

点分树

算法简介

点分治的动态版本，可以动态维护树上路径信息，一般会与线段树或平衡树等数据结构配合使用。

特别适用于一些需要维护与路径长度相关信息的题目。

一般空间复杂度为 $O(n \log n)$ 单次查询或修改时间复杂度为 $O(\log^2 n)$

算法思想

把点分治过程中得到的重心连接，得到一棵虚树，该树有如下性质：

- 1、深度不超过 $\log n$
- 2、所有结点的子树大小和不超过 $n \log n$

由于点分治最多递归 $\log n$ 层，所以性质一成立。

对性质二，考虑每个结点对子树大小和的贡献。

每个结点对每个祖先结点(包括它自己)产生一个贡献，而由性质一，每个节点最多有 $\log n$ 个祖先结点，所以每个结点贡献最多为 $\log n$

所以有所有结点的子树大小和不超过 $n \log n$ 性质二证毕。

有了这两个性质的保障，便可以用点分树暴力地解一些题目。

一般思路为对每个结点，维护该结点在虚树中的子树信息。

对每个结点，若使用线段树或平衡树等数据结构维护子树信息，根据性质二，空间复杂度仅为 $n \log n$

修改和查询都暴力跳 fa 若每次查询线段树或平衡树数据结构维护子树信息，根据性质一，有单次查询或修改时间复杂度为 $O(\log^2 n)$

算法习题

习题一

[洛谷p6329](#)

题意

给出一棵 n 个结点的点权树，相邻点距离为 1 ，接下来 m 个操作，操作有以下两种：

操作0：输出所有到结点 x 距离不超过 k

操作1：将结点 x 点权修改为 y

同时算法要求强制在线，对每次操作的参数 x, y, k 都需要异或上一次输出的答案。

题解

考虑对每个结点，建立两个树状数组，第一个树状数组维护所有在虚树中属于结点 x 的子树且到 x 距离等于 k 的点权和的数组。

第二个树状数组维护所有在虚树中属于结点 x 的子树且到结点 x 在虚树中的父亲的距离等于 k 的点权和的数组。

记 $\text{dist}(x, y)$ 为结点 x 到 y 距离，可以用树剖或者欧拉序列加 ST 表求。

对查询操作，可先查询 x 第一个树状数组中距离小于等于 k 的点权和。然后开始暴力跳 fa

每次加上 fa 结点第一个树状数组中距离小于等于 $k - \text{dist}(x, \text{fa})$ 的点权和。

但是这样会重复计算 fa 结点的子结点的子树贡献。

所以再减去 fa 结点的子结点的第二个树状数组中距离小于等于 $k - \text{dist}(x, \text{fa})$ 的点权和。

最后便可以得到答案，时间复杂度为 $O(\log^2 n)$

对于修改操作，需要同时维护每个结点的第一个和第二个树状数组。

同样先修改 x 第一个树状数组中距离为 0 的点权和(因为 x 到自身距离为 0)。然后开始暴力跳 fa

每次修改 fa 结点第一个树状数组中距离为 $\text{dist}(x, \text{fa})$ 的点权和。

再修改 fa 结点的子结点的第二个树状数组中距离等于 $\text{dist}(x, \text{fa})$ 的点权和。

最后便可以完成修改，时间复杂度也是 $O(\log^2 n)$

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <cctype>
#include <vector>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline void write(LL x){
```

```

    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=1e5+5,MAXM=20,inf=1e6+5;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int n,q,edge_cnt,head[MAXN];
inline void Insert(int u,int v){
    edge[++edge_cnt].to=v;
    edge[edge_cnt].next=head[u];
    head[u]=edge_cnt;
}
int a[MAXN],sz[MAXN],dep[MAXN],mson[MAXN],f[MAXN],tot_sz,root,root_sz;
bool vis[MAXN];
struct LCA{
    int B[MAXN<<1],F[MAXN<<1],pos[MAXN],n;
    int d[MAXN<<1][MAXM],lg2[MAXN<<1];
    inline void push(int index,int depth){
        F[n]=index;
        B[n]=depth;
        n++;
    }
    void dfs(int u,int fa,int depth){
        pos[u]=n;dep[u]=depth;
        push(u,depth);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==fa)
                continue;
            dfs(v,u,depth+1);
            push(u,depth);
        }
    }
    void build(int root){
        dfs(root,-1,0);
        lg2[1]=0;
        _rep(i,2,n)
            lg2[i]=lg2[i>>1]+1;
        _for(i,0,n)
            d[i][0]=i;
        for(int j=1;(1<<j)<=n;j++){
            _for(i,0,n+1-(1<<j)){
                if(B[d[i][j-1]]<B[d[i+(1<<(j-1))][j-1]])
                    d[i][j]=d[i][j-1];
                else
                    d[i][j]=d[i+(1<<(j-1))][j-1];
            }
        }
    }
}

```

