FullStack.Cafe - Kill Your Tech Interview

Q1: Name some characteristics of Array Data Structure ☆

Topics: Arrays Data Structures

Answer:

Arrays are:

- **Finite (fixed-size)** - An array is finite because it contains only limited number of elements.
- **Order** -All the elements are stored one by one , in contiguous location of computer memory in a linear order and fashion
- **Homogenous** - All the elements of an array are of same data types only and hence it is termed as collection of homogenous

Q2: Define Stack ☆

Topics: Stacks Data Structures

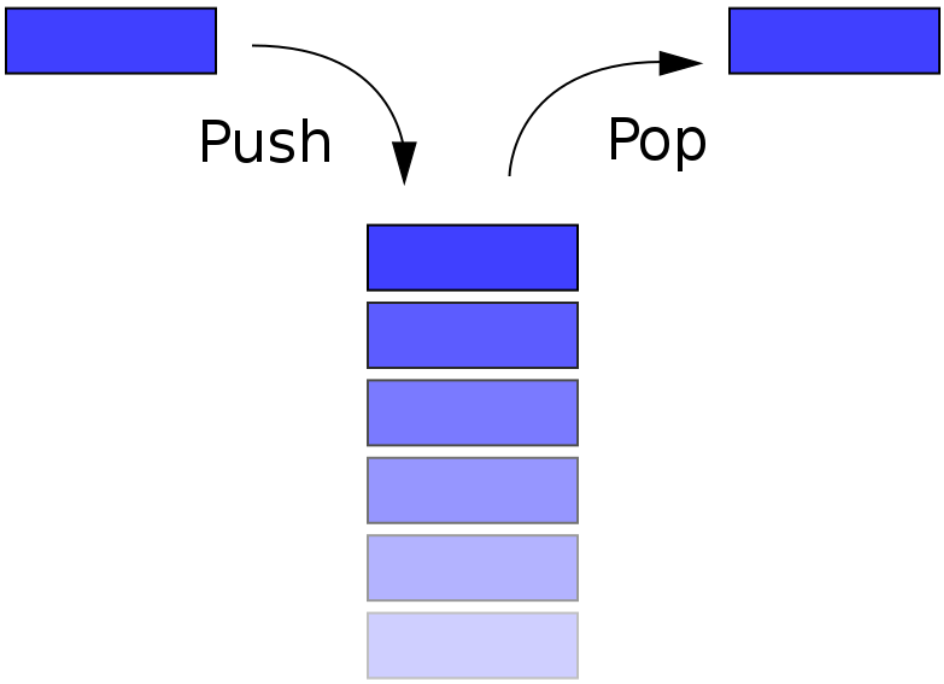
Answer:

A **Stack** is a container of objects that are inserted and removed according to the last-in first-out (**LIFO**) principle. In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack.

There are basically three operations that can be performed on stacks. They are:

1. inserting an item into a stack (**push**).
2. deleting an item from the stack (**pop**).
3. displaying the contents of the stack (**peek** or **top**).

A stack is a limited access data structure - elements can be added and removed from the stack only at the top. push adds an item to the top of the stack, pop removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.



Q3: Explain why Stack is a recursive data structure ☆

Topics: Stacks Data Structures

Answer:

A **stack** is a **recursive** data structure, so it's:

- a stack is either empty or
- it consists of a top and the rest which is a stack by itself;

Q4: Define Linked List ☆

Topics: Linked Lists Data Structures

Answer:

