[ZJOI2007]捉迷藏 解题报告 (动态点分治)

BruceW

[ZJOI2007]捉迷藏

近期做过的码量最大的一题 (当然也是我写丑了....)

CONTENTS

题意

有一个 n 个节点的树 ($n \leq 10^5$), 每个节点为黑色或白色.

有 m 个操作 ($m \leq 5 \times 10^5$), 操作有两种,

1. 将点 x 的的颜色翻转.
2. 查询树上距离最远的黑色点对之间的距离.

思路

首先, 如果没有修改操作的话, 就是一个裸的点分治 (点分治学习笔记).

有修改操作, 那就 动态点分治 .

动态点分治的基本思路是 (个人总结的) : 由于树的结构不会变化, 所以可以先找出重心, 并构建 点分树 (一个连接每一层重心的树), 然后维护点分治中需要知道的数据, 避免了进行递归查找重心带来的时间浪费.

基本思路是简单的, 困难的是归纳出需要知道的数据并对它们进行维护.

(注 : 下文中如果没有特别说明, 则 "儿子", "子树", "父亲", "距离" 等名词都是指在原树上的.)

这道题, 我们在点分治过程中, 对每一个点需要知道的数据为 :

1. 以它为重心时, 在它 管辖范围内的黑色节点 到它的 距离最大值和次大值 , 称为数据 1.
2. 以它为重心时, 它的 管辖范围内的所有点 到上一层重心 (即它在 点分树上的父亲) 的距离, 称为数据 2.

这两个数据的关系是 :

一个点把自己的 数据 2 中的最大值 贡献给 点分树上的父亲 , 点分树上的父亲 根据每个儿子贡献出来的最大值, 得到自己 数据 1 的最大值和次大值 .

显然, 在得到 最大值和次大值 的过程中, 我们需要一个支持排序的数据结构, 这里我们选择使用 堆 .

刚才我们提到的 点分树 , 有一个比较显然的性质 : 它的 树高 是 $\log n$ 级别的.

那么我们就可以在修改一个节点时, 依次维护它的每一个祖先的数据, 并且只有 $O(\log n)$ 的时间复杂度.

但是, 在维护过程中, 我们需要用到删除操作, 而堆本身是不支持删除操作的.

我们可以使用一个小技巧 : 把一个堆 一分为二 , 一个是 排序堆 , 一个是 删除堆 .

当我们需要删除某个值的时候, 只需要把这个值 加入删除堆里,

在取最大值时, 若 排序堆 和 删除堆 的 堆顶元素相同, 就意味着这个元素已经被删除了, 把它们双双弹出.

这样, 我们就得到了一个时间复杂度为 $O(n \log^2 n)$ 的算法, 并且由于堆 (优先队列) 的使用, 会有比较大的常数.

作者的代码在洛谷上会 TLE 一个点, 在 bzoj 上可以通过. 并且代码比较丑陋, 不建议看代码理解. 实在想看代码的话可以去看洛谷的第一篇题解.

