Data Structures - Cheat Sheet

Trees

Red-Black Tree

1. **Red Rule**: A red child must have a black father
2. **Black Rule**: All paths to external nodes pass through the same number of black nodes.
3. All the leaves are black, and the sky is grey.
Rotations are terminal cases. Only happen once per fixup.
If we have a series of *insert-delete* for which the insertion point is known, the amortized cost to each action is $O(n)$.
Height:$\log n \leq h \leq 2\log n$
Limit of rotations: 2 per insert.
Bound of ratios between two branches L, R: $S(R) \leq (S(L))^2$
Completely isomorphic to 2-4 Trees.

B-Tree

d defines the *minimum number of* keys on a node
Height: $h \approx \log_d n$
1. Every node has at most d children and at least $\frac{d}{2}$ children (root excluded).
2. The root has at least 2 children if it isn't a leaf.
3. A non-leaf node with k children contains $k-1$ keys.
4. On B+ trees, leaves appear at the same level.
5. Nodes at each level form linked lists
d is optimized for HDD/cache block size
Insert: Add to insertion point. If the node gets too large, *split.*$O(\log n) \leq O(log_d n)$
Split: The middle of the node (low median) moves up to be the edge of the father node. $O(d)$
Delete: If the key is not in a leaf, switch with succ/pred. Delete, and deal with short node v:
1. If v is the root, discard; terminate.
2. If v has a non-short sibling, steal from it; terminate.
3. Fuse v with its sibling, *repeat* with $p \leftarrow p[v]$.

Traversals

```
Traverse(t):
if t==null then return
→ print (t) //pre-order
Traverse(t.left)
→ (OR) print(t) //in-order
Traverse(t.right)
→ (OR) print(t) //post-order
```

Heaps

	Binary	**Binomial**	**Fibonacci**
`findMin`	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
`deleteMin`	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$
`insert`	$\Theta(\log n)$	$O(\log n)$	$\Theta(1)$
`decreaseKey`	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
`meld`	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$

Binary

Melding: If the heap is represented by an array, link the two arrays together and *Heapify-Up*. $O(n)$.

Binomial

Melding: Unify trees by rank like binary summation. $O(\log n)$

Fibonacci Heap

Maximum degree: $D(n) \leq \lfloor \log_\varphi n \rfloor$; $\varphi = \frac{(1+\sqrt{5})}{2}$
Minimum size of degree k: $s_k \geq F_{k+2}$
Marking: Every node which lost one child is marked.
Cascading Cut: Cut every marked node climbing upwards. *Keeps amortized* $O(\log n)$ *time for* deleteMin. *Otherwise* $O(\sqrt{n})$.
Proof of the φ^k node size bound:
1. All subtrees of junction j, sorted by order of insertion are of degree $D[s_i] \geq i-2$ (Proof: when x's largest subtree was added, since $D[x]$ was $i-1$, so was the subtree. Since then, it could lose only one child, so it is at least $i-2$)
2. $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$; $F_{k+2} \geq \varphi^k$
3. If x is a node and $k = \deg[x]$, $S_x \geq F_{k+2} \geq \varphi^k$. (Proof: Assume induction after the base cases and then $s_k = 2 + \sum_{i=2}^{k} S_{i-2} \geq 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i = F_{k+2}$)

Structures

Median Heap: one min-heap and one max-heap with $\forall x \in \min, y \in \max : x > y$ then the minimum is on the median-heap

Sorting

Comparables

Algorithm	Expected	Worst	Storage
QuickSort	$O(n\log n)$	$O(n^2)$	In-Place
	Partition recursively at each step.		
BubbleSort		$O(n^2)$	In-Place
SelectionSort		$O(n^2)$	In-Place
	Traverse n slots keeping score of the maximum. Swap it with $A[n]$. Repeat for $A[n-1]$.		
HeapSort		$O(n\log n)$	Aux
InsertionSort			Aux
MergeSort		$O(n\log n)$	Aux

Linear Time

BucketSort $\Theta(n)$:
If the range is known, make the appropriate number of buckets, then:
1. Scatter: Go over the original array, putting each object in its bucket.
2. Sort each non-empty bucket (recursively or otherwise)

3. Gather: Visit the buckets in order and put all elements back into the original array.

CountSort $\Theta(n)$:

1. Given an array A bounded in the discrete range C, initialize an array with that size.
2. Passing through A, increment every occurence of a number i in its proper slot in C.
3. Passing through C, add the number represented by i into A a total of $C[i]$ times.

RadixSort $\Theta(n)$:

1. Take the least significant digit.
2. Group the keys based on that digit, but otherwise keep the original order of keys. (This is what makes the LSD radix sort a stable sort).
3. Repeat the grouping process with each more significant digit.