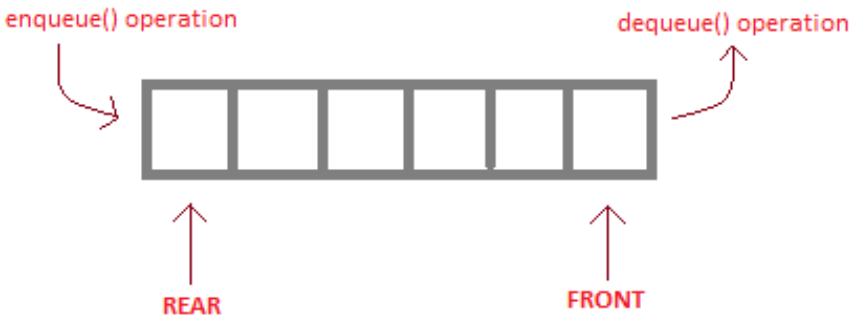
A **linked list** is a linear data structure where each element is a separate object. Each element (we will call it a **node**) of a list is comprising of two items - the **data** and a **reference (pointer)** to the next node. The last node has a reference to **null**. The entry point into a linked list is called the **head** of the list. It should be noted that *head is not a separate node,* but the reference to the first node. If the list is empty then the head is a null reference.

Q5: What is Queue? ☆

Topics: Queues Data Structures

Answer:

A **queue** is a container of objects (a *linear* collection) that are inserted and removed according to the first-in first-out (FIFO) principle. The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.



enqueue( ) is the operation for adding an element into Queue.
dequeue( ) is the operation for removing an element from Queue .

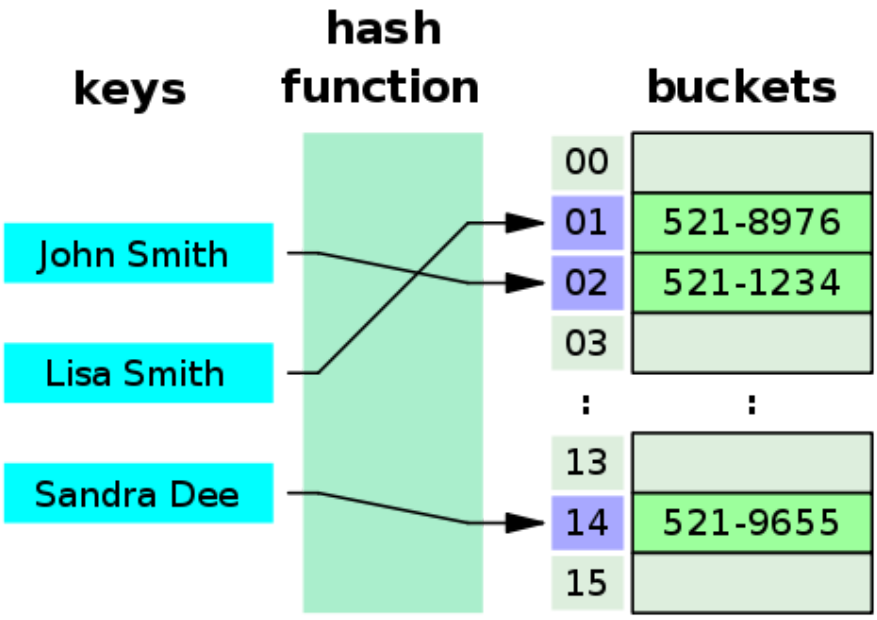
QUEUE DATA STRUCTURE

Q6: What is Hash Table? ☆

Topics: Hash Tables Data Structures

Answer:

A **hash table** (hash map) is a data structure that implements an **associative** array abstract data type, a **structure** that can **map keys to values**. Hash tables implement an associative array, which is indexed by arbitrary objects (keys). A hash table uses a **hash function** to compute an **index**, also called a **hash value**, into an **array of buckets** or slots, from which the desired **value** can be found.



