

IADS summary Fall 2005

November 23, 2005

Anna Pagh



IT University
of Copenhagen

Today's lecture

- Summary of the course
- Exam preparation
 - Problem 2 Exam E2004 (AVL trees)
 - Problem 2 Exam F2005 (Shortest paths)
- Questions
- Evaluation



Data structures

- Set data structures
- Disjoint set data structures
- Graphs



Set data structures

- List
- Stack
- Queue
- Dynamic set (dictionary)
- Priority queue



Dynamic set

- Insert, delete, lookup
- Implementations:
 - Sorted array
 - (Balanced) search tree
 - AVL trees
 - Hash table



Dynamic set

	Insert	Delete	Lookup
Sorted Array	$O(n)$	$O(n)$	$O(\log n)$
AVL tree	$O(\log n)$	$O(\log n)$	$O(\log n)$
Hash table	$O(1)$ exp.	$O(1)$ exp.	$O(1)$ exp.



Priority queue

- Min (or max) priority queue:

A value associated with each element.
Element with minimum (or max.) value can
be found fast.

	Insert	Decrease-Key	Extract-Min
Heap	$O(\log n)$	$O(\log n)$	$O(\log n)$
(Fibonacci heap)	$O(\log n)$	$O(1)$	$O(\log n)$



Disjoint set data structure

- Union-Find
- n elements, m operations
- $\alpha(n)$ is inverse Ackermann function

Linked List w. weighted union	$O(m+n \log n)$
Forest w. union by rank	$O(m \log n)$
Union by rank and path compression	$O(\alpha(n)m)$



Graphs

- Adjacency matrix
 - $O(V^2)$ space
 - $O(1)$ lookup
 - $O(V)$ to check adjacent vertices
- Adjacency list
 - $O(V+E)$ space
 - $O(\text{deg})$ lookup, $\text{deg}=\text{outgoing degree}$
 - $O(\text{deg})$ to check adjacent vertices



Techniques

- Divide-and-conquer
- Dynamic programming
- (Recursion)



Divide-and-conquer

- Divide into smaller problems, solve them, and combine the solved sub-problems into a solution.
- Solve the smaller problems in the same way.
- Merge sort



Dynamic programming

- Useful when a recursive algorithm computes the same values repeatedly.
- Never compute the same value twice.
- Solve (few) small problems, combine into larger. Remember values, to use several times.
- Used for optimization problems.



Analysis

- Running time and space
- Upper bounds - Big-Oh: $O(f(n))$
- Lower bounds - $\Omega(f(n))$
- Recurrences
- Amortized analysis
- Loop invariant proofs



Recurrences

- Substitution method
 - Guess solution and show by induction
- Recurrence tree method
 - Nodes in a tree represents costs
 - Sum the costs at each level and sum all levels
- Master method
 - $T(n) = aT(n/b) + f(n)$



Amortized analysis

- Used to analyze data structures when worst case analysis is too pessimistic.
- The average time in the worst case.
- Considers a sequence of operations performed on a data structure.
- Accounting method and aggregate method.
- Can improve worst case analysis of algorithms using a data structure.



Loop invariant proofs

- To show correctness of an algorithm
- An invariant is defined
 1. Initialization - Invariant true before start
 2. Maintenance - If it is true before the iteration it is also true afterwards
 3. Termination - If invariant is true after the last iteration, then algorithm is correct.

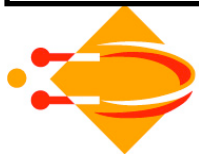


Computational problems

- Sorting
- Graph exploration
- Topological sorting of a graph
- Minimum spanning tree
- Shortest path

Sorting

Algorithm	Time	Constraints/ Comments
Insertion sort	$O(n^2)$	(general)
Merge sort	$O(n \log n)$	(general)
Heap sort	$O(n \log n)$	uses a heap d.s.
Quick sort	$O(n^2)$ worst case $O(n \log n)$ expected	fast in practice, randomized
Counting sort	$O(n)$	$O(n)$ different keys in the universe
Radix sort	$O((b/r)(n+2^r))$ for b - bit numbers, any $r < b$	$O(n)$ for constant length numbers
Lower bound	$\Omega(n \log n)$	Comparison model



Graph exploration

- Breadth first search: $O(V+E)$ time
 - Finds shortest path in unweighted graphs
- Depth first search: $O(V+E)$ time



Topological sorting

- For sorting the vertices in a DAG, such that all edges goes in the same direction.
- Use depth first search.
- Time $O(V+E)$.



Minimum spanning tree

- Cheapest way to connect all vertices in a graph.
- Kruskal: Sort edges and add one at the time unless it introduces a cycle.
 - Time: $O(E \log V)$.
- (Prim: $O(E + V \log V)$.)



Single source shortest path

Algorithm	Time	Graphs
Topological sorting	$O(E+V)$	DAGs
Bellman-Ford	$O(VE)$	General graphs
Dijkstra's	$O(V \log V + E)$	Positive edge weights



Question hour

- Monday January 9 at 13.00 (preliminary)
- Room: see course homepage

