Seoul National University

Data Structure

Fall 2017, Kang

# Programming Assignment 3: Non-Binary Tree (Chapter 6)

Due: Nov. 14, 14:00, submit at eTL

## Reminders

- The points of this homework add up to 100.
- Like all homeworks, this has to be done individually.
- Lead TA: Hyunsik Jeon (jeon185@gmail.com)
- Write a program in Java.
- Do not use Java Collection Framework

## 1. How to submit the programming assignment

1) Create a ***JAR*** file including 'src' folder that contains your sources files, but without 'release' folder. (Refer to '1 – Introduction.pptx' in the first lab session)
    - We will run your ***Main*** class in the JAR file to grade your programming assignments. Before submitting the JAR file, please check if your Main class in the JAR file works correctly in a terminal.
    - You **MUST** obey the I/O specification of the programming assignment, and rules for the submission of the programming assignment.

2) Compress the JAR file with a readme file and the project folder. If you don't use Eclipse, include source files into the compressed file. The readme file needs to include your student id, name, how to execute your program, and any other information TA needs to know.
    - The format of the JAR file is "*PA_##_(StudentID).jar*" where '##' is the programming assignment ID, and *(StudentID)* is your student ID.
        - ex) PA_01_2017-12345.jar
    - All documents should be written in English.

3) The name of the compressed file (zip file) should be '*PA_##_(StudentID).zip*' where '##' is the programming assignment ID, and (StudentID) is your student ID.
    - ex) PA_01_2017-12345.zip
        - ➢ *01: the first programming assignment*
        - ➢ *2015-12345: your student ID*

4) Submit the compress file to the eTL (http://etl.snu.ac.kr/) .

## 2. How to grade your programming assignment

1) We made a grading machine to automatically grade your programming assignment. The machine will run your program, and compare answers and outputs that your program generates for given inputs. If your program cannot generate correct answers for an input file, it will not give you the point corresponding to the input. Our machine will consider the following scenarios:

   - (***Accept***) When your program generates exact outputs for an input file, the machine will give you the point of the input.
   - (***Wrong Answer***) When your program runs normally, but generates incorrect outputs for an input file, including typos, the machine will not give you the point of the input.
   - (***Run Error***) When your program does not run, or is terminated suddenly for some reasons, the machine will not give you the point of an input file because it cannot generate any outputs.
   - (***Time Limit***) When your program runs over a predefined execution time for an input file, our machine will stop your program, and it will not give you the point of the input. The time limit of the execution is ***5 seconds***.

2) We will generate 10 input files, and assign 10 points for each input file. For example, if your program gets 9 accepts, and 1 wrong answer by the machine, the total point will be 90 points. Hence, before submitting your programming assignment, please be sure that your program makes correct answers in reasonable time for any input case.

## 3. Problem

John is a human resource manager in a company. He hires people, measures their abilities, and merges the people or groups. He wants to have a program that can do his work automatically.

In this assignment, we will implement human resource managing program, which is called *HumanResource*. The requirements of *HumanResource* are as follows:

- When hiring a person, John registers the person's ID and the person's ability together. The person is recognized as a group and becomes the leader of the group.
- When a group and another group are merged, the leader of the merged group is the leader of the more powerful group (More powerful means that the sum of people's ability in the group is higher.). The merged group's ability is the sum of two group's ability. If the two group have the same ability, the leader with the lower ID becomes the leader of the merged group.
- John also wants to be able to merge all two pairs. For example, when a person A and another person B are merged, A and B and all their bosses are placed directly under the leader of each group, and the two groups are merged. (hint: path compression)

Here are additional functions John wants to have.

- AbilityOfPerson: Print out the ability of the person.
- DepthOfPerson: Print out how far the leader is from the person.
- AbilityOfGroup: The input has to be the person's ID. Print out the sum of the abilities of all people in the group the person belongs to.
- SizeOfGroup: The input has to be the person's ID. Print out the number

of people in the group the person belongs to.

Here are several assumptions for clarity.

