

树链剖分详解

转载请注明出处，部分内容引自banananana大神的博客

别说你不知道什么是树\(\smile\)(帮你百度一下)

前置知识：dfs序 线段树

先来回顾两个问题：

1，将树从x到y结点最短路径上所有节点的值都加上z

这也是个模板题了吧

我们很容易想到，树上差分可以以 $O(n+m)$ 的优秀复杂度解决这个问题

2，求树从x到y结点最短路径上所有节点的值之和

lca大水题，我们又很容易地想到，dfs $O(n)$ 预处理每个节点的dis（即到根节点的最短路径长度）

然后对于每个询问，求出x,y两点的lca，利用lca的性质 $distance(x,y) = dis(x) + dis(y) - 2 * dis(lca)$ 求出结果

时间复杂度 $O(m\log n+n)$

现在来思考一个bug：

如果刚才的两个问题结合起来，成为一道题的两种操作呢？

刚才的方法显然就不够优秀了（每次询问之前要跑dfs更新dis）

树链剖分华丽登场

树剖是通过轻重边剖分将树分割成多条链，然后利用数据结构来维护这些链（本质上是一种优化暴力）

首先明确概念：

重儿子：父亲节点的所有儿子中子树结点数目最多（size最大）的结点；

轻儿子：父亲节点中除了重儿子以外的儿子；

重边：父亲结点和重儿子连成的边；

轻边：父亲结点和轻儿子连成的边；

重链：由多条重边连接而成的路径；

轻链：由多条轻边连接而成的路径；

公告

友情链接：FFT杨 | T神犇
张神犇

昵称：ivanovcraft
园龄：1年1个月
粉丝：16
关注：7
+加关注

| 2019年6月 | | | | | | |
|---------|----|----|----|----|----|----|
| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
| 26 | 27 | 28 | 29 | 30 | 31 | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 1 | 2 | 3 | 4 | 5 | 6 |

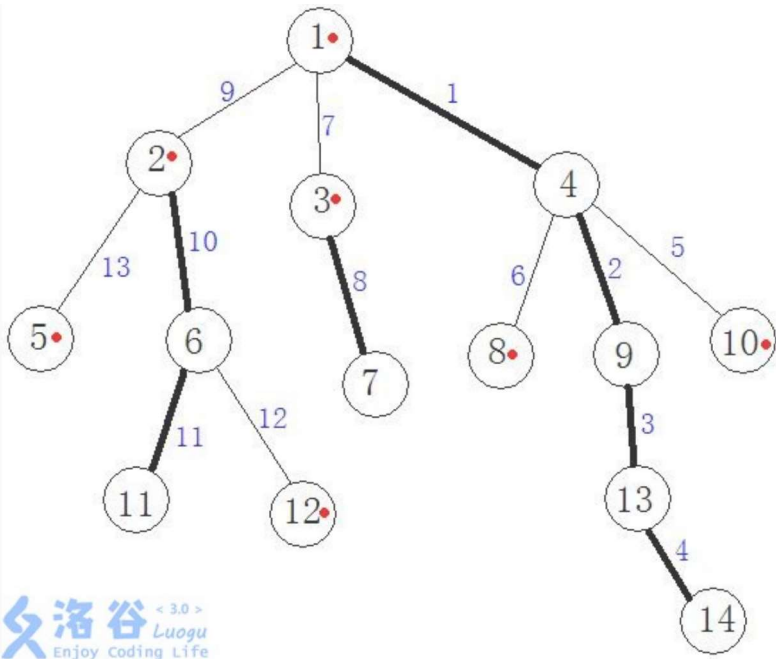
搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

bfs广度优先搜索(2)
cdq分治(2)
dfs深度优先搜索(1)
DP(4)
LCA(1)
NOIP(2)
STL(4)
背包(1)
二分答案(1)
记忆化搜索(1)
链表(3)
模拟(2)
莫队(1)
奇巧(1)
省选(2)
树链剖分(2)



比如上面这幅图中，用黑线连接的结点都是重结点，其余均是轻结点，
2-11就是重链，2-5就是轻链，用红点标记的就是该结点所在重链的起点，也就是下文提到的top结点，
还有每条边的值其实是进行dfs时的执行序号。

