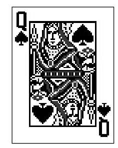
CSE 326: Data Structures Lecture #23

Randomized Data Structures

Henry Kautz

Winter Quarter 2002

Pick a Card



Warning! The Queen of Spades is a very unlucky card!

Randomized Data Structures

- We've seen many data structures with good average case performance on random inputs, but bad behavior on particular inputs
  - Binary Search Trees
- Instead of randomizing the input (since we cannot!), consider randomizing the data structure
  - No bad inputs, just unlucky random numbers
  - Expected case good behavior on any input

What's the Difference?

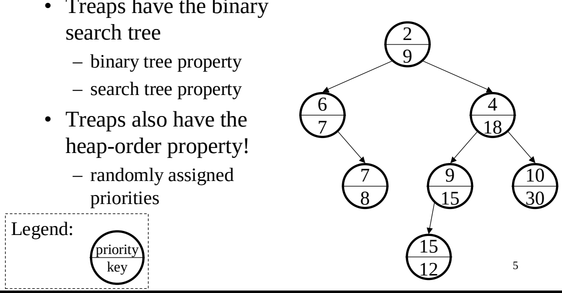
- Deterministic with good average time
  - If your application happens to always use the "bad" case, you are in big trouble!
- Randomized with good expected time
  - Once in a while you will have an expensive operation, but no inputs can make this happen all the time
- *Kind of like an insurance policy for your algorithm!*

Allstate. You're in good hands.

Treap Dictionary Data Structure

- Treaps have the binary search tree
  - binary tree property
  - search tree property
- Treaps also have the heap-order property!
  - randomly assigned priorities

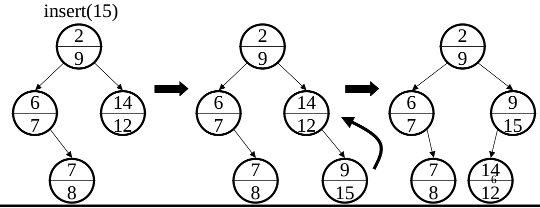
heap in yellow; search tree in blue



Treap Insert

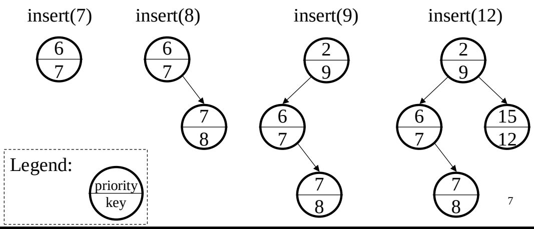
- Choose a random priority
- Insert as in normal BST
- Rotate up until heap order is restored (maintaining BST property while rotating)

insert(15)



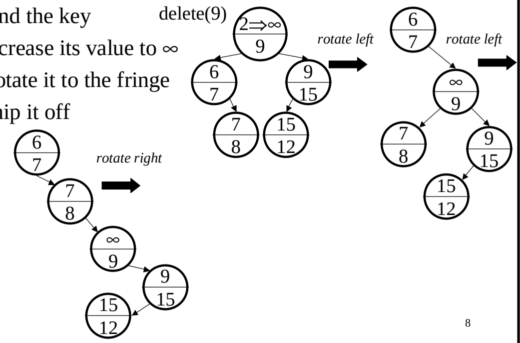
Tree + Heap… Why Bother?

Insert data in sorted order into a treap; what shape tree comes out?

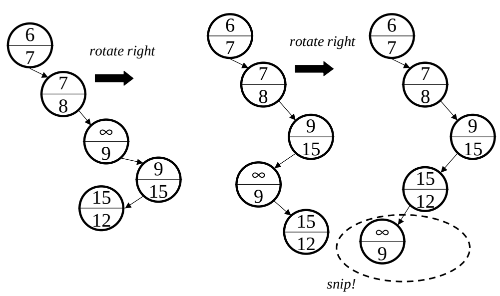


Treap Delete

- Find the key
- Increase its value to ∞
- Rotate it to the fringe
- Snip it off



Treap Delete, cont.



Treap Summary

- Implements Dictionary ADT
  - insert in expected O(log n) time
  - delete in expected O(log n) time
  - find in expected O(log n) time
  - but worst case O(n)
- Memory use
  - O(1) per node
  - about the cost of AVL trees
- Very simple to implement, little overhead – less than AVL trees

Other Randomized Data Structures & Algorithms

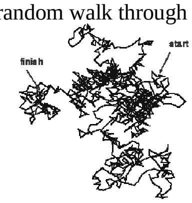
- Randomized skip list
  - cross between a linked list and a binary search tree
  - O(log n) expected time for finds, and then can simply follow links to do range queries
- Randomized QuickSort
  - just choose pivot position randomly
  - expected O(n log n) time for *any* input

Randomized Primality Testing

- No known polynomial time algorithm for primality testing
  - but does not appear to be NP-complete either – in between?
- Best known algorithm:
  1. Guess a random number 0 < A < N
  2. If (A^{N-1} % N) ≠ 1, then N is not prime
  3. Otherwise, 75% chance N is prime
     - or is a “Carmichael number” – a slightly more complex test rules out this case
  4. Repeat to increase confidence in the answer

Randomized Search Algorithms

- Finding a goal node in very, very large graphs using DFS, BFS, and even A* (using known heuristic functions) is often too slow
- Alternative: random walk through the graph



N-Queens Problem

- Place N queens on an N by N chessboard so that no two queens can attack each other
- Graph search formulation:
  - Each way of placing from 0 to N queens on the chessboard is a vertex
  - Edge between vertices that differ by adding or removing one queen
  - Start vertex: empty board
  - Goal vertex: any one with N non-attacking queens (there are many such goals)

Demo: N-Queens

DFS

(over vertices where no queens attack each other)

versus

Random walk

(biased to prefer moving to vertices with fewer attacks between queens)

Random Walk – Complexity?

