Lecture 3: Levels of Abstraction

http://www.cs.virginia.edu/cs216

Recap
• Big-$O$: the set $O(f)$ is the set of functions that grow no faster than $f$
  – There exist positive integers $c, n_0 > 0$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
• Omega ($\Omega$): the set $\Omega(f)$ is the set of functions that grow no slower than $f$
  – There exist positive integers $c, n_0 > 0$ s.t. $f(n) \geq cg(n)$ for all $n \geq n_0$.

Question from last class
Given $f \in O(h)$ and $g \notin O(h)$ which of these are true:

a. For all positive integers $m$, $f(m) < g(m)$.

This is proved false by counterexample.

Question from last class
Given $f \in O(h)$ and $g \notin O(h)$ which of these are true:

a. For all positive integers $m$, $f(m) < g(m)$.

This is exactly our counterexample.

What else might be useful?
Theta (“Order of”)
- Intuition: the set \( \Theta(f) \) is the set of functions that grow as fast as \( f \)
- Definition: \( f(n) \in \Theta(g(n)) \) if and only if both:
  1. \( f(n) \in O(g(n)) \)
  2. \( f(n) \in \Omega(g(n)) \)
- Note: we do not have to pick the same \( c \) and \( n_0 \) values for 1 and 2
- When we say, “\( f \) is order \( g \)” that means \( f(n) \in \Theta(g(n)) \)

Little-oh (o)
- Definition: \( f \in o(g) \): for all positive constants \( c \) there is a value \( n_0 \) such that \( f(n) \leq cg(n) \) for all \( n \geq n_0 \).
- Compare: \( f \in O(g) \): there are positive constants \( c \) and \( n_0 \) such that \( f(n) \leq cg(n) \) for all \( n \geq n_0 \).

Summary
- Big-O: there exist \( c, n_0 > 0 \) such that \( f(n) \leq cg(n) \) for all \( n \geq n_0 \).
- Omega (\( \Omega \)): there exist \( c, n_0 > 0 \) s.t. \( f(n) \geq cg(n) \) for all \( n \geq n_0 \).
- Theta (\( \Theta \)): both O and \( \Omega \) are true
- Little-o: there exists \( n_0 > 0 \) such that \( f(n) \leq cg(n) \) for all \( n \geq n_0 \).

(Trick) Question
If wealth\((n)\) is your net worth \( n \) days after today, would you prefer:
a. wealth\((n)\) \( \in O(n) \)
b. wealth\((n)\) \( \in O(n^2) \)
c. wealth\((n)\) \( \in o(n) \)
d. wealth\((n)\) \( \in \Omega(n) \)
Which of these are satisfied by \( \text{wealth}(n) = 0.0001n \)?
Which is better: \( \text{wealth}(n) = 100000000 \) \( \text{wealth}(n) = 0.0001n \)?
Course Goal 3

Understand how a program executes at levels of abstraction ranging from a high-level programming language to machine memory.

From Lecture 1...

Levels of Abstraction: Program

Real World Problem

High-Level Program

Machine Instructions

Physical Processor

Tic-Tac-Toe

Play Tic-Tac-Toe

Tinkertoy Computer

Tic-Tac-Toe Strategy

Low-level description

Sequence Alignment: Program

Genome Similarity

High-Level Program

Python Interpreter

Align.py

Low-Level Program

Electrons, etc.

Levels of Abstraction: Data

Real World Thing(s)

Data Abstraction

Low-Level Data Structure

Bits

Electrons, etc.
Levels of Abstraction: PS1

- Genome
- Physical World
- Virtual World
- Python List
- Bits
- Electrons, etc.

List Abstract Datatype

- Ordered collection datatype: 
  \(<x_0, x_1, ..., x_{n-1}>\)
- Operations for manipulating and observing elements of list

List Operations (Ch 3)

- Access \((L, i)\): returns the \(i^{th}\) element of \(L\)
- Length \((L)\): returns the number of elements in \(L\)
- Concat \((L, M)\): returns the result of concatenating \(L\) with \(M\). (Elements \(<l_0, l_1, ..., l_{|L|-1}, m_0, m_1, ..., m_{|M|-1}, >\))
- MakeEmptyList(): returns \(<\rangle\)
- IsEmptyList\((L)\): returns true iff \(|L| = 0\).

