Course Review

Cpt S 223

Fall 2012

Final Exam

- When: Monday (December 10) 8–10 AM
- Where: in class (Sloan 150)

- Closed book, closed notes
- Comprehensive

- Material for preparation:
  - Lecture slides & class notes
  - Homeworks & program assignments
  - Weiss book

Syllabus for the final exam

- Math basics (proofs, recursions, etc.)
- Asymptotic notation
- Trees
- Search trees (BST, AVL, B-trees)
- Priority queues (binary heap, binomial heap)
- Hash tables
- Union-find data structure
- Sorting

Course Overview

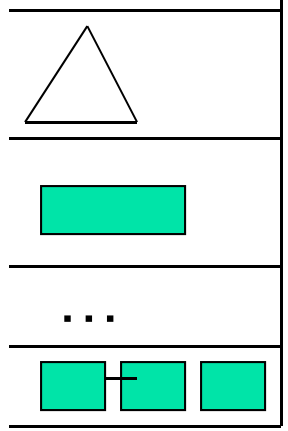
- What are abstract data types?
- How to design data structures for different purposes?
- How to use data structures?
- How to measure how good they are?
- How to pick the “right” data structure(s) while designing algorithms?

Data structure topics covered

Simple linear structures	Linked list, array
FIFO, LIFO	Queue, stack
Searching (ordered)	BST, AVL tree, B+ tree (for disks)
Searching (unordered)	Hash tables
Priority queues	Binary heap, binomial heap
Disjoint set operations	Union-find
Sorting algorithms	Insertion, quick sort, merge sort, heap sort, counting sort, radix sort, bucket sort, etc.
Programming	C++ with STL

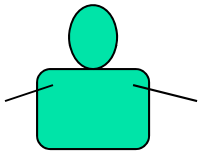
Developmental Cycle

Data Structures
(garage tools)



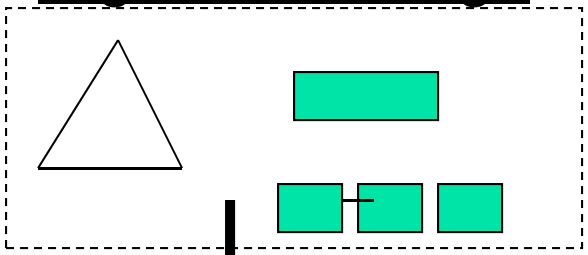
(ideas)

User



Problem
specification

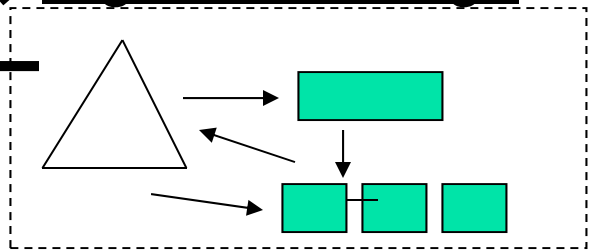
High-level design



input

refine

Algorithm design



- optimize costs
- maximize performance
- space-time tradeoffs
- design simplicity

Experimentation

- Simulations
- real data
- benchmarking

(methods)

Analysis

(testing)

Course Review …

- Program design
  - Tradeoffs: space, time efficiency, design simplicity
  - Runtime measurement, plotting and explanation
  - C++ STL code:
    - vector, list, queue, stack
    - set, map, multiset, multimap
    - priority_queue
    - hash_set, hash_map
  - C++ code:
    - Binomial heap
    - Union-find

Objectives

1. Introduce new & advanced data structures
2. Introduce algorithmic design and analysis
3. Solve problems using different data structures and design techniques, and compare their performance and tradeoffs
4. Implement algorithms and data structures in C++

Math Review

- Series: Definitions, manipulations, arithmetic and geometric series closed form
- Proofs: Know definition, components, and how to use the following
  - Proof by induction
  - Proof by counterexample
  - Proof by contradiction
- Recursion
  - Know definition and rules
  - Analyze running time of recursive algorithm
  - Tail recursion

