CS Content Academy: Data Structures

Outline

1. How Does Data Structures Fit into CS—What Is CS?
2. What Are Data Structures and Algorithms?
3. The Structure of Data Structures and Algorithms
   - 3.1 Part A: Container Hierarchy
   - 3.2 Part B: Algorithm Analysis
   - 3.3 Part C: Searching and Sorting Algorithms
   - 3.4 Part D: Advanced Topics
4. Themes
   - 4.1 ADTs and Contiguous vs. Linked Data Structures
   - 4.2 Recursion vs. Stacks
   - 4.3 Trade-Offs
5. Pedagogy
   - 5.1 Projects
   - 5.2 Execution by Hand
   - 5.3 Visualization
   - 5.4 Active Learning
6. Where to Go from Here

1. What is Computer Science?

Computer Science is the discipline that answers the question

What can we compute and how can we compute it?

What can we compute?

Theory	Practice
What counts as computing—computability theory (automata, etc.)	Applications of automata
What can be computed in principle—computability, decidability	Artificial Intelligence Halting problem, etc.
What computations are tractable—abstract complexity theory	Cryptography

How can we compute things?

Hardware versus software—we only deal with software

How can we compute things, from most to least general

Theory	Practice
Algorithms—abstract complexity theory (P, NP, etc.)	Algorithms and data structures Algorithm analysis—concrete complexity theory
Languages—Formal language theory	Programming languages Compilers and interpreters
Graphs—Graph theory	Networking
Logic	Artificial Intelligence Program Validation
	Operating Systems, Database Systems, Games and Simulations, Graphics and Media, Business Applications, Scientific and Engineering Applications, etc.

Computing as a human activity:

Software engineering

Societal and ethical issues

Cyber security and cyber war

2. What Are Data Structures and Algorithms?

Data structure—an arrangement of data in memory.

Examples: arrays, singly linked lists, linked trees, hash tables, etc.

Algorithm—a finite sequence of steps for accomplishing some computational task.

Examples: division algorithm, factorial function, binary search, insertion sort, etc.

Abstract data type (ADT)—a set of values (the **carrier set**), and operations (the **method set**) on those values.

Examples: integers, strings, stacks, queues, etc.

Data type—an implementation of an abstract data type on a computer.

Examples: Java int, String, ArrayList, etc.

Remarks

1. Virtually every program stores some data, and it is *de facto* arranged in memory, so virtually every program uses data structures.
2. Virtually every program has a finite sequence of steps for doing something, so virtually every program has algorithms.
3. Implementing an ADT (making a data type) requires figuring out how to represent values of its carrier set on the computer and how to realize the elements of its method set in sub-programs.
4 Representing values requires arranging data in memory, so implementing ADTs requires data structures.
5. Realizing operations on a computer requires specifying a finite sequence of steps to carry out the operation, so implementing ADTs requires algorithms.
6. Every class embodies an ADT.
7. Programming is largely the design and implementation of ADTs, typically (nowadays) as classes in an OO language, so programming essentially requires using data structures and algorithms.
8. Computer scientists have been studying data structures for over 60 now, so there is a wealth of material on this topic.

3.1 Structure of DS&A: A Container Hierarchy

A Container Hierarchy

container—an entity that holds finitely many other entities.

dispenser—a non-traversable container.

collection—a traversable container.

iterator—an entity that provides serial access to each member of an associated collection.

stack—a dispenser holding a sequence of elements that can be accessed, inserted, or removed at only one end, called the top.

queue—a dispenser holding a sequence of elements that allows insertions only at one end, called the back or rear, and deletions and access to elements at the other end, called the front.

list—an ordered collection.

set—an unordered collection in which an element may appear at most once.

map—an unordered collection of elements, which are values of an *element type*, accessible using a key, which is a value of a *key type*; also called a table, dictionary, or associative array.

3.2 Structure of DS&A: Algorithm Analysis

Framework

1. Choose a measure for the size of the input.
2. Choose a basic operation to count.
3. Determine whether the algorithm has different complexity for various inputs of size n; if so, then derive measures for the *best case* complexity $B(n)$, *worst case* complexity $W(n)$, and *average case* complexity $A(n)$ as functions of the size of the input; if not, then derive a measure for the *every case* complexity $C(n)$ as a function of the size of the input.
4. Determine the order of growth of the complexity measures for the algorithm.

