提高组7

树的覆盖

题解

树 DP ， $DP[i][j][0/1/2]$ 表示以 i 为根，子树覆盖了 j 个点， $0/1/2$ 代表的意义如下：

0 表示当前节点和子节点都没被选中

1 表示当前节点被选中

2 表示存在子节点被选中

记录当前节点选中与否，是因为如果当前节点被选中，在子树范围之外，还能覆盖父节点。

每加入一棵子树，需要枚举子树的覆盖数量进行状态转移，这样看起来时间复杂度为 $O(n^3)$ ，但加入一个优化，即只枚举到当前子树大小进行转移，时间复杂度为 $O(n^2)$ 。

标准代码

C++ 11

```cpp
#include <bits/stdc++.h>
using namespace std;
const int mod = 1e9 + 7;
const int maxn = 3009;
int f[maxn][maxn][3], temp[maxn][3], a[maxn], siz[maxn], n;
vector<int>vec[maxn];
void dfs(int u, int fa)
{
    siz[u] = 1;
    f[u][0][0] = f[u][1][1] = 1;
    for (auto v : vec[u])
    {
        if (v == fa)    continue;
        dfs(v, u);
        for (int j = siz[u]; j >= 0; j--)
            for (int q = siz[v]; q >= 0; q--)
                for (int z1 = 0; z1 <= 2; z1++)//0表示自己和儿子没人放保安,1表示自己放了,2表示儿子放了
                    for (int z2 = 0; z2 <= 2; z2++)
                    {
                        int z = z1, x = j + q;
                        if (z1 == 0 && z2 == 1) z = 2, x++;
                        else if (z1 == 1 && z2 == 0)    x++;
                        temp[x][z] = (temp[x][z] + 1ll * f[u][j][z1] * f[v][q][z2] % mod) % mod;
                    }
        siz[u] += siz[v];
        for (int j = 0; j <= siz[u]; j++)
            for (int q = 0; q <= 2; q++)
            {
                f[u][j][q] = temp[j][q];
                temp[j][q] = 0;
            }
    }
}
signed main() {
    cin >> n;
    for (int i = 1; i < n; i++)
    {
        int x, y; cin >> x >> y;
        vec[x].push_back(y); vec[y].push_back(x);
    }
    dfs(1, 0);
```

```
    for (int i = 0; i <= n; i++)    cout << (1ll * f[1][i][0] + f[1][i][1] + f[1][i]
[2]) % mod << endl;
```

奇怪的树

题解

首先，显然是要优先选体积小的点，这样才能保证选得最多。

那么如何快速模拟这个过程呢？

考虑 $v_1 = v_2 = \ldots = v_n$ 的情况。

注意到可以把**选点**转化为**排除点**，也即优先排除体积大的点，直到剩余点的体积之和不超过箱子的容积。

并且，由于 v_i 相等，所以一个箱子的最优决策一定都来自于儿子的最优决策。

可以考虑使用可并堆来维护这些点。

对于一般情况，考虑用权值线段树来维护点。

求答案的过程可以用线段树上二分实现。

并且每次使用线段树合并 / 使用 DFS 序将子树转化为区间并结合主席树来解决。

复杂度 $O(n \log n)$ 。

当然，使用平衡树 + 启发式合并也是可以的。

标准代码

C++

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

const int BUFF_SIZE = 1 << 20;
char BUFF[BUFF_SIZE],*BB,*BE;

#define gc() (BB == BE ? (BE = (BB = BUFF) + fread(BUFF,1,BUFF_SIZE,stdin),BB == BE ? EOF : *BB++) : *BB++)
template<class T>
inline void read(T &x)
{
    x = 0;
    char ch = 0,w = 0;
    for(;ch < '0' || ch > '9';w |= ch == '-',ch = gc());
    for(;ch >= '0' && ch <= '9';x = (x << 3) + (x << 1) + (ch ^ '0'),ch = gc());
    w && (x = -x);
}

const int N = 2e5;
int n;
int to[(N << 1) + 5],pre[(N << 1) + 5],first[N + 5];
inline void add_edge(int u,int v)
{
    static int tot = 0;
    to[++tot] = v,pre[tot] = first[u],first[u] = tot;
}
int c[N + 5],a[N + 5];
long long v[N + 5],ans;
int ind[N + 5],len;
struct node
{
    int cnt;
    long long sum;
    int ls,rs;
} seg[(N << 5) + 10];
int rt[N + 5];
void insert(int x,int k,int &p,int tl,int tr)
{
    static int tot = 0;
    !p && (p = ++tot),seg[p].cnt += k,seg[p].sum += ind[x] * k;
    if(tl == tr)
        return ;
    int mid = tl + tr >> 1;
    x <= mid ? insert(x,k,seg[p].ls,tl,mid) : insert(x,k,seg[p].rs,mid + 1,tr);
```

