Laboratory practice No. 1: Recursion

Juan P. Ossa Zapata
Universidad EAFIT
Medellín, Colombia
jpossaz@eafit.edu.co

Mauricio Jaramillo Uparela
Universidad EAFIT
Medellín, Colombia
mjaramillu@eafit.edu.co

August 2, 2019

1) *ONLINE EXERCISES (CODINGBAT)*

1.a. *Recursion I*

i.

```
public int countPairs(String str) {            // c0 * n
  if (str.length() == 2 || str.length() == 1   // c1 * n
    || str.length() == 0) {                    // c1 * n
    return 0;                                  // c2 * n
  } else if (str.charAt(0) == str.charAt(2)) { // c2 * n
    return 1 + countPairs(str.substring(1));   // c2 * T(n-1)
  } else {                                     // c3
    return countPairs(str.substring(1));       // c3 * T(n-1)
  }
}
```

ii.

```
public int countHi2(String str) {              // c0 * n
  if (str.length() == 1 || str.length() == 0) { // c1 * n
    return 0;                                  // c1 * n
  } else if (str.charAt(0) == 'x') {           // c2 * n
    if (str.charAt(1) == 'h'
    && str.charAt(2) == 'i') {                 // c2 * c3 * n
      return countHi2(str.substring(2));       // c2 * c3 * T(n-2)
    } else {                                   // c2 * c4 * T(n-1)
      return countHi2(str.substring(1));       // c2 * c4 * T(n-1)
    }
  } else if (str.charAt(0) == 'h'
    && str.charAt(1) == 'i') {                 // c5 * n
    return 1 + countHi2(str.substring(1));     // c5 * T(n-1)
  } else {                                     // c6 * n
    return countHi2(str.substring(1));         // c6 * T(n-1)
    }
  }
```

iii.

```
  public int countAbc(String str) {                  // c0 * n
    if (str.length() == 0 || str.length() == 1
    || str.length() == 2) {                          // c1 * n
      return 0;                                      // c1 * n
    } else if (str.charAt(0) == 'a'
      && str.charAt(1) == 'b'
      && (str.charAt(2) == 'c'
      || str.charAt(2) == 'a')) {                    // c2 * n
      return 1 + countAbc(str.substring(1));         // c2 * T(n-1)
    } else {                                         // c3 * n
      return countAbc(str.substring(1));             // c3 * T(n-1)
    }
  }
```

iv.

```
  public String parenBit(String str) {
    if (str.length() == 0 || str.length() == 1) {
      return "";
    } else if (str.charAt(0) == '(') {
      int count = 0;
      while (str.charAt(count) != ')') {
        count++;
      }
      count++;
      return str.substring(0, count) + parenBit(str.substring(count));
    } else {
      return parenBit(str.substring(1));
    }
  }
```

v.

```
  public int strCount(String str, String sub) {
    if (str.length() == 0) {
      return 0;
    } else {
      int i = 0;
      while (i < sub.length()) {
        if (sub.charAt(i) == str.charAt(i)) {
          i++;
        } else {
          break;
        }
      }
      if (i == sub.length()) {
        return 1 + strCount(str.substring(i), sub);
      } else {
        return strCount(str.substring(1), sub);
      }
    }
  }
```

1.b. Recursion II

i.
```
public boolean splitArray(int[] nums) {
  return splitArrayAux(nums, 0, 0, 0);
}
public boolean splitArrayAux(int [] nums, int start,
  int first, int second) {
  if (start == nums.length) {
    return first == second;
  } else {
    return splitArrayAux(nums, start + 1,
      first + nums[start], second)
    || splitArrayAux(nums, start + 1, first,
      second + nums[start]);
  }
}
```

ii.
```
public boolean splitOdd10(int[] nums) {
  return splitOdd10Aux(nums, 0, 0, 0);
}
public boolean splitOdd10Aux(int [] nums, int start,
  int first, int second) {
  if (start == nums.length) {
    return (first % 10 == 0) && (second % 2 != 0);
  } else {
    return splitOdd10Aux(nums, start + 1,
      first + nums[start], second) ||
    splitOdd10Aux(nums, start + 1,
      first, second + nums[start]);
  }
}
```

iii.
```
public boolean groupSumClump(int start, int[] nums, int target) {
  if (start >= nums.length) {
    return target == 0;
  }
  int sum = 0;
  int i;
  for (i = start; i < nums.length; i++) {
    if (nums[i] == nums[start]){
      sum += nums[start];
    } else {
      break;
    }
  }
  return groupSumClump(i, nums, target - sum)
  || groupSumClump(i, nums, target);
}
```

iv.

```
public boolean groupSum5(int start, int[] nums, int target) {
  if (start == nums.length) {
    return target == 0;
  } else {
    if (nums[start] % 5 == 0) {
      return groupSum5(start + 1, nums, target - nums[start]);
    } else if (start > 0 && nums[start] == 1
      && nums[start - 1] % 5 == 0) {
      return groupSum5(start + 1, nums, target);
    } else {
      return groupSum5(start + 1, nums, target - nums[start])
      || groupSum5(start + 1, nums, target);
    }
  }
}
```

v.

```
public boolean split53(int[] nums) {
  return split53Aux(nums, 0, 0, 0);
}
public boolean split53Aux(int [] nums, int start,
  int first, int second) {
  if (start == nums.length) {
    return first == second;
  } else {
    if (nums[start] % 5 == 0) {
      return split53Aux(nums, start + 1, first + nums[start], second);
    } else if (nums[start] % 3 == 0) {
      return split53Aux(nums, start + 1, first, second + nums[start]);
    } else {
      return split53Aux(nums, start + 1, first + nums[start], second)
      || split53Aux(nums, start + 1, first, second + nums[start]);
```

```
        }
      }
    }
```

2) *What did you learn about `Stack Overflow`?*

The Stack Overflow error is caused by a bad recursive call -for example you do not make the problem simpler every time you make a recursive call- or when you do not have a stopping condition. In Java,