

Laboratory practice No. 1: Recursion

Juan S. Cárdenas Rodríguez

Universidad EAFIT
Medellín, Colombia
jscardenar@eafit.edu.co

David Plazas Escudero

Universidad EAFIT
Medellín, Colombia
dplazas@eafit.edu.co

August 24, 2017

1) ONLINE EXERCISES (CODINGBAT)

1.a. Recursion I

```
i.      public int countPairs(String str) {                // c0
        if (str.length() <= 2) {                          // c1
            return 0;                                     // c2
        } else if (str.charAt(0) == str.charAt(2)) {      // c3
            return 1 + countPairs(str.substring(1));     // c4 + T(n-1)
        } else {                                         // c4
            return countPairs(str.substring(1));         // T(n-1)
        }
    }
```

Complexity of `countPairs` can be written as:

$$T(n) = \begin{cases} c_0 + c_1 + c_2 & n \leq 2 \\ c_3 + c_4 + T(n-1) & n > 2 \end{cases}$$

Solving the recursive equation yields:

$$T(n) = (c_3 + c_4)n + k$$

Therefore, $T(n)$ is $O(cn + k)$ and applying the sum and product rule $T(n)$ is $O(n)$.

```
ii.     public int countHi2(String str) {                // c0
        if (str.length() == 1 || str.length() == 0) {    // c1
            return 0;                                     // c2
        } else if (str.charAt(0) == 'x') {               // c3
            if (str.charAt(1) == 'h'
                && str.charAt(2) == 'i') {                // c4
```

```

        return countHi2(str.substring(2));        // T(n-2)
    } else {                                       // c5
        return countHi2(str.substring(1));        // T(n-1)
    }
} else if (str.charAt(0) == 'h'
&& str.charAt(1) == 'i') {                       // c5
    return 1 + countHi2(str.substring(1));        // c5
} else {                                         // c6
    return countHi2(str.substring(1));            // T(n-1)
}
}

```

The complexity of `countHi2` can be written as:

$$T(n) = \begin{cases} c_0 + c_1 + c_2 & n \leq 1 \\ c_5 + T(n-1) & n > 1 \end{cases}$$

Solving the recursive equation for this algorithm, yields:

$$T(n) = c_5n + k$$

Then, $T(n)$ is $O(c_5n + k)$ and applying the sum and product rule $T(n)$ is $O(n)$.

iii.

```

public int countAbc(String str) {                // c0
    if (str.length() == 0 || str.length() == 1
|| str.length() == 2) {                         // c1
        return 0;                               // c2
    } else if (str.charAt(0) == 'a'
&& str.charAt(1) == 'b'
&& (str.charAt(2) == 'c'
|| str.charAt(2) == 'a')) {                     // c3
        return 1 + countAbc(str.substring(1));  // c4 + T(n-1)
    } else {                                     // c5
        return countAbc(str.substring(1));      // T(n-1)
    }
}

```

The complexity of `countAbc` can be written as:

$$T(n) = \begin{cases} c_0 + c_1 + c_2 & n \leq 2 \\ c_3 + c_4 + T(n-1) & n > 2 \end{cases}$$

The solution to this recursive equation yields:

$$T(n) = (c_3 + c_4)n + k$$

Therefore, $T(n)$ is $O((c_3 + c_4)n + k)$ and applying the sum and product rule $T(n)$ is $O(n)$.

```
iv.    public String parenBit(String str) {                // c0
        int a = str.length();                             // c1
        if (a <= 1) {                                     // c2
            return "";                                     // c3
        }
        if (str.substring(a - 1).equals("(")) {           // c4
            int paren = str.indexOf("(");                  // c5
            return str.substring(paren);                   // T(n-k)
        }
        return parenBit(str.substring(0,a - 1));          // T(n-1)
    }
```

The complexity of `parenBit` can be written as:

$$T(n) = \begin{cases} c_0 + c_1 + c_2 + c_3 & n \leq 1 \\ c_4 + T(n - 1) & n > 1 \end{cases}$$

Solving the recursive equation yields:

$$T(n) = c_4n + k$$

Then $T(n)$ is $O(c_4n + k)$ and applying the product and sum rules, we obtain that $T(n)$ is $O(n)$.

```
v.    public int strCount(String str, String sub) {
        int a = str.length();
        int b = sub.length();
        if (a < b || b == 0){
            return 0;
        }
        if (str.substring(a - b).equals(sub)) {
            return 1 + strCount(str.substring(0,a - b),sub);
        }
        return strCount(str.substring(0,a - 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) {                // c1
        if (start == nums.length) {             // c2
            return first == second;              // c3
        }
```

```

    } else {                                     // c4
        return splitArrayAux(nums, start + 1,
            first + nums[start], second)
        || splitArrayAux(nums, start + 1, first,
            second + nums[start]);               // c5 + 2T(n-1)
    }
}

```

Complexity of `splitArray` can be written as:

$$T(n) = \begin{cases} c_1 + c_2 + c_3 + c_4 & n = 0 \\ c_5 + 2T(n-1) & n \neq 0 \end{cases}$$

The solution to this recursive equation is:

$$T(n) = k2^{n-1} + (2^n - 1)(c_1 + c_2 + c_3 + c_4 + c_5)$$

Then, $T(n)$ is $O(k2^{n-1} + (2^n - 1)(c_1 + c_2 + c_3 + c_4 + c_5))$.