Q7: What is Heap? ☆

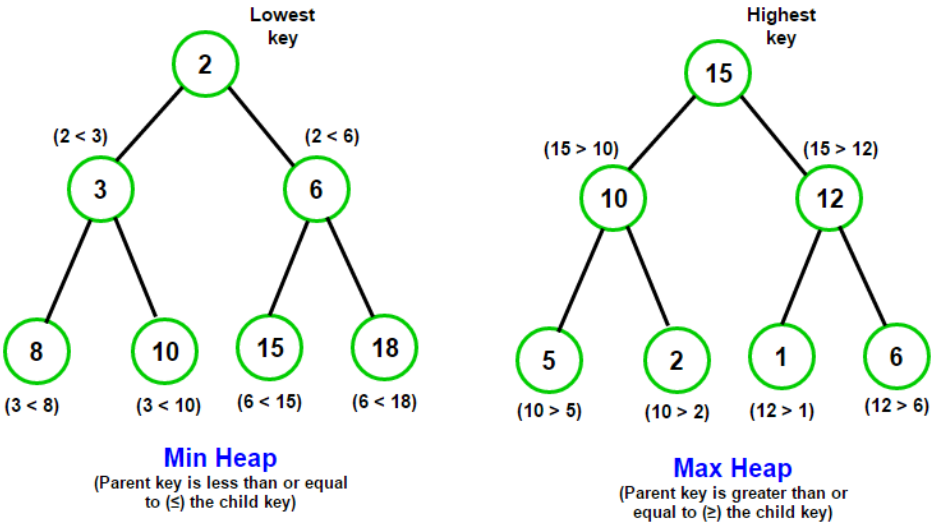
Topics: Heaps and Maps Data Structures

Answer:

A **Heap** is a special Tree-based data structure which is an almost complete tree that satisfies the heap property:

- in a **max heap**, for any given node C, if P is a parent node of C, then the key (the value) of P is greater than or equal to the key of C.
- In a **min heap**, the key of P is less than or equal to the key of C. The node at the "top" of the heap (with no parents) is called the root node.

A common implementation of a heap is the binary heap, in which the tree is a **binary tree.**

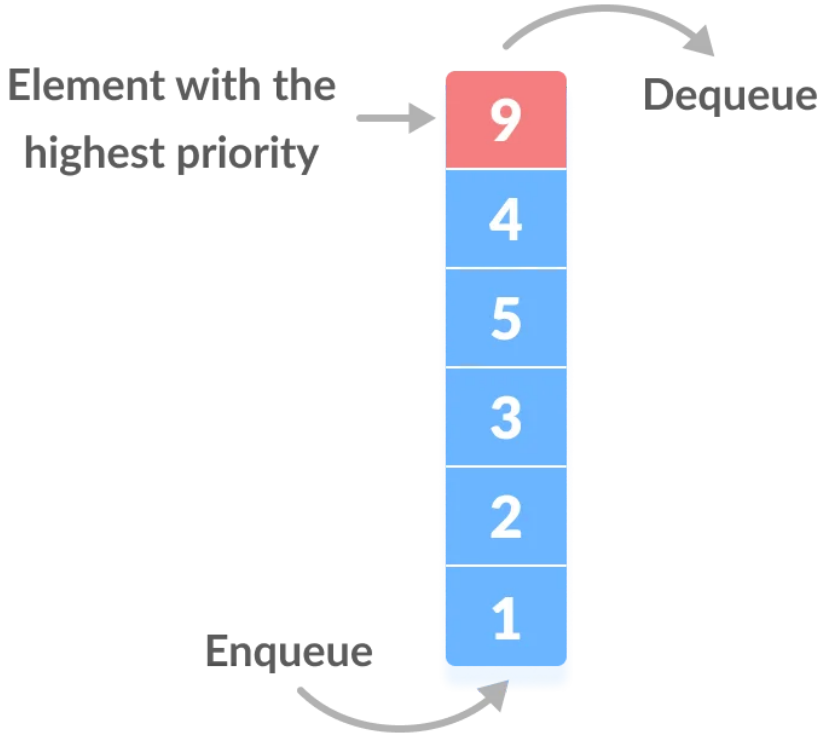


Q8: What is Priority Queue? ☆

Topics: Heaps and Maps Data Structures

Answer:

A **priority queue** is a data structure that stores **priorities** (comparable values) and perhaps associated information. A **priority queue** is different from a "normal" queue, because instead of being a "first-in-first-out" data structure, values come out in order by **priority**. Think of a priority queue as a kind of bag that holds priorities. You can put one in, and you can take out the current highest priority.