- Everyone has different ID.
- The maximum number of people is 100,000.
- Once people are hired by the company, they do not leave the company.
- Path compression occurs only when a merge occurs. If you just find the root of some nodes, path compression does not happen.
- Each person has values of “Parent”, “Ability”, “GroupAbility”, and “GroupSize”. However, “GroupAbility” and “GroupSize” is only used for leaders. Therefore, for leaders, both values must be stored correctly, but non-leaders may have dummy values.

You need to fill in the “ParPtrTree.java”, the “HumanResource.java”, and “Main.java” of “PA03” java project. You can modify the skeleton codes if you want.

## ADT of ParPtrTree.java

1) setAbility

| Function |
|---|
| void setAbility(Integer index, Integer ability) |
| **Description** |
| - This function sets the person indexed by "index" to have ability amount to "ability".<br>- You have to save the ability value to "Ability" array. |

2) setGroupAbility

| Function |
|---|
| void setGroupAbility(Integer index, Integer GA) |
| **Description** |
| - This function sets the person indexed by "index" to have group-ability amount to "GA".<br>- You have to save the GA to "GroupAbility" array.<br>- Group-ability is only useful for the leader. Therefore, non-leaders may have dummy values.<br>- The group-ability value of the leader indicates the total ability of the group. |

3) setGroupSize

| Function |
|---|
| void setGroupSize(Integer index, Integer GS) |
| **Description** |
| - This function sets the person indexed by "index" to have group-size amount to "GS".<br>- You have to save the size to "GroupSize" array.<br>- Group-size is only useful for the leader. Therefore, non-leaders may have dummy values.<br>- The group-size value of the leader indicates the total number of the group. |

4) addGroupAbility

| **Function** |
|---|
| void addGroupAbility(Integer index, Integer GA) |
| **Description** |
| - This function increments the group-ability of the person indexed by “index” by “GA”. |

5) addGroupSize

| **Function** |
|---|
| void addGroupSize(Integer index, Integer GS) |
| **Description** |
| - This function increments the group-size of the person indexed by “index” by “GS” |

6) getAbility

| **Function** |
|---|
| Integer getAbility(Integer index) |
| **Description** |
| - This function returns the ability of the person indexed by “index”. |

7) getGroupAbility

| **Function** |
|---|
| Integer getGroupAbility(Integer index) |
| **Description** |
| - This function returns the group-ability of the group the person indexed by “index” belongs to. <br> - There is no need to handle exceptions when index does not indicate a leader. |

8) getGroupSize

| **Function** |
|---|
| Integer getGroupSize(Integer index) |
| **Description** |
| - This function returns the group-size of the group the person indexed by “index” belongs to.<br>- There is no need to handle exceptions when index does not indicate a leader. |

9) getDepth

| **Function** |
|---|
| Integer getDepth(Integer index) |
| **Description** |
| - This function returns the depth of the person indexed by “index”.<br>- Depth is the number of edges between the leader and the person. Therefore the depth of any leader is 0. |

10) differ

| **Function** |
|---|
| boolean differ(int a, int b) |
| **Description** |
| - This function returns whether the person indexed by “a” and the person indexed by “b” are in the same group. |

11) UNION

| **Function** |
|---|
| void UNION(int a, int b) |
| **Description** |
| - This function merges two trees.<br>- The group including the person indexed by “b” is attached the group including the person indexed by “a”.<br>- Path compression for “a” and “b” are needed before two trees are |

| |
|---|
| merged. |

12) FIND_pathCompression

| Function |
|---|
| Integer FIND_pathCompression(Integer curr) |
| **Description** |
| - This function finds the root node of the “curr” node, and connects the “curr” node to the root node directly.<br>- Finally, it returns the index of the root node. |

13) FIND

| Function |
|---|
| Integer FIND (Integer curr) |
| **Description** |
| - This function finds the root node of the “curr” node. Path Compression is not occurred.<br>- Finally, it returns the index of the root node. |

## ADT of HumanResource.java

1) Hire

**Function**

void Hire(int member, int ability)

**Description**

- This function sets the person indexed by “member” to have the ability amount to “ability”.
- It also initializes group-ability and group-size.

