「CSP2019 校内 6 连测」Day2_Tree

Jiayi Su (ShuYuMo)

2019-11-10 10:02:00

一道趣题。

校内 6 连测 Day2

Tree

面向数据编程：

subtask1

由于题目没有给定根，所以第一个想法就是枚举根，然后在每个点检查子树是否包含每一个颜色，更新答案。
具体可以状压啊，(暴力统计也能过)，需要注意的是状态压缩时，设置状态中 `1 << 50` 这样的语句时不可以的。(应该为 `1LL << 50`)；
subtask2 只有一种颜色，找一个最长的链即可，就是树的直径。### subtask3 发现要是想要 `AC` 这道题，枚举根肯定是没前途的，除非可以确定一个根，然后直接 $O(1)$ 猜答案。

先随便找一个点作为临时的根，然后对于我们枚举到的一个点，我们考虑它作为 LCA 的时候的最优答案，显然真正的根有两种情况：- 在当前点为根的子树中。- 不在他的子树内。对于这几种情况，我们都算一个最远的那个点离他多远，把最远的那个合法的点作为根就可以。

比如，右边这个图，当真正的根是 **6**，当前点是 **3** 的时候，我们就需要知道，他的子树的点集 **T={1,2,3,5,9,8}** 这个集合里面是否包含了每种颜色的点。

这是正解的弱化版本，树形 `DP`，`f[i][j]` 表示在树上结点 `i`，颜色 `j` 的数量。

那么我们就可以用 `f[1]-f[4]` 算得 `T` 中包含的每种颜色的点的个数。我们对于每个点，枚举根在哪个方向，算这个方向上最远的点，复杂度是 $O(n)$ 的。

算 $f[i][j]$ 的总复杂度是 $O(nm)$ 的，枚举根方向再算合不合法，复杂度也是 $O(nm)$。所以总复杂度是 $O(nm)$.

subtask4

归纳上一个 `subtask3` 发现其实对于每一个点，只需要知道两个信息即可：- 这个点为根的子树是否包含所有的颜色 - 整棵树抛去这个点为根的子树是否包含所有的颜色（称之为这个点为根的子树的反树）

还是先选一个临时的根，分开考虑两种情况 - 到一个点之后判断 以 这个点为根的子树是否包含所有的颜色。设以某个点 i 为根的子树所包含的元素种类数为 W_i
考虑一个点对这个点到根这条链上的贡献，每个点都会使得这个点到根这条链上的点所有点 j 的 $W_j + 1$。但是考虑两个相同颜色的点，对答案贡献之后，会使这两个点的 `LCA` 到根这条路径上的点都被加重复了一次。所以这条路径上的点都需要再 -1。
树上的路径操作——树上差分即可完成。注意，需要对同一种颜色的点按照 `DFS` 序排序，然后两两求 `LCA` 可以发现，只有这样做才是对的。

对这个树进行差分，然后能在 DFS 的过程中求得每个点为根的子树中的颜色种类数量。如果颜色种类数量为 m 那就说明这个子树已经包含了所有的颜色，那么这个点就是题目要求的 LCA 之一，然后用向上最长链（可以预处理出来，其实不是向上的最长链，只要这条链的终点不在当前子树中即可，也就是选取这个链的终点当作真正的根）更新答案。- 如何判断一个点为根的的子树的反树包含了所有的点。其实可以反过来想，一个反树包含所有的颜色，也就是某一种颜色并没有被当前子树全部包含。求出每种颜色全部点的 LCA。显然，选取的能够更新答案的点不是这些 LCA 的祖先。预处理每一个种颜色所有点的 LCA，然后进行在那个点打标记。最后 dfs 上传标记即可，设当前点为 now，儿子结点为 to，那么标记数组 $f[i]$ 就等于 $f[now]|| = f[to]$。最后检查每一个没被打过标记的点，用向下最长链（可以预处理，也就是这个点往下和一个最深的叶子结点的距离，也就是选取这个最深的叶子结点当真正的根）更新答案。

这两种情况更新完答案就做完了

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include<cmath>
#include<algorithm>
#include<vector>
#define _R register
using namespace std;
const int _N = 1e6 + 100;
const int _M = 2e6 + 100;
inline int read()
{
    char c = getchar(); int sign = 1; int x = 0;
    while(c > '9' || c < '0') { if(c=='-')sign = -1; c = getchar(); }
    while(c <= '9' && c >= '0') { x *= 10; x += c - '0'; c = getchar(); }
    return x * sign;
}
int head[_N];
struct edges{
    int node;
    int nxt;
}edge[_M];
int tot = 0;
void add(int u, int v){
    edge[++tot].nxt = head[u];
    head[u] = tot;
    edge[tot].node = v;
}
int n, m;
int col[_N];
int root;
int MaxDeep[_N];
int MaxUp[_N];
void dfs1(int now, int fa){
    int Maxdp = -1;
    for(_R int i = head[now];i;i = edge[i].nxt){
        int to = edge[i].node;
```