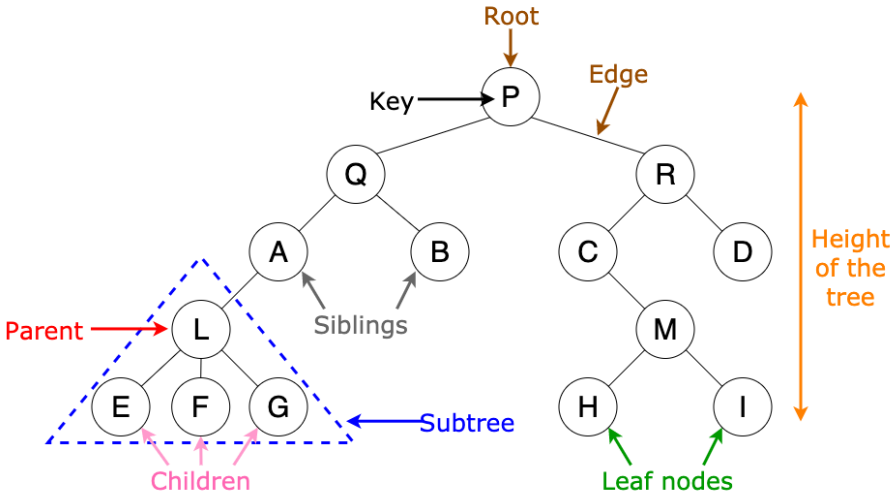
Q9: Define Tree Data Structure ☆

Topics: Trees Data Structures

Answer:

Trees are well-known as a *non-linear* data structure. They don't store data in a linear way. They organize data *hierarchically*.

A **tree** is a collection of entities called **nodes**. Nodes are connected by **edges**. Each node contains a **value** or **data** or **key**, and it may or may not have a **child** node. The first node of the tree is called the **root**. **Leaves** are the last nodes on a tree. They are nodes without children.



Q10: Define Binary Tree ☆

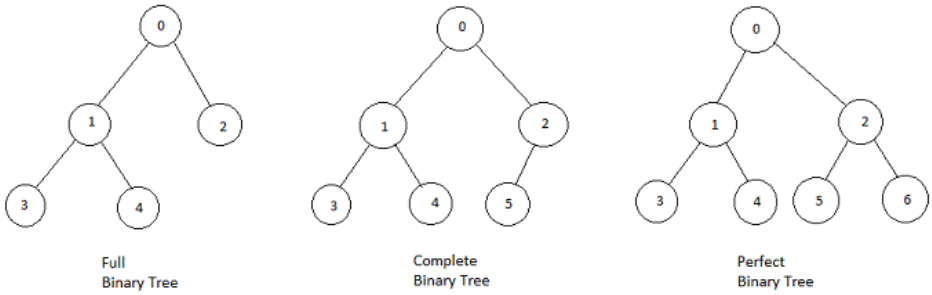
Topics: Binary Tree Trees Data Structures

Answer:

A normal tree has no restrictions on the number of children each node can have. A **binary tree** is made of nodes, where each node contains a "left" pointer, a "right" pointer, and a data element.

There are three different types of binary trees:

- **Full binary tree**: Every node other than leaf nodes has 2 child nodes.
- **Complete binary tree**: All levels are filled except possibly the last one, and all nodes are filled in as far left as possible.
- **Perfect binary tree**: All nodes have two children and all leaves are at the same level.