代码

Copy

```
#include<bits/stdc++.h>
using namespace std;
const int _=1e5+7;
const int __=2e5+7;
const int L=20;
const int inf=0x3f3f3f3f;
struct hep{
  priority_queue<int> A,B;
  void push(int x){ A.push(x); }
  void del(int x){ B.push(x); }
  int top(){
    while(!A.empty()&&!B.empty()&&A.top()==B.top()){ A.pop(); B.pop(); }
    return A.empty() ?-1 :A.top();
  }
  int sec(){
    int x=top();
    if(x==-1) return x;
    A.pop();
    int y=top();
    A.push(x);
    return y;
  }
}h1[_],h2[_],ans;
int n,m,dep[_],f[_][L+7],dis[_],sz[_],rt,minx=inf,ft[_],lt[_];
int lst[_],nxt[__],to[__],tot;
bool vis[_],sta[_];
int gi(){
  char c=getchar(); int x=0,f=1;
  while(c<'0'||c>'9'){ f=c=='-'?-1:1; c=getchar(); }
  while(c>='0'&&c<='9'){ x=(x<<3)+(x<<1)+c-'0'; c=getchar(); }
  return x*f;
}
void add(int x,int y){ nxt[++tot]=lst[x]; to[tot]=y; lst[x]=tot; }
int Lca(int x,int y){
  if(dep[x]<dep[y]) swap(x,y);
```

```
  for(int i=L;i>=0;i--)
    if(dep[f[x][i]]>=dep[y])
      x=f[x][i];
  if(x==y) return x;
  for(int i=L;i>=0;i--)
    if(f[x][i]!=f[y][i]){
      x=f[x][i];
      y=f[y][i];
    }
  return f[x][0];
}
int dist(int x,int y){ return dep[x]+dep[y]-2*dep[Lca(x,y)]; }
void idk(int u,int fa){
  dep[u]=dep[fa]+1;
  f[u][0]=fa;
  for(int i=1;i<=L;i++)
    f[u][i]=f[f[u][i-1]][i-1];
  for(int i=lst[u];i;i=nxt[i]){
    int v=to[i];
    if(v==fa) continue;
    idk(v,u);
  }
}
void pre(int u,int fa,bool id){
  if(id) h1[rt].push(dis[u]+1);
  sz[u]=1; dis[u]=dis[fa]+1;
  for(int i=lst[u];i;i=nxt[i]){
    int v=to[i];
    if(v==fa||vis[v]) continue;
    pre(v,u,id);
    sz[u]+=sz[v];
  }
}
void g_rt(int u,int fa,int sum){
  int maxn=sum-sz[u];
  for(int i=lst[u];i;i=nxt[i]){
    int v=to[i];
    if(v==fa||vis[v]) continue;
    g_rt(v,u,sum);
    maxn=max(maxn,sz[v]);
  }
  if(maxn<minx){ minx=maxn; rt=u; }
}
void upd(int u,int id){
  int t1=h2[u].top(),t2=h2[u].sec();
  if(id){
    if(t1==-1);
    else if(t2==-1) ans.push(0);
    else ans.push(t1+t2);
```

```
  }
  else{
    if(t1==-1);
    else if(t2==-1) ans.del(0);
    else ans.del(t1+t2);
  }
}
void init(int u,int lrt){
  minx=inf;
  pre(u,0,0);
  g_rt(u,0,sz[u]);
  //printf("u: %d rt: %d dis[rt]: %d\n",u,rt,dis[rt]);
  dis[rt]= lrt ?dis[rt] :-1;
  ft[rt]=lrt;
  lt[rt]=dis[rt]+1;    // 因为此时的 dis 是到 lrt 的子节点的距离, 所以要加上 1
  pre(rt,0,1);
  h2[lrt].push(h1[rt].top());
  vis[rt]=1; u=rt;
  for(int i=lst[u];i;i=nxt[i]){
    int v=to[i];
    if(vis[v]) continue;
    init(v,u);
  }
  vis[u]=0;
  h2[u].push(0);
  upd(u,1);
}
void modify(int u){
  sta[u]^=1;
  int len=lt[u],fa=ft[u],t=u;
  upd(u,0);
  //printf("%d %d\n",h2[u].top(),h2[u].sec());
  if(sta[u]){ h2[u].del(0); }
  else h2[u].push(0);
  upd(u,1);
  while(fa){
    int top=h1[u].top(),sec=h2[fa].sec();
    bool f1= len>=top,f2= len>=sec;
    if(f1){
      if(f2) upd(fa,0);
      h2[fa].del(top);
      if(sta[t]){ h1[u].del(len); h2[fa].push(h1[u].top()); }
      else{ h1[u].push(len); h2[fa].push(len); }
      if(f2) upd(fa,1);
    }
    else{
      if(sta[t]) h1[u].del(len);
      else h1[u].push(len);
    }
```

```
    u=ft[u]; fa=ft[u];
    len=dist(t,fa);
  }
}
char ss[_];
void run(){
  m=gi(); char c; int x;
  for(int i=1;i<=m;i++){
    //gets(ss);
    c=getchar();
    //printf("%s %c\n",ss,c);
    if(c=='G') printf("%d\n",ans.top());
    else{
      x=gi();
      //printf("x: %d\n",x);
      modify(x);
      //puts("!!!");
    }
  }
}
int main(){
  //freopen("x.in","r",stdin);
  //freopen("x.out","w",stdout);
  n=gi(); int x,y;
  for(int i=1;i<n;i++){
    x=gi(); y=gi();
    //printf("%d %d\n",x,y);
    add(x,y);
    add(y,x);
  }
  dis[0]=-1;
  idk(1,0);
  init(1,0);
  run();
  return 0;
}
```

标签: 动态点分治, 点分治, 解题报告

« 上一篇: 「学习笔记」点分治

» 下一篇: 「学习笔记」动态点分治