变量声明：

```
const int maxn=1e5+10;
struct edge{
    int next,to;
}e[2*maxn];
struct Node{
    int sum,lazy,l,r,ls,rs;
}node[2*maxn];
int
rt,n,m,r,a[maxn],cnt,head[maxn],f[maxn],d[maxn],size[maxn],son[maxn],rk[maxn],top[maxn],id[ma
xn];
```

| 名称 | 解释 |
|---------|----------------------------|
| f[u] | 保存结点u的父亲节点 |
| d[u] | 保存结点u的深度值 |
| size[u] | 保存以u为根的子树节点个数 |
| son[u] | 保存重儿子 |
| rk[u] | 保存当前dfs标号在树中所对应的节点 |
| top[u] | 保存当前节点所在链的顶端节点 |
| id[u] | 保存树中每个节点剖分以后的新编号（DFS的执行顺序） |

我们要做的就是（树链剖分的实现）：

1，对于一个点我们首先求出它所在的子树大小,找到它的重, 处理出size,son

- 树论(2)
- 树上差分(1)
- 数据结构(13)
- 数论(3)
- 数位DP(1)
- 搜索(3)
- 贪心(4)
- 线段树(2)
- 知识点(4)
- 主席树(2)
- 状压DP(1)

随笔档案

- 2019年2月 (1)
- 2018年11月 (3)
- 2018年10月 (17)
- 2018年9月 (16)
- 2018年8月 (4)
- 2018年7月 (1)
- 2018年6月 (4)
- 2018年5月 (14)

相册

背景

最新评论

- 1. Re:树链剖分详解写的不错 --chdy
- 2. Re:树链剖分详解大佬想问下，按照你的图查询节点14和10是咋跑的，14的话岂不是直接加到了1那里去了，怎么搞到lca呢 --zhuiyicc
- 3. Re:树链剖分详解Orz 写的太好了 --abc2237512422
- 4. Re:树链剖分详解博主写得太好啦!!! --LLL_Amazing
- 5. Re:树链剖分详解orz刚学树链剖分 --暗焱

阅读排行榜

- 1. 树链剖分详解(8275)
- 2. 链表及数组模拟链表(1392)
- 3. NOIP2018游记(729)
- 4. STL整理之map(383)
- 5. STL整理之set(320)

评论排行榜

- 1. 树链剖分详解(14)
- 2. NOIP2018游记(4)
- 3. Luogu P4945 【最后的战役】(2)
- 4. Luogu P4894 【GodFly求解法向量】(2)
- 5. CF1030A 【In Search of an Easy Problem】(2)

推荐排行榜

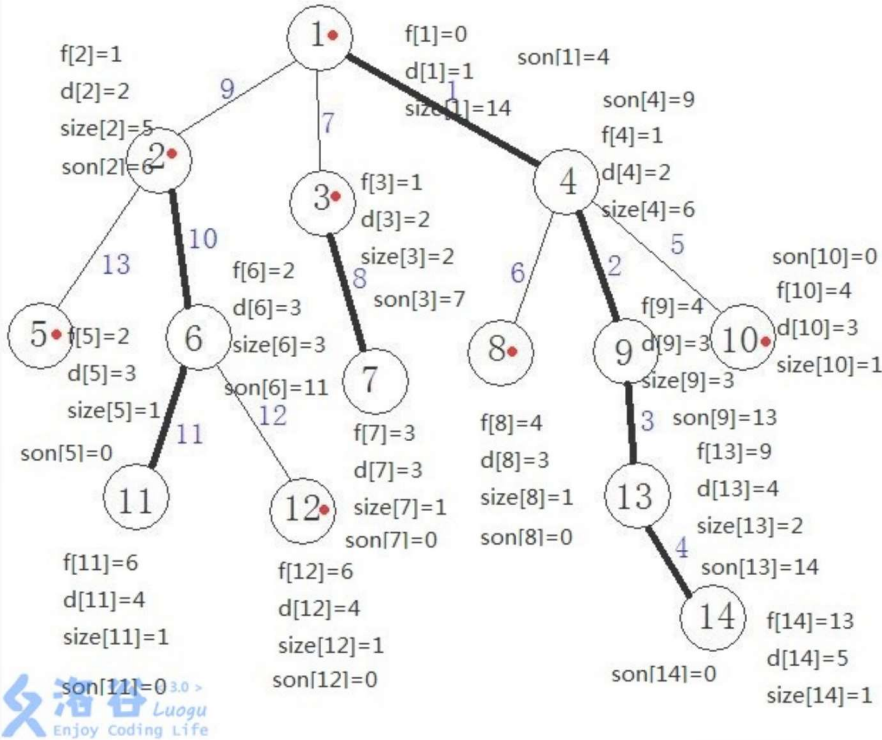
- 1. 树链剖分详解(45)
- 2. NOIP2018游记(4)
- 3. 链表及数组模拟链表(3)
- 4. STL整理之map(2)
- 5. NOIP2002普及T3【产生数】(2)