```cpp
}
int query(long long v,int p,int tl,int tr)
{
    if(tl == tr)
        return min((long long)seg[p].cnt,v / ind[tl]);
    int mid = tl + tr >> 1;
    return v <= seg[seg[p].ls].sum ? query(v,seg[p].ls,tl,mid) : (seg[seg[p].ls].cnt + query(v - seg[seg[p].ls].sum,seg[p].rs,mid + 1,tr));
}
int merge(int x,int y)
{
    if(!x || !y)
        return x | y;
    seg[x].cnt += seg[y].cnt,seg[x].sum += seg[y].sum,
    seg[x].ls = merge(seg[x].ls,seg[y].ls),
    seg[x].rs = merge(seg[x].rs,seg[y].rs);
    return x;
}
int fa[N + 5];
int q[N + 5],head,tail;
int main()
{
    read(n);
    int x,y;
    for(register int i = 1;i < n;++i)
        read(x),read(y),add_edge(x,y),add_edge(y,x);
    for(register int i = 1;i <= n;++i)
        read(c[i]),read(v[i]),read(a[i]),ind[i] = c[i];
    sort(ind + 1,ind + n + 1),len = unique(ind + 1,ind + n + 1) - ind - 1;
    for(register int i = 1;i <= n;++i)
        c[i] = lower_bound(ind + 1,ind + len + 1,c[i]) - ind;
    q[++tail] = 1;
    for(register int p;head < tail;)
    {
        p = q[++head];
        for(register int i = first[p];i;i = pre[i])
            if(to[i] ^ fa[p])
                fa[to[i]] = p,q[++tail] = to[i];
    }
    for(register int i = n;i;--i)
        insert(c[q[i]],1,rt[q[i]],1,len),ans = max(ans,(long long)a[q[i]] * query(v[q[i]],rt[q[i]],1,len)),rt[fa[q[i]]] = merge(rt[fa[q[i]]],rt[q[i]]);
    printf("%lld\n",ans);
}
```

不降序列

题解

减掉 m 后得到的数字小于等于 0 ，字典序小于 1 到 m 的所有数字，但这个操作只能做一次。

每个数 K 只有保持不变和 - m 两种状态，因此考虑 $2-SAT$ 。

题目要求让所有数组保持字典序的不下降，但实际只需要处理相邻的两个数组。

如果存在后一个数组是前一个数组的前缀，根据字典序的规则肯定无解，输出 0 。

如果前一个是后一个的前缀，那么无需额外的处理。排除这两种情况后，那么相邻的两个数组一定存在某一位的不同。

如果 $a[i+1][j] < a[i][j]$ ，那么 $a[i][j]$ 要做减法，同时 $a[i+1][j]$ 必须保持不变。

如果 $a[i+1][j] > a[i][j]$ ，那么二者必须同时保持不变或同时做减法。

因此按照上述规则建边，设 u 表示 $a[i][j]$ 做减法， $u+n$ 表示 $a[i][j]$ 不变， v 表示 $a[i+1][j]$ 做减法，$v+n$ 表示 $a[i+1][j]$ 不变。

1. 如果 $u < v$ ：添加有向边 $v->u$ 和 $u+n->v+n$ 。
2. 如果 $u > v$ ：添加有向边 $u+n->u$ 和 $v->v+n$ 。

跑 $2-Sat$ 即可，具体方案也依此构造即可。

标准代码

C++ 11

