CSE 250

Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Midterm Review

Midterm Procedure

- Exam is during normal class time. Same time, same place.
- Seating is assigned randomly
  - Wait outside the room until instructed to enter
  - Immediately place all bags/electronics at the front of the room
- At your seat you should have:
  - Writing utensil
  - UB ID card
  - One 8.5x11 cheatsheet (front and back) if desired
  - Summation/Log rules will be provided

Content Overview

Analysis Tools/Techniques	**ADTs**	**Data Structures**
Asymptotic Analysis, (Unqualified) Runtime Bounds		
	Seq	Array
Amortized Runtime	Seq, Buffer	ArrayBuffer
	Seq	Linked Lists
Recursive analysis, divide and conquer, Average/Expected Runtime		
	Stack, Queue	
	Graphs	EdgeList, AdjacencyList Adjacency Matrix

Analysis Tools and Techniques

Limit Tests

Case 1: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$ *(f grows faster; g is better)*

Case 2: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ *(g grows faster; f is better)*

Case 3: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = some\ \ constant$ *(f and g “behave” the same)*

Recap of Runtime Complexity

Big-ϴ — Tight Bound

- Growth functions are in the **same** complexity class
- If f(n) ∈ ϴ(g(n)) then an algorithm taking f(n) steps is as "exactly" as fast as one that takes g(n) steps.

Big-O — Upper Bound

- Growth functions in the **same or smaller** complexity class.
- If f(n) ∈ O(g(n)), then an algorithm that takes f(n) steps is *at least as fast* as one taking g(n) (but it may be even faster).

Big-Ω — Lower Bound

- Growth functions in the **same or bigger** complexity class
- If f(n) ∈ **Ω**(g(n)), then an algorithm that takes f(n) steps is *at least as slow* as one that takes g(n) steps (but it may be even slower)

Common Runtimes (in order of complexity)

Constant Time:	$\Theta(1)$
Logarithmic Time:	$\Theta(\log(n))$
Linear Time:	$\Theta(n)$
Quadratic Time:	$\Theta(n^2)$
Polynomial Time:	$\Theta(n^k)$ for some $k > 0$
Exponential Time:	$\Theta(c^n)$ (for some $c \geq 1$)

Formal Definitions

$f(n) \in O(g(n))$ iff exists some constants **c, n_0** s.t.

  $f(n) \leq c * g(n)$ for all **$n > n_0$**

$f(n) \in \Omega(g(n))$ iff exists some constants **c, n_0** s.t.

  $f(n) \geq c * g(n)$ for all **$n > n_0$**

$f(n) \in \Theta(g(n))$ iff **$f(n) \in O(g(n))$** and **$f(n) \in \Omega(g(n))$**

Amortized Runtime

If n calls to a function take $O(T(n))$...

We say the **Amortized Runtime** is $O(T(n) / n)$

The **amortized runtime** of `append` on an `ArrayBuffer` is: $O(n/n) = O(1)$

The **unqualified runtime** of `append` on an `ArrayBuffer` is: $O(n)$

What guarantees do you get?

If *f*(*n*) is a Tight Bound
The algorithm always runs in $cf(n)$ steps

← Unqualified runtime

If *f*(*n*) is a Worst-Case Bound
The algorithm always runs in at most $cf(n)$

If *f*(*n*) is an Amortized Worst-Case Bound
n invocations of the algorithm **always** run in $cnf(n)$ steps

If *f*(*n*) is an Average Bound
…we don't have any guarantees

Inductive Proofs

Solve for $T(n)$

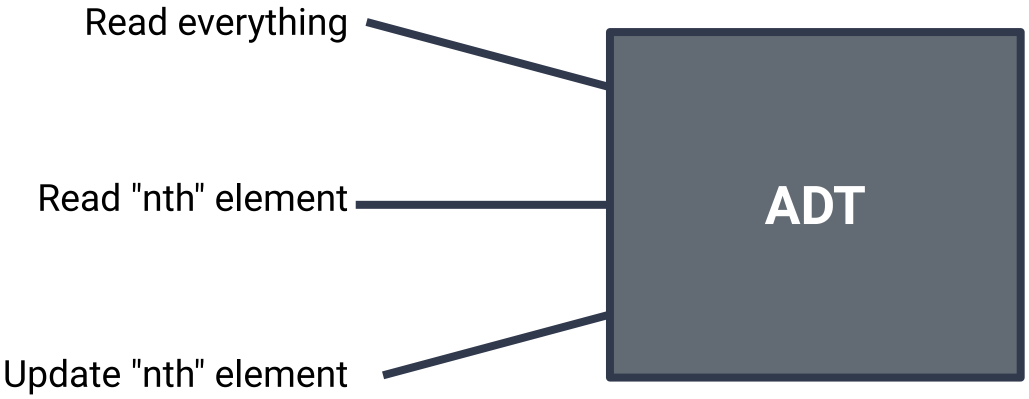
Approach:

1. Generate a hypothesis
2. Prove your hypothesis for the base case
3. Prove the hypothesis for the recursive case *inductively*

ADTs and Data Structures

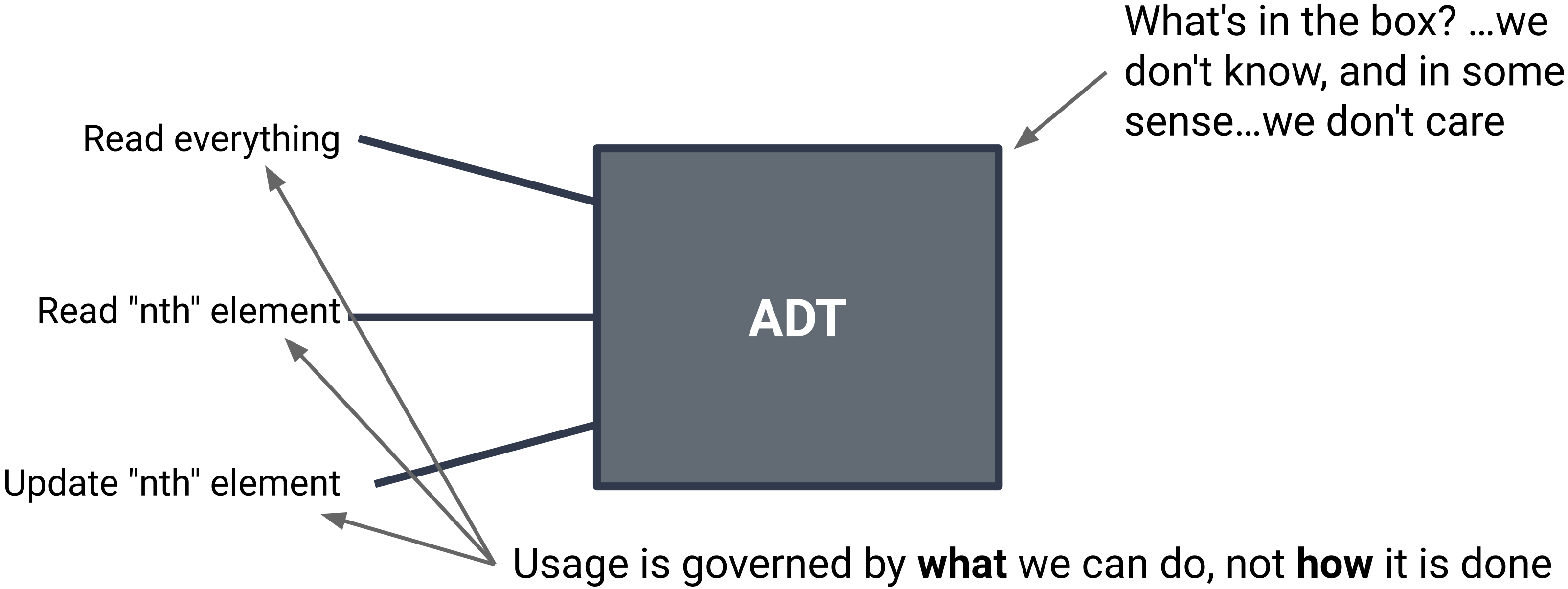
Abstract Data Types (ADTs)

- The specification of what a data structure can do



Abstract Data Types (ADTs)

- The specification of what a data structure can do



Abstract Data Type vs Data Structure

ADT

The interface to a data structure

*Defines **what** the data structure can do*

Many data structures can implement the same ADT

Data Structure

The implementation of one (or more) ADTs

*Defines **how** the different tasks are carried out*

Different data structures will excel at different tasks

Abstract Data Type vs Data Structure

ADT

The interface to

*Defines **what** the*

can

Many data st

implement th

Think about the Linked List we are implementing for PA2.

The internal structure and the mental model of our sequence are very different.

