海蒂旅行问题解答分析

1. 问题的核心：期望值方程

首先，我们回顾一下海蒂旅行的期望花费问题。我们定义 E_u 为**海蒂当前在朋友 u 处，并准备进行下一次旅行时，她预计将花费的总金额**。

根据问题描述的规则：

- 如果朋友 u 是一个**叶子节点**（即 `deg[u] == 1`，只有一个朋友），那么海蒂的旅行将在此结束，她不会再被送往任何地方。因此，$E_u = 0$。
- 如果朋友 u **不是叶子节点**（即 `deg[u] > 1`），她会随机选择一个朋友 v 将海蒂送过去。每个朋友 v 被选中的概率是 $\frac{1}{\deg[u]}$。一旦海蒂被送到 v 处，她将支付旅行费用 $\text{cost}(u, v)$，并且从 v 处开始的预期花费是 E_v。
  因此，非叶子节点的 E_u 可以表示为：

$$E_u = \sum_{v \in \text{neighbors}(u)} \frac{1}{\deg[u]} (\text{cost}(u, v) + E_v)$$

这个方程可以重写为：

$$\deg[u] \cdot E_u = \sum_{v \in \text{neighbors}(u)} (\text{cost}(u, v) + E_v)$$

这是一个包含 N 个变量（$E_0, E_1, \ldots, E_{N-1}$）和 N 个方程的线性方程组。由于社交网络是一棵树，我们可以利用树的结构来高效地解决这个方程组，而不是使用通用的高斯消元法。

2. `ks` 和 `bs` 的意义：线性关系中的系数

解决树上期望值问题的一个标准方法是使用两次深度优先搜索（DFS），或者像你代码中这样，通过一次**自底向上（从叶子到根）的遍历**来计算系数，然后利用这些系数直接得到根节点的期望值。

我们假设树被任意地以节点 0 为根。对于树中的任何非根节点 u，它都有一个唯一的父节点 $\text{parent}(u)$。我们可以尝试将 E_u 表示成一个关于 $E_{\text{parent}(u)}$ 的线性函数：

$$E_u = A_u \cdot E_{\text{parent}(u)} + B_u$$

这里的 A_u 和 B_u 就是我们要在第一次遍历中计算的系数。

- `ks[u]` 存储的就是 A_u。
- `bs[u]` 存储的就是 B_u。

3. A_u 和 B_u 的推导（以及与代码的对应）

让我们将 E_u 的方程展开，并代入其子节点的线性关系：

对于非叶子节点 u（且 $u \neq \text{root}$）：

$$\deg[u] \cdot E_u = (\text{cost}(u, \text{parent}(u)) + E_{\text{parent}(u)}) + \sum_{v \in \text{children}(u)} (\text{cost}(u, v) + E_v)$$

我们将子节点 v 的 E_v 替换为 $A_v \cdot E_u + B_v$（因为 u 是 v 的父节点）：

$$\deg[u] \cdot E_u = (\text{cost}(u, \text{parent}(u)) + E_{\text{parent}(u)}) + \sum_{v \in \text{children}(u)} (\text{cost}(u, v) + A_v \cdot E_u + B_v)$$

现在，我们将所有包含 E_u 的项移到方程的左边，其他项移到右边：

$$\deg[u] \cdot E_u - \sum_{v \in \text{children}(u)} (A_v \cdot E_u) = \text{cost}(u, \text{parent}(u)) + E_{\text{parent}(u)} + \sum_{v \in \text{children}(u)} (\text{cost}(u, v) + B_v)$$

提出 E_u：

$$E_u \left(\deg[u] - \sum_{v \in \text{children}(u)} A_v \right) = E_{\text{parent}(u)} + \text{cost}(u, \text{parent}(u)) + \sum_{v \in \text{children}(u)} (\text{cost}(u, v) + B_v)$$

现在，我们将等式两边都除以 $(\deg[u] - \sum_{v \in \text{children}(u)} A_v)$，得到 E_u 的形式：

$$E_u = \frac{1}{\deg[u] - \sum_{v \in \text{children}(u)} A_v} \cdot E_{\text{parent}(u)} + \frac{\text{cost}(u, \text{parent}(u)) + \sum_{v \in \text{children}(u)} (\text{cost}(u, v) + B_v)}{\deg[u] - \sum_{v \in \text{children}(u)} A_v}$$

对照 $E_u = A_u \cdot E_{\text{parent}(u)} + B_u$：

- A_u **(对应** `ks[u]` **)**：

$$A_u = \frac{1}{\deg[u] - \sum_{v \in \text{children}(u)} A_v}$$

在代码中：
`sumk` 累加了所有子节点 `v` 的 `ks[v]`（即 $\sum A_v$）。
`denominator = (deg_u - sumk)` 。
`rev = pow(denominator, -1, mod)` 是分母的模逆。
`ks[u] = rev` ，这正是 A_u 的计算。

