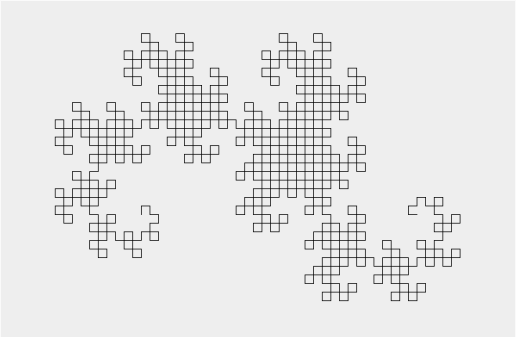
[CI2] Cours 10: Récursion – La courbe du dragon

Daniela Petrişan
Université Paris Cité, IRIF

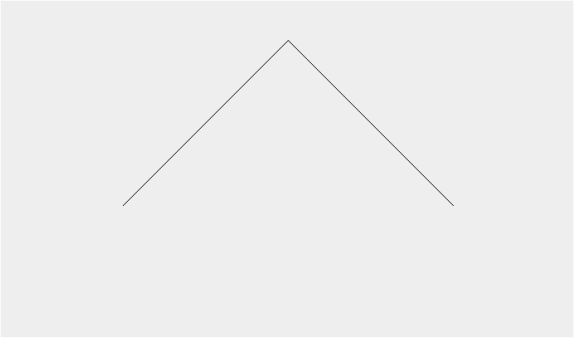
La courbe du dragon



La courbe du dragon est une courbe récursive dont le nom provient de sa ressemblance avec une créature mythique. Elle peut être construite en représentant un virage à droite par `1` et un virage à gauche par `-1`. Chaque courbe d'ordre donné peut ainsi être codée sous forme d'une liste d'entiers.

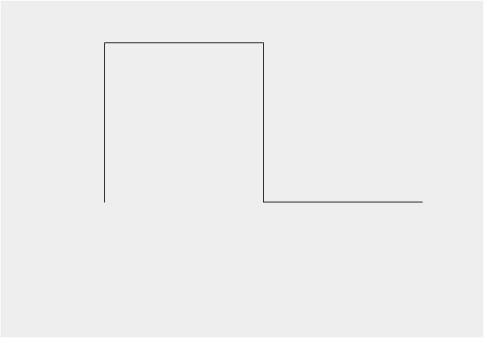
La courbe du dragon d'ordre 1

La courbe du premier ordre : [1]



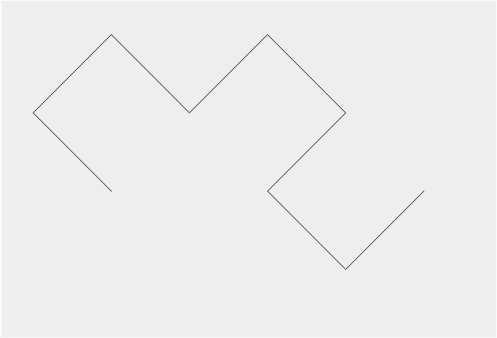
La courbe du dragon d'ordre 2

La courbe du second ordre : [1, 1, -1]



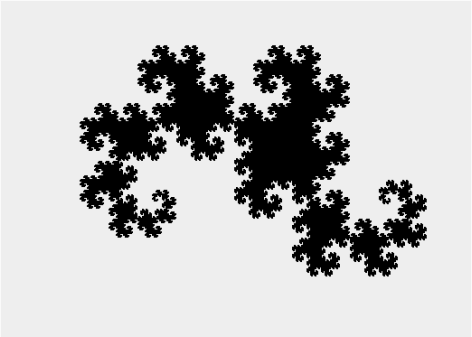
La courbe du dragon d’ordre 3

La courbe d’ordre 3 : [1, 1, -1, 1, 1, -1, -1]



La courbe du dragon d'ordre n

Pour la courbe d'ordre $n > 1$, on concatène le codage de la courbe c d'ordre $n - 1$, celui de la courbe d'ordre 1 et celui de cette même courbe c mais parcourue en sens inverse (en particulier, les virages gauche et droite y sont échangés).



La courbe du dragon d'ordre n

Pour la courbe d'ordre $n > 1$, on concatène le codage de la courbe c d'ordre $n - 1$, celui de la courbe d'ordre 1 et celui de cette même courbe c mais parcourue en sens inverse (en particulier, les virages gauche et droite y sont échangés).

ordre 1 : [1]

ordre 2 : [1, 1, -1] =[1] + [1]+ [-1]

La courbe du dragon d'ordre n

Pour la courbe d'ordre $n > 1$, on concatène le codage de la courbe c d'ordre $n - 1$, celui de la courbe d'ordre 1 et celui de cette même courbe c mais parcourue en sens inverse (en particulier, les virages gauche et droite y sont échangés).

ordre 1 : [1]

ordre 2 : [1, 1, -1]

ordre 3 : [1, 1, -1, 1, 1, -1, -1] = [1, 1, -1] + [1] + [1, -1, -1]

