algorithms and datastructures java Cheat Sheet

by Meliodas via cheatography.com/61459/cs/15877/

DATASTRUCTURES

Linked List

DEFINATION: A linked list is a data structure composed of a group of nodes, each node linking to the next node in the sequence. Each node consist of two elements, #.data storing the contents of the node and #.next a reference to the next node.

ADDING:(To add data from a LL:)

To add a new node with data to t startof LL:

```
1. Create a new node containing data
2. Point the .next of the new node to the current first node
3. Point the start of the list to the new node
```

REMOVING: (To remove data from a LL:)

```
1. If the first node #.data is equal to data:
     a. Point the start of the list to the node after the first node
2. Otherwise:
    a. Find a node *prev just before a node with #.data equal to data
    b. Point prev.next to prev.next.next skipping the node being removed
```

ARRAY

Definition- Stores data elements based on an sequential, most commonly 0 based,index.

Stack (abstract data type) LIFO

A stack is a abstract collection data type where the primary operations are **push** which adds an element and **pop** which removes. It is a Last-In-First-Out (LIFO) data structure, the last element pushed must be the first one popped.

Array Based

Stack (abstract data type) LIFO (cont)

An array implementation uses an simple array and items are added and removed from the end of the array. In a language without dynamic arrays, the stack would keep track of the number of items and the available space, when the stack runs out of space the array would be reallocated to provide more space. In Javas the **array**data structure is dynamically resized and the stack implementation is very simple. Arrays in Java provide native push and pop functions, for this visualisation these are reimplemented.

Linked List Based

The Linked List based implementation is also simple. A **push** adds a new node onto the front of a list, and pop removes the front of the list.

Queue (abstract data type) FIFO

A queue is a abstract collection data type where the primary operations are enqueue which adds an element and dequeue which removes. It is a First-In-First-Out (FIFO) data structure, the first element pushed must be the first one removed.

Linked List Based

The *Linked List based implementation works by keeping a pointer to the start and end of the list. Dequeueing removes the first element of the list, much like the linked list* stack, enqueueing utilises the pointer to the end of the list to add an item at the end of the list in constant time.

Array Based

The array based implementation works by keeping a two arrays **left** and **right**. Dequeue is O(1) even though it includes a O(n) reverse operation. This is because the reverse happens at a 1/n frequency.

Hash table

A hash table is a data structure that maps keys to values. The keys are distributed across a number of buckets by hashing the key to produce a bucket index.
A good hash function produces an even distribution across all the buckets so that no bucket is overloaded.
Even with a good hash function collisions happen, where two keys are hashed to the same slot. Therefore most hash tables have some collision resolution strategy to handle this case.
Separate chaining
In separate chaining (also called open hashing or closed addressing) each bucket has its own list of entries. A good hash table only has very few items in each bucket, and so data structures like a linked list are popular.
This example is hash set, as it only stores **a key**. However all the main logic is the same, a hash table stores **a (key, value)** pair instead of just a key, but all the hashing and equality checks are still performed on the key.

SEARCHING

LinearSearch (sequential search)

Linear search (sequential search) is the most simple approach to find out, whether the array (or a different data structure) contains some element. The principle of linear search is trivial – iterate over all elements stored in the structure and compare them with the searched one. In the worst case – the last element is equal to the searched one or the structure does not contain the element at all – linear search has to perform n comparisons, hence the asymptotic complexity of the algorithm is O(n).

Binary search (half-interval search)

A binary search algorithm finds the **position/index** of inp **input element** in a sorted array. With each iteration it halves the number of items to check.
// **searches the sorted array myArray** for **an item inp**
// **returns the *index* of *inp* or -1 if not found**

SORTING Comparison sorting

Bubble sort

Description
In every step it compares two adjacent elements and if the lower value is on the left side of the higher, bubble sort swaps them (lighter value ascends to the end of the array) and with the same logic algorithm proceeds to the next item.
After one iteration the lowest value is located at the end of the array. Algorithm now repeats the procedure with reduced array (the last element is already sorted). After n-1 iterations is the array completely sorted, because the last bubble is sorted trivially.

Insertion sort

Description
1.One element is sorted trivially
2. Pick element next to the already sorted sequence and insert it to the correct place - move every element of the already sorted sequence, which has a higher value than the element being sorted, one place right, than put the element into the gap (correct place within the sequence).
3. While array contains any unsorted elements GOTO: 2.

Selection sort

The algorithm sorts the list building the sorted section front to back. Each pass through the list finds the minimum element in the unsorted range and swaps the element so that it is at the end of the sorted portion of the list.

Bucket sort (Other sorting)

Bucket sort, or bin sort, is a sorting algorithm that works by partitioning an *array* into a number of **buckets.** Each bucket is then sorted individually, either using a different *sorting algorithm*, or by **recursively applying** the bucket sorting algorithm.
Bucket sort works as follows:
a. Set up an array of initially empty “buckets”.
b. Scatter: Go over the original array, putting each object in its bucket.
c. Sort each non-empty bucket.
d. Gather: Visit the buckets in order and put all elements back into the original array.

RADIX SORT

Description
The fundamental principle of radix sort stems from the definition of the stable sort - sorting algorithm is stable, if it maintains the order of keys, which are equal.
Radix sort iteratively orders all the strings by their n-th character – in the first iteration, the strings are ordered by their last character. In the second run, the strings are ordered in respect to their penultimate character. And because the sort is stable, the strings, which have the same penultimate character, are still sorted in accordance to their last characters. After n-th run the strings are sorted in respect to all character positions.

TREES

A binary search tree is a binary tree data structure where each node is greater than all the nodes in its left sub-tree and smaller than all the nodes in its right sub-tree. Each child is another binary search tree.
The shape of a binary search tree depends on the order of insertions and it can be degenerate. More complex data structures like AVL-trees or red-black trees are self balancing binary search trees and ensure that the shape does not become degenerate leading to worst case run times.
Binary search trees have very fast insertion and deletion when balanced, and the code is a lot simpler than other self balancing binary search trees.

HEAP SORT

Heapsort is based on usage of the **binary heap - data structure** which acts as a **priority queue.** If we insert all elements of the array into the priority queue, the operation poll will always return (and remove) the element of the heap, which has the highest priority.
If we use poll operation n times, we will obtain list of sorted elements.
The heapsort algorithm consists of these steps:
1.Build the heap using all elements of the array.
2.Poll the highest element of the heap.
3.Swap it with the last element of the heap (in array).
4.Reduce the heap size by 1 (elements at the end of the heap are already sorted).
5.Repair the heap (move element swapped in 3 to its correct place in the structure).
6.If there are any elements remaining in the heap **GOTO: 2**.
7.Array is sorted according to the priority of the elements in reverse order.