CS321 Data Structures

Jan 11 2021

Lecture 1

Introduction

Topics

- Expectations
- Syllabus
- Goal – program design/system architecture
- Why Data Structures?

Expectations

- Know Java and can write basic Java programs.
- Are comfortable looking up Java API details.
- Will independently search for solutions to basic programing questions.
- Will correct my arithmetic errors in class – I make arithmetic errors.
- Will look up material independently.

What is a Data Structure?

- Array
- Linked List
- HashTable
- Tree

Introduction to Data Structures

- Why do we have data structures?
- ----????

Introduction to Data Structures

- Why do we have data structures?
  - **To look up data later!**
- All data structures are designed to really do one thing: Let us find the data we want, later, quickly and efficiently.
- All data structures trade off between:
  - Space: memory use.
  - Time: time to add or change data.
  - Complexity: complexity of the algorithm.

Introduction to Data Structures

- This trade off is done for one reason:
  - To optimize the speed at which a data item can be retrieved when an algorithm needs that data item to solve a problem.
- The best data structure for an algorithm is the one that provides the fastest retrieval of a specific data item exactly when it is needed. All other costs being equal.

Introduction to Data Structures

- Good programming is about:
  - Writing good Data Structures that are reliable, robust, efficient, and provide quick access to needed data to the components of the program that need that data.
  - **If you want to write good code write good data structures and organize them carefully.**

Oil Rig Story

- An Oil Company needed to stop all oil rigs simultaneously and recalculate their calibration.
- This took 24 hours of computation time.
- This amounts to many millions of dollars in lost revenue.

Story is Apocryphal = fancy way of saying can not confirm.

Oil Rig Story

- Hired a Computer Scientist to work on this.
- He spent 3 weeks studying the software.
- --company got rather impatient since he was just sitting around.

Oil Rig Story

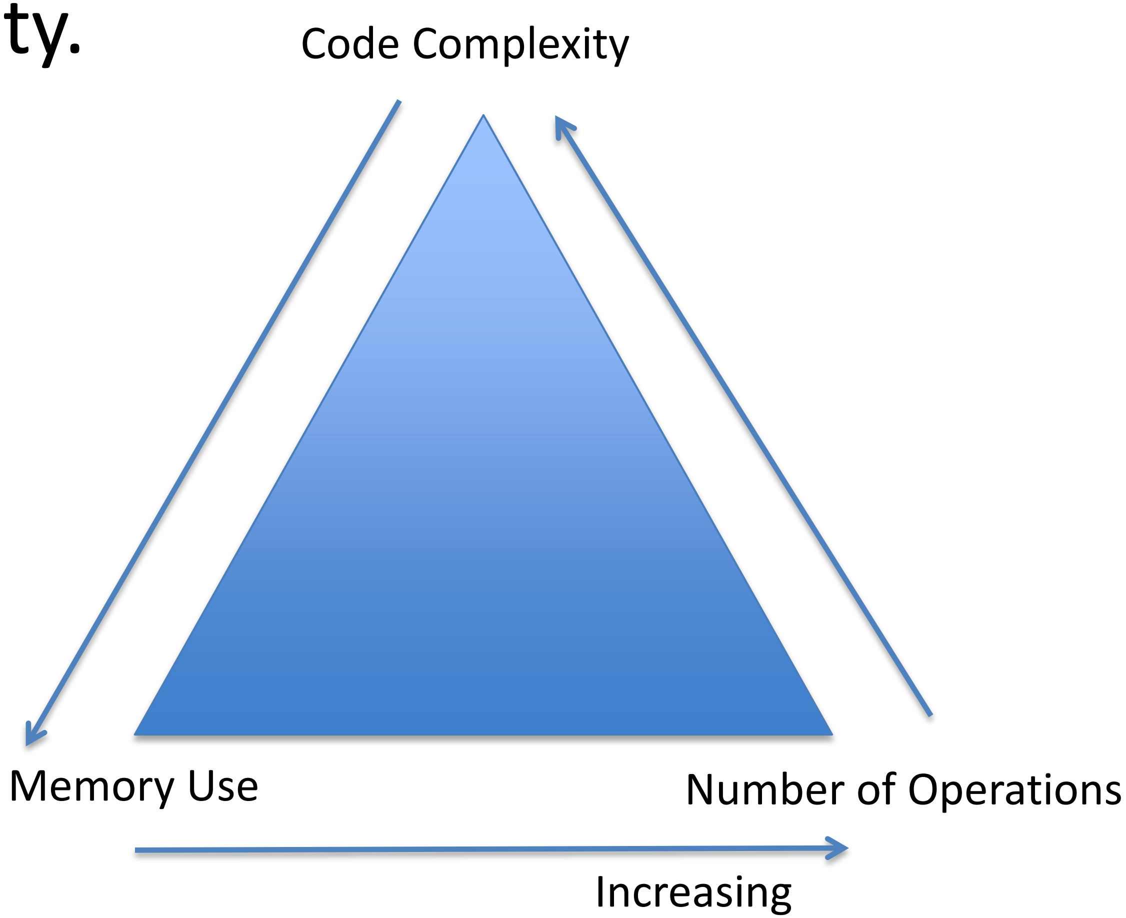
- Hired a Computer Scientist to work on this.
- He spent 3 weeks studying the software.
- Discovered it was basically 3 nested for loops.
- In the center of the for loop was a call to a long computation function.
- This computation computed: A CONSTANT!
- So run time was O(n^3 * C).

