

【BFS】-BOSS的收入

题目描述与示例

题目描述

一个 XX 产品行销总公司，只有一个boss，其有若干一级分销，一级分销又有若干二级分销，每个分销只有唯一的上级分销。规定，每个月，下级分销需要将自己的总收入(自己的+下级上交的)每满 100 元上交 15 元给自己的上级。

现给出一组分销的关系，和每个分销的收入，请找出boss并计算出这个boss的收入。

比如：

收入 100 元，上交 15 元；

收入 199 元（99 元不够 100），上交 15 元，

收入 200 元，上交 30 元。

输入：

分销关系和收入：[[分销id 上级分销的Id 收入]，[分销id 上级分销的id 收入]，[分销id 上级分销的id 收入]]

分销ID范围 0..65535

收入范围 0..65535，单位元

提示：输入的数据只存在 1 个boss，不存在环路

输出：[boss的ID，总收入]

输入描述

第 1 行输入关系的总数量 N

第 2 行开始，输入关系信息，格式： 分销ID 上级分销ID 收入

比如：

5

1 0 100

2 0 199

3 0 200

4 0 200

5 0 200

输出描述

输出： boss的ID 总收入

比如：

0 120

补充说明

给定的输入数据都是合法的，不存在环路，重复的

示例

输入

```
1 5
2 1 0 100
3 2 0 199
4 3 0 200
5 4 0 200
6 5 0 200
```

输出

```
1 0 120
```

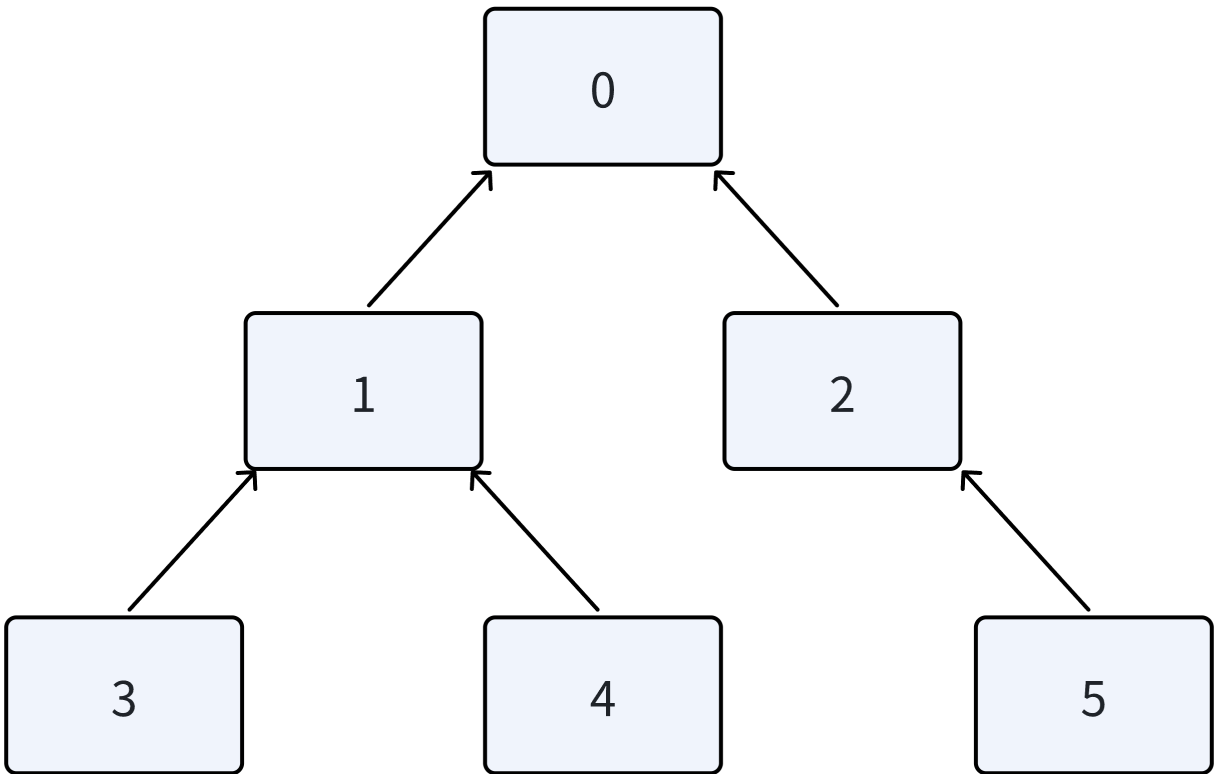
解题思路

拓扑排序BFS解法

很明显这个层层分销的制度，可以使用**树形结构**来表示。譬如对于例子

```
1 5
2 1 0 100
3 2 0 199
4 3 1 200
5 4 1 200
6 5 2 200
```