La courbe du dragon d'ordre n

Pour la courbe d'ordre $n > 1$, on concatène le codage de la courbe c d'ordre $n - 1$, celui de la courbe d'ordre 1 et celui de cette même courbe c mais parcourue en sens inverse (en particulier, les virages gauche et droite y sont échangés).

ordre 1 : [1]

ordre 2 : [1, 1, -1]

ordre 3 : [1, 1, -1, 1, 1, -1, -1]

ordre 4 : [1, 1, -1, 1, 1, -1, -1, 1, 1, 1, -1, -1, 1, -1, -1] = [1, 1, -1, 1, 1, -1, -1] + [1] + [1, 1, -1, -1, 1, -1, -1]

La courbe du dragon d'ordre n

Pour la courbe d'ordre $n > 1$, on concatène le codage de la courbe c d'ordre $n - 1$, celui de la courbe d'ordre 1 et celui de cette même courbe c mais parcourue en sens inverse (en particulier, les virages gauche et droite y sont échangés).

ordre 1 : [1]

ordre 2 : [1, 1, -1]

ordre 3 : [1, 1, -1, 1, 1, -1, -1]

ordre 4 : [1, 1, -1, 1, 1, -1, -1, 1, 1, 1, -1, -1, 1, -1, -1]

ordre 5 : [1, 1, -1, 1, 1, -1, -1, 1, 1, 1, -1, -1, 1, -1, -1, 1, 1, 1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, 1, -1, -1]

La courbe suivante

Écrire une méthode suivant qui prend le codage d'une courbe en paramètre et renvoie le codage de la courbe d'ordre immédiatement supérieur.

```
public List<Integer> suivant(List<Integer> turns) {
     List<Integer> copy = new ArrayList<Integer>(turns);
     Collections.reverse(copy);
     turns.add(1);
     for (Integer turn : copy) {
         turns.add(-turn);
     }
     return turns;
}
```

La courbe du dragon

```
public List<Integer> suivant(List<Integer> turns) {
     List<Integer> copy = new ArrayList<Integer>(turns);
     Collections.reverse(copy);
     turns.add(1);
     for (Integer turn : copy) {
         turns.add(-turn);
     }
     return turns;
}

public List<Integer> getSequence(int n){
     if (n == 1) {
          return new ArrayList<>(Arrays.asList(1));
     }
     return suivant(getSequence(n-1));
}
```

La courbe du dragon avec récursivité croisée

Rappel : Pour la courbe d'ordre $n > 1$, on concatène le codage de la courbe c d'ordre $n - 1$, celui de la courbe d'ordre 1 et celui de cette même courbe c mais parcourue en sens inverse (en particulier, les virages gauche et droite y sont échangés).

La fonction `getSequenceR` renvoie le codage de la courbe d'ordre n.

La fonction `getSequenceL` renvoie le codage de la courbe d'ordre n mais parcourue en sens inverse (en particulier, les virages gauche et droite y sont échangés).

La courbe du dragon avec récursivité croisée

Rappel : Pour la courbe d'ordre $n > 1$, on concatène le codage de la courbe c d'ordre $n - 1$, celui de la courbe d'ordre 1 et celui de cette même courbe c mais parcourue en sens inverse (en particulier, les virages gauche et droite y sont échangés).

La fonction `getSequenceR` renvoie le codage de la courbe d'ordre n.

La fonction `getSequenceL` renvoie le codage de la courbe d'ordre n mais parcourue en sens inverse (en particulier, les virages gauche et droite y sont échangés).

La fonction `getSequenceR` sera définie récursivement en fonction de `getSequenceR` et `getSequenceL`. De même, `getSequenceL` est définie récursivement en termes de `getSequenceR` et `getSequenceL`. On dit alors qu'il y a une récursivité croisée.

La courbe du dragon avec récursivité croisée

La fonction `getSequenceR` sera définie récursivement en fonction de `getSequenceR` et `getSequenceL`. De même, `getSequenceL` est définie récursivement en termes de `getSequenceR` et `getSequenceL`. On dit alors qu'il y a une récursivité croisée.

```
public List<Integer> getSequenceR(int iter) {
     if (iterations == 1) {
            return new ArrayList<>(Arrays.asList(1));
     }
     List<Integer> turnSequence = getSequenceR(iter - 1);
     turnSequence.add(1);
     turnSequence.addAll(getSequenceL(iter - 1));
     return turnSequence;
}
```

La courbe du dragon avec récursivité croisée

La fonction `getSequenceR` sera définie récursivement en fonction de `getSequenceR` et `getSequenceL`. De même, `getSequenceL` est définie récursivement en termes de `getSequenceR` et `getSequenceL`. On dit alors qu'il y a une récursivité croisée.

```
public List<Integer> getSequenceR(int iter) {
    if (iterations == 1) {
        return new
        ↪ ArrayList<>(Arrays.asList(1));
    }
    List<Integer> turnSequence =
    ↪ getSequenceR(iter - 1);
    turnSequence.add(1);
    turnSequence.addAll(getSequenceL(iter - 1));
    return turnSequence;
}
```

```
public List<Integer> getSequenceL(int iter) {
    if (iterations == 1) {
        return new
        ↪ ArrayList<>(Arrays.asList(...));
    }
    ...
    ...
    ...
    ...
    return turnSequence;
}
```