Data Structure

tation of one (or

) ADTs

e different tasks

ried out

ructures will excel

at different tasks

`Seq` Summary

Operation	`Array[T]`	`ArrayBuffer[T]`	`List[T]` *(index)*	`List[T]` *(ref)*
`apply(i)`	$\Theta(1)$	$\Theta(1)$	$\Theta(i)$, $O(n)$	$\Theta(1)$
`update(i, val)`	$\Theta(1)$	$\Theta(1)$	$\Theta(i)$, $O(n)$	$\Theta(1)$
`insert(i, val)`	$\Theta(n)$	$O(n)$	$\Theta(i)$, $O(n)$	$\Theta(1)$
`remove(i, val)`	$\Theta(n)$	$\Theta(n-i)$, $O(n)$	$\Theta(i)$, $O(n)$	$\Theta(1)$
`append(i)`	$\Theta(n)$	$O(n)$, Amortized $\Theta(1)$	$\Theta(i)$, $O(n)$	$\Theta(1)$

Queues vs Stacks (ADTs)

Queue First in, First Out (FIFO)

Stacks Last in, First Out (LIFO / FILO)

Recap

Stacks: Last In First Out (LIFO)

- Push (put item on top of the stack) $\Theta(1)$ (or amortized $O(1)$)
- Pop (take item off top of stack) $\Theta(1)$
- Top (peek at top of stack) $\Theta(1)$

Queues: First in First Out (FIFO)

- Enqueue (put item on the end of the queue) $\Theta(1)$ (or amortized $O(1)$)
- Dequeue (take item off the front of the queue) $\Theta(1)$
- Head (peek at the item in the front of the queue) $\Theta(1)$

A (Directed) Graph ADT

Two type parameters (`Graph[V,E]`)

`V`: The vertex label type

`E`: The edge label type

Vertices

…are elements (like Linked List Nodes)

…store a value of type `V`

Edges

…are also elements

…store a value of type `E`

Edge List Summary

- `addEdge, addVertex`: $O(1)$
- `removeEdge`: $O(1)$
- `removeVertex`: $O(m)$
- `vertex.incidentEdges`: $O(m)$
- `vertex.edgeTo`: $O(m)$
- **Space Used:** $O(n) + O(m)$

Adjacency List Summary

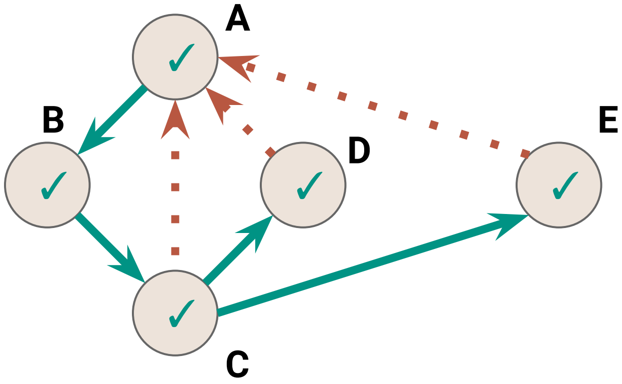
- `addEdge, addVertex`: $O(1)$
- `removeEdge`: $O(1)$
- `removeVertex`: $O(\deg(\text{vertex}))$
- `vertex.incidentEdges`: $O(\deg(\text{vertex}))$
- `vertex.edgeTo`: $O(\deg(\text{vertex}))$
- **Space Used:** $O(n) + O(m)$

Adjacency Matrix Summary

- `addEdge, removeEdge`: $O(1)$
- `addVertex, removeVertex`: $O(n^2)$
- `vertex.incidentEdges`: $O(n)$
- `vertex.edgeTo`: $O(1)$
- **Space Used:** $O(n^2)$

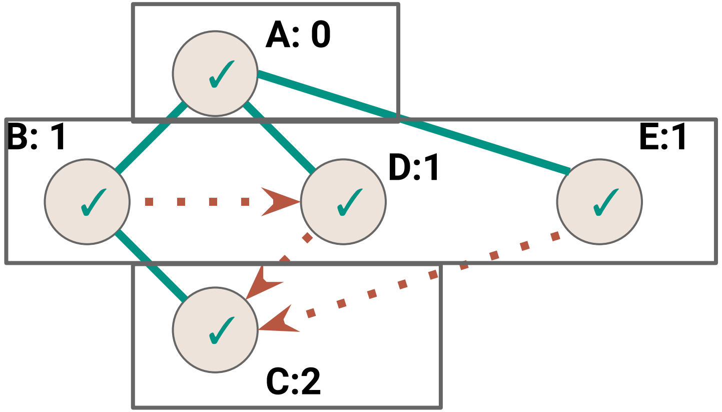
DFS vs BFS

DFS (LIFO order...Stacks)



BACK Edge(*v*,*w*): ***w*** is an ancestor of ***v*** in the discovery tree

BFS (FIFO order...Queues)



CROSS Edge(*v*,*w*): ***w*** is at the same or next level as ***v***

DFS Traversal vs BFS Traversal

Application	DFS	BFS
Spanning Trees	✓	✓
Connected Components	✓	✓
Paths/Connectivity	✓	✓
Cycles	✓	✓
Shortest Paths		✓
Articulation Points	✓	