Selection

QuickSelect	$O(n)$	$O(n^2)$
5-tuple Select		

Hashing

Universal Family: a family of mappings $H. \forall h \in H. h : U \to [m]$ is universal iff $\forall k_1 \neq k_2 \in U : Pr_{h \in H}[h(k_1) = h(k_2)] \leq \frac{1}{m}$

Example: If $U = [p] = \{0, 1, \ldots, p-1\}$then $H_{p,m} = \{h_{a,b} \mid 1 \leq a \leq p; 0 \leq b \leq p\}$ and every hash function is $h_{a,b}(k) = ((ak + b)\, mod\, (p))\, mod\, (m)$

Linear Probing: Search in incremental order through the table from $h(x)$ until a vacancy is found.

Open *Addressing*: Use $h_1(x)$ to hash and $h_2(x)$to permute. No pointers.

Open *Hashing*:

Perfect Hash: When one function clashes, try another. $O(\infty)$.

Load Factor α**:** The length of a possible collision chain. When $|U| = n$, $\alpha = \frac{\mathbf{m}}{\mathbf{n}}$.

Methods

Modular: Multipilicative, Additive, Tabular(byte)-additive

Performance

Chaining	$\mathbb{E}[X]$	**Worst Case**
Successful Search/Del	$\frac{1}{2}(1 + \alpha)$	n
Failed Search/Verified Insert	$1 + \alpha$	n

Probing

Linear: $h(k, i) = (h'(k) + i) \bmod m$

Quadratic:$h(k, i) = \left(h'(k) + c_1 i + c_2 i^2\right) \bmod m$

Double: $h(k, i) = (h_1(k) + ih_2(k))\, \mathbf{mod}\, m$

$\mathbb{E}[X]$	**Unsuccessful Search**	**Successful Search**
Uni. Probing	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$
Lin. Probing	$\frac{1}{2}\left(1 + \left(\frac{1}{1-\alpha}\right)^2\right)$	$\frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$

So Linear Probing is slightly worse but better for cache.

Collision Expectation: $\mathbb{P}[X \leq 2\mathbb{E}[X]] \geq \frac{1}{2}$

So:

1. if $m = n$ then $\mathbb{E}[|Col| < n] \geq \frac{n}{2}$
2. if $m = n^2$ then $\mathbb{E}[|Col| < 1] \geq \frac{1}{2}$ And with 2 there are no collisions.

Two-Level Hashing

The number of collisions per level: $\sum_{i=0}^{n-1} \binom{n_i}{2} = |Col|$

1. Choose $m = n$ and h such that $|Col| < n$.
2. Store the $n_i$elements hashed to i in a small table of size n_i^2 using a *perfect* hash function h_i.

Random algorithm for constructing a *perfect* two level hash table:

1. Choose a random h from $H(n)$ and compute the number of collisions. If there are more than n collisions, repeat.
2. For each cell i,if $ni > 1$, choose a random hash function from $H(ni2)$. If there are any collisions, repeat.

Expected construction time $- O(n)$

Worst Case search time - $O(1)$

Union-Find

MakeSet(x)	Union(x, y)	Find(x)
$O(1)$	$O(1)$	$O(\alpha(k))$

Union by Rank: The larger tree remains the master tree in every union.

Path Compression: every *find* operation first finds the master root, then repeats its walk to change the subroots.

Recursion

Master Theorem: for $T(n) = aT\left(\frac{n}{b}\right) + f(n);\ a \geq 1,\ b > 1,\ \varepsilon > 0$:

$$\begin{cases} T(n) = \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{\log_b (a) - \varepsilon}\right) \\ T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right) & f(n) = \Theta\left(n^{\log_b a} \log^k n\right); k \geq 0 \\ T(n) = \Theta(f(n)) & \begin{array}{l} f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right) \\ af\left(\frac{n}{b}\right) \geq cf(n) \end{array} \end{cases}$$

Building a recursion tree: build one tree for running times (at $T(\alpha n)$) and one for $f(n)$.

Orders of Growth

$f = O(g)$	$\limsup_{x \to \infty} \frac{f}{g} < \infty$	$f = o(g)$	$\frac{f}{g} \overset{x \to \infty}{\to} 0$
$f = \Theta(g)$	$\lim_{x \to \infty} \frac{f}{g} \in \mathbb{R}^+$		
$f = \Omega(g)$	$\liminf_{x \to \infty} \frac{f}{g} > 0$	$f = \omega(g)$	$\frac{f}{g} \overset{x \to \infty}{\to} \infty$

Amortized Analysis

Potential Method: Set Φ to examine a parameter on data stucture D_i where i indexes the state of the structure. If c_i is the actual cost of action i, then $\hat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1})$. The total potential amortized cost will then be $\sum_{i=1}^n \hat{c_i} = \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^n c_i + \Phi(D_i) - \Phi(D_0)$

Deterministic algorithm: Always predictable.

Stirling's Approximation: $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \Rightarrow \log n! \sim x \log x - x$

Scribbled by Omer Shapira, based on the course "Data Structures" at Tel Aviv University.

Website: http://www.omershapira.com