```
    }  
  }  
}  
int query(int a,int b){  
  int lef=pos[a],rig=pos[b],len=abs(rig-lef)+1;  
  if(lef>rig)  
    swap(lef,rig);  
  if(B[d[lef][lg2[len]]]<B[d[rig-(1<<lg2[len])+1][lg2[len]]])  
    return F[d[lef][lg2[len]]];  
  else  
    return F[d[rig-(1<<lg2[len])+1][lg2[len]]];  
}  
}lca;  
int get_dis(int a,int b){  
  return dep[a]+dep[b]-(dep[lca.query(a,b)]<<1);  
}  
#define lowbit(x) x&(-x)  
struct BIT{  
  int n;  
  vector<int>c;  
  void build(int n){  
    this->n=n;  
    c.resize(n+1);  
  }  
  void add(int pos,int v){  
    ++pos;  
    while(pos<=n){  
      c[pos]+=v;  
      pos+=lowbit(pos);  
    }  
  }  
  int query(int pos){  
    pos=min(pos+1,n);  
    int ans=0;  
    while(pos){  
      ans+=c[pos];  
      pos-=lowbit(pos);  
    }  
    return ans;  
  }  
}  
}tree_1[MAXN],tree_2[MAXN];  
void find_root(int u,int fa){  
  sz[u]=1;mson[u]=0;  
  for(int i=head[u];i;i=edge[i].next){  
    int v=edge[i].to;  
    if(vis[v]||v==fa)  
      continue;  
    find_root(v,u);  
    sz[u]+=sz[v];  
    mson[u]=max(mson[u],sz[v]);  
  }  
}
```

```

    }
    mson[u]=max(mson[u],tot_sz-sz[u]);
    if(mson[u]<root_sz){
        root=u;
        root_sz=mson[u];
    }
}
void build_tree(int u,int fa){
    int cur_sz=tot_sz;
    vis[u]=true;f[u]=fa;
    tree_1[u].build((cur_sz>>1)+1);tree_2[u].build(cur_sz+1);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v])
            continue;
        tot_sz=sz[v]>sz[u]?cur_sz-sz[u]:sz[v];root_sz=MAXN;
        find_root(v,u);
        build_tree(root,u);
    }
}
void update(int u,int val){
    tree_1[u].add(0,val);
    for(int i=u;f[i];i=f[i]){
        int d=get_dis(u,f[i]);
        tree_1[f[i]].add(d,val);
        tree_2[i].add(d,val);
    }
}
int query(int u,int k){
    int ans=tree_1[u].query(k);
    for(int i=u;f[i];i=f[i]){
        int d=get_dis(u,f[i]);
        if(d>k)
            continue;
        ans+=tree_1[f[i]].query(k-d);
        ans-=tree_2[i].query(k-d);
    }
    return ans;
}
int main()
{
    n=read_int(),q=read_int();
    _rep(i,1,n)
    a[i]=read_int();
    int u,v;
    _for(i,1,q){
        u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    lca.build(1);
}

```

```
root_sz=MAXN;
tot_sz=n;
find_root(1,0);
build_tree(root,0);
_rep(i,1,n)
update(i,a[i]);
int last=0,opt,x,y;
while(q--){
    opt=read_int(),x=read_int()^last,y=read_int()^last;
    if(opt==0)
        enter(last=query(x,y));
    else{
        update(x,y-a[x]);
        a[x]=y;
    }
}
return 0;
}
```

习题二

洛谷p2056

题意

给出一棵 n 个结点的树，相邻点距离为 1 ，一开始所有点都为黑点，接下来 m 个操作，操作有以下两种：

操作1：改变结点 x 的颜色(黑色变白色，白色变黑色)。

操作2：询问树上距离最远的黑色点对的距离，如果只有一个黑点输出 0 ，如果没有黑点输出 -1 。

题解

考虑对每个结点 x 建立两个大根堆，第一个堆维护每棵在虚树中属于结点 x 的子树到结点 x 的最大距离。

第二个堆维护所有在虚树中属于结点 x 的子树的结点到 x 在虚树中的父亲结点的距离。

最后建立一个答案堆，维护每个结点的答案，其中每个结点的答案为每个结点第一个堆的最大值和次大值和。

对于操作1，同样是暴力跳 fa 沿途维护结点的第一个堆和第二个堆。

关于堆的删除操作，可以考虑建两个堆，如果第一个堆存所有的数，第二个堆存要删除的数。

每次取第一个堆最大元素前先检测两个堆顶元素是否相同，相同则同时 $\text{pop}()$

另外，求两个结点的距离好像可以通过 bfs 预处理完成，可以避免欧拉序列加 ST 表常数过大的问题。

记 $\text{dist}(x,y)$ 为结点 x 到 y 距离，可以用树剖或者欧拉序列加 ST 表求。

对于操作2，直接查询答案堆最大元素即可。

具体一些细节见代码。

