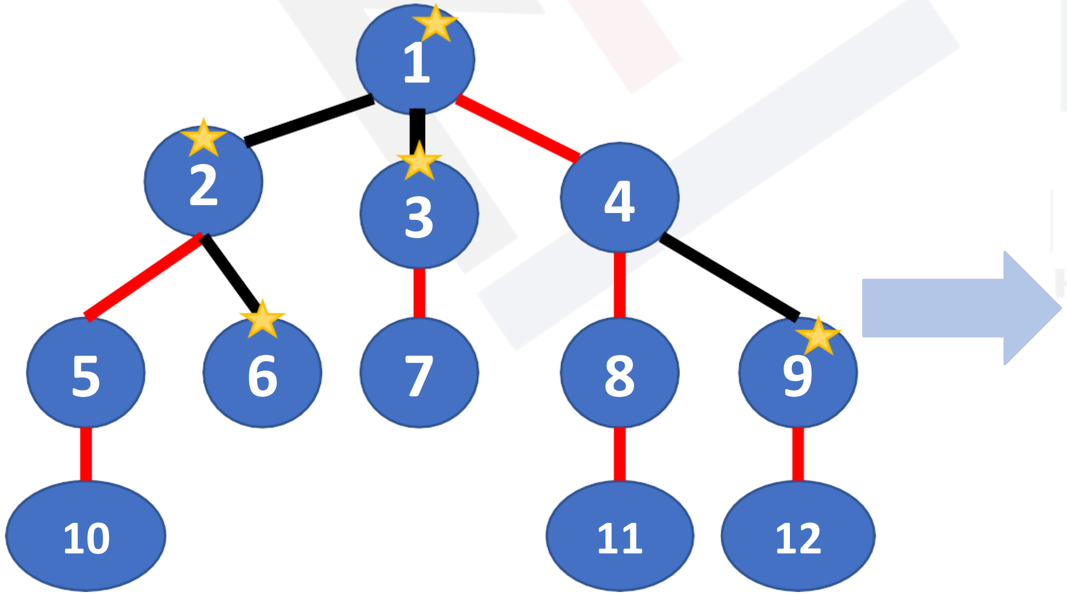
线段树维护序列下标id	1	2	3	4	5	6	7	8	9	10	11	12
树上节点编号rev	1	4	8	11	9	12	3	7	2	5	10	6