可以画成如下树形结构



每一个上级的收入不仅取决于他自己的收入，还取决于其直接下属的收入。

以这个例子为例，如果我们想计算节点 0 从节点 1 得到多少收入，就必须先计算节点 1 的总收入。而如果想计算节点 1 的总收入，又必须先计算节点 1 从节点 3 和节点 4 分别获得多少收入。

很显然这存在依赖关系：**我们必须先把下层节点的总收入计算完之后，才能将总收入进行抽成，来计算当前节点的总收入。**

对于这种存在依赖的问题，我们可以使用**拓扑排序**来完成。直接套模板即可完成。

注意本题并没有直接告知根节点的ID，因此需要找到唯一的非子节点来作为根节点。

*自底向上的DFS解法

当然，熟悉树和递归的同学，也容易想到能够用**自底向上的DFS**来完成这个题目。

本文不做赘述，后面提供包含详尽注释的代码。

代码

解法一：拓扑排序BFS

Python

```
1 # 题目：【BFS】2024E-BOSS的收入
2 # 分值：200
3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：BFS/拓扑排序
5 # 代码看不懂的地方，请直接在群上提问
6
7 from collections import defaultdict, deque
8
```

```
9 # 输入边的个数
10 n = int(input())
11
12 # 构建邻接表, key是子节点, value是父节点
13 # 由于每一个节点最多只有一个父节点
14 # 所以parents邻接表的value无需设置默认值为list
15 parents = dict()
16
17 # 构建总收入哈希表, key是节点名, value是该节点的收入
18 total_money = defaultdict(int)
19
20 # 构建入度哈希表, key是节点名, value是入度
21 indegree = defaultdict(int)
22
23
24 # 循环n行, 输入n行
25 for _ in range(n):
26     # 在实际考试中, 发现必须加入这里的try-except语句才能够满分
27     # 题目的输入存在一些未知的错误, 不加上只能够通过95%的用例
28     try:
29         # 输入子节点c, 父节点p, 子节点最初的收入money
30         # c表示children, p表示parent
31         c, p, money = map(int, input().split())
32         # 在parents邻接表中储存c的父节点为p
33         parents[c] = p
34         # 在总收入哈希表中初始化c的收入, 记录为money
35         total_money[c] = money
36         # 后续需要进行拓扑排序, 父节点p的入度+1
37         indegree[p] += 1
38     except:
39         break
40
41
42 # 寻找boss根节点root, 根节点存在以下特征:
43 # 1. 入度不为0 (位于indegree的key中)
44 # 2. 不是任何一个节点的子节点 (不位于parents的key中)
45 for node in indegree.keys():
46     if node not in parents:
47         root = node
48         break
49
50 # 构建队列q维护拓扑排序过程
51 q = deque()
52 # 所有入度为0的节点, 都是初始的叶节点, 存入队列q中
53 for c in parents.keys():
54     if indegree[c] == 0:
55         q.append(c)
```

```

56
57 # 拓扑排序过程
58 while q:
59     # 弹出队头元素，为当前节点c
60     c = q.popleft()
61     # 如果遍历到根节点root，则直接退出循环
62     if c == root:
63         break
64     # 获得当前节点c的父节点p
65     p = parents[c]
66     # 父节点的入度+1
67     indegree[p] -= 1
68     # 弹出的当前节点c的收入已经计算完毕
69     # 将其收入整除100后乘15，是提供给父节点p的分销佣金
70     # 将该分销佣金加入父节点p的收入中
71     total_money[p] += total_money[c] // 100 * 15
72     # 若此时父节点的入度为0，则说明其所有子节点均已考虑
73     # 该父节点p的总收入计算完毕，将其加入队列
74     if indegree[p] == 0:
75         q.append(p)
76
77 # 输出root的id以及其收入total_money[root]
78 print(f"{root} {total_money[root]}")

```

Java

```

1 import java.util.*;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Scanner scanner = new Scanner(System.in);
7         int n = scanner.nextInt();
8
9         // 构建邻接表，key是子节点，value是父节点
10        Map<Integer, Integer> parents = new HashMap<>();
11
12        // 构建总收入哈希表，key是节点名，value是该节点的收入
13        Map<Integer, Integer> totalMoney = new HashMap<>();
14
15        // 构建入度哈希表，key是节点名，value是入度
16        Map<Integer, Integer> indegree = new HashMap<>();
17
18        // 循环n行，输入n行
19        for (int i = 0; i < n; i++) {

