Map/Dictionary Implementation Recap

Dictionary operation	Unsorted array	Sorted array	Singly linked unsorted	Singly linked sorted	balanced BST	hashtable
Search(A, k)	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$	O(log n)	O(1) expected
Insert(A, x)	$O(1)$	$O(n)$	$O(1)$	$O(n)$	O(log n)	O(1) expected
Delete(A, x) or Delete(A,k) (given location of *x*)	$O(1)$*	$O(n)$	O(1) *	O(1) *	O(log n)	n/a
Remove(A,x) or Remove(A,k) (not given location of *x*)	O(n) requires search + delete	O(n)	O(n)	O(n)	O(log n)	O(1) expected

OrderedDictionary

Ordered Dictionary operation	Unsorted array	Sorted array	Singly linked unsorted	Singly linked sorted	balanced BST	hashtable
Search(A, k)						
Insert(A, x)						
Delete(A, x)						
Successor(A, x)						
Predecessor(A, x)						
Minimum(A)						
Maximum(A)						

Recap – ADTs

We've considered major categories of ADTs for collections, characterized by the access they provide for their elements, and commons ADTs within those categories

- *containers* – based on position, not element value
  - Sequence/List – linear structure with access at any position
  - Stack – insert/remove at the same end (top)
  - Queue – insert/remove at opposite ends (front, back)
- *dictionary* – based on element's key (lookup)
  - Dictionary/Map – find(k), insert(k,v), remove(k)
  - OrderedDictionary – also max/min, successor(k), predecessor(k)
- *priority queue* – ordered, based on element's key
  - PriorityQueue – insert(x), findMin (or max), removeMin (or max)

Recap – Data Structures

We've seen some useful data structures – arrays, linked lists, binary trees, general trees.

We've seen some clever ways to use and adapt basic data structures to achieve efficient implementations of ADTs.

- sorted array/linked list (vs. unsorted)
- circular arrays
- linked list with tail pointers
- array-based implementation of binary trees
- search trees (binary, multiway) – trees with ordering property for elements
- balanced search trees (AVL, 2-4) – search trees with structural property to maintain log n height
- hashtables – arrays with clever conversion of key to array index
- heaps – trees with ordering + structural properties

Specialized Data Structures

Be aware that there's more out there.

- other implementations
  - dictionaries: splay trees, red-black trees, b-trees, skip lists
  - priority queues: bounded height PQs, Fibonacci heaps, pairing heaps
- string data structures
  - e.g. suffix trees/arrays for pattern matching
  - e.g. prefix trees
- geometric data structures
  - e.g. BSP, kd-trees for fast searching in space
- graph data structures
- set data structures

Match each of the following scenarios with the most appropriate data structure used in its implementation.

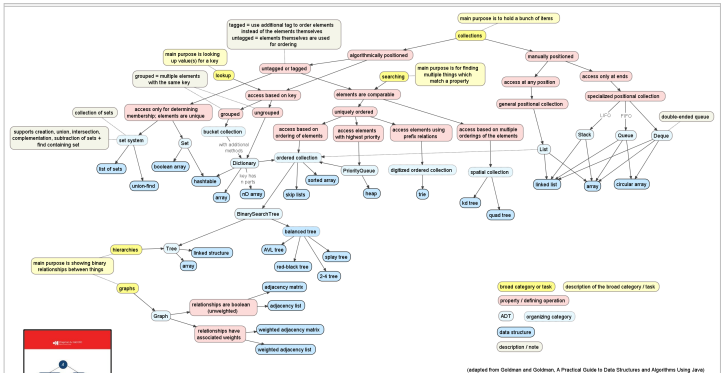
Question	Map/Dictionary	kd-trees	set	suffix trees/arrays	priority queue
to store configuration options and their values for an application managing runtime settings	✓ 4	1	1	0	0
counting word frequency in a document	✓ 4	0	0	1	1
a symbol table used by a compiler to associate variable names with their properties (type, scope, memory location) during code compilation	✓ 5	0	1	0	0
in a text adventure game, resolving the name entered by the user to its corresponding object representation	✓ 5	0	0	1	0
managing processes or threads in an operating system so that the most important tasks are serviced first	0	0	0	0	✓ 6
in best-first and A* search algorithms, to explore the most promising node first	0	0	0	0	✓ 6
in an event-driven simulation, to ensure that scheduled events are processed in the correct chronological order	0	0	0	1	✓ 5

Match each of the following scenarios with the most appropriate data structure used in its implementation.

Question	Map/Dictionary	kd-trees	set	suffix trees/arrays	priority queue
in image recognition, storing feature descriptors to accelerate searches for similar image patches	0	✓ 4	1	1	0
in robot navigation, determine the nearest obstacles in order to plan collision-free paths through an environment	0	✓ 5	0	0	1
in GIS, to find all locations (such as restaurants or hospitals) within a specific area	0	✓ 6	0	0	0
in ray tracing (computer graphics), finding the nearest object hit by a ray	0	✓ 5	0	0	1
in pattern recognition, speeding up nearest-neighbor searches when classifying handwritten digits or facial recognition	0	✓ 4	0	1	1

Match each of the following scenarios with the most appropriate data structure used in its implementation.

Question	Map/Dictionary	kd-trees	set	suffix trees/arrays	priority queue
keeping track of visited nodes during search	1	0	✓ 5	0	0
storing a unique collection of permissions or roles for a user within a security context	0	0	✓ 6	0	0
removing duplicate email addresses from a large collection of email addresses	0	1	✓ 5	0	0
in plagiarism detection, to find the longest substring appearing in two or more other strings	0	0	0	✓ 6	0
building indexes for large text databases	3	0	1	✓ 2	0
finding all occurrences of a pattern or substring within a large text	0	0	0	✓ 6	0



A Practical Guide to Data Structures and Algorithms using Java, Goldman and Goldman

The Algorithm Design Manual, Skiena