维护

- **路径修改**：将x到y的路径上所有节点增加s。
- 修改方法：
- 如果x和y在同一条重链上

  直接用数据结构修改x至y]间的值
- 如果x和y不在同一条重链上

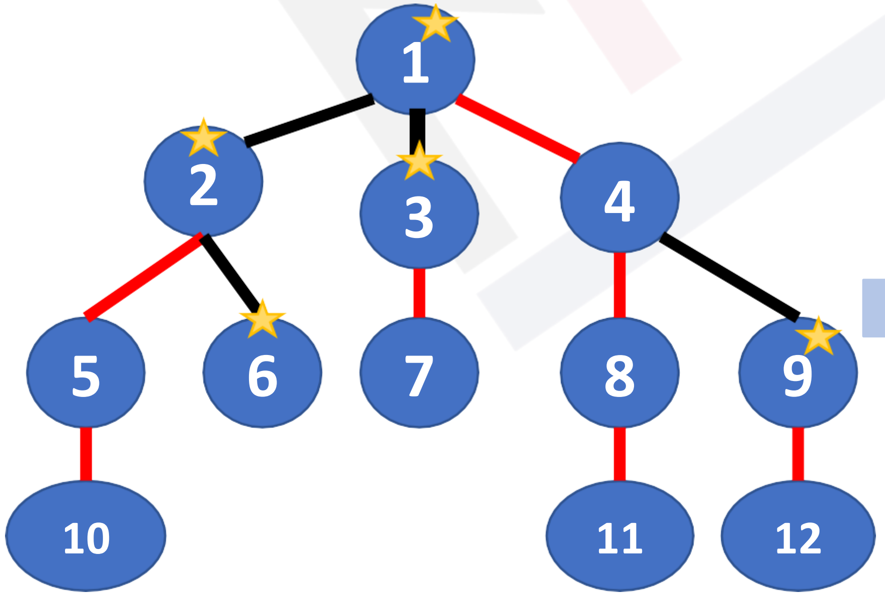
  一边进行修改，一边将x和y往同一条重链上靠



线段树维护序列下标id	1	2	3	4	5	6	7	8	9	10	11	12
树上节点编号rev	1	4	8	11	9	12	3	7	2	5	10	6

维护

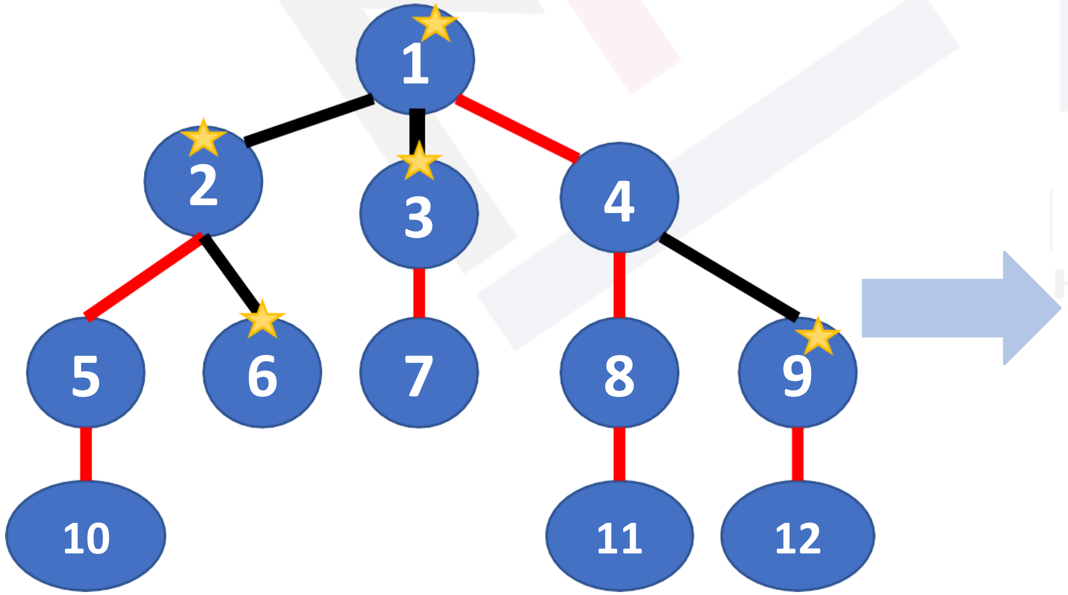
- **路径修改**：将x到y的路径上所有节点增加s。
- 修改方法：x和y同时往LCA跳，经过的路径增加s.
- 使用top[]求解LCA。(top[x]为x所在重链的顶部节点)
- （1）top[x]与top[y]不同时，top深度大的节点跳到top位置，假如top[x]深度大，x=top[x];同时再跳到父亲节点所在的重链上，x=father[x]；
- （2）继续比较top[x]是否与top[y]相同，如果top[x]==top[y]，x和y深度小的为LCA，否则重复执行(1)。
- 因为重路径顶部结点朝祖先节点方向，与寻找LCA的方向相同。