- Random walk – also known as an “absorbing Markov chain”, “simulated annealing”, the “Metropolis algorithm” (Metropolis 1958)
- Can often prove that if you run long enough will reach a goal state – but may take exponential time
- In some cases can prove that with high probability a goal is reached in polynomial time
  - e.g., 2-SAT, Papadimitriou 1997
- Widely used for real-world problems where actual complexity is unknown – scheduling, optimization
  - N-Queens – probably polynomial, but no one has tried to prove formal bound

Traveling Salesman

Recall the **Traveling Salesperson (TSP) Problem**: Given a *fully connected, weighted* graph G = (V,E), is there a cycle that visits all vertices exactly once and has total cost ≤ K?

- NP-complete: reduction from Hamiltonian circuit

- Occurs in many real-world transportation and design problems
- Randomized simulated annealing algorithm demo

Final Review

(“We’ve covered way too much in this course…

What do I really need to know?”)

Be Sure to Bring

- 1 page of notes
- A hand calculator
- Several #2 pencils

19

Final Review: What you need to know

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \qquad \sum_{i=0}^{N} A^i = \frac{A^{N+1}-1}{A-1}$$

- Basic Math
  - Logs, exponents, summation of series
  - Proof by induction
- Asymptotic Analysis
  - Big-oh, Theta and Omega
  - Know the definitions and how to show f(N) is big-O/Theta/Omega of (g(N))
  - How to estimate Running Time of code fragments
    - *E.g.* nested "for" loops
- Recurrence Relations
  - Deriving recurrence relation for run time of a recursive function
  - Solving recurrence relations by expansion to get run time

20

Final Review: What you need to know

- Lists, Stacks, Queues
  - Brush up on ADT operations – Insert/Delete, Push/Pop *etc*.
  - Array versus pointer implementations of each data structure
  - Amortized complexity of stretchy arrays
- Trees
  - Definitions/Terminology: root, parent, child, height, depth *etc*.
  - Relationship between depth and size of tree
    - Depth can be between O(log N) and O(N) for N nodes

21

Final Review: What you need to know

- Binary Search Trees
  - How to do Find, Insert, Delete
    - Bad worst case performance – could take up to O(N) time
  - AVL trees
    - Balance factor is +1, 0, -1
    - Know single and double rotations to keep tree balanced
    - All operations are O(log N) worst case time
  - Splay trees – good amortized performance
    - A single operation may take O(N) time but in a sequence of operations, average time per operation is O(log N)
    - Every Find, Insert, Delete causes accessed node to be moved to the root
    - Know how to zig-zig, zig-zag, etc. to "bubble" node to top
  - B-trees: Know basic idea behind Insert/Delete

22

Final Review: What you need to know

- Priority Queues
  - Binary Heaps: Insert/DeleteMin, Percolate up/down
    - Array implementation
    - BuildHeap takes only O(N) time (used in heapsort)
  - Binomial Queues: Forest of binomial trees with heap order
    - Merge is fast – O(log N) time
    - Insert and DeleteMin based on Merge
- Hashing
  - Hash functions based on the mod function
  - Collision resolution strategies
    - Chaining, Linear and Quadratic probing, Double Hashing
  - Load factor of a hash table

23

Final Review: What you need to know

- Sorting Algorithms: Know run times and how they work
  - Elementary sorting algorithms and their run time
    - Selection sort
  - Heapsort – based on binary heaps (max-heaps)
    - BuildHeap and repeated DeleteMax's
  - Mergesort – recursive divide-and-conquer, uses extra array
  - Quicksort – recursive divide-and-conquer, Partition in-place
    - fastest in practice, but $O(N^2)$ worst case time
    - Pivot selection – median-of-three works best
  - Know which of these are stable and in-place
  - Lower bound on sorting, bucket sort, and radix sort

24

Final Review: What you need to know

- Disjoint Sets and Union-Find
  - Up-trees and their array-based implementation
  - Know how Union-by-size and Path compression work
  - No need to know run time analysis – just know the result:
    - Sequence of M operations with Union-by-size and P.C. is $\Theta(M\ \alpha(M,N))$ – just a little more than $\Theta(1)$ amortized time per op
- Graph Algorithms
  - Adjacency matrix versus adjacency list representation of graphs
  - Know how to Topological sort in $O(|V| + |E|)$ time using a queue
  - Breadth First Search (BFS) for unweighted shortest path

Final Review: What you need to know

- Graph Algorithms (cont.)
  - Dijkstra's shortest path algorithm
  - Depth First Search (DFS) and Iterated DFS
    - Use of memory compared to BFS
  - A* - relation of g(n) and h(n)
  - Minimum Spanning trees – Kruskal's algorithm
  - Connected components using DFS or union/find
- NP-completeness
  - Euler versus Hamiltonian circuits
  - Definition of P, NP, NP-complete
  - How one problem can be "reduced" to another (e.g. input to HC can be transformed into input for TSP)

Final Review: What you need to know

- Multidimensional Search Trees
  - k-d Trees – find and range queries
    - Depth logarithmic in number of nodes
  - Quad trees – find and range queries
    - Depth logarithmic in inverse of minimal distance between nodes
    - But higher branching fractor means shorter depth if points are well spread out (log base 4 instead of log base 2)
- Randomized Algorithms
  - expected time vs. average time vs. amortized time
  - Treaps, randomized Quicksort, primality testing