「杂题记录」彩色挂饰

Jiayi Su (ShuYuMo)

2021-01-19 15:35:06

一道 dp + 图论 + 状压好题

「杂题记录」彩色挂饰

题意简述

给定一张 n 个点 m 条边的无向图，有一些点有颜色，定义颜色相同的点组成的连通块为同色连通块。需要对没有颜色的点进行染色，最小化同色连通块的数量。

无向图满足每个点双的大小 $\leq s$ ，给出的颜色分别从为 1 到 k 编号。

$n \leq 10^5, 2 < k \leq 20, s \leq 6, n-1 \leq m < ns$

对于 10 % 的数据，$s = 2$.

分析

$s = 2$ 是一档很有启发性的部分分，可以感受到这就是一个树。

不妨设根节点编号为 1 直接树形 dp，设 $f(u, c)$ 为把以 u 为根节点的子树，u 染成颜色 c ，其余的任意染，最小的同色连通块数量。

$$f(u, c) = 1 + \sum_{(u,v)\in E} (f(v, c) - 1)$$

一个图怎么做？考虑用圆方树将无向图转化为树，然后进行树形 dp。$f(u, c)$ 的定义转化为以 u 为根的子树，圆点 或 方点的父亲染成的颜色。感觉是一类圆方树上 dp 的标准套路。对于圆点：

$$f(u, c) = 1 + \sum_{(u,v)\in E} (f(v, c) - 1)$$

对于方点: 简单 **dp** 一下就可以了。注意到 $s \leq 6$ ，瞎搞一下就可以了。注意以下的集合均指每个将一个点双上的点编号离散后的编号点集。

设：定义在点上的函数 $A(x)$ 为与 x 相连的点集。

设：定义在集合上的函数 $C(S)$ ，表示点集 S 是否连通。

$$\begin{aligned}
&C(\emptyset) = 1 \\
&C(S) = \bigvee_{x\in S} \left(C(S \setminus \{x\}) \wedge (A(x) \cap S \neq \emptyset)\right)
\end{aligned}$$

设：函数 $G(S,c)$ 为将连通块 S 涂成 c ，最小代价。每个连通块对应的子树也算入答案。

$$G(S,c)=\begin{cases}\mathrm{INF}, & C(S)=0\\ 1+\sum\limits_{x\in S}(f(x,c)-1) & \end{cases}$$