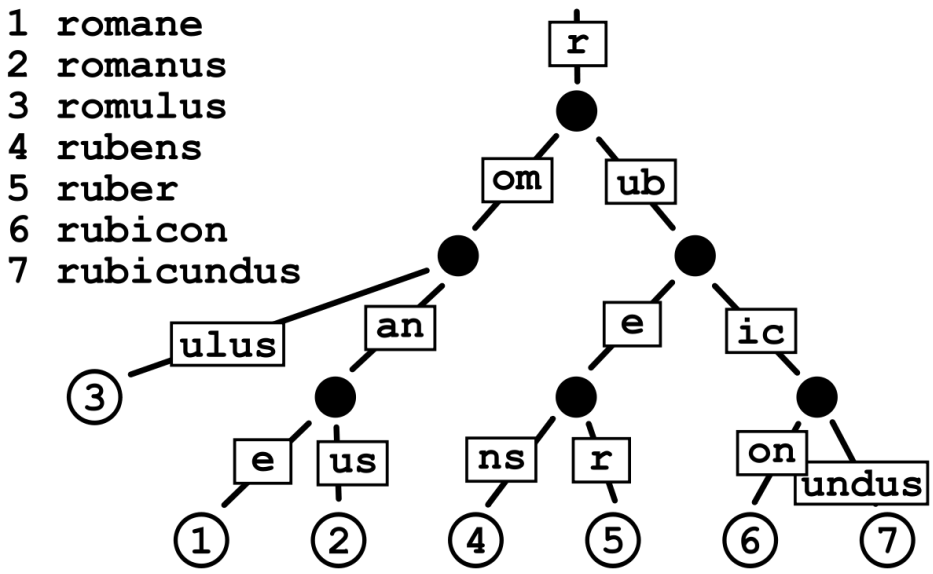


Q11: What is Trie? ☆

Topics: Trie Data Structures

Answer:

Trie (also called **digital tree **or **prefix tree**) is a *tree-based data structure*, which is used for efficient *retrieval* of a key in a large data-set of strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated; i.e., **the value of the key is distributed across the structure**. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. Each complete English word has an arbitrary integer value associated with it (see image).



Q12: What is String in Data Structures? ☆

Topics: Strings Data Structures

Answer:

A **string** is generally considered as a **data type** and is often implemented as an **array data structure** of bytes (or words) that stores a sequence of elements, typically characters, using some character encoding. String may also denote more general arrays or other sequence (or list) data types and structures.

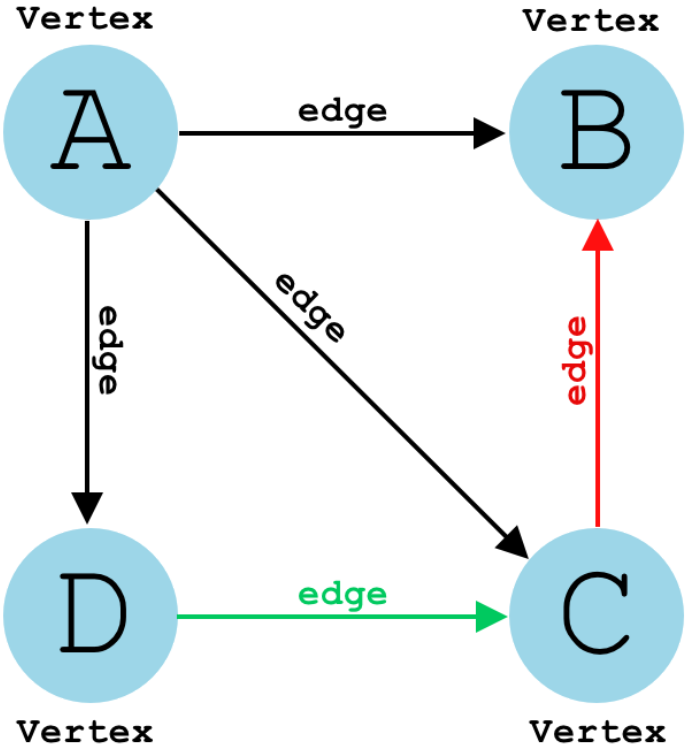
Q13: What is a Graph? ☆

Topics: Graph Theory Data Structures

Answer:

A **graph** is a common data structure that consists of a finite set of **nodes** (or **vertices**) and a set of **edges** connecting them. A pair `(x,y)` is referred to as an edge, which communicates that the **x vertex** connects to the **y vertex**.