```

```

20         try {
21             // 输入子节点c, 父节点p, 子节点最初的收入money
22             int c = scanner.nextInt();
23             int p = scanner.nextInt();
24             int money = scanner.nextInt();
25
26             // 在parents邻接表中储存c的父节点为p
27             parents.put(c, p);
28
29             // 在总收入哈希表中初始化c的收入, 记录为money
30             totalMoney.put(c, money);
31
32             // 后续需要进行拓扑排序, 父节点p的入度+1
33             indegree.put(p, indegree.getOrDefault(p, 0) + 1);
34
35         } catch (Exception e) {
36             break;
37         }
38     }
39
40     // 寻找boss根节点root, 根节点存在以下特征:
41     // 1. 入度不为0 (位于indegree的key中)
42     // 2. 不是任何一个节点的子节点 (不位于parents的key中)
43     int root = -1;
44     for (int node : indegree.keySet()) {
45         if (!parents.containsKey(node)) {
46             root = node;
47             break;
48         }
49     }
50
51     // 构建队列q维护拓扑排序过程
52     Queue<Integer> q = new LinkedList<>();
53     // 所有入度为0的节点, 都是初始的叶节点, 存入队列q中
54     for (int c : parents.keySet()) {
55         if (indegree.getOrDefault(c, 0) == 0) {
56             q.offer(c);
57         }
58     }
59
60     // 拓扑排序过程
61     while (!q.isEmpty()) {
62         // 弹出队头元素, 为当前节点c
63         int c = q.poll();
64
65         // 如果遍历到根节点root, 则直接退出循环
66         if (c == root) {

```

```

67         break;
68     }
69
70     // 获得当前节点c的父节点p
71     int p = parents.get(c);
72
73     // 父节点的入度-1
74     indegree.put(p, indegree.get(p) - 1);
75
76     // 弹出的当前节点c的收入已经计算完毕
77     // 将其收入整除100后乘15，是提供给父节点p的分销佣金
78     // 将该分销佣金加入父节点p的收入中
79     totalMoney.put(p, totalMoney.getOrDefault(p, 0) +
totalMoney.get(c) / 100 * 15);
80
81     // 若此时父节点的入度为0，则说明其所有子节点均已考虑
82     // 该父节点p的总收入计算完毕，将其加入队列
83     if (indegree.get(p) == 0) {
84         q.offer(p);
85     }
86 }
87
88 // 输出root的id以及其收入totalMoney.get(root)
89 System.out.println(root + " " + totalMoney.get(root));
90
91 scanner.close();
92 }
93 }
94

```

C++

```

1  #include <iostream>
2  #include <unordered_map>
3  #include <queue>
4
5  using namespace std;
6
7  int main() {
8      int n;
9      cin >> n;
10
11     // 构建邻接表，key是子节点，value是父节点
12     unordered_map<int, int> parents;
13

```



```

14 // 构建总收入哈希表, key是节点名, value是该节点的收入
15 unordered_map<int, int> total_money;
16
17 // 构建入度哈希表, key是节点名, value是入度
18 unordered_map<int, int> indegree;
19
20 // 循环n行, 输入n行
21 for (int i = 0; i < n; i++) {
22     // 在实际考试中, 发现必须加入这里的try-catch语句才能够满分
23     // 题目的输入存在一些未知的错误, 不加上只能够通过95%的用例
24     try {
25         // 输入子节点c, 父节点p, 子节点最初的收入money
26         int c, p, money;
27         cin >> c >> p >> money;
28
29         // 在parents邻接表中储存c的父节点为p
30         parents[c] = p;
31
32         // 在总收入哈希表中初始化c的收入, 记录为money
33         total_money[c] = money;
34
35         // 后续需要进行拓扑排序, 父节点p的入度+1
36         indegree[p]++;
37     } catch (...) {
38         break;
39     }
40 }
41
42 // 寻找boss根节点root, 根节点存在以下特征:
43 // 1. 入度不为0 (位于indegree的key中)
44 // 2. 不是任何一个节点的子节点 (不位于parents的key中)
45 int root = -1;
46 for (const auto& pair : indegree) {
47     int node = pair.first;
48     if (parents.find(node) == parents.end()) {
49         root = node;
50         break;
51     }
52 }
53
54 // 构建队列q维护拓扑排序过程
55 queue<int> q;
56 // 所有入度为0的节点, 都是初始的叶节点, 存入队列q中
57 for (const auto& pair : parents) {
58     int c = pair.first;
59     if (indegree[c] == 0) {
60         q.push(c);