数组) ,
解释:比如说点1, 它有三个儿子2, 3, 4
2所在子树的大小是5
3所在子树的大小是2
4所在子树的大小是6
那么1的重儿子是4

ps:如果一个点的多个儿子所在子树大小相等且最大
那随便找一个当做它的重儿子就好了
叶节点没有重儿子, 非叶节点有且只有一个重儿子

2, 在dfs过程中顺便记录其父亲以及深度(即处理出f,d数组), 操作1,2可以通过一遍dfs完成

```
void dfs1(int u,int fa,int depth) //当前节点、父节点、层次深度
{
    f[u]=fa;
    d[u]=depth;
    size[u]=1; //这个点本身size=1
    for(int i=head[u];i;i=e[i].next)
    {
        int v=e[i].to;
        if(v==fa)
            continue;
        dfs1(v,u,depth+1); //层次深度+1
        size[u]+=size[v]; //子节点的size已被处理, 用它来更新父节点的size
        if(size[v]>size[son[u]])
            son[u]=v; //选取size最大的作为重儿子
    }
}
//进入
dfs1(root,0,1);
```

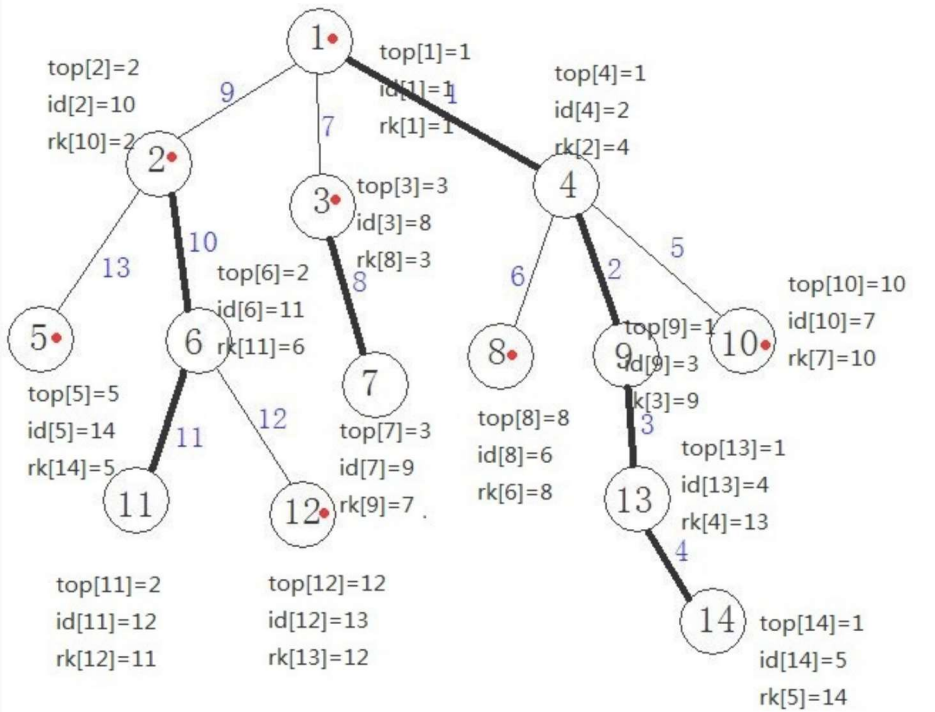


dfs跑完大概是这样的, 大家可以手动模拟一下

3, 第二遍dfs, 然后连接重链, 同时标记每一个节点的dfs序, 目的是为了用数据结

构来维护重链，我们在dfs时保证一条重链上各个节点dfs序连续（即处理出数组top,id,rk）

```
void dfs2(int u,int t) //当前节点、重链顶端
{
    top[u]=t;
    id[u]=++cnt; //标记dfs序
    rk[cnt]=u; //序号cnt对应节点u
    if(!son[u])
        return;
    dfs2(son[u],t);
    /*我们选择优先进入重儿子来保证一条重链上各个节点dfs序连续，
    一个点和它的重儿子处于同一条重链，所以重儿子所在重链的顶端还是t*/
    for(int i=head[u];i;i=e[i].next)
    {
        int v=e[i].to;
        if(v!=son[u]&&v!=f[u])
            dfs2(v,v); //一个点位于轻链底端，那么它的top必然是它本身
    }
}
```



dfs跑完大概是这样的，大家可以手动模拟一下

4，两遍dfs就是树链剖分的主要处理，通过dfs我们已经保证一条重链上各个节点dfs序连续，那么可以想到，我们可以通过数据结构（以线段树为例）来维护一条重链的信息