Graphs are used to solve real-life problems that involve representation of the problem space as a **network**. Examples of networks include telephone networks, circuit networks, social networks (like LinkedIn, Facebook etc.).



Q14: What are Dynamic Arrays? ☆☆

Topics: Arrays Data Structures

Answer:

A **dynamic array** is an array with a big improvement: *automatic resizing*.

One limitation of arrays is that they're *fixed* size, meaning you need to specify the number of elements your array will hold ahead of time. A dynamic array expands as you add more elements. So you don't need to determine the size ahead of time.

Q15: Why and when should I use Stack or Queue data structures instead of Arrays/Lists? ☆☆

Topics: Queues Stacks Data Structures

Answer:

Because they help manage your data in more a *particular* way than arrays and lists. It means that when you're debugging a problem, you won't have to wonder if someone randomly inserted an element into the middle of your list, messing up some invariants.

Arrays and lists are random access. They are very flexible and also easily *corruptible*. If you want to manage your data as FIFO or LIFO it's best to use those, already implemented, collections.

More practically you should:

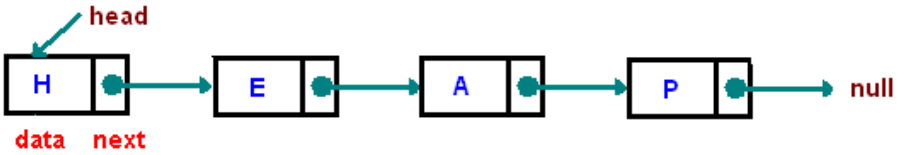
- Use a queue when you want to get things out in the order that you put them in (FIFO)
- Use a stack when you want to get things out in the reverse order than you put them in (LIFO)
- Use a list when you want to get anything out, regardless of when you put them in (and when you don't want them to automatically be removed).

Q16: What are some types of Linked List? ☆☆

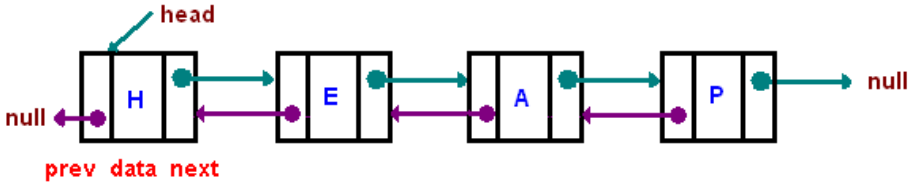
Topics: Linked Lists Data Structures

Answer:

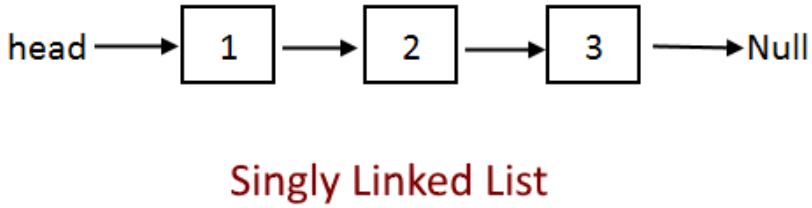
- A **singly linked list**



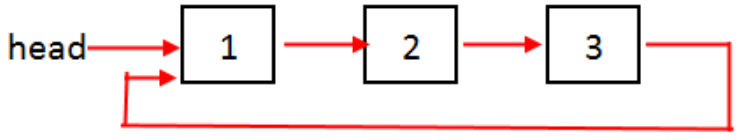
- A **doubly linked list** is a list that has two references, one to the next node and another to previous node.



