Data Structure

Lecture#26: Conclusion

U Kang
Seoul National University

In This Lecture

- Review what we have learned
- Common ideas
- What to remember from this course

Goals of this Course

1. Reinforce the concept that costs and benefits exist for every data structure.
2. Learn the commonly used data structures.
   - These form a programmer's basic data structure ``toolkit.''
3. Understand how to measure the cost of a data structure or program.
   - These techniques also allow you to judge the merits of new data structures that you or others might invent.

What We Have Learned (1)

- Algorithm analysis techniques
  - Big-Oh, best/worst/average case running time
- Why designing better algorithm is better than buying $$ computers
- List, stack, queue, dictionary
  - Array based vs link based
- Binary tree
  - Traversals, BST, Huffman tree
- Priority queue
  - Max heap

What We Have Learned (2)

- Non-binary trees
  - Parent pointer implementation, Union/find
- Sorting
  - Quicksort, mergesort, heapsort, binsort, radix sort, …
- Searching
  - (Quadratic) binary search, jump search, self-organizing list
- Hashing
  - Collision resolution
- Graph
  - DFS, BFS, topological sort, shortest path, MST

Common Ideas (1)

- Time/space tradeoff
  - To save space, use more CPU time
  - To save CPU time, use more space
    - Array list vs. linked list
    - Singly linked list vs. doubly linked list
    - Open hashing vs. closed hashing
- Caching
  - Put frequently accessed items near to you
    - List ordered by frequency, self-organizing list

Common Ideas (2)

- Knowing the data helps
  - Sparsity of data
    - Adjacency list vs. adjacency matrix in graph
  - Access frequency pattern of data
    - Lists ordered by frequency
  - Read only vs. Read/Write
    - Array vs. linked list
    - Sequential implementation vs. dynamic node implementation for general tree

Common Ideas (3)

- Asymptotic analysis: crucial tool for CS
  - Big-(Theta, Oh, Omega) complexity matters
  - Quickly grasps the solution
    - How long would it take to copy 100 GB of files from a hard disk to another hard disk, assuming hard disk I/O rate of 30 MB/sec?
- Randomness helps
  - Good average-case behavior
    - Hashing, Quicksort
- Recursion
  - Enables solving complicated problems easily
    - Quicksort, DFS

Common Ideas (4)

- Divide and conquer
  - Mergesort, quicksort
- Constraints on problems lead to efficient method
  - Stack and Queue
  - Maxheap: removemax() and insert() in O(log n)
- Tree – much smaller height than # of items
  - Allows fast search: O(n) => O(log n)
    - BST, MaxHeap

CS problems ⇔ offline problems

- CS problems, ideas ⇔ offline problems, ideas
  - Time-space tradeoff
  - Caching (e.g. note of frequent phone numbers on your desk)
  - Sorting (e.g. merge sort)
  - Divide and conquer
  - Tree – much smaller height than # of items
- Understanding how to solve CS problems helps solve offline problems
  - Furthermore, it often gives theoretical lower bound
- Good intuition of solving offline problems helps find a good solution of CS problems

What to Remember (1)

- For each task/algorithm,
  - What is the right data structure for it? What is the strength/weakness of the data structure?
  - What is the running time of the algorithm? What is the main idea of the algorithm?

What to Remember (2)

- You should be able to explain a concept in less than 1 minute
  - What is the problem that the algorithm/data structure tries to solve? What is the application?
  - What is the solution? What is the main idea?
  - What is the advantage of the solution? What is the limitation?
  - In plain terms so that people with no background can understand
- Then, you can summarize this course in less than 20 minutes; you become smarter.

Related Course

- Algorithm
- Database
- …

- And, math-related courses (e.g., linear algebra, statistics)

Thank You!