题目描述

微商模式比较典型，下级每赚 100 元就要上交 15 元，给出每个级别的收入，求出金字塔尖上的人收入。

输入描述

第一行输入N，表示有N个代理商上下级关系
接下来输入N行，每行三个数：

> 代理商代号 上级代理商代号 代理商赚的钱

输出描述

输出一行，两个以空格分隔的整数，含义如下

> 金字塔顶代理商 最终的钱数

用例

输入	3 1 0 223 2 0 323 3 2 1203
输出	0 105
说明	2的最终收入等于323 + 1203/100*15=323 + 180 0的最终收入等于（323 + 180 + 223）/ 100 * 15 = 105

输入	4 1 0 100 2 0 200 3 0 300 4 0 200
输出	0 120
说明	无

题目解析

这道题看上去平平无奇，但是实际操作起来却有点无从下口。

首先，从用例来看，上下级关系，不是连续，比如2的上级不是1，而是0。

因此，我们需要用到输入中的上下级关系。

我的解题思路如下：

将每行输入转为一个数组，数组包含三个要素：[本级id，上级id，本级收入]，这样就会得到一个二维数组。

然后对二维数组排序，按照上级id进行降序（因为0是最高级）。

定义一个agent数组，agent[i]代表第i级的收入。

遍历二维数组，根据遍历得到的一维数组：[本级id，上级id，本级收入]

可以得到 agent[上级id] += 本级收入 / 100 * 15,

但是此时 agent[上级id] 可能并未初始化，因此我们需要判断是否需要初始化。

另外，我们还需要注意，遍历到第二个一维数组时，agent[本级id]可能有了下级上交的收入，因此：本级收入 += agent[本级id]，当然前提是agent[本级id]有值。

最终我们就可以通过agent[0]获取到金字塔顶级代理的收入了。

本题的顶级代理商可能不是0，因此我们需要在对二维数组排序后，取得顶级代理商的代号first

JavaScript算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let n;
rl.on("line", (line) => {
  lines.push(line);

```

```
  if (lines.length == 1) {
    n = lines[0] - 0;
  }

  if (n && lines.length == n + 1) {
    lines.shift();
    const arr = lines.map((line) => line.split(" ").map(Number));
    console.log(getResult(arr));
    lines.length = 0;
  }
});

function getResult(arr) {
  const agent = {};

  arr.sort((a, b) => b[1] - a[1]);
  const first = arr.at(-1)[1];

  arr.forEach((ele) => {
    let [id, preId, money] = ele;
    if (agent[id]) money += agent[id];

    if (!agent[preId]) agent[preId] = 0;
    agent[preId] += Math.floor(money / 100) * 15;
  });

  return `${first} ${agent[first]}`;
}
```

Java算法源码

```
import java.util.Arrays;
import java.util.HashMap;
import java.util.Scanner;

public class Main {
  // 输入获取
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();
    int[][] arr = new int[n][3];
    for (int i = 0; i < n; i++) {
      arr[i][0] = sc.nextInt();
      arr[i][1] = sc.nextInt();
      arr[i][2] = sc.nextInt();
    }

    System.out.println(getResult(arr));
  }

```

```java
  // 算法入口
  public static String getResult(int[][] arr) {
    HashMap<Integer, Integer> agent = new HashMap<>();

    Arrays.sort(arr, (a, b) -> b[1] - a[1]);

    int first = arr[arr.length - 1][1];

    for (int[] ele : arr) {
      int id = ele[0];
      int preId = ele[1];
      int money = ele[2];

      if (agent.containsKey(id)) money += agent.get(id);
      agent.put(preId, agent.getOrDefault(preId, 0) + money / 100 * 15);
    }

    return first + " " + agent.get(first);
  }
}
```

Python算法源码

```python
# 输入获取
n = int(input())
arr = [list(map(int, input().split())) for _ in range(n)]


# 算法入口
def getResult():
    agent = {}

    arr.sort(key=lambda x: -x[1])

    first = arr[-1][1]

    for id, preId, money in arr:
        if agent.get(id) is not None:
            money += agent[id]

        if agent.get(preId) is None:
            agent[preId] = 0

        agent[preId] += money // 100 * 15

    return f"{first} {agent[first]}"


# 算法调用
print(getResult())
```