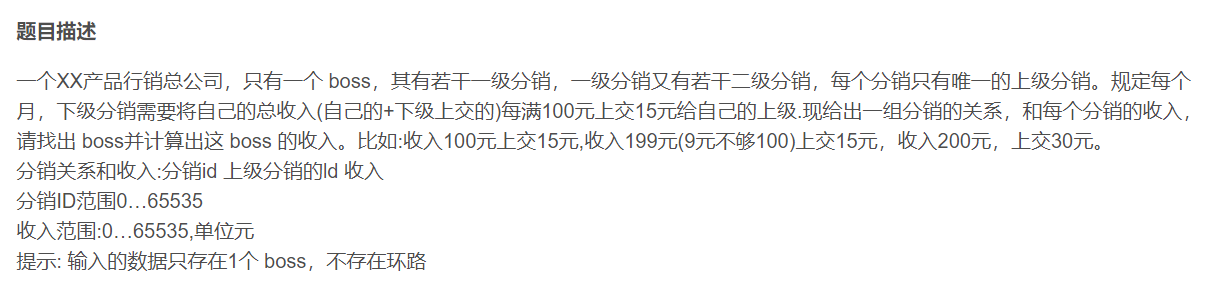
# **E卷-boss的收入[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**



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

补充说明

给定的输入数据都是合法的，不存在重复

示例1

输入

5

1 0 100

2 0 200

3 0 300

4 0 200

5 0 200

输出

0 150

示例2

输入

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=10

思路:使用一个队列进行广度优先搜索(BFS)。从队列中取出当前分销IDx，获取其上级分销ID y，更新收入并更新上级分销的下级计

数器 in。如果某个上级分销没有下级分销了，将其加入队列。重复此过程直至队列为空。

Java

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt(); // 读入关系的总数量N

long[] fa = new long[100010]; // 记录每个分销的上级分销ID

long[] mon = new long[100010]; // 记录每个分销的收入

Arrays.fill(fa,

-1); // 初始化fa数组，全部设置为-1，表示未知上级

List<Integer> in = new ArrayList<>(Collections.nCopies(100010,

0)); // 记录每个分销被定义为上级的次数

Set<Integer> st = new HashSet<>(); // 用于存储所有出现的分销ID

for (int i = 0; i < n; i++) {

int x = scanner.nextInt();

int y = scanner.nextInt();

int z = scanner.nextInt();

st.add(x); // 将分销ID x 加入集合

st.add(y); // 将上级分销ID y 加入集合

in.set(y, in.get(y) + 1); // 记录上级分销ID y 被定义为上级的次数

fa[x] = y; // 设置分销 x 的上级分销ID 为 y

mon[x] = z; // 设置分销 x 的收入为 z

}

Queue<Integer> que = new

LinkedList<>(); // 创建一个队列用于处理没有下级的分销

for (int x : st)

if (in.get(x) == 0)

que.offer(x); // 如果某分销ID没有下级分销，将其加入队列

long ans = 0; // 最终输出的 boss 收入

int x = 0; // 最终输出的 boss ID

while (!que.isEmpty()) {

x = que.poll(); // 获取队列中的第一个元素

ans = mon[x]; // 收入设置为当前的收入（会随程序运行最后是 boss 的收入）

int y = (int) fa[x]; // 获取分销 x 的上级分销ID

if (y == -1)

continue; // 如果上级分销ID是-1，跳过此次循环（可能是根节点）

in.set(y, in.get(y) - 1); // 上级分销 y 的下级数量减1

if (in.get(y) == 0) que.offer(

y); // 如果上级分销 y 没有下级分销了，将其加入队列

// 计算并累加上级分销 y 的收入：当前收入的15%上交给上级

mon[y] += mon[x] / 100 \* 15;

}

// 输出最终的 boss ID 和 boss 的总收入

System.out.println(x + " " + ans);

}

}

Python3

def main():

import sys

input = sys.stdin.read

data = input().split()

n = int(data[0]) # 读取关系的总数量N

index = 1

fa = [-1] \* 100010 # 记录每个分销的上级分销ID

mon = [0] \* 100010 # 记录每个分销的收入

in\_degree = [0] \* 100010 # 记录每个分销被定义为上级的次数

st = set() # 用于存储所有出现的分销ID

for i in range(n):

x = int(data[index])

y = int(data[index + 1])

z = int(data[index + 2])

index += 3

st.add(x)

st.add(y)

in\_degree[y] += 1 # 记录上级分销ID y 被定义为上级的次数

fa[x] = y # 设置分销 x 的上级分销ID 为 y

mon[x] = z # 设置分销 x 的收入为 z

queue = [] # 创建一个队列用于处理没有下级的分销

for x in st:

if in\_degree[x] == 0:

queue.append(x) # 如果某分销ID没有下级分销，将其加入队列

ans = 0 # 最终输出的 boss 收入

x = 0 # 最终输出的 boss ID

while queue:

x = queue.pop(0) # 获取队列中的第一个元素

ans = mon[x] # 收入设置为当前的收入（会随程序运行最后是 boss 的收入）

y = fa[x] # 获取分销 x 的上级分销ID

if y == -1:

continue # 如果上级分销ID是-1，跳过此次循环（可能是根节点）

in\_degree[y] -= 1 # 上级分销 y 的下级数量减1

if in\_degree[y] == 0:

queue.append(y) # 如果上级分销 y 没有下级分销了，将其加入队列

# 计算并累加上级分销 y 的收入：当前收入的15%上交给上级

mon[y] += mon[x] // 100 \* 15

# 输出最终的 boss ID 和 boss 的总收入

print(x, ans)

if \_\_name\_\_ == "\_\_main\_\_":

main()

C++

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

ll fa[100010],

mon[100010]; // 声明两个数组，fa用来记录每个分销的上级分销ID，mon用来记录每个分销的收入

int main() {

int n;

cin >> n; // 读入关系的总数量N

vector<int> in(100010); // 记录每个分销被定义为上级的次数

set<int> st; // 用于存储所有出现的分销ID

memset(fa, -1, sizeof(

fa)); // 初始化fa数组，全部设置为-1，表示未知上级

for (int i = 0; i < n; i++) {

int x, y, z;

cin >> x >> y >> z; // 读入每条关系信息：分销ID上级分销ID收入

st.insert(x); // 将分销ID x加入集合

st.insert(y); // 将上级分销ID y加入集合

in[y]++; // 记录上级分销ID y被定义为上级的次数

fa[x] = y; // 设置分销 x 的上级分销ID为 y

mon[x] = z; // 设置分销 x 的收入为 z

}

queue<int> que; // 创建一个队列用于处理没有下级的分销

for (int x : st)

if (in[x] == 0)

que.push(x); // 如果某分销ID没有下级分销，将其加入队列

int ans = 0; // 最终输出的boss收入

int x = 0; // 最终输出的boss ID

while (!que.empty()) {

x = que.front(); // 获取队列中的第一个元素

ans = mon[x]; // 收入设置为当前的收入（会随程序运行最后是boss的收入）

que.pop(); // 弹出这个元素

int y = fa[x]; // 获取分销 x 的上级分销ID

if (y == -1)

continue; // 如果上级分销ID是-1，跳过此次循环（可能是根节点）

in[y]--; // 上级分销 y 的下级数量减1

if (in[y] == 0) que.push(

y); // 如果上级分销 y 没有下级分销了，将其加入队列

// 计算并累加上级分销y的收入：当前收入的15%上交给上级

mon[y] += mon[x] / 100 \* 15;

}

// 输出最终的boss id 和 boss的总收入

cout << x << " " << ans << endl;

return 0;

}

C语言

#include <stdio.h>

#include <string.h>

#define MAX 100010

long long fa[MAX], mon[MAX];

int main() {

int n;

scanf("%d", &n); // 读入关系的总数量N

int in[MAX] = {0}; // 记录每个分销被定义为上级的次数

int st[MAX] = {0}; // 标记分销ID是否出现

memset(fa, -1, sizeof(

fa)); // 初始化fa数组，全部设置为-1，表示未知上级

for (int i = 0; i < n; i++) {

int x, y, z;

scanf("%d %d %d", &x, &y, &z);

st[x] = 1; // 将分销ID x 标记为出现

st[y] = 1; // 将上级分销ID y 标记为出现

in[y]++; // 记录上级分销ID y 被定义为上级的次数

fa[x] = y; // 设置分销 x 的上级分销ID 为 y

mon[x] = z; // 设置分销 x 的收入为 z

}

int queue[MAX], front = 0,

rear = 0; // 创建一个队列用于处理没有下级的分销

for (int i = 0; i < MAX; i++)

if (st[i] && in[i] == 0)

queue[rear++] = i; // 如果某分销ID没有下级分销，将其加入队列

long long ans = 0; // 最终输出的 boss 收入

int x = 0; // 最终输出的 boss ID

while (front < rear) {

x = queue[front++]; // 获取队列中的第一个元素

ans = mon[x]; // 收入设置为当前的收入（会随程序运行最后是 boss 的收入）

int y = fa[x]; // 获取分销 x 的上级分销ID

if (y == -1)

continue; // 如果上级分销ID是-1，跳过此次循环（可能是根节点）

in[y]--; // 上级分销 y 的下级数量减1

if (in[y] == 0) queue[rear++] =

y; // 如果上级分销 y 没有下级分销了，将其加入队列

// 计算并累加上级分销 y 的收入：当前收入的15%上交给上级

mon[y] += mon[x] / 100 \* 15;

