CSC 212 Programming Assignment # 3
Implementing and Using General Trees
Due date: 26/11/2019

Guidelines:	This is an **individual** assignment. The assignment must be submitted to **Web-CAT**

1 Introduction

The electric power grid is the network infrastructure responsible for the generation and transmission of electricity. We have three types of elements in a power grid:

- A generator: this is a station that generates electricity. A generator has a power limit that it cannot exceed.
- A transmitter: this is a station that transmits electricity. A transmitter can be connected directly to a generator or to another transmitter. A transmitter has a maximum power that can pass through it.
- A consumer: this can be a household, a shop, a factory etc. A consumer can only be attached to a transmitter (not directly to a generator), and no element can be attached to it. A consumer has maximum consumption power that it cannot exceed.

For simplicity, we assume that the power grid can be represented as a general (non-binary) tree. The grid contains only one generator located at the root, whereas consumers are located at leaf nodes. See Figure 1 for an example.

We are interested in the following:

1. Finding how much power is generated, transmitted or consumed by an element. This is the total power required by the consumers in the corresponding subtree.

   Example 1. *In the grid shown in Figure 1: Elements 6 and 9 are consuming 0 power, since there are no consumers attached to them. Element 16 consumes 4, Element 8 consumes 10, and Element 4 consumes 19.*

2. Whether there is an overload in the grid. An overload takes place when the power required at an element exceeds its capacity. By definition, overloads can only happen at the generator or the transmitters (but not consumers).

   Example 2. *In the grid shown in Figure 1, the only element that has an overload is Element 4, since it consumes 19, and its limit is 18.*

In this programming assignment, we will use general trees to represent a power grid.

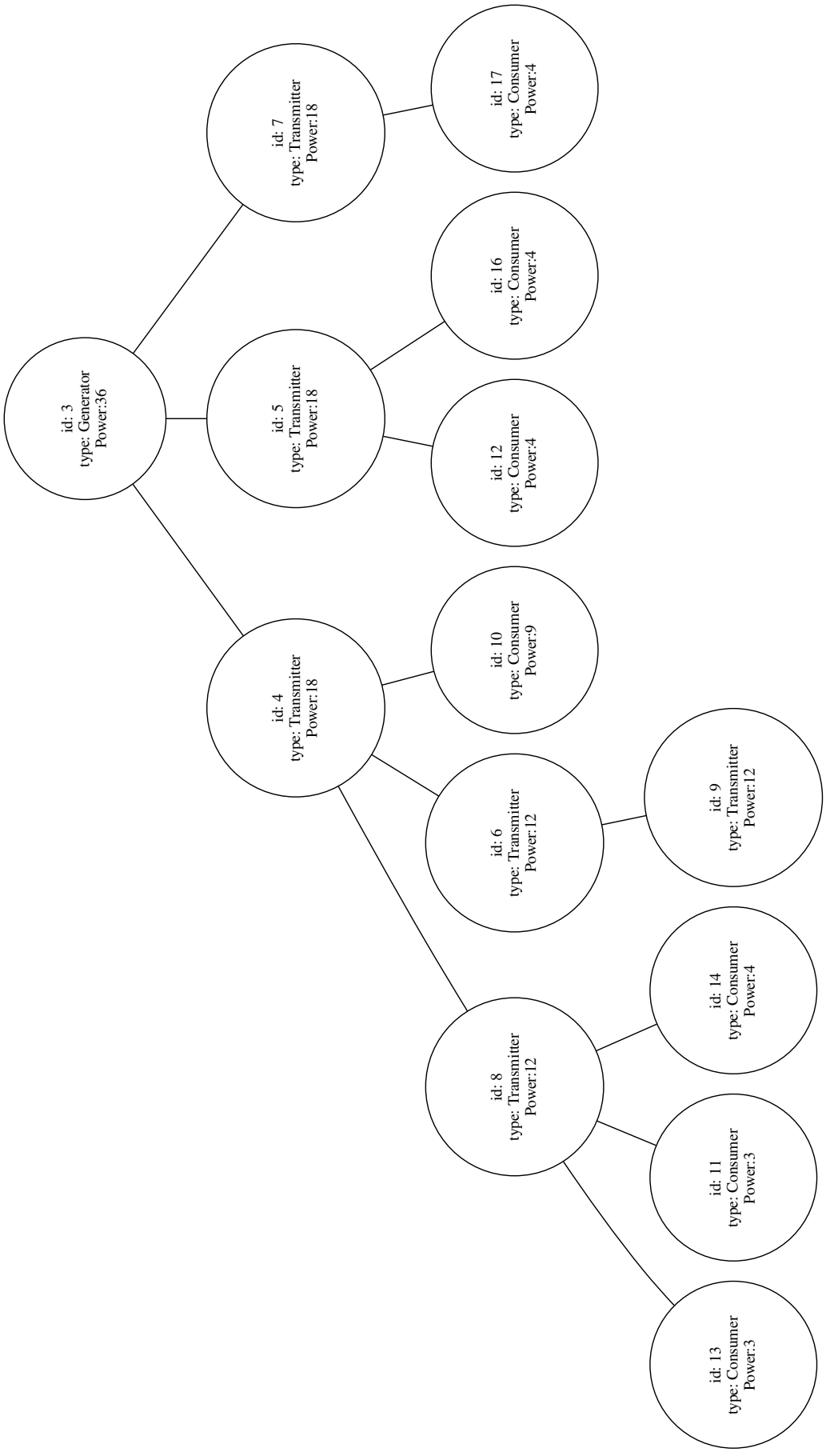

Figure 1: Example of a power grid.

