WORLD STAR™
Date :
Page :

Stack : Ordered Collection of Homogeneous Options.

→ Last in First Out principle.

→ Data insertion or deletion only from one end.

→ Gives access only to TOP element.

E.g. Web history, Undo.

Operations :

1. Create 2. check empty 3. push 4. pop 5. peek
6. Destroy.

Linked List → Stack

Advantages:

→ Constant time to push/pop element.

→ Infinite size.

Disadvantages:

→ Slowest compared to all other implementations.

push() adds element to head.
pop() removes head element
peek() displays head element

Storage : O(n).

Array Stack :

→ Problems : 1. Size must be known in advance.
2. If less space allocated, frequent OVERFLOW
3. Code needs to be modified for reallocation of space.
4. If more space allocated, sheer wastage of memory

→ Solution : Multiple Stacks : To have more than 1 stack in same array of sufficient size

Multiple Stacks

→ Array of size n is used to represent 2 stacks (A, B)
The value of n is such that it does not exceed the combined size of A and B.
If stack A grows from left to right, then stack B will grow from the right to left at the same time.

Applications :

1. Infix to Postfix. } conversion
2. Infix to Prefix }
3. Postfix } evaluation.
4. Prefix }
5. Recursion
6. Tower of Hanoi.

Also Used in :

→ Compilers
→ OS
→ Virtual machines → manipulating numbers
→ Algorithms → Backtracking
→ A.I → Finding Path

Evaluation :

1. Postfix : 1. Scan every character from left to right.
   2. If operand, push it to the stack
   3 If operator, pop top 2 operands and apply the operator.
   4. Push the result back to stack.
   5. Repeat the steps 1 to 4.

* For Prefix scan right to left [rest is same]

Reverse a string :

→ Push characters of string one by one into the stack
→ Pop characters in the stack one by one and put them back to string.

String is Pallindrome :

→ Push characters of string one by one to the stack.
→ Pop elements one by one and compare with original string from left to right.
→ If all comparisons are equal, the string is pallindrome

Recursion : Two main major cases :

→ Base case : No need to make further calls to same function.
→ Recursive case : 1. Problem is divided into subparts.
2. Function calls itself but with subparts.
3. Result is obtained by combining the solutions of subparts

In each recursive call, there is a need to save :

- Current values of parameters.
- Local variables
- The return address : The address where the control has to return after call.

WORLD STAR™
Date :
Page :

Queue

→ First in First Out Principle.
→ Insertion at one end – Rear.
→ Deletion at one end – Front.
→ Gives access only to elements [At front & Rear].

Operations:
1. Create 2. Check empty 3. Enqueue 4. Dequeue
5. Destroy.

Types : 1. Simple
2. Circular : Last node connected to First node.
3. Doubly Ended : Deletions / Insertions can be done at both ends.
4. Priority : Elements have predifined priority.
High : First removed Low : Last removed

* Simple:
Array Representation:
→ Useful when ~~an~~ queue is small
→ Size should be known beforehand.

Linked Representation:
→ Start pointer of linked list (head) is used as front.
→ Another pointer, pointing to the last element is used as Rear.
→ Insertions at Rear.
→ Deletions from Front.
→ If queue is empty FRONT = REAR = NULL

Storage : ~~O~~ O(n)
Time : O(1). [For any operation except destroy.]

WORLD STAR™
Date :
Page :

★ Circular :
Linked List :
→ Same as simple, but REAR points to FIRST
→ The new node will have, data, address of next node

★ Priority Queue :
Linked List :
→ Sorted List.
1. Insertion is done based on priority. [High to Low]
2. Deletion is done from the front.
3. O(n) to insert and O(1) to delete.

→ Unsorted List
1. Insertion is done from the rear.
2. Deletion is done based on priority. Highest priority element is search and deleted.
3. O(1) to insert and O(n) to delete.

A new node has 1) Data 2) Priority 3) Address of next node.

Array :
1. Separate arrays are used for each priority number.
2. Every queue has its front and rear pointers
3. Two dimensional arrays are used for these purpose
4. Each queue is implemented using circular arrays/queues

Date :
Page :

Doubly Ended :

→ Also known as head-tail Linked List
→ NO insertion/deletion from middle.
→ In computer's memory, DEqueue is implemented using either circular array/circular doubly L.L.
→ In ~~De~~ DEqueues, two pointers are maintained
- LEFT
- RIGHT

- Input Restricted
1. Insertions can only be done at one end.
2. Deletions from either end is allowed.

- Output Restricted
1. Insertions from either end is allowed.
2. Deletions ~~from~~ ~~ei~~ can only be done at one end.

Applications of Queue:

→ Waiting List for single shared resource like printer, disk, CPU.
→ Playlist for jukebox to add songs to the end, and ~~fr~~play from the front.
→ OS for handling interrupts.
→ To transfer data asynchronously between 2 processor
→ Josephus Problem.

WORLD STAR™
Date :
Page :

Linked List :

→ Linear Collection of data elements.
→ Can be used to implement other data structures.
→ Types : Singly Linked List.
Doubly Linked List.
Circular Linked List.

