Raising Pay

Purpose: multiway trees

Due: Oct 30^{th}

Description

The Egalitarian Company makes 10 gallon hats. The companies management is very strict. Every employee (except the owner) at the company has a single boss. Each employee who is not a boss of someone is called a worked. The owner of the company also tries to live up to the name of the company when he awards raises, in that every worker gets a raise or no one does. A boss only sends a raise request up the line if the percentage of people directly under him making requests is greater than $t\%$. If the owner receives more than $t\%$ requests, then all the workers will get a raise. Now the Hat Union represents the workers in the company and wants to know the minimum number of workers needed to make requests so that all the workers will receive a raise.

Input

The input will come entirely from the keyboard (stdin) and will consist of test cases. Each test case will be on 2 lines. The first line will contain two integers n and t, ($1 \leq n$ and $1 \leq t \leq 100$) separated by a single space. Here n represents the number of employees (not including the owner) and the threshold to send the raise request up the line. The owner is indicated with the number 0, while the each of the other integers represents a single employee.'The second line will contain n integers (each separated by a single space) indicating the bosses. The i^{th} integer names the boss of the i^{th} employee. The end of input is indicated by 0 employees and 0 threshold.

Output

For each test case first print the resulting organizational chart. Where all the employees under the same boss are ordered. On the next line print the number of workers needed to request a raise, for the company to grant all workers raises.

Sample Input

```
3 40
0 0 0
3 100
0 0 0
```

```
14 60
0 0 1 1 2 2 2 5 7 5 7 5 7 5
0 0
```

Corresponding Sample Output

```
0
    1
    2
    3
2
0
    1
    2
    3
3
0
    1
         3
         4
    2
         5
              8
             10
             12
             14
         6
         7
              9
             11
             13
5
```

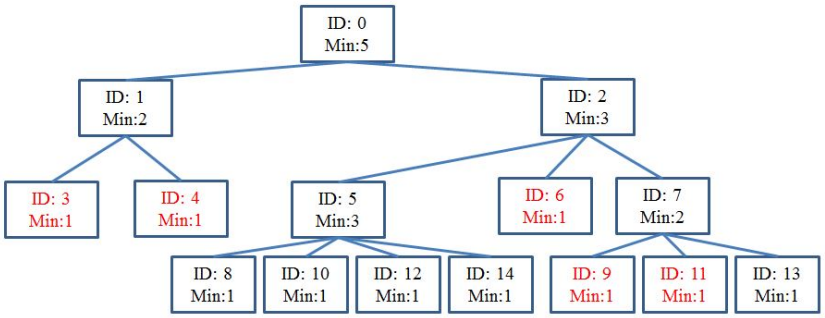

Figure 1: Organizational Chart for the third example. If the workers in red ask for raises, everyone will get a raise.

How the program will be graded

Memo

What	pts	Oct 30^{th}
Name	1	
Class Diagram (note that a struct is a class) [1]	5	

[1] The class diagram should conform to UML standards, showing the methods, the properties, and the relationships between all classes.

What	pts	Due Oct
Name	1	23^{th}
Description	2	23^{th}
Style	5	30^{th}
pre/post conditions	5	30^{th}
Functionality and use of the stl		
Forest::find(id)	5	23^{th}
Forest::insert(boss, underling)	5	23^{th}
Forest::print()	5	23^{th}
Rest	16	30^{th}

Required Starting Data Structure

You can add to this code but you implement what is here.

```
struct Node
{
    int id;
    int minRequests4Raise;
    vector< Node *> underlings;
    Node(int n = 0) // constructor
};

class Tree
{
private:
    Node * root;
    void print(Node * ptr, int level) // private version of print
    Node * find(Node * root, int target);  // private version of find
public:
    Tree(Node * rt = nullptr) { root = rt; }
    Tree(int id);
    Node * getRoot();
    void printTree();                  // public print
    Node * find(int target);           // public find
};

class Forest
{
private:
    vector< Tree > trees;
public:
    void print();
    Node * find(int target);
    void insert(int boss, int underling);
    Forest();
    void clear();
};
```

Figure 2: Required Starting Code

Suggested Insertion Algorithm

(Assumes the forest contains at least contains a single tree with at least 1 Node in it) (i.e. the owner)) There are 4 cases to consider when you read a boss, empolyee pair

1. If neither the boss nor employee is already in the forest
   (a) Create a tree with the boss as the root and the employee as the child of the root.
   (b) `push_back` that tree to the forest.
2. If the boss is in the forest but not the employee
   (a) Just make a new node with the employee as the child of the boss
3. The employee is in the forest but NOT his boss.
   (a) This employee must be the root of a tree (as there can only be 1 boss/employee)
   (b) Make a new tree with the boss as the root and make the employee his child
   (c) Replace the childs tree with the tree whose root is the boss
4. Both the boss and employee are in the forest
   (a) Add the employee as a child of the boss
   (b) Delete the i^{th} tree from the forest , you can use :
   ```
   nameOfForest.trees.erase(nameOfForest.trees[nameOfForest.trees.begin() + i])
   ;
   ```