- A **multiply linked list** - each node contains two or more link fields, each field being used to connect the same set of data records in a different order of same set(e.g., by name, by department, by date of birth, etc.).
- A **circular linked list** - where last node of the list points back to the first node (or the head) of the list.



Singly Linked List



Circular Linked List

Q17: Name some disadvantages of Linked Lists? ☆☆

Topics: Linked Lists Data Structures

Answer:

Few disadvantages of linked lists are :

- They use more memory than arrays because of the storage used by their pointers.
- Difficulties arise in linked lists when it comes to reverse traversing. For instance, singly linked lists are cumbersome to navigate backwards and while doubly linked lists are somewhat easier to read, memory is wasted in allocating space for a back-pointer.
- Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access.
- Random access has linear time.
- Nodes are stored incontiguously (no or poor cache locality), greatly increasing the time required to access individual elements within the list, especially with a CPU cache.
- If the link to list's node is accidentally destroyed then the chances of data loss after the destruction point is huge. Data recovery is not possible.
- Search is linear versus logarithmic for sorted arrays and binary search trees.
- Different amount of time is required to access each element.
- Not easy to sort the elements stored in the linear linked list.

Q18: What is Complexity Analysis of Queue operations? ☆☆

Topics: Queues Data Structures

Answer:

- Queues offer random access to their contents by shifting the first element off the front of the queue. You have to do this repeatedly to access an arbitrary element somewhere in the queue. Therefore, **access** is $O(n)$.
- Searching for a given value in the queue requires iterating until you find it. So **search** is $O(n)$.
- Inserting into a queue, by definition, can only happen at the back of the queue, similar to someone getting in line for a delicious Double-Double burger at In 'n Out. Assuming an efficient queue implementation, queue **insertion** is $O(1)$.
- Deleting from a queue happens at the front of the queue. Assuming an efficient queue implementation, queue **deletion** is `$O(1)$.