Orders of Growth

n	$\lg n$	n	$n \lg n$	n^2	n^3	2^n	$n!$
10	3.3	10	33	100	1000	1024	3,628,800
100	6.6	100	660	10,000	1,000,000	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
1000	10	1000	10,000	1,000,000	10^9		
10,000	13	10,000	130,000	10^8	10^{12}		
100,000	17	100,000	1,700,000	10^{10}	10^{15}		
1,000,000	20	1,000,000	$2 \cdot 10^7$	10^{12}	10^{18}		

Example: Find the Minimum Value in an Array

Strategy: Examine every element of the array and remember the smallest so far.

```
1 public int selectionSort( int[] A ) {
2    int min = A[0];
3    for ( int i = 1; i < A.length; i++ )
4       if ( A[i] < min ) min = A[i];
5    return min;
6 }
```

1. Input size measure: size of the array n
2. Basic operation: comparison of keys
3. Always does the same thing for inputs of size n, so figure out $C(n)$
4. Order of growth: $O(n)$

3.3 Structure of DS&A: Searching and Sorting Algorithms

Sorting Algorithms

Consider and analyze (time permitting) several standard sorting algorithms, including bubble sort, selection sort, insertion sort, Shell sort, merge sort, quicksort, and heapsort.

Example: Selection Sort

Strategy: repeatedly find the smallest element in the array and place it at the start of the unsorted portion.

```
1 public void selectionSort( int[] A ) {
2    for ( int i = 0; i < A.length-1; i++ ) {
3       int minIndex = i;
4       for ( int j = i+1; j < A.length; j++ )
5          if ( A[j] < A[minIndex] ) minIndex = j;
6       int tmp = A[i]; A[i] = A[minIndex]; A[minIndex] = tmp;
7    }
8 }
```

1. Input size measure: size of the array n
2. Basic operation: comparison of keys
3. Always does the same thing for inputs of size n, so figure out $C(n)$
4. Order of growth: $O(n^2)$

Searching Algorithms

Consider and analyze sequential search and binary search. Introduce binary search trees and consider algorithms on binary trees, including traversal, insertion, deletion, and search.

Example: Sequential Search

Strategy: examine each element until the key is found or the end of the array or list is reached.

```
1 public int sSearch( int[] A, int key ) {
2    for ( int i = 0; i < A.length-1; i++ )
3       if ( key == A[i] ) return i;
4    return -1;
5 }
```

1. Input size measure: size of the array n
2. Basic operation: comparison of keys
3. Behavior varies depending on the key and the array, so figure out $B(n)$, $W(n)$, and $A(n)$
4. $B(n) = 1$, $W(n) = n$, $A(n) \approx n/2$, which are all $O(n)$

3.4 Structure of DS&A: Advanced Topics

More containers:

- Bags
- Dequeues
- Randomizer (random queue)
- Quad trees
- Etc.

Graphs

- Matrix vs adjacency list representations
- Directed vs undirected graphs
- Graph ADT
- Breadth-first vs depth-first search
- Graph algorithms: spanning trees, minmum paths, etc.

Balanced Trees

- AVL trees
- 2-3 trees and red-black trees
- B-trees

Strings

- Sorting
- Tries
- String search algorithms
- Regular expressions
- Data compression

More ...

4.1 Themes: ADTs and Contiguous vs. Linked Data Structures

Every container is first considered as an ADT, and then we tink about how to represent carrier set elements contiguously or using linked structures. Algorithms for these structures are then considered/analyzed to decide when each is best to use.

Example: HashSets vs TreeSets

HashSets can store, remove, and find data a little more quickly than TreeSets, but TreeSets can be traversed in order and HashSets can't.

4.2 Themes: Recursion vs. Stacks

Every algorithm using stacks can be replaced with one using recursion and vice-versa. Sometimes a job is easier to do with a stack, sometimes it is easier with recursion, and sometimes it doesn't matter. Also, recursion can be eliminated without a stack for tail-recursive algorithms. We consider stack-based vs recursive algorithms and elimination of tail-recursion as we study various containers and searching and sorting algorithms.