回顾上文的那个题目，修改和查询操作原理是类似的，以查询操作为例，其实就是个LCA，不过这里使用了top来进行加速，因为top可以直接跳转到该重链的起始结点，轻链没有起始结点之说，他们的top就是自己。需要注意的是，每次循环只能跳一次，并且让结点深的那个来跳到top的位置，避免两个一起跳从而擦肩而过。

```
int sum(int x,int y)
{
    //...
```

```

int ans=0,fx=top[x],fy=top[y];
while(fx!=fy)    //两点不在同一条重链
{
    if(d[fx]>=d[fy])
    {
        ans+=query(id[fx],id[x],rt);    //线段树区间求和,处理这条重链的贡献
        x=f[fx],fx=top[x];    //将x设置成原链头的父亲结点,走轻边,继续循环
    }
    else
    {
        ans+=query(id[fy],id[y],rt);
        y=f[fy],fy=top[y];
    }
}
//循环结束,两点位于同一重链上,但两点不一定为同一点,所以我们还要统计这两点之间的贡献
if(id[x]<=id[y])
    ans+=query(id[x],id[y],rt);
else
    ans+=query(id[y],id[x],rt);
return ans;
}

```



大家如果明白了树链剖分,也应该有举一反三的能力(反正我没有),修改和LCA就留给大家自己完成了

5, 树链剖分的时间复杂度

树链剖分的两个性质:

- 1, 如果 (u, v) 是一条轻边, 那么 $\text{size}(v) < \text{size}(u)/2$;
- 2, 从根结点到任意结点的路所经过的轻重链的个数必定都小于 $\log n$;

可以证明, 树链剖分的时间复杂度为 $O(n \log^2 n)$

几道例题:

1, 树链剖分模板

就是刚才讲的

上代码:

View Code

2, [NOI2015]软件包管理器

观察到题目要求支持两种操作

1, **install x**: 表示安装软件包x

2, **uninstall x**: 表示卸载软件包x

对于操作一, 我们可以统计x到根节点未安装的软件包的个数, 然后区间修改为已安装

对于操作二, 我们可以统计x所在子树已安装软件包的个数, 然后将子树修改为未安装

上代码:



```

#include<iostream>
#include<cstdio>
#define int long long
using namespace std;
const int maxn=1e5+10;
struct edge{
    int next,to;
}e[2*maxn];
struct Node{

```



```

    int l,r,ls,rs,sum,lazy;
}node[2*maxn];
int rt,n,m,cnt,head[maxn];
int f[maxn],d[maxn],size[maxn],son[maxn],rk[maxn],top[maxn],tid[maxn];
int readn()
{
    int x=0;
    char ch=getchar();
    while(ch<'0' || ch>'9')
        ch=getchar();
    while(ch>='0' && ch<='9')
    {
        x=(x<<1)+(x<<3)+ch-'0';
        ch=getchar();
    }
    return x;
}
void add_edge(int x,int y)
{
    e[++cnt].next=head[x];
    e[cnt].to=y;
    head[x]=cnt;
}
void dfs1(int u,int fa,int depth)
{
    f[u]=fa;
    d[u]=depth;
    size[u]=1;
    for(int i=head[u];i;i=e[i].next)
    {
        int v=e[i].to;
        if(v==fa)
            continue;
        dfs1(v,u,depth+1);
        size[u]+=size[v];
        if(size[v]>size[son[u]] || !son[u])
            son[u]=v;
    }
}
void dfs2(int u,int t)
{
    top[u]=t;
    tid[u]=++cnt;
    rk[cnt]=u;
    if(!son[u])
        return;
    dfs2(son[u],t);
    for(int i=head[u];i;i=e[i].next)
    {
        int v=e[i].to;
        if(v!=son[u] && v!=f[u])
            dfs2(v,v);
    }
}
void pushup(int x)
{
    int lson=node[x].ls,rson=node[x].rs;
    node[x].sum=node[lson].sum+node[rson].sum;
    node[x].l=node[lson].l;
    node[x].r=node[rson].r;
}
void build(int li,int ri,int cur)
{
    if(li==ri)
    {
        node[cur].ls=node[cur].rs=node[cur].lazy=-1;
        node[cur].l=node[cur].r=li;
        return;
    }
    int mid=(li+ri)>>1;
    node[cur].ls=cnt++;
    node[cur].rs=cnt++;
    build(li,mid,node[cur].ls);