Implementation :

→ Each node contains 2 parts → 1) Data 2) Pointer to next node
→ The left part contains data (int data type, array, structure)
→ The right part contains pointer to next node.
→ The next/right part of last node stores a special value NULL.
→ HEAD/START stores the address of first node.
- Every node contains pointer to another node of same type, therefore, it is also called a self-referential data type.

✱ Singly Linked List :

→ Simple type of Linked List.
→ All nodes linked in sequential manner, Linear Linked List.
One way chain. [Has a beginning and end.].
→ Allows traversal in of data only in one way.

Operations:

1. Creation 2. Insertion 3. Deletion 4. Traversal
5. Searching.

★ Circular Linked List:

→ The next/right part of the last node points the first node of the Linked List.
→ Rest is same as Singly Linked List.
→ The last node pointing the first node gives access to the preceding elements in the list.
→ Types:
- Circular Singly Linked List
- Circular Doubly Linked List
→ Has no beginning or end.
→ Widely used in OS for task maintainance.

★ Doubly Linked List:

→ Requires more space per node and more expensive basic operations.
→ Maintains 2 pointers and data.
→ 2 pointers: 1) Pointer to the previous node.
2) Pointer to the next node
→ Searching becomes twice as efficient.

→ Labelled/Weighted Graph: Every edge in the graph is assigned some data.
Denoted by w(e)
In weighted graph, the data represents some weight/length.

→ Multi-Graph : A graph with multiple edges and/or loops is called multigraph.

→ Complete Graph : Every vertex is directly connected to every other vertex.
If n is number of vertices, for complete graph, number of edges = $\frac{n(n-1)}{2}$

Applications :
→ Driving distance/time map.
→ Street Map.
→ Communication Network.

* Directed Graph:
→ Out-degree ⇒ [outdeg(u)] Number of ~~egd~~ edges that originate at u.

→ In-degree ⇒ [indeg(u)] Number of edges that terminate at u

* Representation of Graphs: 1) Adjacency Matrix.
2) Adjacency List/Multi-List

→ Labelled/Weighted Graph: Every edge in the graph is assigned some data.
Denoted by w(e)
In weighted graph, the data represents some weight/length

→ Multi-Graph : A graph with multiple edges and/or loops is called multigraph.

→ Complete Graph : Every vertex is directly connected to every other vertex.
If n is number of vertices, for complete graph, number of edges = $\frac{n(n-1)}{2}$

Applications :
→ Driving distance/time map.
→ Street Map.
→ Communication Network

* Directed Graph:
→ Out-degree ⇒ [outdeg(u)] Number of edges that originate at u.

→ In-degree ⇒ [indeg(u)] Number of edges that terminate at u

* Representation of Graphs: 1) Adjacency Matrix
2) Adjacency List/Multi-List

☆ Adjacency Matrix : [Bit matrix / Boolean matrix].

→ For a graph having n nodes, adjacency matrix will be a ~~of n*n (n²)~~ (n×n) matrix.

- The entry aij will be set to 1 if the vertices Vi and Vj are adjacent to each other.
- Else the entry aij will be set to 0.
- Adjacency matrix depends on ordering of nodes.
- Change in ordering will result in a different adjacency matrix.

→ A simple graph (that has no loops), the adjacency matrix will have 0's on diagonal.
→ For undirected graph, matrix is symmetric.
→ Memory use : $O(n^2)$
→ Number of 1's in adjacency matrix = Number of edges in the graph.

For weighted graph : The adjacency matrix has the weights of edges ~~insted~~ instead of 1's.

☆ Adjacency List :

→ Structure that contains list of all nodes.
→ Every node is linked to its own list that is a list of all its neighbours.
→

Advantages :

1) Easy to follow and clearly shows neighbours of a node.
2) Better to store graphs having a small size/moderate size.
3) Adding new nodes is easier compared to adjacency matrix.

→ Sum of lengths of all adjacency lists of a graph
= Number of edges [Directed Graph]
= 2 × Number of edges [Non-Directed Graph]

* Traversal Algorithms:
→ Breadth First Search. (BFS) [Uses queue]
→ Depth First Search. (DFS) [Uses Stack]

* BFS
→ Start by putting any one of the vertex at rear end of queue. [If first element front and rear is same]
→ Take ~~&~~ Front element of queue and add it to visited list.
→ Create a list of that vertex's adjacent nodes. Add the ones not present in visited list to the rear end of queue
→ Repeat above steps until queue is empty.

Time Complexity : O(V+E)
Space Complexity : O(V)

Applications:
→ To build index by search index.
→ For GPS navigation.
→ Path Finding Algorithms.
→ Cycle ~~detected~~ detection in undirected graphs
→ In minimum spanning tree.
→ Ford-Fulkerson algorithm, to find maximum flow in a network.

WORLD STAR™
Date :
Page :

DFS :

→ Start by putting any one of the vertex to top of stack.
→ Take the top element and add it to visited list.
→ Create a list of that vertex's adjacent nodes. Add the ones not present in the visited list to the top of stack.
→ Repeat above steps until stack is empty.

Time Complexity : O(V+E)
Space Complexity : O(V)

Applications :

→ For finding a path
→ To test if the graph is bipartite.
→ For finding the strongly connected components of the graph
→ For detecting cycles in a graph.