June 19, 2017

Contents

1

Data Structures

1.1 Data Types

Primative Types

Primatives: Wiki Link

These are the basic elements:

1. Bool
2. char
3. float
4. double
5. int
6. string
7. reference
8. enum

Composite Types

Composites: Wiki Link

Abstract Data Types

Abstract Types: Wiki Link

Nodes	A vertex connected to other vertexes by edges, in a tree nodes are connected by edges
Edge	Connections to other nodes
Subtrees	A connected group of children nodes connected to a parentthats not the root
Digraph	
Root	Top node of a tree
Child	A node directly connected to another, away from the root
Parent	Directly connected node towards the root. Can only have 1 parent as this causes cycles.
Sibling	Node with same parent
Decendant	Node accessible by parent to child
Ancestor	Node accessible by traversing child to parent
Leaf	(or external node) A node with no children (degree 1)
Branch	(or internal node) A node with children (degree >1)
Degree	The number of edges on a nodes
Path	Sequence of nodes and edges to reach another node
Level	1+connections to root of a node. (R)-( )-( )-(A) A has level4.
Node Height	Number of edges on the longest path between that node and a leaf.
Depth	Number of edges from the root to a given node
Forest	A set of disjointed trees
Branching Factor	Maximum number of children per node

Linear Data Structures

1.2 Abstract Data Types

Abstract data types (ADT) → its only a data structure if you're talking about the implimentation.

Trees

Tree: Wiki Link

A tree is a data structure made up of nodes connected by edges without having a cycle in it (a node cannot call itself in anyway). A linear list is a trivial tree.

Search Tree

Search Tree: Wiki Link

Tree data structure used for locating specific keys from within a set. Needs to be relatively balanced to be efficient.

Binary Trees

Binary Trees: Allisons Link

Binary Trees: Wiki Link

Hash Tree

Hash Array Tree: Wiki Link
Merkle Tree: Wiki Link

Trie

Trie: Wiki Link
Type of search tree. No node stores the key associated with its node, the position in the tree decides its key. All of the decendants of a node have the same prefix, the root is an empty string.
Commonly used for autocomplete and predictive text.
A trie can replace hash table:

- Worst case lookup is better
- No key collisions
- Buckets are only necessary if a key identifies multiple values
- Can provide alphabetical ordering

Drawbacks:

- Tends to be slower than a hash for lookups
- Floats can cause nasty long search chains
- Can require more memory as the keys are split up instead of contiguous

Heap

The Heap: Wiki Link
Tree data type, subtype of a priority queue. two types → min and max. In a min/max heap the root is the lowest/highest value in the tree.
The binary heap was introduced for the heap sort algorithm. The heap is partially ordered.

- Heap Property: with P → parent node, C → child node, then the key of P is ordered with respect to C. This applies for every child and parent.
- The root is the lowest or highest value
- Items always go in the next free slot. If it isnt in the right place compare to its parent and swap is the parent is smaller/larger.

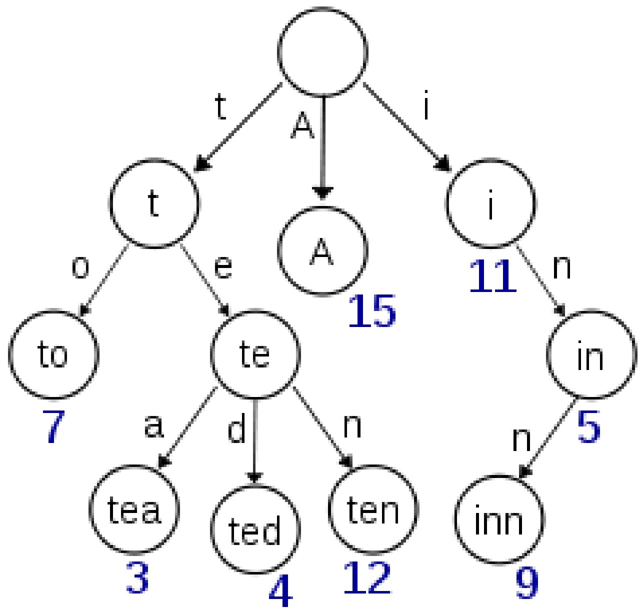


Figure 1.1: A trie data type visualization

Table 1.1: **Time Complexity**

Algo	Ave	Worst
Space	O(n)	O(n)
Search	O(1)	O(n)
Insert	O(1)	O(n)
Delete	O(1)	O(n)

Sets

Sets: Wiki Link

1.3 Hashes

Hash function

Hash Table

Hash Table: Wiki Link

Hash tables tend to be faster than other table data structures, degrading to the same average lookup time of an unordered array only in the worst case senario. If the hash function is complex and the entry count small, this advantage can be lost.