- B_u **(对应** `bs[u]` **)**：

$$B_u = \frac{\text{cost}(u, \text{parent}(u)) + \sum_{v \in \text{children}(u)} (\text{cost}(u, v) + B_v)}{\deg[u] - \sum_{v \in \text{children}(u)} A_v}$$

在代码中：
`sumb` 累加了所有子节点 `v` 的 `bs[v]`（即 $\sum B_v$）。
`sumc` 累加了所有子节点 `v` 的 `cost(u,v)`（即 $\sum \text{cost}(u, v)$）。
`bs[u] = (sumb + sumc) * rev % mod` 。
注意： 这里的 `bs[u]` 计算中**缺少了** $\text{cost}(u, \text{parent}(u))$ 这一项。这通常意味着这个项会在第二次 DFS 中被处理，或者，对于根节点来说，这个项不存在。

4. 叶子节点的处理（基准情况）

对于叶子节点 u（且 $u \neq \text{root}$）：
根据规则，$E_u = 0$。
所以，当 `u` 是叶子节点时，`ks[u]` 和 `bs[u]` 都应该为 0。
代码中 `if len(path[u]) == 1 and u > 0: continue` 这一行，使得叶子节点的 `ks[u]` 和 `bs[u]` 保持其初始值 0，这与 $E_u = 0$ 的定义是吻合的。

5. 为什么 `bs[0]` 是最终答案？

代码的最后一行是 `print(bs[0])` 。这表示根节点（节点 0）的 B_0 值就是最终答案。
让我们看看根节点 E_0 的方程。根节点没有父节点，所以它的方程是：

$$\deg[0] \cdot E_0 = \sum_{v \in \text{children}(0)} (\text{cost}(0, v) + E_v)$$

我们将子节点 v 的 E_v 替换为 $A_v \cdot E_0 + B_v$：

$$\deg[0] \cdot E_0 = \sum_{v \in \text{children}(0)} (\text{cost}(0, v) + A_v \cdot E_0 + B_v)$$

$$\deg[0] \cdot E_0 = \sum_{v \in \text{children}(0)} (\text{cost}(0, v) + B_v) + E_0 \sum_{v \in \text{children}(0)} A_v$$

移项并提出 E_0：

$$E_0 \left(\deg[0] - \sum_{v \in \text{children}(0)} A_v \right) = \sum_{v \in \text{children}(0)} (\text{cost}(0, v) + B_v)$$

所以：

$$E_0 = \frac{\sum_{v \in \text{children}(0)} (\text{cost}(0, v) + B_v)}{\deg[0] - \sum_{v \in \text{children}(0)} A_v}$$

现在，我们对照代码中计算 `bs[0]` 的部分：

- `sumk` 对应 $\sum_{v \in \text{children}(0)} A_v$
- `sumb` 对应 $\sum_{v \in \text{children}(0)} B_v$
- `sumc` 对应 $\sum_{v \in \text{children}(0)} \text{cost}(0, v)$
- `denominator = (deg_u - sumk)` 对应分母 $\deg[0] - \sum_{v \in \text{children}(0)} A_v$
- `rev` 是分母的模逆。
- `bs[0] = (sumb + sumc) * rev % mod`，这正是 $\frac{\sum_{v \in \text{children}(0)} (\text{cost}(0,v)+B_v)}{\deg[0]-\sum_{v \in \text{children}(0)} A_v}$ 的计算。

因此，`bs[0]` 实际上直接计算出了 E_0 的值。由于问题没有明确指定起始点，通常这种情况下会默认计算从根节点（这里是节点 0）开始的期望花费。

总结正确性

这段代码通过一次**自底向上的动态规划**（利用 BFS 队列的逆序遍历实现了后序遍历的效果），巧妙地解决了树上期望值计算的线性方程组。

1. 它定义了 $E_u = A_u \cdot E_{\text{parent}(u)} + B_u$ 这样的线性关系。
2. 在第一次遍历中，它为每个节点计算了 A_u（`ks[u]`）和 B_u（`bs[u]`）。这些计算依赖于其子节点已经计算好的 A_v 和 B_v。
3. 对于叶子节点，其 $E_u = 0$，所以 $A_u = 0, B_u = 0$ 作为基准。

4. 对于根节点（节点 0），由于它没有父节点，其 E_0 的方程形式简化，并且可以直接通过其子节点的 A_v 和 B_v 计算出来。代码中的 `bs[0]` 正是这个计算结果。

这种方法避免了显式地构建和求解整个线性方程组，而是利用了树的结构特性，在 $O(N)$ 的时间复杂度内高效地得到了答案。