Q19: What are some types of Queue? ☆☆

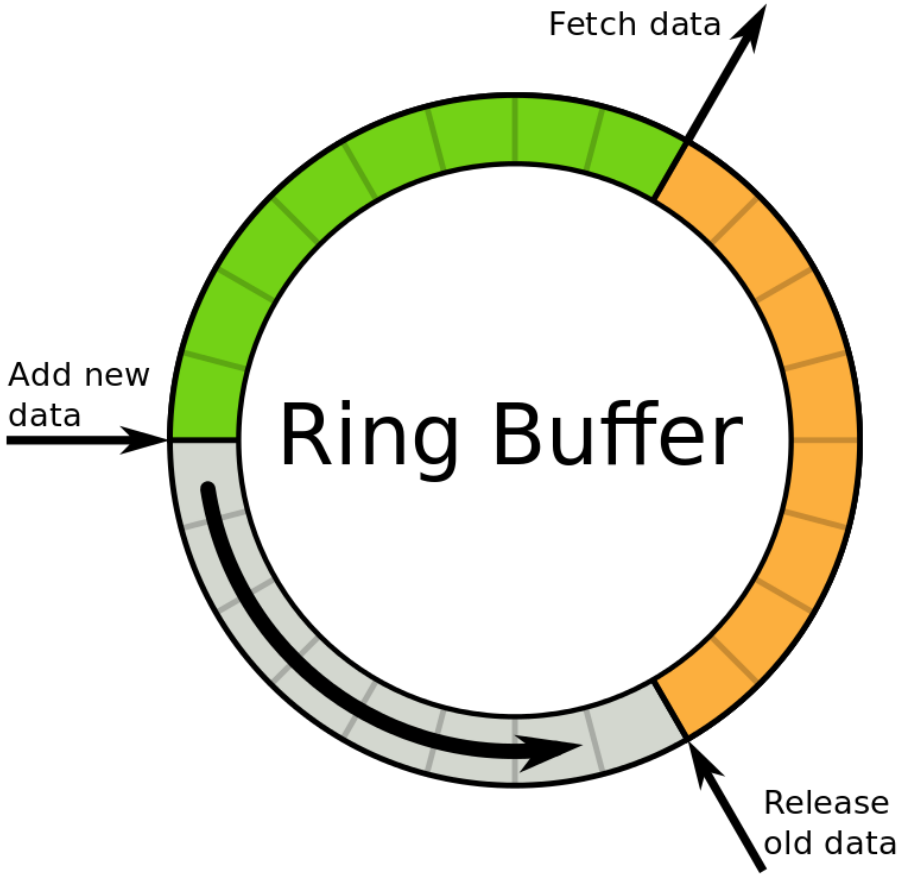
Topics: Queues Data Structures

Answer:

Queue can be classified into following types:

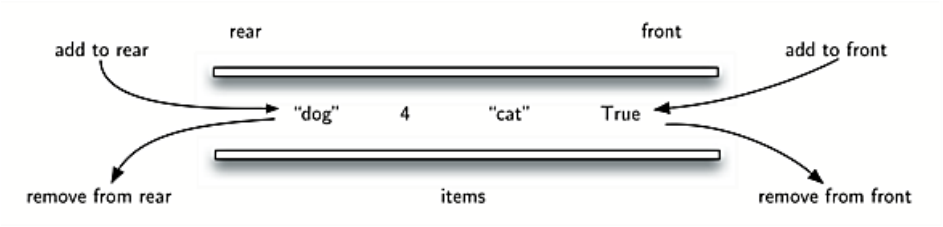
- **Simple Queue** - is a linear data structure in which removal of elements is done in the same order they were inserted i.e., the element will be removed first which is inserted first.

- **Circular Queue** - is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called **Ring Buffer**. Circular queue avoids the wastage of space in a regular queue implementation using arrays.

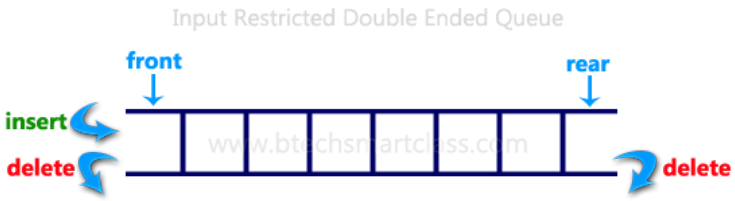


- **Priority Queue** - is a type of queue where each element has a priority value and the deletion of the elements is depended upon the priority value

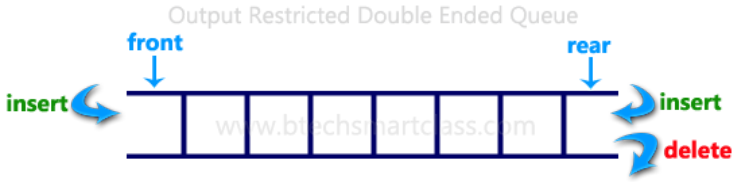
- In case of **max-priority queue**, the element will be deleted first which has the largest priority value
- In case of **min-priority queue** the element will be deleted first which has the minimum priority value.
- **De-queue (Double ended queue)** - allows insertion and deletion from both the ends i.e. elements can be added or removed from rear as well as front end.



- **Input restricted deque** - In input restricted double ended queue, the insertion operation is performed at only one end and deletion operation is performed at both the ends.



- **Output restricted deque** - In output restricted double ended queue, the deletion operation is performed at only one end and insertion operation is performed at both the ends.



Q20: What is the space complexity of a Hash Table? ☆☆

Topics: Hash Tables Data Structures

Answer:

The space complexity of a datastructure indicates how much space it occupies in relation to the amount of elements it holds. For example a space complexity of `O(1)` would mean that the datastructure alway consumes constant space no matter how many elements you put in there. `O(n)` would mean that the space consumption grows linearly with the amount of elements in it.

A **hashtable** typically has a space complexity of `O(n)`.