```

```

61     }
62 }
63
64 // 拓扑排序过程
65 while (!q.empty()) {
66     // 弹出队头元素，为当前节点c
67     int c = q.front();
68     q.pop();
69
70     if (c == root)
71         break;
72
73     // 获得当前节点c的父节点p
74     int p = parents[c];
75
76     // 父节点的入度-1
77     indegree[p]--;
78
79     // 弹出的当前节点c的收入已经计算完毕
80     // 将其收入整除100后乘15，是提供给父节点p的分销佣金
81     // 将该分销佣金加入父节点p的收入中
82     total_money[p] += total_money[c] / 100 * 15;
83
84     // 若此时父节点的入度为0，则说明其所有子节点均已考虑
85     // 该父节点p的总收入计算完毕，将其加入队列
86     if (indegree[p] == 0) {
87         q.push(p);
88     }
89 }
90
91 // 输出root的id以及其收入total_money[root]
92 cout << root << " " << total_money[root] << endl;
93
94 return 0;
95 }
96

```

时空复杂度

时间复杂度： $O(N)$ 。需要遍历每一个节点。

空间复杂度： $O(N)$ 。入度哈希表和邻接表所占空间。

*解法二：自底向上的DFS

Python

```
1 # 题目：【BFS】2024E-BOSS的收入
2 # 分值：200
3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：自底向上DFS
5 # 代码看不懂的地方，请直接在群上提问
6
7 from collections import defaultdict
8
9
10 # 自底向上的dfs函数
11 # node为当前节点
12 # neighbor_dic为邻接表
13 # total_money为每一个节点的总收入
14 def dfs(node, neighbor_dic, total_money):
15     # 如果当前节点node不是一个父节点，则说明其为叶子节点
16     # 叶节点的总收入无需修改
17     # 直接返回
18     if node not in neighbor_dic:
19         return
20
21     # 考虑当前节点的所有子节点child
22     for child in neighbor_dic[node]:
23         # 自底向上，先对子节点进行DFS调用，更新子节点的总收入
24         dfs(child, neighbor_dic, total_money)
25         # 子节点的DFS调用后，子节点的总收入已经计算完毕
26         # 将其更新入当前节点中
27         total_money[node] += total_money[child] // 100 * 15
28
29     return
30
31 # 输入边的个数
32 n = int(input())
33
34 # 构建表示树形结构的邻接表
35 # 其中key是节点名，value是其所有子节点children构成的列表
36 neighbor_dic = defaultdict(list)
37
38 # 构建总收入哈希表，key是节点名，value是该节点的收入
39 total_money = defaultdict(int)
40
```

```

41
42 # 循环n行，输入n行
43 for _ in range(n):
44     # 在实际考试中，发现必须加入这里的try-except语句才能够满分
45     # 题目的输入存在一些未知的错误，不加上只能够通过95%的用例
46     try:
47         # 输入子节点c，父节点p，子节点最初的收入money
48         # c表示children, p表示parent
49         c, p, money = map(int, input().split())
50         # 在邻接表中储存节点p的子节点包含c
51         neighbor_dic[p].append(c)
52         # 在总收入哈希表中初始化c的收入，记录为money
53         total_money[c] = money
54     except:
55         break
56
57
58 # 寻找根节点，存在于neighbor_dic中，
59 # 且未不位于total_money中的节点为根节点
60 for p in neighbor_dic:
61     if p not in total_money:
62         root = p
63         break
64
65 # 递归入口，调用根节点root
66 dfs(root, neighbor_dic, total_money)
67
68 # 输出root的id以及其收入total_money[root]
69 print(f"{root} {total_money[root]}")

```

Java

```

1 import java.util.*;
2
3 public class Main {
4
5     // 自底向上的DFS函数
6     // node为当前节点
7     // neighborDic为邻接表
8     // totalMoney为每一个节点的总收入
9     public static void dfs(int node, Map<Integer, List<Integer>> neighborDic,
10 Map<Integer, Integer> totalMoney) {
11         // 如果当前节点node不是一个父节点，则说明其为叶子节点
12         // 叶节点的总收入无需修改，直接返回
13         if (!neighborDic.containsKey(node)) {

