Salary Inversion

Two requirements:

- Your solution to this problem must use a recursive function.
- Your solution must not use global variables.

Solutions that violate either of these requirements will receive no credit.

A company is trying to determine if there are workers who make more than their bosses. A *boss* has one or more direct reports. The *CEO* of the company has zero or more direct reports, but does not have a boss. A *worker* has no direct reports. The *management chain* for a worker includes their boss, their boss's boss, etc up to the CEO of the company.

Each employee has a salary and is denoted as Employee N, where N is a unique identifier for the employee. For example, lets assume a company has employees with the following salaries:

- Employee 1 = \$6000
- Employee 2 = \$5000
- Employee 3 = \$3000
- Employee 4 = \$2500
- Employee 5 = \$5500
- Employee 6 = \$10000
- Employee 7 = \$9000
- Employee 8 = \$1000

Each employee reports to a single direct boss, with the exception of Employee 1 who is always the CEO at the company. Using the list of employees above, the company's direct reporting structure is as follows:

- (Boss: Employee 1, Direct Reports: (Employee 2, Employee 3, Employee 8))
- (Boss: Employee 2, Direct Reports: (Employee 4, Employee 5))
- (Boss: Employee 3, Direct Reports: (Employee 6))
- (Boss: Employee 6, Direct Reports: (Employee 7))

Your goal is to determine the number of workers who have at least one person in their management chain who makes less than they do.

Given this reporting structure, Employees 4, 5, 7, and 8 are workers. Two of whom–5 and 7–make more than at least one boss in their management chain. Employee 5 makes more than their direct boss Employee 2. Employee 7 makes more than Employee 3, their boss's boss, and Employee 1, the CEO.

The reporting structure for the company will be represented using the `Tree` class. Here is the header file for the `Tree` class:

```
#ifndef _Tree_
```

```
#define _Tree_
#include <vector>
#include <string>

/**
Class for representing an employee and the people who report to them.

Public attributes:
 empNum: the employee's employee number represented as an integer
 salary: the employee's salary represented as an integer
 children: a list of the employee's direct reports

 Public methods:
  addDirectReport: add a new direct report to the tree
  printTree: print the contents of the tree.
 **/
class Tree {
public:
int empNum;
int salary;
std::vector<Tree*> children;

/**
Constructor for Tree

 @param empNum the employee's employee number represented as an integer
 @param salary the employee's salary represented as an integer
**/
Tree(int _empNum, int _salary);

/**
add a new direct report

 @param t the new direct report represented as a Tree
**/
void addDirectReport(Tree* t);

/**
print the contents of the tree, indented to illustrate the tree structure

 @param tabs a string with spaces that tracks the indentation level
**/
void printTree(std::string tabs);
};

#endif
```

and here is the code for the `Tree` class:

```
#include "Tree.h"
#include <vector>
#include <string>
#include <iostream>

/**
Class for representing an employee and the people who report to them.

Public attributes:
 empNum: the employee's employee number represented as an integer
 salary: the employee's salary represented as an integer
 children: a list of the employee's direct reports

 Public methods:
  addDirectReport: add a new direct report to the tree
  printTree: print the contents of the tree.
 **/
Tree::Tree(int _empNum, int _salary): empNum(_empNum), salary(_salary),
children(std::vector<Tree*>()) {}

/**
add a new direct report

 @param t the new direct report represented as a Tree
**/
void Tree::addDirectReport(Tree* t) {
    children.push_back(t);
}

/**
print the contents of the tree, indented to illustrate the tree structure

 @param tabs a string with spaces that tracks the indentation level
**/
void Tree::printTree(std::string tabs) {
    std::cout << tabs << empNum << " " << salary << "\n";
    for (Tree* t : children) {
        t->printTree(tabs + "  ");
    }
}
```

Your task is to complete the function `solve` in `problem8.cpp`. Given the root of the tree (the `Tree` node representing the CEO), your function should return the number of workers who make more than at least one boss in their management chain.

Here is the code for the header file for this task:

```
#ifndef _Problem8_
#define _Problem8_
#include "Tree.h"
#include <string>
#include <vector>

class Problem8 {
public:

/**
 * @brief   find the number of Workers (ie. employees with no direct reports)
 *          who make more than at least one of their bosses.
 *
 * @param   t the root of the tree
 * @return  the number of workers who make more than at least one of their bosses
 */
static int solve(Tree* t);
};

#endif
```

And here is the skeleton code for this task:

```
#include "Problem8.h"
#include "Tree.h"
#include <string>
#include <vector>

/**
 * @brief   find the number of Workers (ie. employees with no direct reports)
 *          who make more than at least one of their bosses.
 *
 * @param   t the root of the tree
 * @return  the number of workers who make more than at least one of their bosses
 */
int Problem8::solve(Tree* t) {
    // YOUR CODE HERE
    // Return is included to ensure that the skeleton code compiles.
    return -1;
}
```

When you take the placement exam, you will be expected to copy the code above into the relevant files and then complete the specified function. For the practice problems, for your convenience, we have provided, four files:

- `Tree.h` - the header file for the `Tree` class,

- `Tree.cpp` - the code for the `Tree` class,
- `Problem8.h` - the header file for the problem, and
- `Problem8.cpp` - the skeleton code for the problem.