```

#include <iostream>
#include <cstdio>
#include <queue>
#include <algorithm>
#include <cstring>
#include <cctype>
#include <vector>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n'||c=='\r')c=getchar();
    return c;
}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=1e5+5,MAXM=20,inf=1e6+5;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int n,q,edge_cnt,head[MAXN];
inline void Insert(int u,int v){
    edge[++edge_cnt].to=v;
    edge[edge_cnt].next=head[u];
    head[u]=edge_cnt;
}
int sz[MAXN],mson[MAXN],f[MAXN],tot_sz,root,root_sz;
int dep[MAXN],fdis[MAXM][MAXN],ans[MAXN];
bool vis[MAXN],turn_off[MAXN];
struct Heap{
    priority_queue<int> q1,q2;
    void debug(){

```

```
        enter(q1.size()-q2.size());
    }
    void Insert(int x){
        q1.push(x);
    }
    void Delate(int x){
        if(q1.top()==x)
            q1.pop();
        else
            q2.push(x);
    }
    int get_first(){
        while((!q1.empty())&&(!q2.empty())&&(q1.top()==q2.top()))
            q1.pop(),q2.pop();
        if(q1.empty())
            return -inf;
        else
            return q1.top();
    }
    int get_pair(){
        if(q1.size()-q2.size()<2)
            return -inf;
        int t1=get_first(),t2;
        q1.pop();
        t2=get_first();
        q1.push(t1);
        return t1+t2;
    }
}Ans,heap_1[MAXN],heap_2[MAXN];
void add_ans(int x){
    ans[x]=heap_1[x].get_pair();
    if(ans[x]!=-inf)
        Ans.Insert(ans[x]);
}
void delate_ans(int x){
    if(ans[x]!=-inf)
        Ans.Delate(ans[x]);
}
void bfs(int s){
    queue<int>q;
    q.push(s);
    fdis[++dep[s]][s]=0;
    while(!q.empty()){
        int u=q.front();q.pop();
        heap_2[s].Insert(fdis[dep[s]-1][u]);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(vis[v])
                continue;
            if(fdis[dep[s]][u]+1<fdis[dep[s]][v]){
```

```

        fdis[dep[s]][v]=fdis[dep[s]][u]+1;
        dep[v]++;
        q.push(v);
    }
}
}
}
void find_root(int u,int fa){
    sz[u]=1;mson[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v]||v==fa)
            continue;
        find_root(v,u);
        sz[u]+=sz[v];
        mson[u]=max(mson[u],sz[v]);
    }
    mson[u]=max(mson[u],tot_sz-sz[u]);
    if(mson[u]<root_sz){
        root=u;
        root_sz=mson[u];
    }
}
void build_tree(int u,int fa){
    int cur_sz=tot_sz;
    vis[u]=true;f[u]=fa;
    bfs(u);
    if(heap_2[u].get_first()!=-inf)
        heap_1[fa].Insert(heap_2[u].get_first());
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[v])
            continue;
        tot_sz=sz[v]>sz[u]?cur_sz-sz[u]:sz[v];root_sz=MAXN;
        find_root(v,u);
        build_tree(root,u);
    }
    heap_1[u].Insert(0);
    ans[u]=heap_1[u].get_pair();
    if(ans[u]!=-inf)
        Ans.Insert(ans[u]);
}
void Add(int u){
    n++;
    delate_ans(u);
    heap_1[u].Insert(0);
    add_ans(u);
    for(int i=u;f[i];i=f[i]){
        int d=fdis[dep[i]-1][u];
        delate_ans(f[i]);
        if(heap_2[i].get_first()!=-inf)

```

```
        heap_1[f[i]].Delate(heap_2[i].get_first());
        heap_2[i].Insert(d);
        heap_1[f[i]].Insert(heap_2[i].get_first());
        add_ans(f[i]);
    }
}
void Delate(int u){
    n--;
    delate_ans(u);
    heap_1[u].Delate(0);
    add_ans(u);
    for(int i=u;f[i];i=f[i]){
        int d=fdis[dep[i]-1][u];
        delate_ans(f[i]);
        heap_1[f[i]].Delate(heap_2[i].get_first());
        heap_2[i].Delate(d);
        if(heap_2[i].get_first()!=-inf)
            heap_1[f[i]].Insert(heap_2[i].get_first());
        add_ans(f[i]);
    }
}
int query(){
    if(n==0)
        return -1;
    else if(n==1)
        return 0;
    else
        return Ans.get_first();
}
int main()
{
    n=read_int();
    int u,v;
    _for(i,1,n){
        u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    mem(fdis,127/3);
    root_sz=MAXN;
    tot_sz=n;
    find_root(1,0);
    build_tree(root,0);
    q=read_int();
    char opt;
    int x;
    while(q--){
        opt=get_char();
        if(opt=='G')
            enter(query());
    }
}
```

```
    else{
        x=read_int();
        turn_off[x]^=1;
        if(turn_off[x])
            Delate(x);
        else
            Add(x);
    }
}
return 0;
}
```

From:

<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:

https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E7%82%B9%E5%88%86%E6%A0%91&rev=1592032846

Last update: 2020/06/13 15:20