Annotated Syllabus for Data Structures Using Ada95

Mark Allen Weiss
School of Computer Science
University Park
Florida International University
Miami, FL 33199

PREREQUISITES: Some Discrete Math, knowledge of Ada (including exceptions and generic packages), and the equivalent of two programming courses.

TEXT: M. A. Weiss, *Data Structures and Algorithm Analysis in Ada*, Benjamin/Cummings, 1993. Additional handouts concerning Ada95 will be provided during the semester.

SOURCE CODE: Ada95 code is available online. The handout *Summary of Ada95 Code* describes differences from Ada83.

COURSE OUTLINE: 15 weeks, 2 meetings of 75 minutes per week are assumed. Twenty-five lectures are listed, thus leaving time for reviews, a midterm, quizzes, and discussions that describe programming assignments. Additional topics can be added if time permits.

1. **Discussion of Big-Oh: Part I**

   Explain the basic idea that more data means more running time, and then show the different types of running times (Figures 2.3 and 2.4). Remark that some curves are better than others. At this point, there are three questions:

   (a) How do you know which curve your algorithm is on?

   (b) How do you design algorithms to be on the best curve?

   (c) How can you predict the amount of time a program will take?

   Provide a few quick examples and mention the basic rules, as in the text. Do the $O(n^3)$ and then the $O(n^2)$ maximum subsequence sum problem algorithms. Also, begin filling in some of the entries in Figure 2.2. You can take the $n = 10$ and $n = 100$ entries as givens and then estimate the others based on the asymptotic running times.

   Reading: Sections 2.0 to 2.4.2.

2. **Discussion of Big-Oh: Part II**

   Continue with the $O(n \log n)$ maximum subsequence sum problem algorithm. This is a good place to review recursion and Ada array slicing. You should finish by doing the linear-time algorithm, and perhaps fill in a proof of correctness.

   Reading: Sections 2.4.3 and 2.4.4, and Section 1.3.

3. **Discussion of Big-Oh: Part III**

   You will need to begin by reviewing logarithms. This is a difficult concept for many students. Mention the general rule that if we reduce the size of a problem by a constant factor in constant time, then the running time will be logarithmic. Give the binary search as an example; you can do it either recursively or nonrecursively. You can mention that the binary search can be used to implement a data structure that supports access in $O(\log n)$ time, and both insertions and deletions in $O(n)$ time, and that other mechanisms for reducing these bounds will be seen later in the course. Exponentiation is another example that I think is easy for students to follow. If you have time, do Euclid's algorithm.

   Reading: Sections 2.4.4 to 2.4.6, and Section 1.2.2.

4. **Ada95: A Quick Look at Inheritance and Other Changes**

5. **Ada95: A Quick Look at Inheritance and Other Changes**

   This is two lectures. Describe the trivial changes in Ada95 and then discuss the basics of inheritance. Explain tagged types and show how new types are derived. Discuss the `Shape` class in the handout.

   Reading: The handouts *Using GNAT on Unix* and *Using Inheritance for Heterogeneity.*

6. **Stacks and Queues: Applications**

   Describe what the stack and queue operations are, but do not get into the implementations. For the stack, describe the symbol balancing algorithm and the infix to postfix algorithm. You may also want to mention in passing how stacks are used to keep track of function and procedure calls. For the queue, mention some applications. *Note*: the description in the text is an abstract description; in practice, the top of the stack stores the current environment, and not the most recent environment, so when function F calls G, an activation record for G is pushed onto the stack, and it is popped when G returns.

   Reading: Sections 3.3.1, 3.3.3, 3.4.1, and 3.4.3.

7. **Stacks and Queues: Implementations**

   First describe the array implementation of the stack. Then describe a simple linked list implementation. Note that this requires using the finalization package, so here is the first instance of an Ada95 change. You will want to spend some time to describe finalization, but it seems to be a simple concept for many students. Finally you can describe, but not implement the array and linked list versions of the queue. An alternative scenario is to do the linked list implementation of the stack last, so that it leads into the next lecture more smoothly.

Reading: Sections 3.3.2 and 3.4.2, and the handout *Using Finalization in Ada95*.

8. **Linked Lists**

   Since the stack lecture has already described a simple example of linked lists, this lecture is simplified somewhat and boils down to describing header nodes, searching, and deletion.

   Reading: Sections 3.0 to 3.6. Refer to *Summary of Ada95 Code*.

9. **Trees, Binary Search Trees, and Traversals**

10. **Trees, Binary Search Trees, and Traversals**

    This is a two-part lecture. Give the basic definition of a tree, and show the file system application. Then show a binary tree. I like to define trees recursively. Describe the basic algorithms for a binary search tree, including deletion, and give some code. Note that the Ada95 versions are more complex because each recursive routine is embedded in a public routine. Show the inorder traversal for printing the tree in sorted order. Show the postorder traversal for computing the height of the tree. Mention also the preorder and level order traversals.

    Reading: Sections 4.0, 4.2, 4.3, and 4.6.

