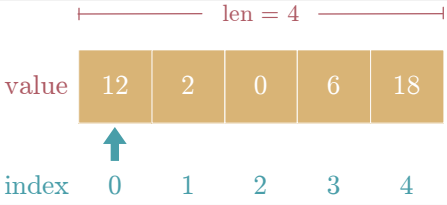
Cheat Sheet – Data Structures

1. List

- Ordered collection of elements
- The position of each element is defined by the *index*
- The elements can be accessed in *any order*



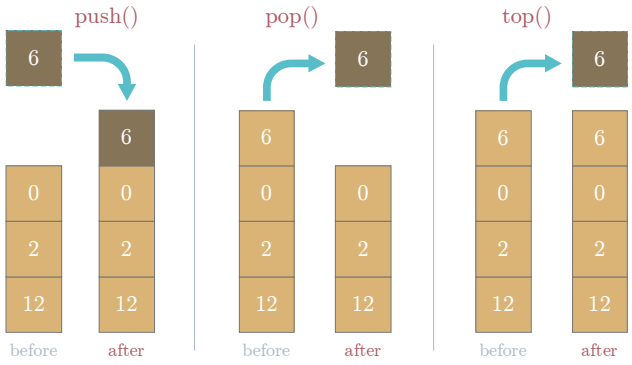
2. Linked List

- Each linked list element contains both the values and the address (pointer) to the next linked list element.
- Hence the linked list can only be traversed sequentially going through each element at a time

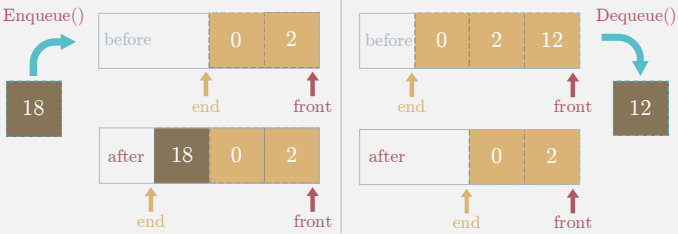
- Linked List does not have their order defined by their physical placement in the memory.
- Contiguous elements of the linked list are not placed adjacent to each other in the memory.

3. Stack

- Stack is a sequential data structure which maintains the order of elements as they were inserted in.
- Last In First Out (**LIFO**) order, which means that the elements can only be accessed in the ***reverse order*** as they were inserted into the stack.
- The element to be ***inserted last,*** will the ***first one to get removed*** from the stack.
- **Push()** adds an element at the ***head*** of the stack, while **pop()** removes an element from the ***head*** of the stack
- A real-life example of a stack is a stack of kitchen plates



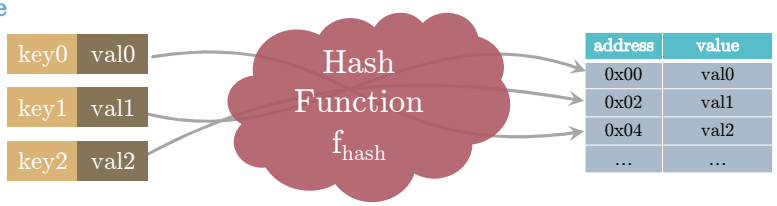
4. Queue



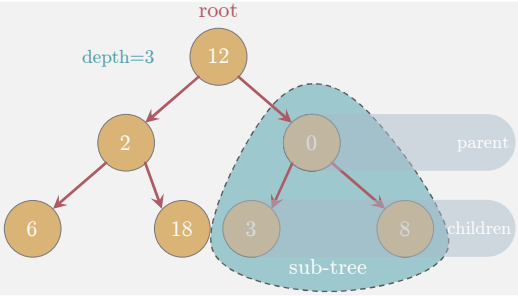
- A queue is a sequential data structure that maintains the order of elements as they were inserted in
- First In First Out (**FIFO**), the element to be ***inserted first***, will the ***first one to get removed*** from the queue
- Whenever an element is added (Enqueue()) it is added to the ***end*** of the queue. On the other hand, element removal (Dequeue()) is done from the ***front*** of the queue.
- A real-life example is a check-out line at a grocery store

5. HashTable

- Creates ***paired assignments*** (key mapped to values) so the pairs can be accessed in constant time
- For each *(key, value)* pair, the key is passed through a hash function to create a unique physical address for the value to be stored in the memory.
- Hash function can end up generating the same physical address for different keys. This is called a ***collision***.



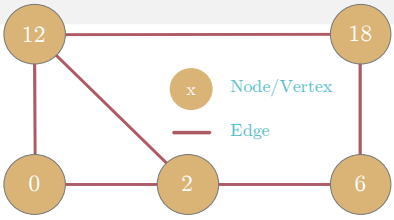
4. Tree



- Maintains a hierarchical relation between its elements.
- Root Node – The node at the top of the tree
- Parent Node – Any node that has at least one child
- Child Node – The successor of a parent node is known as a child node. A node can be both a parent and a child node. The root is never a child node.
- Leaf Node– The node which does not have any child node.
- Traversing – Passing through the nodes in a certain order, e.g BFS, DFS

4. Graph

- A graph is a *pair of sets (V, E),* where **V** is set of all the vertices, **E** is set of all edges.
- A neighbor of a node is set of all vertices connected with that node through an edge.
- As opposed to trees, a graph can be cyclic, which means starting from a node and following the edges, you can end up on the same node



Source: https://www.cheatsheets.aqeel-anwar.com Tutorial: Click here