```

```

    build(mid+1,ri,node[cur].rs);
    pushup(cur);
}
void pushdown(int x)
{
    int lson=node[x].ls,rson=node[x].rs;
    node[lson].sum=node[x].lazy*(node[lson].r-node[lson].l+1);
    node[rson].sum=node[x].lazy*(node[rson].r-node[rson].l+1);
    node[lson].lazy=node[x].lazy;
    node[rson].lazy=node[x].lazy;
    node[x].lazy=-1;
}
void update(int li,int ri,int c,int cur)
{
    if(li<=node[cur].l&&node[cur].r<=ri)
    {
        node[cur].sum=c*(node[cur].r-node[cur].l+1);
        node[cur].lazy=c;
        return;
    }
    if(node[cur].lazy!=-1)
        pushdown(cur);
    int mid=(node[cur].l+node[cur].r)>>1;
    if(li<=mid)
        update(li,ri,c,node[cur].ls);
    if(mid<ri)
        update(li,ri,c,node[cur].rs);
    pushup(cur);
}
int query(int li,int ri,int cur)
{
    if(li<=node[cur].l&&node[cur].r<=ri)
        return node[cur].sum;
    if(node[cur].lazy!=-1)
        pushdown(cur);
    int tot=0;
    int mid=(node[cur].l+node[cur].r)>>1;
    if(li<=mid)
        tot+=query(li,ri,node[cur].ls);
    if(mid<ri)
        tot+=query(li,ri,node[cur].rs);
    return tot;
}
int sum(int x)
{
    int ans=0;
    int fx=top[x];
    while(fx)
    {
        ans+=tid[x]-tid[fx]-query(tid[fx],tid[x],rt)+1;
        update(tid[fx],tid[x],1,rt);
        x=f[fx];
        fx=top[x];
    }
    ans+=tid[x]-tid[0]-query(tid[0],tid[x],rt)+1;
    update(tid[0],tid[x],1,rt);
    return ans;
}
signed main()
{
    n=readn();
    for(int i=1;i<n;i++)
    {
        int x=readn();
        add_edge(x,i);
        add_edge(i,x);
    }
    cnt=0;
    dfs1(0,-1,1);
    dfs2(0,0);
    cnt=0;
    rt=cnt++;
    build(1,n,rt);

```

```
m=readn();
for(int i=1;i<=m;i++)
{
    int x;
    string op;
    cin>>op;
    x=readn();
    if(op=="install")
        printf("%lld\n",sum(x));
    else if(op=="uninstall")
    {
        printf("%lld\n",query(tid[x],tid[x]+size[x]-1,rt));
        update(tid[x],tid[x]+size[x]-1,0,rt);
    }
}
return 0;
}
```

3,[SDOI2011]染色

有一些思维含量的题

统计颜色段数量时不能简单地区间加法

线段树还应维护区间最左颜色和区间最右颜色

合并时


如果S(l,k)的右端与S(k+1,r)的左端颜色相同，那么S(l,r)=S(l,k)+S(k+1,r)-1 (减去重复的那一个)

否则S(l,r)=S(l,k)+S(k+1,r)正常合并

分类: 树链剖分 , 数据结构 , 线段树 , 知识点

好文要顶 关注我 收藏该文



 **ivanovcraft**
关注 - 7
粉丝 - 16

+加关注

» 下一篇: [Luogu P3384 【模板】树链剖分](#)

posted @ 2018-05-10 12:54 ivanovcraft 阅读(8275) 评论(14) 编辑 收藏

评论列表

- #1楼 2018-07-08 11:07 YoungNeal

orz

支持(0) 反对(0)
- #2楼 2018-08-08 16:23 Cwolf9

对于博主开头提的第一个问题：
1，将树从x到y结点最短路径上所有节点的值都加上z。
差分我知道，我想问要求x-到y节点路径的权值和的解法是什么？用树状数组维护吗？还是说dfs一遍？

支持(0) 反对(0)
- #3楼[楼主] 2018-08-08 21:16 ivanovcraft

完整解决方案是这样的：
首先定义s[i]表示对点i所打的标记
1，每次修改链x-y时（权值加c），令s[x]+=c,s[y]+=c,s[lca(x,y)]-=c,s[father] 45 ;
2，打好标记后，再dfs一遍，(for each son[x])令s[x]+=s[son[x]]
3，这样，一个点的权值就是原本的权值a[i]加上s[i]