```

```

13         return;
14     }
15
16     // 考虑当前节点的所有子节点child
17     for (int child : neighborDic.get(node)) {
18         // 自底向上, 先对子节点进行DFS调用, 更新子节点的总收入
19         dfs(child, neighborDic, totalMoney);
20         // 子节点的DFS调用后, 子节点的总收入已经计算完毕
21         // 将其更新入当前节点中
22         totalMoney.put(node, totalMoney.getOrDefault(node, 0) +
totalMoney.get(child) / 100 * 15);
23     }
24 }
25
26 public static void main(String[] args) {
27     Scanner scanner = new Scanner(System.in);
28     int n = scanner.nextInt();
29
30     // 构建表示树形结构的邻接表
31     Map<Integer, List<Integer>> neighborDic = new HashMap<>();
32
33     // 构建总收入哈希表, key是节点名, value是该节点的收入
34     Map<Integer, Integer> totalMoney = new HashMap<>();
35
36     // 循环n行, 输入n行
37     for (int i = 0; i < n; i++) {
38         try {
39             // 输入子节点c, 父节点p, 子节点最初的收入money
40             int c = scanner.nextInt();
41             int p = scanner.nextInt();
42             int money = scanner.nextInt();
43
44             // 在邻接表中储存节点p的子节点包含c
45             neighborDic.computeIfAbsent(p, k -> new ArrayList<>()).add(c);
46
47             // 在总收入哈希表中初始化c的收入, 记录为money
48             totalMoney.put(c, money);
49         } catch (Exception e) {
50             break;
51         }
52     }
53
54     // 寻找根节点, 存在于neighborDic中, 且未不位于totalMoney中的节点为根节点
55     int root = -1;
56     for (int p : neighborDic.keySet()) {
57         if (!totalMoney.containsKey(p)) {
58             root = p;

```

```

59         break;
60     }
61 }
62
63 // 递归入口，调用根节点root
64 dfs(root, neighborDic, totalMoney);
65
66 // 输出root的id以及其收入totalMoney.get(root)
67 System.out.println(root + " " + totalMoney.get(root));
68
69 scanner.close();
70 }
71 }
72

```

C++

```

1  #include <iostream>
2  #include <unordered_map>
3  #include <vector>
4  #include <queue>
5
6  using namespace std;
7
8  // 自底向上的DFS函数
9  // node为当前节点
10 // neighborDic为邻接表
11 // totalMoney为每一个节点的总收入
12 void dfs(int node, unordered_map<int, vector<int>>& neighborDic,
13 unordered_map<int, int>& totalMoney) {
14     // 如果当前节点node不是一个父节点，则说明其为叶子节点
15     // 叶节点的总收入无需修改，直接返回
16     if (neighborDic.find(node) == neighborDic.end()) {
17         return;
18     }
19     // 考虑当前节点的所有子节点child
20     for (int child : neighborDic[node]) {
21         // 自底向上，先对子节点进行DFS调用，更新子节点的总收入
22         dfs(child, neighborDic, totalMoney);
23         // 子节点的DFS调用后，子节点的总收入已经计算完毕
24         // 将其更新入当前节点中
25         totalMoney[node] += totalMoney[child] / 100 * 15;
26     }
27 }

```

```

28
29 int main() {
30     int n;
31     cin >> n;
32
33     // 构建表示树形结构的邻接表
34     unordered_map<int, vector<int>> neighborDic;
35
36     // 构建总收入哈希表, key是节点名, value是该节点的收入
37     unordered_map<int, int> totalMoney;
38
39     // 循环n行, 输入n行
40     for (int i = 0; i < n; i++) {
41         try {
42             // 输入子节点c, 父节点p, 子节点最初的收入money
43             int c, p, money;
44             cin >> c >> p >> money;
45
46             // 在邻接表中储存节点p的子节点包含c
47             neighborDic[p].push_back(c);
48
49             // 在总收入哈希表中初始化c的收入, 记录为money
50             totalMoney[c] = money;
51         } catch (...) {
52             break;
53         }
54     }
55
56     // 寻找根节点, 存在于neighborDic中, 且未不位于totalMoney中的节点为根节点
57     int root = -1;
58     for (const auto& pair : neighborDic) {
59         int p = pair.first;
60         if (totalMoney.find(p) == totalMoney.end()) {
61             root = p;
62             break;
63         }
64     }
65
66     // 递归入口, 调用根节点root
67     dfs(root, neighborDic, totalMoney);
68
69     // 输出root的id以及其收入totalMoney[root]
70     cout << root << " " << totalMoney[root] << endl;
71
72     return 0;
73 }

```

时空复杂度

时间复杂度： $O(N)$ 。需要遍历每一个节点。

空间复杂度： $O(N)$ 。邻接表所占空间。