Example: Recursion Elimination from Binary Search

```
01 public int bSearchRecursive( int[] A, int key ) {
02    return bSearchHelper(A, key, 0, A.length-1);
03 }
04
05 private int bSearchHelper( int[] A, int key, int lo, int hi ) {
06    if ( hi < lo ) return -1;
07    int m = (lo+hi)/2;
08    if ( key == A[m] ) return m;
09    else if ( key < A[m] ) return bSearchHelper(A, key, lo, m-1);
10    else                   return BSearchHelper(A, key, m+1, hi);
11 }
12
13 public int bSearch( int[] A, int key ) {
14    int lo = 0;
15    int hi = A.length-1;
16    while (lo <= hi) {
17       int m = (lo+hi)/2;
18       if ( key == A[m] ) return m;
19       else if ( key < A[m] ) hi = m-1;
20       else lo = m+1;
21    }
22    return -1;
23 }
```

Recursive binary search is tail recursive, so recursion can be eliminated, speeding up the algorithm slightly and using less memory to run it.

4.3 Themes: Trade-Offs

- Time vs Space
- Efficiency vs Complexity/Reliability
- Needed vs Unneeded Operations

5.1 Pedagogy: Projects

Data structures and algorithms are essential to programming, so learning about them is also about learning how to program.

The best way to learn to program is to do it.

Example Projects

- Implement the container hierarchy
- Write a program to compare the time to sort various lists with algorithms

```
Comparison of all sorts on small slices of random data.
   N     Bubble   Select   Insert    Shell    Merge   QBasic QImprove    Heap   Inspct
10000     0.282    0.148    0.074    0.001    0.001    0.001    0.001   0.002    0.001
20000     1.127    0.592    0.298    0.003    0.003    0.002    0.002   0.005    0.002
40000     4.515    2.365    1.190    0.007    0.006    0.004    0.004   0.010    0.004
80000    18.003    9.453    4.725    0.015    0.012    0.008    0.008   0.022    0.008
```

- Use containers to do something, like a checkout simulation using queues, or a concordance using tree maps
- Write a simple game that uses interesting data structures, like Boggle

5.2 Pedagogy: Execution by Hand

This is an essential skill that students need to be taught.

Example

```
01 public int fib( int n ) {
02    if ( n <= 1 ) return 1;
03    return fib(n-1)
04           + fib(n-2);
05 }
```

Compute `fib(4)`

Exercise

Compute `bSearch(new int[] {1, 3, 6, 7, 12, 19, 23, 25}, 5)`

5.3 Pedagogy: Visualization

Visualization can sometimes help students understand how an algorithm works.

Visualization can also demonstrate how algorithms to accomplish the same task differ.

Investigate

- Sorting Algorithm Animations: http://www.sorting-algorithms.com
- Data Structure Visualizations: http://www.cs.usfca.edu/~galles/visualization
- Sort Timing and Visualization: http://math.hws.edu/TMCM/java/xSortLab/

5.4 Pedagogy: Active Learning

Labs

- Labs are a valuable way to teach programming and to give students a supportive environment for code development

POGIL

- POGIL stands for Process Oriented Guided Inquiry Learning
- POGIL is very popular and widely used in Chemistry
- My in-class activities are based on this approach

Process

1. Students prepare before class (read)
2. Students take a graded readiness quiz individually
3. Students take a graded IFAT-form quiz as a group
4. Students work through the activity in class in groups
5. Often, students do homework individually

Examples: Attached

6. Where to Go From Here

Books

- Most college data structure and algorithms textbooks are perfectly fine
- They mainly differ in the languages used and the level of presentation
- An advanced book using Java: Robert Sedgewick and Kevin Wayne, *Algorithms, 4th Edition*, Addison-Wesley, 2011
- A less advanced book using Ruby (and it's free): Christopher Fox. *Concise Notes on Data Structures and Algorithms: Ruby Edition*, Ventus Publishing ApS, 2012 (http://bookboon.com/en/concise-notes-on-data-structures-and-algorithms-ebook)

The Web

- MOOCS (Coursera)
- Various (Stanford CS Education Library, etc.)
- Free video lectures (YouTube, freevideolectures.com).