11. **AVL Trees**

    Discuss the basics of AVL trees, showing single and double rotations.

    Reading: Section 4.4.

12. **B-Trees**

    Discuss the B-tree. You can either use the 2-3 tree example from the text or do a 5-tree. If you opt for the 5-tree, be sure to tell your students to rotate their papers (so it is 8.5 inches high and 11 inches wide) and have them start drawing from the bottom.

    Reading: Section 4.7.

13. **Hash Tables: Part I**

    In the first lecture, describe the limitations of the binary search tree, namely the problem with sorted input and the practical annoyance of balancing the tree. You can show that constant time per operation is doable by giving as an example the bit array. Then discuss the basic idea of hashing and examine some hash functions. Review the MOD operator, and emphasize that `N MOD 11` is `N` if `N` is nonnegative and less than 11 (many students think it is 0). Next discuss separate chaining and linear probing. For linear probing, explain the optimistic analysis of the insertion cost, and why primary clustering makes linear probing poor for high load factors.

Reading: Sections 5.0 to 5.4.1.

14. **Hash Tables: Part II**

    The second lecture describes quadratic probing and explains how the implementation avoids multiplications. It also describes rehashing. At this point, you should discuss the array doubling strategy in Ada95.

    Reading: Sections 5.4.2 and 5.5. Note that the quadratic probing code in the original text was somewhat buggy.

15. **Priority Queues: Part I**

    Begin by explaining what a priority queue is, and the operating system scheduler application. Explain that the structure we are looking for should have properties that are better than a binary search tree but worse than a queue. Describe the binary heap structure and then the heap order property as in the text. Then show how `Insert` and `Delete_Min` are implemented.

    Reading: Sections 6.0 to 6.3.3.

16. **Priority Queues: Part II, and Heapsort**

    Show the linear-time heap construction algorithm and then describe heapsort.

    Reading: Sections 6.3.4 and 7.5.

17. **Sorting: Insertion Sort, Shellsort**

    Describe insertion sort briefly, and give a quick analysis. Go through the lower-bound proof for adjacent-exchange sorting algorithms, and then describe Shellsort.

    Reading: Sections 7.0 to 7.4.

18. **Sorting: Mergesort, Quicksort**

    First discuss the basic merging algorithm, and then discuss the recursive mergesort. An analysis can be performed to verify the $O(n \log n)$ behavior (if it was not explicitly done for the recursive maximum subsequence sum problem). Then begin the discussion of quicksort.

    Reading: Sections 7.6 and 7.7.

19. **Sorting: Quicksort, Lower Bounds**

Continue quicksort, describing the median-of-three selection, the partitioning algorithm, and the cutoff for small files. Give an implementation, and if you have time and mathematically sophisticated students, prove the $O(n \log n)$ average case bound. Then quickly do the lower bound for sorting, emphasizing that it only tells you how many comparisons are *necessary* for any algorithm.

Reading: Sections 7.7 (continued) and 7.9.

20. **Graph Algorithms: Part I - Overview**

Give the basic definitions and show the representation of a graph. This includes the adjacency matrix versus the adjacency list and the internal mapping of vertex names to numbers. List the problems that will be covered in the forthcoming lectures as well as some other applications.

Reading: Sections 9.0 and 9.1.

21. **Graph Algorithm: Part II - Topological Sort**

Give the basic algorithm and then show how a queue makes it linear time. Include a sketch that shows that cycle detection works. Do several examples.

Reading: Section 9.2.

22. **Graph Algorithms: Part III - Breadth-First Search**

Again, give the basic algorithm and then show how a queue makes it linear time. Then show how actual shortest paths (rather than the length of the shortest path) are maintained and printed out. Do several examples.

Reading: Sections 9.3 and 9.3.1.

23. **Graph Algorithms: Part IV - Dijkstra's Algorithm**

Give the basic algorithm and then describe how the priority queue improves the worst-case running time for sparse graphs. Do several examples. Mention the problems with negative edges and negative-cost cycles.

Reading: Sections 9.3.2 and 9.3.3.

24. **Graph Algorithms: Part V - Kruskal's Algorithm and Union/Find**

Kruskal's algorithm is simple to describe; the basic problem becomes one of detecting if two vertices in the partially constructed spanning forest are connected. Show that this is a problem of maintaining equivalence classes and then describe the union/find algorithm. Be careful to explain that the trees that form in the union/find algorithm are not part of the spanning tree.

Reading: Sections 9.5.0, 9.5.2, and Sections 8.0 to 8.5.

25. **Graph Algorithms: Part VI - Depth First Search and Strong Components**

    You can mention that testing whether a graph is strongly connected can be done in linear time by running a breadth-first search from any vertex v, then reversing the edges in the graph and running a second breadth-first search from v. If both searches visit every vertex, the graph is strongly connected, otherwise it is not. If the graph is not strongly connected, this algorithm does not find the strong components, but it does provide the intuition for the algorithm in the text. Describe depth-first search and then give the strong components algorithm.

    Reading: Sections 9.6.0, 9.6.4, and 9.6.5.