Algorithmic Analysis

- Why analyze an algorithm?
- Line-by-line analysis
- Input:
  - Best-case, worst-case and average-case analysis
- Problem:
  - Upper bound, lower bound
- Rate of growth for algorithms:
  - Definitions and notation (O, Ω, Θ, o, w)

Abstract Data Types

- Lists
  - Operations: Insert, Delete, Search
  - Implementations: vectors, singly-linked lists, double-linked lists, sentinels
  - Analysis of operations for each implementation
- Stacks (LIFO)
  - Operations: Push, Pop, Top
  - Implementations: linked-list, vector
  - Analysis of operations for each implementation
- Queues (FIFO)
  - Operations: Enqueue, dequeue
  - Implementations: linked-list, vector
  - Analysis of operations for each implementation
- Standard Template Library (STL)
  - Use of vector, list, stack and queue template classes
  - Use of iterators
- Know all the tradeoffs (in time & space) between all these data structures

Trees (in memory)

- Definitions: root, leaf, child, parent, ancestor, descendant, path, height, depth
- Binary tree: Definition, traversals
- Storing/representation:
  - All children: use array or list
  - Store pointers to only Leftmost child and right sibling
  - Other representations possible
- Tree traversals
  - Inorder, postorder and preorder

Search Trees

- Binary search tree (BST)
  - Definition
  - Operations: Insert, Delete, Search, FindMin, FindMax, traversals
  - Know how to perform these on a BST and show resulting BST
  - Know worst-case and average-case analysis of performance
- Balanced BST (AVL trees)
  - Definition
  - Operations: Rotations & Cases, Insert, Lazy Delete, Search, FindMin, FindMax
  - Know how to perform these on an AVL tree and show resulting AVL tree
  - Know worst-case performance
- STL set and map classes
  - Differences
  - How to use them

Disk-based Search Trees

- B+ trees
  - Definition and properties
  - Input parameters: B, D, K
  - M and L, and how to choose them
  - Operations: Insert, Delete, Search
  - Know how to perform these on a B+ tree and show resulting B+ tree
  - Know worst-case performance
  - Know how to calculate height of a B+ tree

Priority Queues

- Binary heap, Binomial heaps
- Need to know:
  - <u>Binary heap</u>
    - Structure property:
      - complete binary tree except the last level (filled breadth-first left to right)
    - Heap order property
      - Min heap : each node's value is less than or equal to its children
  - <u>Binomial heap</u>
    - Structure property:
      - A forest of binomial trees (similar to binary representation)
    - Heap order property:
      - Min heap: within each binomial tree, same heap order like in binary heap

Implementation: Need to know…

- Binary heap
  - How to convert tree representation to array and vice versa
  - When to use percolateUp and percolateDown
  - Computation complexities of all basic heap operations on binary heap
    - Insert, buildheap, deleteMin

- Binomial heap
  - Array of pointers to each binomial tree
    - log n binomial tree pointers
  - Know relationships between binomial *heap*, binomial *trees* and their properties
  - Computation complexities of all basic heap operations on binary heap
    - Insert, deleteMin, merge

Run-times for each heap operation

	Insert	DeleteMin	Merge
Binary heap	O(lg n) worst-case, O(1) amortized using buildheap	O(lg n)	O(n)
Binomial Heap	O(lg n) worst-case O(1) amortized (or equivalently, O(m) for m successive calls to insert)	O(lg n)	O(lg n)

Other heap operations

- deleteMax
- decreaseKey
- increaseKey
- remove

Need to know all the complexities and be ready to write pseudocodes for the above

Also be prepared to think of writing basic application code that use heap functionalities

Union-Find data structure

- Union-find
  - Supports two operations on disjoint sets:
    - Union(a,b)
    - Find(a)
- One application is equivalence class computation:
  - Maximal subsets defined by equivalence relation
  - Disjoint subsets
- Array implementation