线段树维护序列下标id	1	2	3	4	5	6	7	8	9	10	11	12
树上节点编号rev	1	4	8	11	9	12	3	7	2	5	10	6

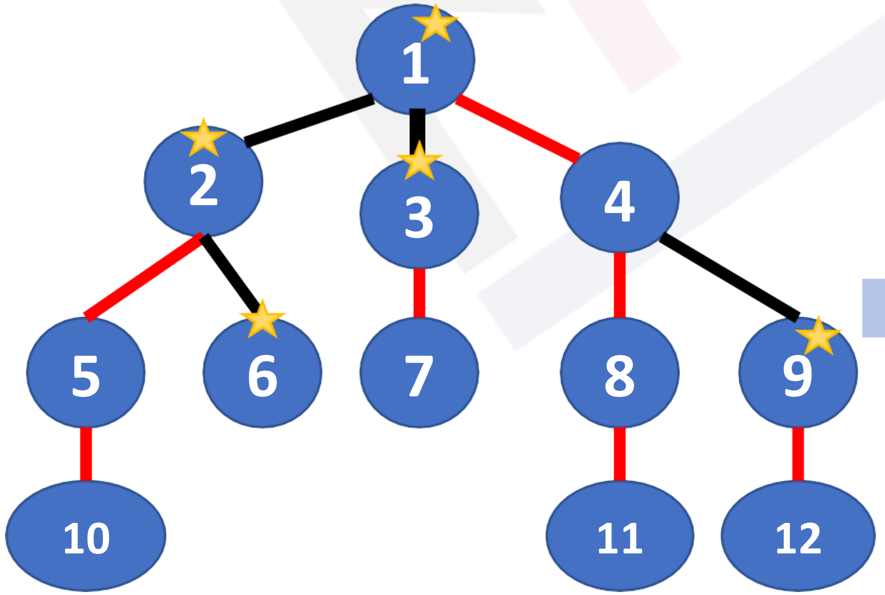
维护

- **路径修改**：将x到y的路径上所有节点增加s。
- 当节点x跳到top[x]位置，那么需要修改x到top[x]路径节点，
- 对应线段树下标 id[top[x]] ~ id[x] 区间
- 当top[x]==top[y]时，x和y在同一条重链，修改区间id[x]~id[y](x深度<y深度)



线段树维护序列下标id	1	2	3	4	5	6	7	8	9	10	11	12
树上节点编号rev	1	4	8	11	9	12	3	7	2	5	10	6

- **路径修改**：将x到y的路径上所有节点增加s。
- 例如修改树上路径(10,8)
- 节点10跳到top[10]=2，修改路径id[2]~id[10]，即线段树区间[9,11]
- 节点2跳到节点1，此时top[1]==top[8]，修改路径id[1]~id[8]，即线段树区间[1,3]



线段树维护序列下标id	1	2	3	4	5	6	7	8	9	10	11	12
树上节点编号rev	1	4	8	11	9	12	3	7	2	5	10	6

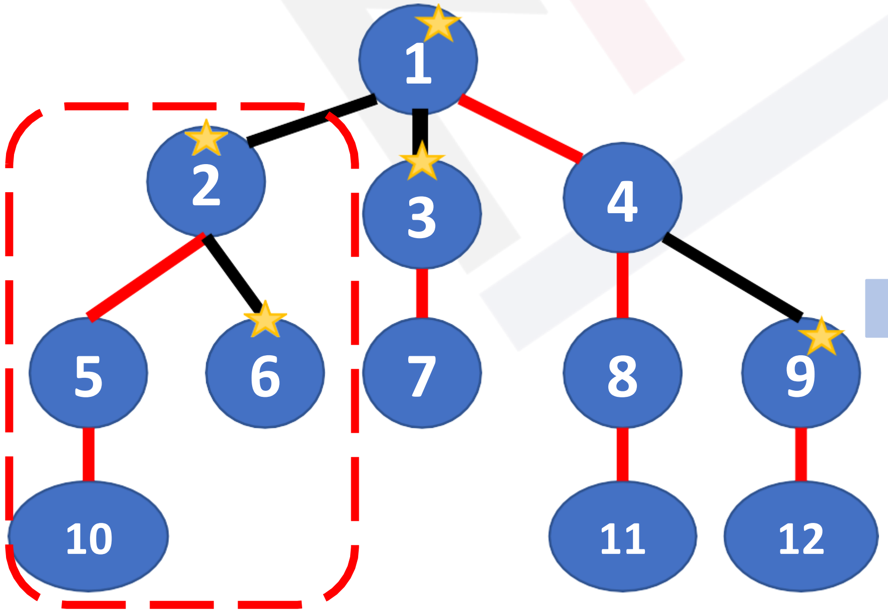
- **路径修改**：将x到y的路径上所有节点增加s。
- 线段树**多段区间修改**
- 时间复杂度：$O(\log^2 N)$

```
void update(int x,int y,int s)
{
    int fx=top[x],fy=top[y];
    while(fx!=fy)                              //不在同一条重链上
    {
        if(deep[fx]<deep[fy]) {swap(x,y);swap(fx,fy);}   //选择深度大的往上跳
        change(1,1,n,id[fx],id[x],s);                    //修改树上路径 fx->x
        x=father[fx];
        fx=top[x];
    }
    if(deep[x]>deep[y]) swap(x,y);   //已经跳到同一条重路径上了
    change(1,1,n,id[x],id[y],s);     // 修改树上路径 x->y
}
```

