COURSE DESCRIPTION

Dept., Number  | CSC 130  | Course Title        | Data Structures and Algorithm Analysis
Semester hours | 3        | Course Coordinator  | Jinsong Ouyang
URL (if any):   |          |                      | http://gaia.ecs.csus.edu/~ouyangj//

Catalog Description

Specification, implementation, and manipulation of complex data structures: multi-lists, trees, sets, and graphs. Design and analysis of algorithms. Recursion and stack-based memory management. Advanced searching and sorting. NP completeness. Prerequisites: At least a C-grade in CSC 20 and CSC 28; however, CSC 28 may be taken concurrently.

Textbook


Course Goals

1. Thorough coverage of standard data structures.
2. Design techniques for advanced data structures.
3. Analysis of algorithms.
4. Fundamental algorithm design techniques.

Prerequisites by Topic

Thorough understanding of:
- Design and implementation of medium size programs using multi-level decomposition, data abstraction, and procedural abstraction.
- Programming style and program documentation concepts.
- Program development process, including the distinction between compiling, linking, and executing programs.
- Use of a symbolic debugger, testing and debugging techniques.
- Records/structures, classes, and objects, not including inheritance and polymorphism.
- One dimensional arrays of structured types.
- Sub-programs and their appropriate use, including understanding of forward declarations of subprograms and why they are needed.
- Parameter passing and its implications.
- Scope rules.
- Selected sorting and searching algorithms such as: insertion sort, and binary search.

Basic understanding of:
- Lifetime of variables.
• Dynamic memory allocation and use of pointers.
• Recursion as a problem solving technique, and its implementation in programming languages.
• Multi dimensional arrays.
• Representation of data structures, including simulated pointers (cursor representation).
• Evaluation of various alternatives to select appropriate data structures.
• Strings, their various representations, and operations.
• Stack, queue, linear list ADTs, their representation (contiguous/sequential, linked) and associated algorithms.
• Tools and techniques to allow generic data types, e.g. templates.

Exposure to:
  • Big-O notation and ability to analyze simple algorithms.
  • Exception handling.

Major Topics Covered in the Course

1. Design, representation and implementation of data structures (4 hours).
2. Algorithm analysis and design: Big-O notation; time requirement; space requirement; counting techniques (4 hours).
3. Linear lists and multilists: application of stacks and queues, priority queues (3 hours).
4. Trees: binary and n-ary trees; traversals; threaded trees; heaps; binary search trees, AVL trees, B-trees, and general search trees (8 hours).
5. Sets and their representation: bit map, hash table, union-find (3 hours).
6. Graphs: traversals; spanning trees; shortest paths (6 hours).
7. Recursion and stack-based memory management (3 hours).
8. Advanced sorting: heapsort, treesort, radix sort; comparison with other sort algorithms (5 hours).
9. Problem space searching: BFS and DFS; heuristic search, local search (e.g., hill-climbing), game-playing methods (minimax, alpha-beta pruning), constraint satisfaction (backtracking and heuristic repair) (5 hours).
10. NP-completeness (1 hour).
11. Tests (3 hours).

Outcomes

Thorough understanding of:
  • Representation of data structures (contiguous/sequential, linked).
  • Data abstraction and procedural abstraction.
  • Tools and techniques to allow generic data types, e.g. templates.
  • Dynamic memory allocation and use of pointers.
  • Recursion as a problem solving technique, and its implementation in programming languages.
• Application of stack, queue, linear list ADTs.
• Representation of trees, including cursor representation.
• Standard algorithms to manipulate trees, including binary search trees, AVL trees, heaps and Btrees.
• Representation of graphs.
• Standard algorithms to manipulate graphs, including breadth and depth first search, spanning trees, shortest path.
• Representation of sets: union-find, hashing.
• Evaluation of various alternatives to select appropriate data structures.

Basic understanding of:
• Exception handling.
• Concepts of analysis of algorithms: asymptotic behavior, best, average, worst performance.
• Analysis of non-recursive algorithms.
• Advanced searching and sorting algorithms and their performance.

Exposure to:
• Program solving strategies, e.g., greedy method, divide and conquer, backtracking.
• Analysis of recursive algorithms.
• Complexity classes, including introduction to NP-Completeness.

Laboratory Projects

Several projects of varying complexity selected to demonstrate applications of data structures. Examples include:

1. Multilists.
2. Recursion using back-tracking.
3. Trees.
5. Advanced sorting and searching.

Estimated Curriculum Category Content (Semester hours)

<table>
<thead>
<tr>
<th>Area</th>
<th>Core</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Software Design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comp. Arch.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Structures</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prog. Languages</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Oral and Written Communications**

No significant component.

**Social and Ethical Issues**

No significant component.

**Theoretical Content**

See “Major Topics” above.

**Problem Analysis**

Students are given problems to solve with some guidance on the approach and the data structures to be used. However, it is their responsibility to analyze the given problem.

**Solution Design**

Design is part of each laboratory assignment. Students are expected turn in the design for their program. Design issues are emphasized throughout the course.