Oil Rig Story

- The Fix!
- Compute the CONSTANT outside of the for loops and store it in a variable for later use.
- The variable is our data structure!
- This reduced the total run time from 24 hours to 8 hours!
- The Computer Scientist enjoyed a notable bonus!

So what does this mean?

- Triangle of optimization Space, Time, Complexity.



Sorting Compared

Method/Structure	Space	Complexity	Time
Insertion Sort/Array	O(n)	Simple	$O(n^2)$
Quicksort/Array	O(2*n)	Complex	$O(n^2)$ (A(n*log(n))
Heapsort/Tree	O(n)	Complex	O(n*log(n))
Mergesort/Array	O(2*n)	Simple	O(n*log(n))
Counting Sort/Array	O(k = range of n)	Simple	O(n)

Binary Search Trees

Method/Structure	Space	Complexity	Time
Binary Search Tree	O(n)	Simple	O(n)
AVL Tree	O(n)	Complex	O(log(n))

Abstract Data Type

Search Tree Object Variant #1

```
Atree = createTree();
insertKey (int key, Data *data);
deleteKey(key);
Data *findKey(key);
deleteTree();
```

Search Tree Object Variant #2

```
Atree = createTree();
Node *insertKey (int key, Data *data);
Node *deleteKey(key);
Node *findKey(key);
deleteTree();
```

Abstract Data Type

- An abstract data type (ADT) is the set of minimal methods necessary to define a data structure without regard to how the structure is actually implemented.

Abstract Data Type: Examples

Stack

- Astack = newStack();
- Astack.push(int value);
- Astack.pop(int value);
- deleteStack(astack);

- The internals of the stack could be implemented as an array, tree, linked list but the ADT does not change.

Linked List

- Llist = newList();
- Llist.append(int value);
- Llist.delete(int value);
- Llist.find(int value);
- deleteList(astack);

- This is a linked list ADT but an extremely limited one.

Why Does this Matter?

- It defines a standard by which data structures can be compared and analyzed.
- You can switch different implementations of a linked list without worrying about compatibility if their ADT's are the same.
- It lets you select the specific implementation of an ADT that best matches your applications requirements: speed versus size versus complexity.
- Java has lots of these.

Binary Search Trees

- Both have exactly the same ADT.
- However, a BST with no balancing will have very simple code.
- A Balanced BST (AVL) will have more complex code but better run time performance.
- Both have same memory consumption.

Linked List

- Linked List as an Array: wastes space but simple to code and fixed size!
- Liked List using pointers: space efficient, possible to corrupt memory, or lose pointers, not a fixed size.

Stacks?

- Stack as a linked list?
- Stack implemented using an array?

How do we select a Data Structure?

- Does its ADT have the methods we need?
- How much space does it use?
- What is its worst and best case performance?
- How complex is the code to implement it?

Coming Up

- First assignment comes out on Wednesday.

What is Big O notation?

- A way to approximately count algorithm complexity.
- A way to describe the worst case running time of algorithms.
- A tool to help improve algorithm performance.
- Can be used to count operations and memory usage.

Bounds on Operations

- An algorithm takes some number of steps to complete:
- a + b is a single operation, takes 1 op.
- Adding up N numbers takes N-1 steps.
- O(1) means ‘on order of 1’ operation.
- O(c) means ‘on order of constant’.
- O(n) means ‘ on order of N steps’.
- O(n^2) means ‘ on order of N*N steps’.

O(n) times for sorting algorithms.

Technique	O(n) operations	O(n) memory use
Insertion Sort	$O(N^2)$	O(N)
Bubble Sort	$O(N^2)$	O(N)
Merge Sort	O(N * log(N))	O(N)
Heap Sort	O(N * log(N))	O(N)
Quicksort	$O(N^2)$	O(N)