CPT108 Data Structures and Algorithms

Lecture 24

Review

Course Review

Three major components

- Problem solving and problem analysis
- Data structures
- Algorithms

Problem solving and Problem analysis

Problem analysis

- Problem and data abstraction
- Inputs and outputs specification
- Constraints and assumptions

Program design

- Program decomposition
  - Divide problem into smaller *subproblem* that can be solved separately at different time, by different person
- Data modelling and design
- Choice of algorithms
  - Complexity and the asymptotic notation (Big-Oh (O), Big-Omega (Ω), and Big-Theta (Θ))
- Algorithm tracing and verification

Problem solving and Problem analysis (cont.)

Problem solving techniques

- Data abstraction and modelling
  - Data-directed design i.e., objects and abstract data type (ADT)
  - Information hiding i.e., abstraction for external use hiding internal detail, levels of abstraction, etc.
- Divide-and-conquer
  - Divide the problem into smaller, more manageable subproblems that looks similar to the initial problem
  - Then solve these subproblems and put their solutions together to solve the original problem
- Recursion
  - Stopping case
  - Recursion step
  - Eventuality

Problem solving and Problem analysis (cont.)

Abstract data type (ADT) (Classes)

- Specification
  - Data members and methods (functions)
  - `public`, `private`, `protected`
- Object lifetime and scope
- Object creation (constructor) and destruction (destructor)
- The `this` pointer

Problem solving and Problem analysis (cont.)

Basics of analysis

- What is Big-Oh (O), Big-Omega (Ω), and Big-Theta (Θ)?
- How to determine the relative relationship of two functions?
  - Refer to increasing rate of some standard functions, such as: n, $n \lg n$, n^2, 2^n, etc.

Practice on some examples from lecture notes, assignment, and problems from the optional textbook!

- How to analyze basic code segment, e.g., loop?
- How to analyze the complexity of recursive functions?
  - e.g., use of recurrence equations

Data structures

Data type

- Simple data types
- Arrays
- Classes

Abstract data types (ADTs) (Classes)

- Sets
- Heaps
- Hashtables
- Lists (or Collections)
  - Linked lists (singly linked lists, doubly linked lists, and circular linked lists)
  - Stacks
  - Queues
- Trees, such as, but not limited to, binary tree
- Graphs

Data structures (cont.)

- Two fundamental ones: Arrays and linked lists
- Three basic operations: search, insertion, and deletion
- Arrays
    - What are the steps to resize an array?

Data structures (cont.)

Trees

- Basic definitions and concepts
  - Path, length of a path, height and depth of a node
- Tree traversal
  - Preorder
  - Postorder
  - Inorder
- Searching, insertion and deletion in binary search tree (BST)
  - How do they work?
  - What is the time complexity?

Data structures (cont.)

Heaps

- A complete binary tree implemented with an array
- How to build a max- (or min-) heap
- Properties and application (e.g., priority queues)
- Searching, insertion and deletion in heap
  - How do they work?
  - What is the time complexity?

Data structures (cont.)

Hashtables

- Hash codes
- Hash functions
    - Goal: to generate hash codes that are evenly distributed
        - What approaches can we use for different data types?
        - Use of prime number in hash function
- Collision handling
    - Separate chaining
    - Open addressing
        - Linear probing – problem: Primary clustering
        - Quadratic probing – problem: Secondary clustering
        - Double hashing
- How to expand a hashtable?
    - How does it different from resizing an array?

Algorithms

- Searching
  - Linear search
  - Binary search
  - Binary search tree (BST) (Binary tree search)
- Sorting
  - Comparison-based approach
    - Insertion sort, selection sort, bubble sort
    - Merge sort, quick sort (“divide-and-conquer”-based)
    - Heap sort (“tree”-based)
  - Non-comparison-based approach
    - Counting sort
    - Radix sort
- Graph traversal

Algorithms (cont.)

Sorting algorithms

- How does the algorithm work?
  - First, you should have general idea on how the algorithm works
  - Then, read the pseudocode, which possibly including some implementation details.
  - Try to practice the algorithm execution on some examples
- What is the time complexity?
  - You may need to identify the worst-case and best-case sometime.
  - Also, you should be careful about the complexity of recursive functions (be familiar with the techniques for solving such problems)

Complexities of Different Sorting algorithms

	Worst	Best
Selection sort	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n^2)$	$O(n)$
Bubble sort	$O(n^2)$	$O(n^2)$
Improved bubble sort	$O(n^2)$	$O(n)$
Merge sort	$O(n \lg n)$	$O(n \lg n)$
Quick sort	$O(n^2)$	$O(n \lg n)$
Heap sort	$O(n \lg n)$	$O(n \lg n)$
Counting sort	$O(n+k)$	$O(n+k)$
Radix sort	$O(d(n+k))$	$O(d(n+k))$

Algorithms (cont.)

Graph

- General graph concepts
  - Edge, vertex, degree, path, cycle
  - Subgraph, connected component
  - Graph representation:
    - Adjacency matrix
    - Adjacency list

  Pay attention to their features!
- Path searching
  - Breadth first search (BFS)
  - Depth first search (DFS)

  How does the algorithm work?
  What is the time complexity with different graph representation?

Differences between BFS and DFS

	BFS	DFS
Definition	Traversal begins at the *root* node and walk through all nodes on the same level before moving on to the next level	Traversal begins at the *root* node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes
Conceptual Difference	Builds the tree *level by level*	Builds the tree *subtree by subtree*
Data structure	Queue (FIFO)	Stack (LIFO)
Suitable for	Searching vertices *closer* to the given source	Finding paths (or solutions) that are *away* from source
Applications	Finding Shortest path, bipartite graphs, GPS navigation, etc.	Cycles or loops detection, finding strongly connected components (SCC), etc.
Path generation	Traversals according to the tree level	Traversals according the tree depth
Backtracking	Not required	Required to follow a backtrack
Memory	More memory	Less memory
Loops	*Cannot* be trapped into *finite* loops	*Can* be trapped into *infinite* loops

Differences between BFS and DFS

	Adjacency list		Adjacency matrix	
	Time complexity	Auxiliary space	Time complexity	Auxiliary space
BFS	$O(\|V\| + \|E\|)$	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\|^2)$
DFS	$O(\|V\| + \|E\|)$	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\|^2)$

Suggested review methods

- Read through all lecture notes carefully. Try to fully understand all concepts, definitions, algorithms, and analysiss

  Don’t blindly remember the details without understanding it!

- Practice the algorithm execution and complexity analysis by yourself once
- Review homework and assignment
- Work on the problems available at the optional textbook and reference books!

Note: Please do not recite the lecture notes!

You should be able to answer the questions based on your understanding!

All the best

and good luck in the examination! ☺