```
        if(to == fa) continue;
        dfs1(to, now);
        Maxdp = max(Maxdp, MaxDeep[to]);
    }
    MaxDeep[now] = Maxdp + 1;
}
void dfs2(int now, int fa)
{
    int MaxVal1 = -1;int MaxNode1;
    int MaxVal2 = -1;int MaxNode2;
    for(_R int i = head[now];i;i = edge[i].nxt){
        int to = edge[i].node;
        if(to == fa) continue;
        if(MaxVal1 < MaxDeep[to]) {
            MaxVal2 = MaxVal1;MaxNode2 = MaxNode1;
            MaxVal1 = MaxDeep[to];MaxNode1 = to;
        } else MaxVal2 = max(MaxVal2, MaxDeep[to]);
    }
    for(_R int i = head[now];i;i = edge[i].nxt){
        int to = edge[i].node;
        if(to == fa) continue;
        MaxUp[to] = max((MaxDeep[to] == MaxVal1 ? MaxVal2 + 2 : MaxVal1 + 2), MaxUp[now] + 1);
    }
    for(_R int i = head[now];i;i = edge[i].nxt){
        int to = edge[i].node;
        if(to == fa) continue;
        dfs2(to, now);
    }
}
namespace LCA{
    const int Log = 20;
    int anc[_N][Log + 5];
    int deep[_N];
    void dfs(int now, int fa, int dp){
        anc[now][0] = fa;
        deep[now] = dp;
        for(_R int i = head[now];i;i = edge[i].nxt){
            int to = edge[i].node;
            if(to == fa) continue;
            dfs(to, now, dp + 1);
        }
    }
    void work()
    {
        dfs(root, root, 1);
        for(_R int j = 1;j <= Log;j++){
            for(_R int i = 1;i <= n;i++){
```

```
                anc[i][j] = anc[anc[i][j - 1]][j - 1];
            }
        }
    }
    int query(int x, int y){
        if(deep[x] < deep[y]) swap(x, y);
        int dis = deep[x] - deep[y];
        int now = 0;
        while(dis != 0){
            if(dis & 1) x = anc[x][now];
            now ++;
            dis >>= 1;
        }
        if(x == y) return x;
        for(_R int i = Log;i >= 0;i--){
            if(anc[x][i] == anc[y][i])continue;
            x = anc[x][i];
            y = anc[y][i];
        }
        return anc[x][0];
    }
}

int tar[_N];
int colorLCA[_N];
int ans = 0;
namespace Status1{
    int S[_N];
    int LCnd[_N];
    vector<pair<int, int> >dfn[_N];//DFN Id
    int tot = 0;
    void dfs_pre(int now, int fa){
        dfn[col[now]].push_back(make_pair(++tot, now));
        for(_R int i = head[now];i;i = edge[i].nxt){
            int to = edge[i].node;
            if(to == fa) continue;
            dfs_pre(to, now);
        }
    }
    int dfs(int now, int fa){
        int v = S[now];
        for(_R int i = head[now];i;i = edge[i].nxt){
            int to = edge[i].node;
            if(to == fa) continue;
            v += dfs(to, now);
        }
        if(v == m) ans = max(ans, MaxUp[now] + 1);
```

```
        return v;
    }
    void work(){
        dfs_pre(root, root);
        for(_R int i = 1;i <= m;i++) sort(dfn[i].begin(), dfn[i].end());
        for(_R int i = 1;i <= m;i++){
            S[dfn[i][0].second] ++;
            for(_R int j = 1;j < dfn[i].size();j++){
                S[LCA::query(dfn[i][j - 1].second, dfn[i][j].second)] --;
                S[dfn[i][j].second] ++;
            }
        }
        dfs(root, root);
    }
}

namespace Status2{
    void dfs(int now, int fa){
        for(_R int i = head[now];i;i = edge[i].nxt){
            int to = edge[i].node;
            if(to == fa) continue;
            dfs(to, now);tar[now] |= tar[to];
        }
    }
    void work(){
        dfs(root, root);
        for(_R int i = 1;i <= n;i++){
            if(tar[i] == 0){//反树 包含全集；
                ans = max(ans, MaxDeep[i] + 2);
            }
        }
    }
}
signed main()
{
    n = read(), m = read();
    int maxx = -1;
    for(_R int i = 1;i <= n;i++) col[i] = read(), maxx = max(maxx, col[i]);
    for(_R int i = 1;i < n;i++){
        int tmpx = read(), tmpy = read();
        add(tmpx, tmpy);
        add(tmpy, tmpx);
    }
    root = 1;
    dfs1(root, root);
    dfs2(root, root);
    LCA::work();
```

```
    for(_R int i = 1;i <= n;i++){
        int &cl = colorLCA[col[i]];
        if(cl == 0) cl = i;
        else cl = LCA::query(cl, i);
    }
    for(_R int i = 1;i <= m;i++) tar[colorLCA[i]] = 1;

    Status1::work();

    Status2::work();

    cout << ans << endl;
    return 0;
}
```