1、关联容器 map 保存<key, value>数据，能通过 key 快速存储或查找记录。请设计一个 map，能够满足以下要求：
1. map 的容量 size 是一个固定值 N，即 map 最多保存 N 个<key, value>记录；
2. map insert 一个<key, value>前，首先判断这个 key 是否已经在 map 中存在：
1) 如果存在：记这个已存在的记录为<key, old_value>，若 old_value<value，则把 old_value 更新为 value；否则，不做更新。
2) 如果不存在：
若 size<N，则执行 map 的 insert，保存这个<key, value>，且 size+=1;
若 size==N，先淘汰掉一个**更新时间**最早的记录，再执行 map 的 insert，保存这个<key, value>，size 保持为 N 不变。
说明：记录的更新时间默认为其被 insert 进 map 的时间，之后的某一时刻 T，如果这个记录的 value 被更新，那么，该记录的更新时间就变为 T。

解析：
```
import java.util.*;
public class Main{
    public static void main(String[] args){
        Scanner in = new Scanner(System.in);
        int n = Integer.parseInt(in.nextLine());
        int curSize = 0;
        LinkedHashMap<String, Long> map = new LinkedHashMap<>();
        while(in.hasNext()){
            String curInput = in.nextLine();
            String[] stringArray = curInput.split(" ");
            String curKey = stringArray[0];
            long curValue = Long.parseLong(stringArray[1]);
            if(map.containsKey(curKey)){
                if(map.get(curKey) < curValue){
                    map.remove(curKey);
                    map.put(curKey, curValue);
                }
            }else{
                if(curSize < n){
                    curSize++;
                }else{
                    for(String s : map.keySet()){
                        System.out.println(s+" "+map.get(s));
                        map.remove(s);
                        break;
                    }
                }
                map.put(curKey, curValue);
            }
```

```
        }
    }
}
```

2、【题干描述】：

我们共有 n 台服务器，每台服务器可以和若干个子服务器传输数据，n 台服务器组成一个树状结构。

现在要将一份数据从 root 节点开始分发给所有服务器。

一次数据传输需要一个小时时间，

一个节点可以同时对 k 个儿子节点进行并行传输，

不同节点可以并行分发。

问，全部分发完成，最短需要多少小时？

【示例】：

当共有 5 台服务器，其树状结构为

```
          0
        /   \
      1       2
    /   \
  3       4
```

假设每一台服务器同时可以对 1 个儿子节点（k=1）并行传输，最优的数据传输过程示例如下：

第一个小时，0 -> 1；

第二个小时，1->3 & 0->2；

第三个小时，1 -> 4；

所以当 k=1 时，全部分发完成最短需要 3 个小时。

假设每一台服务器同时可以对 2 个儿子节点（k=2）并行传输，最优的数据传输过程示例如下：

第一个小时，0 -> 1 & 0 -> 2；

第二个小时，1 -> 3 & 1 -> 4；

所以当 k=2 时，全部分发完成最短需要 2 个小时。

解析：

```
import java.util.*;

/**
 * 1. 使用 hashmap 构建树形结构
 * 2. 递归分治法：假设已知 各个子节点的分发时间：
 *         则： 2.1 对子节点的分发时间进行降序排列
 *              2.2 当前节点优先对 子节点分发时间 较长的节点进行分发,
 *             如:子节点分发时间 4 2 1 1 1 0, k=2, 则: 先后分发顺序(42)(11)(10),也就是
 *                          (42)+1 (11)+2 (10)+3 => 5 3 3 3 4 3 => 分三轮分发, 最终所需时间为 5 h。
 *              2.3 当节点为叶子结点, 分发时间为 0
 *              2.4 最后使用分治法进行递归分发,相当于从叶子节点向上推。
```

```
 */
class Node{

    int id = 0;
    List<Node> sons = new ArrayList<>();
    Node parent = null;
}

public class Main {

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

        HashMap<Integer , Node> hashMap = new HashMap<>();

        int K = sc.nextInt();
        int N = sc.nextInt();

        if(K == 0 || N == 0){
            System.out.println(0);
        }

        sc.nextLine();

        //树形构建
        for(int i = 0;i < N;i++){
            int node_n = sc.nextInt();
            //指定父节点
            int parent_id = sc.nextInt();
            Node my = null;
            if(hashMap.containsKey(parent_id)){
                //父节点存在
                my = hashMap.get(parent_id);
            }else{
                //父节点不存在
                my = new Node();
                my.id = parent_id;
                hashMap.put(parent_id,my);
            }
            //儿子节点
            for(int j = 0;j < node_n - 1;j++){
                int son_id = sc.nextInt();
                if(hashMap.containsKey(son_id)){
                    //儿子节点存在
```

```
hashMap.get(parent_id).sons.add(hashMap.get(son_id));
                }else{
                    //儿子节点不存在
                    Node son = new Node();
                    son.id = son_id;
                    son.parent = my;
                    hashMap.put(son.id,son);
                    hashMap.get(parent_id).sons.add(son);
                }
            }
            sc.nextLine();
        }

        //根节点存在
        Node root = hashMap.get(0);
        //分治法
        System.out.println(FZ(root , K));

    }

    public static int FZ(Node node , int k){
        int max = 0;
        if(node.sons.size() == 0){
            return 0;
        }else{
            Integer[] sons_time = new Integer[node.sons.size()];
            int i = 0;
            //统计分支时间
            for(Node son:node.sons){
                sons_time[i++] = FZ(son , k);
            }
            //合并分支与当前时间
            //降序排序
            Arrays.sort(sons_time, new Comparator<Integer>() {
                @Override
                public int compare(Integer o1, Integer o2) {
                    return o2 - o1;
                }
            });
            //计算当前节点为root所需要的时间
            int add = 1;
            for(int j = 0; j < sons_time.length ; j++){
                if(j!=0 && (j%k) == 0){
```

```
                    add++;
                }
                sons_time[j] += add;
                if(sons_time[j] > max){
                    max = sons_time[j];
                }
            }
        }
        return max;
    }
}
```