维护

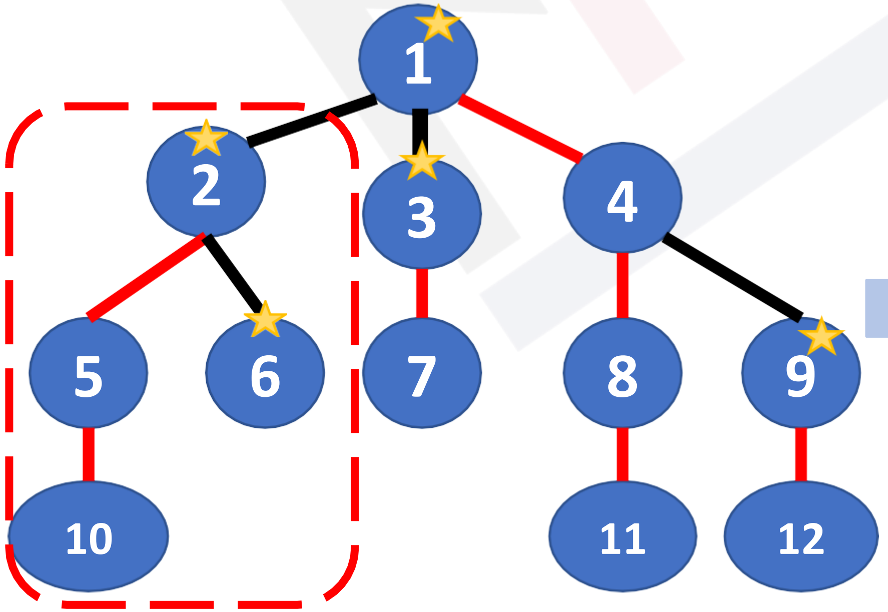
- **路径询问**：求x到y的路径的和，最大值。
- 线段树**多段区间询问**
- 时间复杂度：$O(\log^2 N)$

- **子树修改**：将x为根的子树全部增加s。
- 对于DFS序，子树在连续的位置，应该相当于：
- 区间修改：[id[x],id[x]+size[x]-1]



线段树维护序列下标id	1	2	3	4	5	6	7	8	9	10	11	12
树上节点编号rev	1	4	8	11	9	12	3	7	2	5	10	6

- **询问子树**：求x为根的子树权值和。
- 区间询问：[id[x],id[x]+size[x]-1]



线段树维护序列下标id	1	2	3	4	5	6	7	8	9	10	11	12
树上节点编号rev	1	4	8	11	9	12	3	7	2	5	10	6

- 树链剖分解决**静态树**上路径信息
- 原理是将树剖分成多条链，将链看作序列，使用线段树进行维护。
- 常见操作：
  - 树上单点修改，线段树单点修改
  - 树上路径修改，线段树多区间修改
  - 树上子树修改，线段树区间修改
  - 树上路径询问，线段树多区间询问
  - 树上子树询问，线段树区间询问
- 树链剖分在信息学竞赛中越来越多的被考 到，也是解题很有用的工具。
- 平时多写写，容易有感觉，提高正确率。

[ZJOI2008]树的统计

问题描述:

一棵树上有n个节点，编号分别为1到n，每个节点都有一个权值w。

我们将以下面的形式来要求你对这棵树完成一些操作：

1. CHANGE u t : 把结点u的权值改为t
2. QMAX u v: 询问从点u到点v的路径上的节点的最大权值
3. QSUM u v: 询问从点u到点v的路径上的节点的权值和

$1 \le n \le 30000, 1 \le q \le 200000$

[ZJOI2008]树的统计

西南大学附属中学 High School Affiliated to Southwest University

问题分析：

```cpp
#include<bits/stdc++.h>
#define N 200100
using namespace std;
int head[N],nxt[N],to[N],w[N],tot;
int n,m;
int father[N],deep[N],size[N];//父节点，深度，子树结点数
int son[N],top[N];//重儿子，所在重路径的顶部结点(深度最小的结点)
int id[N],rev[N],t;//在线段树中的下标(dfs序)，线段树中下标的结点,即rev[id[u]]=u ,dfs序号
int sum[2*N],Max[2*N]; //线段树
int ans_sum,ans_max;
void Add(int u,int v)
{
    nxt[++tot]=head[u];
    to[tot]=v;
    head[u]=tot;
}
void dfs1(int u,int dad)
{
    size[u]=1;  //本身结点数为1
    father[u]=dad;
    deep[u]=deep[dad]+1;
    for(int i=head[u];i!=-1;i=nxt[i])
    {
        int v=to[i];
        if(v!=dad)
        {
            dfs1(v,u);
            size[u]+=size[v];
            if(size[v]>size[son[u]]) son[u]=v;//找重儿子
        }
    }
}
```

```cpp
void dfs2(int u,int dad)
{
    int v=son[u];
    if(v)                          //优先选择重儿子
    {
        id[v]=++t;                 //dfs序列
        top[v]=top[u];
        rev[t]=v;
        dfs2(v,u);
    }
    for(int i=head[u];i!=-1;i=nxt[i])
    {
        v=to[i];
        if(!top[v])
        {
            id[v]=++t;
            top[v]=v;
            rev[t]=v;
            dfs2(v,u);
        }
    }
}
```

[ZJOI2008]树的统计

问题分析：

```
void built(int k,int l,int r)   //线段树建树
{
    if(l==r)    {Max[k]=sum[k]=w[rev[l]];return;}   //初始化
    int mid=(l+r)/2;
    built(2*k,l,mid);
    built(2*k+1,mid+1,r);
    sum[k]=sum[2*k]+sum[2*k+1];
    Max[k]=max(Max[2*k],Max[2*k+1]);
}
void change(int k,int l,int r,int x,int v)  //单点修改
{
    if(l>r) return;
    if(l==r&&x==l)  {sum[k]=Max[k]=v;return;}
    int mid=(l+r)/2;
    if(x>=l&&x<=mid) change(2*k,l,mid,x,v);
    if(x>=mid+1&&x<=r) change(2*k+1,mid+1,r,x,v);
    sum[k]=sum[2*k]+sum[2*k+1];
    Max[k]=max(Max[2*k],Max[2*k+1]);
}
void query(int k,int l,int r,int x,int y)
{
    if(x>r||y<l) return ;
    if(x<=l&&r<=y)
    {
        ans_sum+=sum[k];
        ans_max=max(ans_max,Max[k]);
        return;
    }
    int mid=(l+r)/2;
    if(x<=mid) query(2*k,l,mid,x,y);
    if(y>=mid+1) query(2*k+1,mid+1,r,x,y);
}
```

```
void ask(int u,int v)
{
    int fu=top[u],fv=top[v];
    while(fu!=fv)   //不在同一条重链上
    {
        if(deep[fu]<deep[fv]) {swap(u,v);swap(fu,fv);}   //选择深度大的往上跳
        query(1,1,n,id[fu],id[u]);  //访问路径 fu->u
        u=father[fu];
        fu=top[u];
    }
    if(deep[u]>deep[v]) swap(u,v);   //已经跳到同一条重路径上了
    query(1,1,n,id[u],id[v]);
}
```

[ZJOI2008]树的统计

问题分析：

```
int main()
{
    memset(head,-1,sizeof(head));
    scanf("%d",&n);
    int x,y;
    for(int i=1;i<n;i++)
    {
        scanf("%d%d",&x,&y);
        Add(x,y);
        Add(y,x);
    }
    for(int i=1;i<=n;i++)
        scanf("%d",&w[i]);
    scanf("%d",&m);
    dfs1(1,0);
    id[1]=++t;  //初始化根节点
    top[1]=1;
    rev[1]=1;
    dfs2(1,0);
    built(1,1,n);
    char s[10];
    for(int i=1;i<=m;i++)
    {
        scanf("%s%d%d",s,&x,&y);
        if(s[0]=='C')   change(1,1,n,id[x],y);
        else
        {
            ans_sum=0;
            ans_max=0x80000000;     //int最小值
            ask(x,y);               //路径询问
            if(s[0]=='Q'&&s[1]=='M')    printf("%d\n",ans_max);
            if(s[0]=='Q'&&s[1]=='S')    printf("%d\n",ans_sum);
        }

    }
    return 0;
}
```