设：函数 $H(S)$ 为将连通块 S 涂上任意一种相同的颜色，最小代价。

$$H(S)=\min_{c=1}^{k}G(S,c).$$

设：函数 $L(S)$ 为将连通块 S 切成若干块，每一块涂上相同的颜色，最小代价。

$$L(S)=\min\{H(S),\ \min_{s\subseteq S}L(s)+L(S/s)\}$$

就可以求出

$$f(u,c)=\min_{S,u\in S}G(S,c)+L(S\ )$$

转移只需要 80 行就写完了。

```cpp
#include <cstdio>
#include <cstring>
#include <iostream>
#include <vector>
#include <algorithm>
#include <stack>
#include <cassert>
#include <set>

using namespace std;

const int _ = 4e5 + 100;
const int _K = 30;

int Col[_];
int n, m, k, s;
int head[_];
struct edges{ int node, nxt; } edge[_ << 1]; int tot = 0;
void add(int u, int v){ tot++; edge[tot].node = v; edge[tot].nxt  = head[u]; head[u] = tot; }

vector<int> G[_];
set<int> GG[_];

int dfn[_], low[_], dfc = 0, tmp;
int cnt;
stack<int> S;
void tarjan(int now) {
    dfn[now] = low[now] = ++dfc; S.push(now);
    for(int i = 0; i < (int)G[now].size(); i++) {
        int ex = G[now][i];
```

```
        if(!dfn[ex]) {
            tarjan(ex);
            low[now] = min(low[now], low[ex]);
            if(dfn[now] == low[ex]) {
                ++cnt;
                add(now, cnt); add(cnt, now);
                do add(cnt, tmp = S.top()), add(tmp, cnt), S.pop(); while(tmp != ex);
            }
        } else low[now] = min(low[now], dfn[ex]);
    }
}
int popcnt(int S) { int ans = 0; while(S) ans += ((S & 1) != 0), S >>= 1; return S;  }
int F[_][_K];
const int _S = 15;
int ver = 0;
int _INT_MAX_;
void d(int now, int fa){
    for(int i = head[now]; i; i = edge[i].nxt) {
        int ex = edge[i].node; if(ex == fa) continue;
        d(ex, now);
    }
    if(now <= n) {
        for(int i = 1; i <= k; i++) {
            if(Col[now]) if(Col[now] != i) { F[now][i] = _INT_MAX_; continue; }
            int &ans = F[now][i] = 1;
            for(int j = head[now]; j; j = edge[j].nxt) {
                if(edge[j].node == fa) continue;
                ans += F[edge[j].node][i] - 1;
            }
        }
    } else {
        static int Link[_S], iLink[_], NodeCnt; NodeCnt = 0;
        for(int i = head[now]; i; i = edge[i].nxt) {
            int ex = edge[i].node;
            Link[++NodeCnt] = ex;
            iLink[ex] = NodeCnt;
        }
        static int A[_S]; static bool C[1 << _S];
        for(int i = 1; i <= NodeCnt; i++) {
            int &ans = A[i] = 0;
            for(int j = 1; j <= NodeCnt; j++){ if(i == j) continue;
                int ex = Link[j]; if(GG[Link[i]].find(ex) == GG[Link[i]].end()) continue;
                ans |= (1 << (j - 1));
            }
        }
        for(int i = 0; i < (1 << NodeCnt); i++) C[i] = 0;
        C[0] = 1; // C[S] 重标号后的点集 S 是否连通.
```

```
        for(int i = 1; i <= NodeCnt; i++) C[1 << (i - 1)] = 1;
        for(int i = 1; i < (1 << NodeCnt); i++){
            bool &ans = C[i];
            for(int j = 1; j <= NodeCnt; j++){
                if(i & (1 << (j - 1))) ; else continue;
                ans = ( ans || (C[i ^ (1 << (j - 1))] && ((A[j] & i) != 0)) );
            }
        }
        static int G[1 << _S][_K]; // G[S][c] 把联通块 S 涂上颜色 c. 需要的次数.
        for(int i = 1; i < (1 << NodeCnt); i++){
            for(int j = 1; j <= k; j++){
                int &ans = G[i][j];
                if(!C[i]) { ans = _INT_MAX_; continue; }
                ans = 1;
                for(int l = 1; l <= NodeCnt; l++){
                    if(i & (1 << (l - 1))) if(l != iLink[fa]) ans += F[Link[l]][j] - 1;// , assert(F[Li
                }
            }
        }
        static int H[1 << _S]; // H[S] 把 连通块 S 涂上同一种颜色 所需要的 最小代价。
        H[0] = 0;
        for(int i = 1; i < (1 << NodeCnt); i++){
            int &ans = H[i] = _INT_MAX_;
            for(int j = 1; j <= k; j++) ans = min(ans, G[i][j]);
        }
        static int L[1 << _S]; L[0] = 0; // L[S] 把 连通块 S 分成若干块，每一块分别涂上相同的颜色 最小代价。
        for(int i = 1; i < (1 << NodeCnt); i++){
            int &ans = L[i] = H[i];
            for(int S0 = i; S0; S0 = (S0 - 1) & i){
                ans = min(ans, L[S0] + L[i ^ S0]);
            }
        }
        for(int i = 1; i <= k; i++){
            int &ans = F[now][i] = _INT_MAX_;
            if(Col[fa]) if(Col[fa] != i) continue;
            int S = (1 << (NodeCnt)) - 1; S ^= (1 << (iLink[fa] - 1));
            ans = min(ans, G[(1 << (iLink[fa] - 1))][i] + L[S]);
            for(int j = S; j ; j = (j - 1) & S){
                ans = min(ans, G[j | (1 << (iLink[fa] - 1))][i] + L[S ^ j]);
            }
        }
    }
}

int main(){
    // freopen("in.txt", "r", stdin);
    ios::sync_with_stdio(false);
```

```
    cin >> n >> m >> k >> s; cnt = n; _INT_MAX_ = 3 * n;
    for(int i = 1; i <= n; i++) cin >> Col[i];
    for(int i = 1; i <= m; i++){
        int u, v; cin >> u >> v;
        G[u].push_back(v); GG[u].insert(v);
        G[v].push_back(u); GG[v].insert(u);
    }
    tarjan(1);
    d(1, 1);
    int ans = _INT_MAX_;
    for(int i = 1; i <= k; i++) ans = min(ans, F[1][i]);
    printf("%d", ans);
    return 0;
}
```