STACK

A Stack is a linear datastruct ure that follows (Last In First Out) LIFO Principle. This means that the last element in serted into the Stack is the first element to be removed.

In Computer science Stacks are used to implement recursion. Recursion is a Programming method that allows a function to Call itself. This Can be used to Solving Problems that Can be broken down to Smaller Similar Problems.

=> Operations on Stack

- Push: Add element to the top of the Stacks
- Pop: Remove the top element from the Stacks
- Peek: Returns the top element from the Stacks without removing it.
- isEmpty: Checks if the Stack is empty
- isFull: Checks if the Stack is Full or not.

=> Applications of Stack

1) Expression Evaluation:

Stacks Can be used to evaluate mathe matical expressions. (eg: $2+4/(5*(7-9))$)

2) Backtracking:

Stacks Can be used to implement backtra cking. For example, the Problem of finding all Possible Paths through a maze Can be Solved by Stacks

3) Undo / redo :

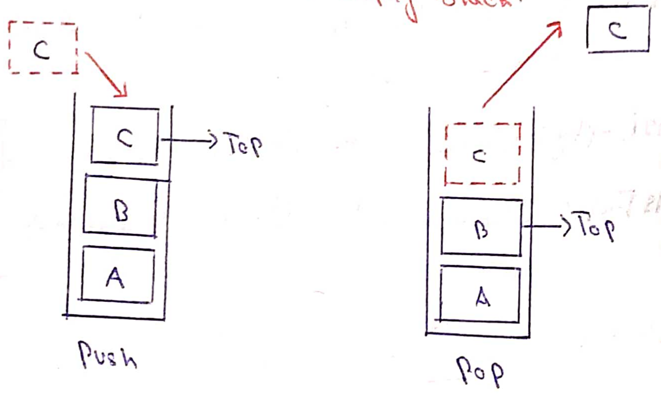
Stacks keeps the Changes that had done in a document. It allows users to undo or redo Changes

4) Function Calls :

Stacks are used to implement Function Calls, when a Function is Called the arguements and local variables are stored into the stack when the function returns all these are popped off.

=> Stack overflow: Stack overflow is a Conditions that occur when we trying to insert a new element on a Full Stack,

=> Stack Underflow: Stack under flow is a Condition that occurs when we trying to remove an element from an Empty Stack.



=> Stack Using Arrays

```
class Stack {
  List <dynamic> _Stack = [];
 bool get isEmpty => _Stack.isEmpty;
Void Push(dynamic item) => _Stack.add(item);
dynamic Pop () => isEmpty ? throwException('Stack
 is empty'): _Stack.removeLast();
```

dynamic peek() => isEmpty ? throw Exception ('Stack is Empty') : _Stack.last;

=> Types of Stacks

1) Array based Stack

An array based Stack uses a fixed Size array to Store elements.

2) Linked list based Stack

It uses a linked-list Structure to Store elements. Each element in this Stack is represented by a node.

3) Dynamic Array based Stack

A dynamic array-based Stack is Similar to the array-based stack, but it's Size is not fixed.

4) Two-Stack Queue

A two-Stack Queue is a queue implemented using two stacks. It utilizes two Stack for enqueue and other Stack for dequeue

5) Min Stack

A min Stack that keeps track of the minimum Element in Constant time. It is typically implemented by another stack.

6) Priority Stack

A Priority Stack is a Variation of a Stack that assigns a Priority Value to Each element. Elements with higher Priority removed first.

Queue

In Computer Science queue is a fundamental datastructure that follows the FIFO (First-In-First-Out) Principle. It represents a Collection of elements where the first element added is first one to be removed.

=> Operations in queue

1) Enqueue -> This operation adds an element to the end of the queue. The newly added element becomes the last one in the queue.

2) Dequeue -> This operation removes the element at the front of the queue. The element that has been in the queue the longest is removed and queue is updated quickly.

=> Applications of queue

1) Task Scheduling: used is Os to schedule tasks / Processes.

2) Print Pooling: when multiple users sent print requests queue manage it.

3) Message Queue: In distributed systems or message-Oriented system queue is used to enable asynchronous Communication b/w different Components.