Example application: Equivalence class computation

Steps in Union(x,y):

1. $EqClass_x$ = Find (x)
2. $EqClass_y$ = Find (y)
3. $EqClass_{xy}$ = UnionSets ($EqClass_x$, $EqClass_y$)

Equivalence class algorithm:

- Initially, put each element in a set of its own
- FOR EACH element pair (a,b):
  - Check [a R b = true]
  - IF a R b THEN
    - Union(a,b)

Variations in Union

- *Simple Union* (aka Arbitrary Union)
  - Perform arbitrary linking of roots
- *Smart Unions*
  1. *Union-by-Rank (aka. union-by-height)*
     - Connect shorter tree under taller tree
  2. *Union-by-Size*
     - Connect smaller size tree under larger size tree (size(tree) = #nodes in the tree)

<u>Remember:</u> union always calls find internally

Variations of Find

- *Simple Find*
  - Simply returns the root id without modifying the tree
- *Smart Find*
  - Uses *path compression*
    - Link all nodes along the path from x to the root directly to the root
  - Returns the root id, but the resulting tree could have been modified as a result of path compression

Heuristics & their Gains

	Worst-case run-time for m operations
Simple Union, Simple Find	O(m n)
Union-by-size, Simple Find	O(m log n)
Union-by-rank, Simple Find	O(m log n)
Arbitrary Union, Path compression Find	O(m log n)
Union-by-rank, Path compression Find	**O(m Inv.Ackermann(m,n)) = O(m log*n)**

Extremely slow Growing function

This is the default version of the union-find data structure

Hashing

- Hash functions
  - purpose: string to integer, integer to integer
- Choice of a “good” hash functions
  - Reduce chance of collision
  - Relatively smaller key value
  - Does not need huge hash table size
- Hash Tables use hash functions
- Hash table size should be a prime
- Load factor
  - Measure to tell how crowded a hash table is
- Know algorithms & analysis for the following
  - Collision resolution by chaining
  - Collision resolution by open-addressing
    - Linear probing, quadratic probing
    - Double hashing
  - Rehashing
- Think of applications where hash tables could work out better than other data structures.

Sorting

- Need to know:
  - Runtime complexity results:
    - $\Theta(n \log n)$ for comparison sorting (e.g., merge sort, heap sort, using balanced BST)
      - $O(n^2)$ methods also exist (e.g., insertion sort, quick sort (worst case))
    - $\Theta(n)$ for non-comparison based sorting (e.g., counting sort)
    - $O(nk)$ for radix sort, bucket sort (where k is the number of bits for each number)
  - Since we just scratched the surface on the algorithmic details of some sorting algorithms in class, I don't expect you to know all the internal details of sorting algorithms. But I do expect you to know the main runtime results of the above sorting methods, so that if you want to ***use sorting to solve a specific exam problem***, then you should be able to simply say that in your pseudocode and state how much time it will cost.

Other General Tips for the Final Exam

- Work out examples of basic operations for different data structures
- Show steps to convey your thought process
  - Think from the perspective of the grader when writing solutions
  - State any non-trivial assumption you make

- Write less and to the point (use figures where possible, to cut down on the text)
- Don’t leave questions unanswered
  - If unable to solve, write at least your approach idea(s)
  - I do give partial credits

Other General Tips for the Final Exam…

<u>On the nature of questions:</u>

- Many questions will be subjective, having you think about different options and compare, or giving you an application situation and having you design a solution for it (present ideas and argue why that will be good).
  - A good example of such a question will be the design for PA4.
  - Think of real world applications for the different data structures.
- Some questions will probe you for a better understanding of the properties that underlie the data strsuctures we discussed. For example (not limited to):
  - Shown an insertion sequence is it possible to arrive at a given BST?
  - Shown a union sequence is it possilbe to have arrived at a shown union-find forest structure?
  - #leaves, height related properties of B+ tree,
  - Etc.

Thank You & Good Luck !

COURSE EVALUATIONS !!