COMP4019 - Lab Session 5 – Old Friends; Amortized Cost; Graphs

Xavier Carpent & Ian Knight

November 4, 2021

Complexity Involving Multiple Variables

We (somewhat furtively) started talking about algorithms where the input is specified by two or more variables. Let's investigate how to characterize the complexity of such operations.

- Consider the following three functions. What is their complexity? What is different about F3 (think back to the coin change example)?

```
function F1(A, B)
    for a ∈ A do
        Print(a)
    end for
    for b ∈ B do
        Print(b)
    end for
```

```
function F2(A, B)
    for a ∈ A do
        for b ∈ B do
            Print(a + b)
        end for
    end for
```

```
function F3(S)
    for i ∈ {1, ..., S/2} do
        Print((i, S − i))
    end for
```

- Can you simplify $O(\log n - m)$? What if you know that $m = O(\log n)$? What if you know that $m = \Omega(\log n)$?
- Compare $\Theta(\sum_{i=1}^{n} k_i)$ and $\Theta(\max_{1 \leq i \leq n} k_i)$.
- Compare $O(t^h)$ and $O(h^t)$.

Note: in the artificial examples above, different variable names are used on purpose to reinforce the fact that, while n (and to some extent m) are very often used as input size, it is not always the case. Remember that it is always important to specify what is n, m, etc. (except perhaps when there is no ambiguity).

Amortized Complexity

Consider an array *counter* that represents an integer in binary format. The following function increments the number represented by *counter*. What is its worst case complexity? What is its amortized complexity?

```
function INCREMENT(counter)
    i ← 0
    n ← LENGTH(counter)
    while i < n and counter[i] = 1 do
        counter[i] ← 0
        i ← i + 1
    if i < n then
        counter[i] ← 1
return AM
```

Graphs

Below is an imperative-style pseudocode for a function that converts a graph $G = (V, E)$ from a *Adjacency List* representation to a *Adjacency Matrix* representation. Write (in pseudocode or the language of your choice) a function that performs the opposite. Analyze the complexity of both in terms of $|V|$ and $|E|$. Is the situation different for a *sparse* graph ($|E| = O(|V|)$) versus for a *dense* graph ($|E| = O(|V|^2)$)?

```
function AL2AM(AL)
    n ← LENGTH(AL)
    AM ← ARRAY(n)                    ▷ Returns an empty array of length n
    for i ∈ {1, ..., n} do
        AM[i] ← ARRAY(n)
        for j ∈ {1, ..., n} do
            AM[i][j] ← 0
        end for
    end for
    for i ∈ {1, ..., n} do
        for v ∈ AL[i] do
            AM[i][j] ← 1
        end for
    end for
return AM
```