La courbe du dragon avec récursivité croisée

La fonction `getSequenceR` sera définie récursivement en fonction de `getSequenceR` et `getSequenceL`. De même, `getSequenceL` est définie récursivement en termes de `getSequenceR` et `getSequenceL`. On dit alors qu'il y a une récursivité croisée.

```
public List<Integer> getSequenceR(int iter) {
    if (iterations == 1) {
        return new
        ↪ ArrayList<>(Arrays.asList(1));
    }
    List<Integer> turnSequence =
    ↪ getSequenceR(iter - 1);
    turnSequence.add(1);
    turnSequence.addAll(getSequenceL(iter - 1));
    return turnSequence;
}
```

```
public List<Integer> getSequenceL(int iter) {
    if (iterations == 1) {
        return new
        ↪ ArrayList<>(Arrays.asList(-1));
    }
    ...
    ...
    ...
    ...
    return turnSequence;
}
```

La courbe du dragon avec récursivité croisée

La fonction `getSequenceR` sera définie récursivement en fonction de `getSequenceR` et `getSequenceL`. De même, `getSequenceL` est définie récursivement en termes de `getSequenceR` et `getSequenceL`. On dit alors qu'il y a une récursivité croisée.

```
public List<Integer> getSequenceR(int iter) {
     if (iterations == 1) {
            return new
             ↪  ArrayList<>(Arrays.asList(1));
     }
     List<Integer> turnSequence =
      ↪  getSequenceR(iter - 1);
     turnSequence.add(1);
     turnSequence.addAll(getSequenceL(iter - 1));
     return turnSequence;
}
```

```
public List<Integer> getSequenceL(int iter) {
     if (iterations == 1) {
            return new
             ↪  ArrayList<>(Arrays.asList(-1));
     }
     List<Integer> turnSequence =
      ↪  getSequenceR(iter - 1);
     ...
     ...
     ...
     return turnSequence;
}
```

La courbe du dragon avec récursivité croisée

La fonction `getSequenceR` sera définie récursivement en fonction de `getSequenceR` et `getSequenceL`. De même, `getSequenceL` est définie récursivement en termes de `getSequenceR` et `getSequenceL`. On dit alors qu'il y a une récursivité croisée.

```
public List<Integer> getSequenceR(int iter) {
    if (iterations == 1) {
        return new
        ↪ ArrayList<>(Arrays.asList(1));
    }
    List<Integer> turnSequence =
    ↪ getSequenceR(iter - 1);
    turnSequence.add(1);
    turnSequence.addAll(getSequenceL(iter - 1));
    return turnSequence;
}
```

```
public List<Integer> getSequenceL(int iter) {
    if (iterations == 1) {
        return new
        ↪ ArrayList<>(Arrays.asList(-1));
    }
    List<Integer> turnSequence =
    ↪ getSequenceR(iter - 1);
    turnSequence.add(-1);
    ...
    return turnSequence;
}
```

La courbe du dragon avec récursivité croisée

La fonction `getSequenceR` sera définie récursivement en fonction de `getSequenceR` et `getSequenceL`. De même, `getSequenceL` est définie récursivement en termes de `getSequenceR` et `getSequenceL`. On dit alors qu'il y a une récursivité croisée.

```
public List<Integer> getSequenceR(int iter) {
    if (iterations == 1) {
        return new
        ↪ ArrayList<>(Arrays.asList(1));
    }
    List<Integer> turnSequence =
    ↪ getSequenceR(iter - 1);
    turnSequence.add(1);
    turnSequence.addAll(getSequenceL(iter - 1));
    return turnSequence;
}
```

```
public List<Integer> getSequenceL(int iter) {
    if (iterations == 1) {
        return new
        ↪ ArrayList<>(Arrays.asList(-1));
    }
    List<Integer> turnSequence =
    ↪ getSequenceR(iter - 1);
    turnSequence.add(-1);
    turnSequence.addAll(getSequenceL(iter - 1));
    return turnSequence;
}
```

La courbe du dragon avec récursivité croisée

La fonction `getSequenceR` sera définie récursivement en fonction de `getSequenceR` et `getSequenceL`. De même, `getSequenceL` est définie récursivement en termes de `getSequenceR` et `getSequenceL`. On dit alors qu'il y a une récursivité croisée.

```
public List<Integer> getSequenceR(int iter) {
    if (iterations == 1) {
        return new
        ↪ ArrayList<>(Arrays.asList(1));
    }
    List<Integer> turnSequence =
    ↪ getSequenceR(iter - 1);
    turnSequence.add(1);
    turnSequence.addAll(getSequenceL(iter - 1));
    return turnSequence;
}
```

```
public List<Integer> getSequenceL(int iter) {
    if (iterations == 1) {
        return new
        ↪ ArrayList<>(Arrays.asList(-1));
    }
    List<Integer> turnSequence =
    ↪ getSequenceR(iter - 1);
    turnSequence.add(-1);
    turnSequence.addAll(getSequenceL(iter - 1));
    return turnSequence;
}
```