```cpp
#include<bits/stdc++.h>
using namespace std;
const int maxN = 500005;
int a[maxN], b[maxN], c[maxN], vis[maxN];
vector<int>g[maxN];
int f, s, t;
void dfs(int x)
{
    c[++c[0]] = x;
    vis[x] = 1;
    if (x == t)f = 0;
    for (int i = 0; i < g[x].size(); i++)if (!vis[g[x][i]])dfs(g[x][i]);
}
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    s = m + 1, t = m + 2;
    for (int i = 1; i <= n; i++)
    {
        scanf("%d", &b[0]);
        for (int j = 1; j <= b[0]; j++)scanf("%d", &b[j]);
        if (i == 1) { for (int j = 0; j <= b[0]; j++)a[j] = b[j]; }
        int k = 1;
        while (k <= a[0] && k <= b[0] && a[k] == b[k])k++;
        if (k <= a[0] && k > b[0]) { printf("No\n"); return 0; }
        else if (k <= a[0] && k <= b[0])
        {
            if (a[k] > b[k])g[s].push_back(a[k]), g[b[k]].push_back(t); else g[b[k]].push_back(a[k]);
        }
        for (int j = 0; j <= b[0]; j++)a[j] = b[j];
    }
    c[0] = 0;
    f = 1;
    dfs(s);
    if (!f) { printf("No\n"); return 0; }
    printf("Yes\n%d\n", c[0] - 1);
    for (int i = 2; i <= c[0]; i++)printf("%d ", c[i]);
    return 0;
}
```

电报

题解

题面虽然看上去很复杂，但是可以把题意进行抽象化。考虑把每个字母看作图中的节点，关系看做边，那么“句子出现的概率”就等同于起点到终点的整条路径，除了终点以外每个节点的度数的乘积。该题其实问的就是：

> 给定 N 个节点， M 条边的无向图。从点 1 开始，每次等概率地选择与其相邻的任意一个节点并移动一步，给每条边赋上一个 $1 \sim m$ 的互不相同的权值，问第一次到达 m 的期望经过的边的权值和。

贪心选择期望经过次数最大的边赋上最小的权值，这个问题就变成了求每条边的期望经过次数。

由于没有对 m 的范围进行限定，那么 m 的最大值可以达到 $O(n^2)$ ，这是无法接受的，因此我们考虑先统计点的期望次数，计算完点的期望经过次数，就可以通过下述式子求出边的期望经过次数。

$$g_i = [u \neq n]\frac{f_u}{d_u} + [v \neq n]\frac{f_v}{d_v} \quad E_i = (u, v)$$

我们设 deg_i 表示第 i 个点的度数， f_i 表示第 i 个点期望经过次数，则点的期望经过次数满足如下关系。（ n 点不会对边的期望次数造成影响，所以考虑 n 点的期望经过次数是没有意义的。）

$$f_i = \begin{cases} f_1 = \sum_{(i,j)\in E, j\neq n} \frac{f_j}{deg_j} + 1 & i = 1 \\ f_i = \sum_{(i,j)\in E, j\neq n} \frac{f_j}{deg_j} & 1 < i < n \end{cases}$$

这看似是动态规划，但是由于满足后效性，环环相扣，该问题并不能用动态规划求解，观察到上述式子其实是一个 $n-1$ 个变量， $n-1$ 个式子的一次方程，使用高斯消元求解即可。之后就是贪心赋值了。

标准代码

C++ 11

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define N 510
#define eps 1e-10

int n, m, h[N], num = 0, du[N];
double a[N][N], b[N * N], ans = 0;
struct edge {
    int to, next;
}data[N * N];
inline void Gauss() {
    for (int i = 1; i <= n; ++i) {
        int r = i;
        for (int j = i + 1; j <= n; ++j) if (fabs(a[j][i]) > fabs(a[r][i])) r = j;
        if (r != i) for (int j = i; j <= n + 1; ++j) swap(a[r][j], a[i][j]);
        for (int j = i + 1; j <= n; ++j) {
            double t = a[j][i] / a[i][i];
            for (int k = i; k <= n + 1; ++k) a[j][k] -= a[i][k] * t;
        }
    }for (int i = n; i >= 1; --i) {
        for (int j = i + 1; j <= n; ++j) a[i][n + 1] -= a[i][j] * a[j][n + 1];
        a[i][n + 1] /= a[i][i];
    }
}
int main() {
    ios::sync_with_stdio(false);
    int x, y;
    cin >> n >> m;
    for (int i = 1; i <= m; ++i) {
        cin >> x >> y;
        data[++num].to = y; data[num].next = h[x]; h[x] = num; du[x]++;
        data[++num].to = x; data[num].next = h[y]; h[y] = num; du[y]++;
    }
    for (int x = 1; x < n; ++x) {
        a[x][x] = 1;
        for (int i = h[x]; i; i = data[i].next) {
            int y = data[i].to;
            if (y != n) a[x][y] -= 1.0 / du[y];
        }
    }
    a[1][n + 1] = 1; a[n][n + 1] = 1; a[n][n] = 1;
    Gauss();
    for (int i = 1; i <= m; ++i) {
```

```
        int x = data[i << 1].to, y = data[i * 2 - 1].to;
        if (x != n) b[i] += a[x][n + 1] / du[x];
        if (y != n) b[i] += a[y][n + 1] / du[y];
    }
    sort(b + 1, b + m + 1);
    for (int i = 1; i <= m; ++i)
        ans += b[i] * (m - i + 1);
    cout << ans << endl;
    return 0;
}


```