2 Implementing a general tree

Write a the class `LinkedGT` that implements the following interface representing a general tree (There is no limit on the number of children that a node can have. The constructor of your class should not take any parameters.):

```
public interface GT<T> {

  // Return true if the tree is empty
  boolean empty();

  // Return true if the tree is full
  boolean full();

  // Return the data of the current node
  T retrieve();

  // Update the data of the current node
  void update(T e);

  // If the tree is empty e is inserted as root. If the tree is not empty
     , e is added as a child of the current node. The new node is made
     current and true is returned.
  boolean insert(T e);

  // Return the number of children of the current node.
  int nbChildren();

  // Put current on the i-th child of the current node (starting from 0),
      if it exists, and return true. If the child does not exist, current
      is not changed and the method returns false.
  boolean findChild(int i);

  // Put current on the parent of the current node. If the parent does
     not exist, current does not change and false is returned.
  boolean findParent();

  // Put current on the root. If the tree is empty nothing happens.
  void findRoot();

  // Remove the current subtree. The parent of current, if it exists,
     becomes the new current.
  void remove();
}
```

3 Representing a power grid

We will represent a power grid using a general tree. Elements of the grid are represented by the class `PGElem` below:

```
public class PGElem {
  private int id;
  private ElemType type;
  private double power;

  public PGElem(int id, ElemType type, double power) {
    this.id = id;
```

```
    this.type = type;
    this.power = power;
  }

  public int getId() {
    return id;
  }

  public ElemType getType() {
    return type;
  }

  public double getPower() {
    return power;
  }
}
```

The enumeration `ElemType` is defined as follows:

```
public enum ElemType {
  Generator, Transmitter, Consumer
}
```

Using these definitions, write the class `PowerGridUtils`:

```
public class PowerGridUtils {

  // Return the IDs of all elements in the power grid pg in pre-order.
  public static Queue<Integer> collectPreorder(GT<PGElem> pg);

  // Searches the power grid pg for the element with ID id. If found, it
     is made current and true is returned, otherwise false is returned.
  public static boolean find(GT<PGElem> pg, int id);

  // Add the generator element gen to the power grid pg. This can only be
      done if the grid is empty. If successful, the method returns true.
     If there is already a generator, or gen is not of type Generator,
     false is returned.
  public static boolean addGenerator(GT<PGElem> pg, PGElem gen);

  // Attaches pgn to the element id and returns true if successful. Note
     that a consumer can only be attached to a transmitter, and no
     element can be be attached to it. The tree must not contain more
     than one generator located at the root. If id does not exist, or
     there is already aelement with the same ID as pgn, pgn is not
     attached, and the method retrurns false.
  public static boolean attach(GT<PGElem> pg, PGElem pgn, int id);

  // Removes element by ID, all corresponding subtree is removed. If
     removed, true is returned, false is returned otherwise.
  public static boolean remove(GT<PGElem> pg, int id);

  // Computes total power that consumed by a element (and all its subtree
     ). If id is incorrect, -1 is returned.
  public static double totalPower(GT<PGElem> pg, int id);

  // Checks if the power grid contains an overload. The method returns
     the ID of the first element preorder that has an overload and -1 if
     there is no overload.
  public static int findOverload(GT<PGElem> pg);
}
```

4 Deliverable and rules

You must deliver:

1. Source code submission to Web-CAT. You have to upload the following class in a zipped file:

   - `LinkedGT.java`
   - `PowerGridUtils.java`

   Notice that you should **not upload** the interfaces and classes given to you.

The submission **deadline** is: **26/11/2019**.

You have to read and follow the following rules:

1. The specification given in the assignment (**class and interface names, and method signatures**) must not be modified. Any change to the specification results in compilation errors and consequently the mark zero.

2. All data structures used in this assignment **must be implemented** by the student. The use of Java collections or any other data structures library is strictly forbidden.

3. This assignment is an individual assignment. Sharing code with other students will result in harsh penalties.

4. Posting the code of the assignment or a link to it on public servers, social platforms or any communication media including but not limited to Facebook, Twitter or WhatsApp will result in disciplinary measures against any involved parties.

5. The submitted software will be evaluated automatically using Web-Cat.

6. All submitted code will be automatically checked for similarity, and if plagiarism is confirmed penalties will apply.

7. You may be selected for discussing your code with an examiner at the discretion of the teaching team. If the examiner concludes plagiarism has taken place, penalties will apply.