Employee Importance

You are given a data structure of employee information, which includes the employee's **unique id**, his **importance value** and his **direct** subordinates' id.

For example, employee 1 is the leader of employee 2, and employee 2 is the leader of employee 3. They have importance value 15, 10 and 5, respectively. Then employee 1 has a data structure like [1, 15, [2]], and employee 2 has [2, 10, [3]], and employee 3 has [3, 5, []]. Note that although employee 3 is also a subordinate of employee 1, the relationship is **not direct**.

Now given the employee information of a company, and an employee id, you need to return the total importance value of this employee and all his subordinates.

Example 1:

```
Input: [[1, 5, [2, 3]], [2, 3, []], [3, 3, []]], 1
Output: 11
Explanation:
Employee 1 has importance value 5, and he has two direct subordinates: employee 2 and
 employee 3. They both have importance value 3. So the total importance value of empl
oyee 1 is 5 + 3 + 3 = 11.
```

Note:

1. One employee has at most one **direct** leader and may have several subordinates.
2. The maximum number of employees won't exceed 2000.

Solution 1

```java
class Solution {
    public int getImportance(List<Employee> employees, int id) {
        int total = 0;
        Map<Integer, Employee> map = new HashMap<>();
        for (Employee employee : employees) {
            map.put(employee.id, employee);
        }
        Queue<Employee> queue = new LinkedList<>();
        queue.offer(map.get(id));
        while (!queue.isEmpty()) {
            Employee current = queue.poll();
            total += current.importance;
            for (int subordinate : current.subordinates) {
                queue.offer(map.get(subordinate));
            }
        }
        return total;
    }
}




class Solution {
    public int getImportance(List<Employee> employees, int id) {
        Map<Integer, Employee> map = new HashMap<>();
        for (Employee employee : employees) {
            map.put(employee.id, employee);
        }
        return getImportanceHelper(map, id);
    }

    private int getImportanceHelper(Map<Integer, Employee> map, int rootId) {
        Employee root = map.get(rootId);
        int total = root.importance;
        for (int subordinate : root.subordinates) {
            total += getImportanceHelper(map, subordinate);
        }
        return total;
    }
}
```

written by shuxu original link here

Solution 2

```
        int total = 0;
        public int getImportance(List<Employee> employees, int id) {
            Employee manager = employees.stream().filter(e -> e.id == id).collect(C
ollectors.toList()).get(0);
            total += manager.importance;
            manager.subordinates.forEach(subId -> getImportance(employees, subId));
            return total;
        }
```

written by fishercoder original link here

Solution 3

// without STL

```
class Solution {
public:
    int getImportance(vector<Employee*> employees, int id) {

        unordered_map<int, Employee*> map;

        for(const auto element : employees){

            map[element->id] = element;
        }

        return help(map, id);
    }

    int help(unordered_map<int, Employee*>& map, const int id){

        auto sum = map[id]->importance;

        for(const auto element : map[id]->subordinates){

            sum += help(map, element);
        }

        return sum;
    }
};
```

// with STL

```
class Solution {
public:
    int getImportance(vector<Employee*> employees, int id) {

        unordered_map<int, Employee*> map;

        std::transform(employees.begin(), employees.end(), std::inserter(map, map.en
d()), [&](Employee* element){

            return std::make_pair(element->id, element);
        });

        return help(map, id);
    }

    int help(unordered_map<int, Employee*>& map, const int id){

        return map[id]->importance + std::accumulate(map[id]->subordinates.begin(),
map[id]->subordinates.end(), 0, [&](int sum, int id){

            return sum + help(map, id);
        });
    }
};
```

written by gemhung original link here