2) Merge

**Function**

boolean Merge(int member1, int member2)

**Description**

- This function merges the group including “member1” and the group including “member2”.
- Before the two groups are merged, member1 and member2 have to be placed directly under each leader.
- If “member1” and “member2” already in the same group, it returns false. Of course path compressions for “member1” and “member2” do not happen.

3) AbilityOfPerson

**Function**

void AbilityOfPerson(int member)

**Description**

- This function prints out the ability of the person indexed “member”.

4) DepthOfPerson

**Function**

void DepthOfPerson(int member)

**Description**

- This function prints out the depth of the person indexed “member”.

5) AbilityOfGroup

| **Function** |
|---|
| void AbilityOfGroup(int member) |
| **Description** |
| - This function prints out the ability of the group the person indexed “member” belongs to. |

6) SizeOfGroup

| **Function** |
|---|
| void SizeOfGroup(int member) |
| **Description** |
| - This function prints out the size of the group the person indexed “member” belongs to. |

## 4. Specification of I/O

### 1) Hire

| Input form | Output form |
|---|---|
| `hire (person_id) (ability)` | `No output` |

**Description**

- (person_id) is the ID of the person.
- (ability) is the ability of the person.

| Example Input | Example Output |
|---|---|
| `hire 1 1` | |

### 2) Merge

| Input form | Output form |
|---|---|
| `merge (person1_id) (person2_id)` | `The group of (person1_id) and the group of (person2_id) are merged.` |

**Description**

- (person1_id) is the ID of the one person.
- (person2_id) is the ID of the other person.
- The Id has be raged between 0 and 99999.
- If two people are in the same group, you have to print out "The two members are in the same group already.". Otherwise, you have to print out "The group of (person1_id) and the group of (person2_id) are merged.

| Example Input | Example Output |
|---|---|
| `merge 1 2` | `The group of 1 and the group of 2 are merged.` |

### 3) Ability of person

| Input form | Output form |
|---|---|
| `aop (person_id)` | `(ability)` |

**Description**

- (person_id) is the ID of the person.

- (ability) is the ability of the person.

| Example Input | Example Output |
|---|---|
| `aop 1` | `1` |

4) Depth of person

| Input form | Output form |
|---|---|
| `dop (person_id)` | `(depth)` |

| Description |
|---|

- (person_id) is the ID of the person.
- (depth) is the depth of the person.

| Example Input | Example Output |
|---|---|
| `dop 1` | `1` |

5) Ability of group

| Input form | Output form |
|---|---|
| `aog (person_id)` | `(ability of the group)` |

| Description |
|---|

- (person_id) is the ID of the person.
- (ability of the group) is the ability of the group the person belongs to.

| Example Input | Example Output |
|---|---|
| `aog 1` | `1` |

6) Size of group

| Input form | Output form |
|---|---|
| `sog (person_id)` | `(size of the group)` |

| Description |
|---|

- (person_id) is the ID of the person.
- (size of the group) is the size of the group the person belongs to.

| Example Input | Example Output |
|---|---|
| `sog 1` | `1` |

## 5. Sample Input and Output

The grading machine expects the sample output for given the sample input. Hence, for the sample input, your program should print the same lines in the sample output. If your program prints different lines from the sample output for the sample input, the grading machine will not give you the point of the input. See more details of the policy on the grading machine in Section 2.

### 1) Sample Input

```
hire 1 1
hire 2 2
hire 3 3
merge 1 2
merge 1 2
aop 1
aop 2
aop 3
dop 1
dop 2
dop 3
aog 1
aog 2
aog 3
sog 1
sog 2
sog 3
```

## 2) Sample output

```
The group of 1 and the group of 2 are merged.
The two members are in the same group already.
1
2
3
1
0
0
3
3
3
2
2
1
```