A hash table is an associative array (i.e. a dictionary) that maps keys to values. Puts a key through a hash function to find/retrieve an index to an array of buckets

Table 1.2: Hash table terms

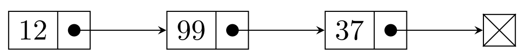
Key	The name of a value/ attribute
Bucket	The array elements. Typically a dynamic array
Slot	Synonym for bucket
Hash function	computes the index from a key
Load Factor	$\frac{entries}{buckets}$ The higher the load factorthe slower the search, the lower the load facter the more memory wasted.

Collision Resolution

Separate Chaining	Buckets have a list of their own entries to a single has, if the hash matches and the key doesn't do a linear search of the list
Open Addressing	On a collision, the new entry takes the next open slot. Useful if memory is an issue and the entries are smaller that $\sim 4\times$ sizeof(*)

Linked Lists

Singley linked lists



Doubley linked lists

Queue

Queue: Wiki Link

Fifo, any added item are added to the end/tail of the structure (inqueued). Items are only removed from the front/head (dequeued).

- Simulate waiting lines
- Buffers for I/O

Priority Queue

Stack

The Stack: Wiki Link

Opposite of a queue → LIFO

Ring Buffer

Ring Buffer: Wiki Link

Union

Tagged Union

Heuristics → efficient algos that get a good although not necessarily perfect solution

2

Algorithms

2.1 Asymptotics

How the algorithm grows as N $\rightarrow \infty$.

Algorithms by David Sedgewick: Page 67
David Mount Notes: Link
Big-O Notation: Wiki Link
Time Complexity: Wiki Link

Notation

Θ: The asymptotic class of an algo.

$$\Theta(g(n)) \equiv \left\{ f(n), 0 <= c_1 g(n) <= f(n) <= c_2 g(n) | c_1, c_2, n_0 \in |\Re| \text{ and } n_0 <= n \right\} \tag{2.1}$$

For a algo to be Θ(g(n)) it needs to be both O(g(n)) and Ω(g(n)). A Θ(g(n)) grows at exactly g(n).

O: Upper asymptotic bound for an algo. An algo that has O(g(n)) grows at or slower than that rate.

$$O(g(n)) \equiv \left\{ f(n) | 0 <= f(n) <= c g(n) | c, n_0 \in |\Re| \text{ and } n_0 <= n \right\} \tag{2.2}$$

Ω: The lower bound on the growth. An algo w/ Ω(g(n)) grows at or faster than g(n).

$$\Omega(g(n)) \equiv \left\{ f(n) | 0 <= c g(n) <= f(n) | c_1, c_2, n_0 \in |\Re| \text{ and } n_0 <= n \right\} \tag{2.3}$$

Table 2.1: Asymptotic growth types

Notation	Name	Example
O(1)	Constant	Seeing if a binary number is even or odd
O(log log n)	Double Logarthmic	
O(log n)	Logarithmic	Finding and item in a sorted array with binary search
O((log n)^c) w/ $c > 1$	Polylogarithmic	
O(n^c) w/ $0 < c < 1$	Fractional power	Searching in a kd-tree
O(n)	Linear	Find an item in an unsorted list
O(n log*n)	n log-star n	Union-find
O(n log n)	quasilinear/linearithmic	FFT
O(n^2)	Quadratic	Common limit on sorting
O(n^c)	Polynomial	LU decomposition

Performace

Worst Case

Best Case

Average Case

Asymptotic Analysis

(Strong) Induction

Iteration

Recurrance

Master's Theorem

2.2 Algorithm Types

Divide & Conquer

Selection

2.3 Searching

Binary Search

Binary Search: Wiki Link

Linear Search

Linear Search: Wiki Link

2.4 Selection

Sieve Technique

2.5 Sorting

Merge Sort

Heap Sort

Quick Sort

Bubble Sort

Insertion Sort

Selection Sort

Count Sort

Radix Sort

3

Math

3.1 Series

Arithmetic:

$$\sum_{i=1}^{n} i = 1 + 2 + ... + n = \frac{n(n+1)}{2} \tag{3.1}$$

Geometric:

$$\sum_{i=0}^{n} x^i = 1 + x + x^2 + ... + x^n = \frac{x^{n+1} - 1}{x - 1} \tag{3.2}$$

Harmonic:

$$H_n \equiv \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n} \approx ln(n) \tag{3.3}$$