华为OD机试 - 文件目录大小（Java & JS & Python）

题目描述

一个文件目录的数据格式为：目录id，本目录中文件大小，(子目录id列表）。

其中目录id全局唯一，取值范围[1, 200]，本目录中文件大小范围[1, 1000]，子目录id列表个数[0,10]例如：1 20 (2,3) 表示目录1中文件总大小是20，有两个子目录，id分别是2和3

现在输入一个文件系统中所有目录信息，以及待查询的目录 id ，返回这个目录和及该目录所有子目录的大小之和。

输入描述

第一行为两个数字M，N，分别表示目录的个数和待查询的目录id，

- 1 ≤ M ≤ 100
- 1 ≤ N ≤ 200

接下来M行，每行为1个目录的数据：

> 目录id 本目录中文件大小 (子目录id列表)

子目录列表中的子目录id以逗号分隔。

输出描述

待查询目录及其子目录的大小之和

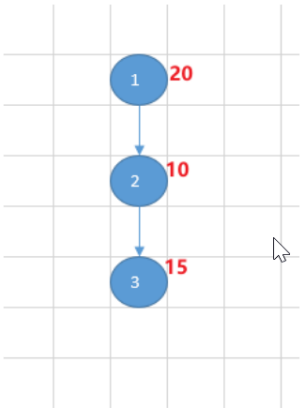
用例

输入	3 1 3 15 () 1 20 (2) 2 10 (3)
输出	45
说明	目录1大小为20，包含一个子目录2 (大小为10)，子目录2包含一个子目录3(大小为15)，总的大小为20+10+15=45

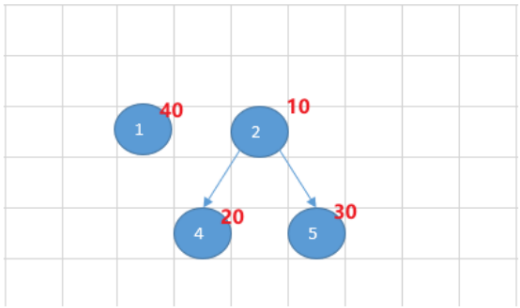
输入	4 2 4 20 () 5 30 () 2 10 (4,5) 1 40 ()
输出	60
说明	目录2包含2个子目录4和5，总的大小为10+20+30 = 60

题目解析

用例1图示：



用例2图示：



本题的目录与子目录的关系，可以看出是一棵树，而计算某个目录及其子目录的文件大小总和，其实就是遍历某个节点下所有分支。

因此，本题可以使用深度优先搜索dfs来解决。下面代码中深度优先搜索，是基于栈实现的，而不是基于递归。基于栈实现dfs的好处是，可以避免较深递归产生的执行栈溢出。

```
import java.util.*;
import java.util.stream.Collectors;

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

    int m = sc.nextInt();
    int n = sc.nextInt();

    HashMap<Integer, ArrayList<Integer>> children = new HashMap<>();
    HashMap<Integer, Integer> cap = new HashMap<>();
    for (int i = 0; i < m; i++) {
      int fa_id = sc.nextInt();
      int fa_cap = sc.nextInt();
```

```
16
17        children.putIfAbsent(fa_id, new ArrayList<>());
18        cap.putIfAbsent(fa_id, fa_cap);
19
20        String ch_str = sc.next();
21        if (ch_str.length() > 2) {
22          children
23              .get(fa_id)
24              .addAll(
25                  Arrays.stream(ch_str.substring(1, ch_str.length() - 1).split(","))
26                      .map(Integer::parseInt)
27                      .collect(Collectors.toList()));
28        }
29      }
30
31      System.out.println(getResult(children, cap, n));
32    }
33
34    public static int getResult(
35        HashMap<Integer, ArrayList<Integer>> children, HashMap<Integer, Integer> cap, int
    target) {
36      int ans = 0;
37
38      LinkedList<Integer> stack = new LinkedList<>();
39      stack.add(target);
40
41      while (stack.size() > 0) {
42        Integer id = stack.pop();
43        if (!cap.containsKey(id)) continue;
44        ans += cap.get(id);
45        stack.addAll(children.get(id));
46      }
47
48      return ans;
49    }
50  }
```