4) BFS [Breadth First Search]: used to implement graph algorithms like BFS.

5) Buffering: Queues are often used as buffers in data Processing Systems.

6) Event handling: Queues are used to handle asynchronous events in Systems.

=> Types of Queues

1) Simple Queue: Also known as Standard queue, it follows the FIFO (FIRST IN FIRST OUT) Principle. Elements are inserted at the rear and removed from the front.

2) Circular Queue: It is Similar to Simple queue but the rear and front are connected when the rear reaches the end of the queue, it wraps around to the front creating a circular ~~queue~~ Structure. This allows for efficient space utilization and eliminates the need for shifting elements.

3) Priority Queue: in Priority queue elements are assigned a Priority value, and the element with the highest Priority removed first.

=> Complexities of Various Sorts in data Structure

1) merge Sort

Time Complexity

Best Case: O(n log n)

Average Case: O(n log n)

Worst Case: O(n log n)

Space Complexity: O(n)

3) Bubble_Sort

• Time Complexity

Best Case: $O(n)$

Average Case: $O(n^2)$

worst Case: $O(n^2)$

• Space Complexity: $O(1)$

4) Insertion Sort

• Time Complexity

Best Case: $O(n)$

Average Case: $O(n^2)$

worst Case: $O(n^2)$

Space Complexity: $O(1)$

5) Selection Sort

• Time Complexity

Best Case: $O(n^2)$

Average Case: $O(n^2)$

worst Case: $O(n^2)$

• Space Complexity: $O(1)$

6) Quick_Sort

• Time Complexity

Best Case: $O(n \log n)$

Average Case: $O(n \log n)$

worst Case: $O(n^2)$

• Space Complexity: $O(\log n)$-$O(n)$

=> Hashing

Hashing is a technique or process of Mapping Keys, Values into hash tables by using a hash function.

Hashing is designed to solve the problem of needing to efficiently find or store an item in a collection

It is done for faster access to elem

The efficiency of mapping depends on the efficiency of the hash function used.

-> Hashing is a technique which uses less key comparisons and searches the element in

- O(n) time in the worst case
- O(1) time in an average case

=> Types of hash functions

1) Division Method
2) Mid Square method
3) Digit folding method

-> Division Method

In this method the hash function is dependent upon the remainder of a division.

h (key) = record % table size

-> Mid Square Method

Consider that if we want to place a record of 3101 and the size of table is 1000, Location = (middle 3 digit)

3101 * 3101 = 96$\boxed{162}$01

h (3101) = 162

-> Digit Folding Method

In this method the key is divided into separate parts and by using some simple operations these parts are

Combined to Produce a hash key.

For example : Consider a record of 124655 012 then it will be divided into Parts ie, 124 , 655 , 012 . After dividing the Parts Combine these Parts by adding it.

$H(key) = 124 + 655 + 012 = 791$

=> Collision

Q what is Collision ?

It is a Situation in which the hash function returns the Same hash key for more than one record.

=> Collision handling method.

1) Chaining
2) Double hashing
3) Linear Probing
4) Quadratic Probing

~~1) Chaining~~

=> Applications of Hash tables

1) Hash tables are used in a variety of database applications , Such as

- indexing : used to index data in database
- Caching : Hash table is used to implement a Cache. Cache is a temporary Storage to Store recently accessed data.

• ~~Loading:~~

• Load balancing: Hash tables can be used to implement Load balancing. Load balancing is a technique that is used to distribute requests across multiple Servers.

2) Cryptography: Hash tables are used in Cryptography to Create message digests. message digests are a unique value that is Calculated from message

3) Machine Learning: Hash tables are used in Machine learning to Create feature vectors.

4) Natural Language Processing: Hash tables are used in natural language Processing to Create word embeddings. A word embedding is a vector representation of a word.

=> Types of hash tables

1) Open addressing: Uses Array to store
2) Chaining: Uses Linked list to store

```
dynamic larg = _stack [0];
dynamic second = _stack [0];

for (int i = 0; i < _stack.length; i++)
  {
    if (_stack[i] > large)
      second = large;
      large = _stack [i];
  }
else if (_stack[i] > second && _stack[i] < large)
       {
return } second = _stack[i];
```