Therefore, applying the sum and product rule $T(n)$ is $O(2^n)$.

ii.

```

public boolean splitOdd10(int[] nums) {
    return splitOdd10Aux(nums, 0, 0, 0);
}

public boolean splitOdd10Aux(int [] nums, int start,
    int first, int second) {               // c1
    if (start == nums.length) {           // c2
        return (first % 10 == 0) && (second % 2 != 0); // c3
    } else {                               // c4
        return splitOdd10Aux(nums, start + 1;
            first + nums[start], second) ||
            splitOdd10Aux(nums, start + 1,
                first, second + nums[start]); // c5 + 2T(n-1)
    }
}

```

Complexity of `splitOdd10` can be written as:

$$T(n) = \begin{cases} c_1 + c_2 + c_3 + c_4 & n = start \\ c_5 + 2T(n-1) & n \neq start \end{cases}$$

The solution to this recursive equation is:

$$T(n) = k2^{n-1} + (2^n - 1)(c_1 + c_2 + c_3 + c_4 + c_5)$$

Then, $T(n)$ is $O(k2^{n-1} + (2^n - 1)(c_1 + c_2 + c_3 + c_4 + c_5))$.

Therefore, applying the sum and product rule $T(n)$ is $O(2^n)$.

```

iii.    public boolean groupSumClump(int start, int[] nums,
        int target) {                                // c1
        if (start >= nums.length) {                  // c2
            return target == 0;                        // c3
        }
        int sum = 0;                                  // c4
        int i;                                         // c5
        for (i = start; i < nums.length; i++) {       // c6 * n
            if (nums[i] == nums[start]) {              // c7 * n
                sum += nums[start];                    // c8 * n
            } else {                                   // c9 * n
                break;                                 // c10
            }
        }
        return groupSumClump(i, nums, target - sum)
        || groupSumClump(i, nums, target);            // 2T(n-1)
    }

```

Can be written as:

$$T(n) = \begin{cases} c_3 & n \leq \text{start} \\ c_1 + c_2 + c_4 + c_5 + (c_6 + c_7 + c_8)n + 2T(n-1) & n > \text{start} \end{cases}$$

The solution to this recursive equation is:

$$T(n) = 2^{n-1}(c + 4c_1) + c_2(2^n - 1) - c_1(n + 2).$$

Then, $T(n)$ is $O(2^{n-1}(c + 4c_1) + c_2(2^n - 1) - c_1(n + 2))$

Therefore, applying the sum and product rule $T(n)$ is $O(2^n)$.

```

iv.    public boolean groupSum5(int start, int[] nums, int target) {
        if (start == nums.length) {                  // c1
            return target == 0;                        // c2
        } else {                                     // c3
            if (nums[start] % 5 == 0) {                // c4
                return groupSum5(start + 1, nums,
                    target - nums[start]);              // c5 + T(n-1)
            } else if (start > 0 && nums[start] == 1
                && nums[start - 1] % 5 == 0) {          // c6
                return groupSum5(start + 1, nums, target); // c7 + T(n-1)
            } else {                                   // c8
                return groupSum5(start + 1, nums,
                    target - nums[start])
                || groupSum5(start + 1, nums, target);  // c9 + 2T(n-1)
            }
        }
    }

```

```
    }
}
```

Taking into account that the case $c_9 + 2T(n - 1)$ is the worst out of all, we can write the recursive equation as:

$$T(n) = \begin{cases} c_2 & n = start \\ c_1 + c_3 + c_4 + c_6 + c_8 + c_9 + 2T(n - 1) & n \neq start \end{cases}$$

The solution to this recursive equation is:

$$T(n) = (c_1 + c_3 + c_4 + c_6 + c_8 + c_9)(2^n - 1) + c_2^{n-1}$$

```
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]);
            }
        }
    }
}
```

[3]

2) *ArrayMax*

3) *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.[2] Java Stack memory is used for execution of a thread. Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method.[1]

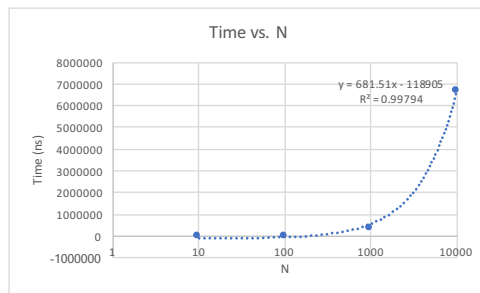


Figure 1: Time vs. N for ArrayMax

N	Time (ns)
10	6000
100	27000
1000	346000
10000	6717000

Table 1: ArrayMax's data.

References

- [1] Pankaj. Java heap space vs. stack – memory allocation in java. <http://www.journaldev.com/4098/java-heap-space-vs-stack-memory>, 2017.
- [2] Sean. What is a stackoverflowerror? <https://stackoverflow.com/questions/214741/what-is-a-stackoverflowerror>, 2008.
- [3] WolframAlpha. <https://www.wolframalpha.com/>.