}

// 输出最终的 boss ID 和 boss 的总收入

printf("%d %lld\n", x, ans);

return 0;

}

JsNode

const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

const MAX = 100010;

let fa = new Array(MAX).fill(-1); // 记录每个分销的上级分销ID

let mon = new Array(MAX).fill(0); // 记录每个分销的收入

let inDegree = new Array(MAX).fill(0); // 记录每个分销被定义为上级的次数

let st = new Set(); // 用于存储所有出现的分销ID

let n = 0;

let relations = [];

let isFirstLine = true;

rl.on("line", (line) => {

if (isFirstLine) {

n = parseInt(line.trim());

isFirstLine = false;

} else {

relations.push(line.trim().split(" ").map(Number));

if (relations.length === n) {

processRelations();

}

}

});

function processRelations() {

// 处理输入的数据，建立分销关系和收入

relations.forEach((relation) => {

let [x, y, z] = relation;

st.add(x);

st.add(y);

inDegree[y]++; // 记录上级分销ID y 被定义为上级的次数

fa[x] = y; // 设置分销 x 的上级分销ID 为 y

mon[x] = z; // 设置分销 x 的收入为 z

});

let queue = []; // 创建一个队列用于处理没有下级的分销

st.forEach((x) => {

if (inDegree[x] === 0) {

queue.push(x); // 如果某分销ID没有下级分销，将其加入队列

}

});

let ans = 0; // 最终输出的 boss 收入

let x = 0; // 最终输出的 boss ID

while (queue.length > 0) {

x = queue.shift(); // 获取队列中的第一个元素

ans = mon[x]; // 收入设置为当前的收入（会随程序运行最后是 boss 的收入）

let y = fa[x]; // 获取分销 x 的上级分销ID

if (y === -1) continue; // 如果上级分销ID是-1，跳过此次循环（可能是根节点）

inDegree[y]--; // 上级分销 y 的下级数量减1

if (inDegree[y] === 0) {

queue.push(y); // 如果上级分销 y 没有下级分销了，将其加入队列

}

// 计算并累加上级分销 y 的收入：当前收入的15%上交给上级

mon[y] += Math.floor(mon[x] / 100) \* 15;

}

// 输出最终的 boss ID 和 boss 的总收入

console.log(x, ans);

}

GO

package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

const MAX = 100010

func main() {

var fa [MAX]int64 // 记录每个分销的上级分销ID

var mon [MAX]int64 // 记录每个分销的收入

inDegree := make([]int, MAX) // 记录每个分销被定义为上级的次数

st := make(map[int]bool) // 用于存储所有出现的分销ID

scanner := bufio.NewScanner(os.Stdin)

scanner.Scan()

n, \_ := strconv.Atoi(scanner.Text()) // 读入关系的总数量N

for i := 0; i < MAX; i++ {

fa[i] = -1 // 初始化fa数组，全部设置为-1，表示未知上级

}

for i := 0; i < n; i++ {

scanner.Scan()

values := strings.Split(scanner.Text(), " ")

x, \_ := strconv.Atoi(values[0])

y, \_ := strconv.Atoi(values[1])

z, \_ := strconv.Atoi(values[2])

st[x] = true

st[y] = true

inDegree[y]++ // 记录上级分销ID y 被定义为上级的次数

fa[x] = int64(y) // 设置分销 x 的上级分销ID 为 y

mon[x] = int64(z) // 设置分销 x 的收入为 z

}

queue := make([]int, 0) // 创建一个队列用于处理没有下级的分销

for x := range st {

if inDegree[x] == 0 {

queue = append(queue, x) // 如果某分销ID没有下级分销，将其加入队列

}

}

var ans int64 // 最终输出的boss收入

var x int // 最终输出的boss ID

for len(queue) > 0 {

x = queue[0] // 获取队列中的第一个元素

queue = queue[1:]

ans = mon[x] // 收入设置为当前的收入（会随程序运行最后是boss的收入）

y := int(fa[x]) // 获取分销 x 的上级分销ID

if y == -1 {

continue // 如果上级分销ID是-1，跳过此次循环（可能是根节点）

}

inDegree[y]-- // 上级分销 y 的下级数量减1

if inDegree[y] == 0 {

queue = append(queue, y) // 如果上级分销 y 没有下级分销了，将其加入队列

}

// 计算并累加上级分销 y 的收入：当前收入的15%上交给上级

mon[y] += (mon[x] / 100) \* 15

}

// 输出最终的boss ID 和boss的总收入

fmt.Println(x, ans)

}