Is this a sufficient list of List operations?

Revised List Operations

- Access \((L, i)\): returns \(L[i]\)
- Length \((L)\): returns \(|L|\)
- Concat \((L, M)\): returns the result of concatenating \(L\) with \(M\).
- MakeEmptyList(): returns \(<\rangle\)
- IsEmptyList\((L)\): returns true iff \(|L| = 0\).
- Append \((L, e)\): returns the result of appending \(e\) to \(L\): \(<l_0, l_1, ..., l_{|L|-1}, e>\)

Can define using Append, Access, Length

Are all of these operations necessary?

Constructing Lists

- The book’s list operations have no way of constructing any list other than the empty list!
- We need at least:
  - Append \((L, e)\): returns the result of appending \(e\) to \(L\):
    \(<l_0, l_1, ..., l_{|L|-1}, e>\)

Necessary List Operations

- Access \((L, i)\): returns \(L[i]\)
- Length \((L)\): returns \(|L|\)
- MakeEmptyList(): returns \(<\rangle\)
- Append \((L, e)\): returns the result of appending \(e\) to \(L\):
  \(<l_0, l_1, ..., l_{|L|-1}, e>\)

Note that we have defined an immutable list. There are no operations for changing the value of a list, only making new lists.
### Continuous Representation

![Continuous Representation Diagram]

- **Length:** 3
- **Data:** 1, 2, 3

### Linked Representation

![Linked Representation Diagram]

- **Node Info:** 1, 2, 3
- **Next Node:** None

We need a special value for `Next` when there is no Next node:
- Book: Λ
- C: 0
- Python: None
- Scheme, Java: null

### Necessary List Operations

- **Access** \((L, i)\): returns \(L[i]\)
- **Length** \((L)\): returns \(|L|\)
- **MakeEmptyList**: returns `<>`
- **Append** \((L, e)\): returns the result of appending \(e\) to \(L\): `<l_0, l_1, ..., l_{|L|-1}, e>`

Can we implement all of these with both representation choices?

### Which Representation is Better?

- **Time of Length**: \((n \text{ is number of elements})\)
  - Continuous: \(O(1)\)
  - Linked: \(O(n)\)
  - Are these bounds tight? \(\Theta\)
- **What about other operations?**
- **Other factors to consider?**

Will explore this more next week

### Python Lists

- **Provide necessary operations**:
  - **Access** \((L, i)\): \(L[i]\)
  - **Length** \((L)\): \(\text{len}(L)\)
  - **MakeEmptyList**: \(L = []\)
  - **Append** \((L, e)\): \(L.append(e)\)
Python List Operations

• insert: `L.insert(i, e)`
  Returns `<l₀, l₁, ..., lᵢ, e, lᵢ, ..., l_|L|>`

• concatenation: `L + M`
  Returns `<l₀, l₁, ..., l_|L|-1, m₀, m₁, ..., m_|M|-1>`

• slicing: `L[from:to]`
  Returns `<l_from, l_from + 1, ..., l_to-1>`

• Lots more

How are they implemented?

• PS1
  – Try to “guess” by measuring performance of different operations
  – Unless you can do exhaustive experiments (hint: you can’t) you can’t be assured of a correct guess

• Around PS4:
  – Look an lower abstraction level: C code for the Python List implementation

Charge

• Problem Set 1 due Monday
• Point of PSs is to learn:
  – You can (and should) discuss your approaches and ideas with anyone
  – You should discuss and compare your answers to 1-6 with your assigned partner and produce a consensus best answer that you both understand and agree on

• Take advantage of Small Hall On-Call Hours:
  • Wednesday 7-8:30pm
  • Thursday 4-5:30pm, 6:30-8:30pm
  • Friday 11am-12:30, 3:30-5pm
  • Saturdays 3-6pm
  • Sunday 3:30-9:30pm
• Monday: Dynamic Programming