

Data Structures

Adrien Tremblay

Data Structure(DS)

A **data structure** is a way of organizing data so that it can be used effectively.

- Essential ingredient to creating.
- Help manage and organize data.
- Make code cleaner and easier to understand.
- Good coders know what data structure fits every given scenario.

Abstract Data Type (ADT)

An **Abstract Data Type** is an abstraction of a data structure which provides only the interface to which a data structure must adhere to.

- Language/implementation non-specific.

Examples:

Abstraction (ADT)	Implementation (DS)
List	Dynamic Array, Linked List
Queue	Linked List based queue, Array based queue, stack based queue
Map	tree Map, Hash Map
Vehicle	Golf Cart, Bike, Smart Car

Dynamic and Static arrays

Static Array

A **Static Array** is a fixed length container containing n elements **indexable** from range $[0, n-1]$

- Contiguous in memory (next to each other).
- Fixed size (cannot grow or shrink in size).

Used in

- Storing sequential data.
- Temporarily storing data.
- Buffers.
- In lookup tables.
- A lot of other places>

Complexity

Action	Static Array	Dynamic Array
Access	$O(1)$	$O(1)$
Search	$O(n)$	$O(n)$
Insertion	N/A	$O(n)$
Appending	N/A	$O(1)$
Deletion	N/A	$O(n)$

Dynamic Arrays

- Can grow or shrink in size
- AKA Arraylists
- Typically implemented using a static array.
 - Create static array with non-zero size.
 - Add elements, keeping track of size.
 - If size goes over limit, create a new array of double the size and copy all elements over.

Linked Lists

A **Linked List** is a sequential list of nodes that hold data which point to other nodes.

- The last node always points to null.
- **Head**: first element
- **Tail**: last element

Used in

- Implementation of List, Queue, and Stack.
- Creating circular lists.
- Model real world things like trains.
- In hashtables, graphs.

Singly Linked vs Doubly Linked

- Singly only contains reference to the next node.
- Doubly also contains reference to the previous node.

Complexity

Action	Singly Linked	Doubly Linked
Search	$O(n)$	$O(n)$
Insert at head	$O(1)$	$O(1)$
Insert at tail	$O(1)$	$O(1)$
Remove at head	$O(1)$	$O(1)$
Remove at tail	$O(1)$	$O(1)$
Remove in middle	$O(n)$	$O(n)$

Stack

A stack is a one-ended linear data structure which models a real world stack by having two primary operations: *push*, *pop*.

- There is a pointer to the top element called *Top*.
- *LIFO*: Last in first out.

Uses

- Undo mechanisms for text editors.
- Used to match brackets.
- Model piles of stuff.
- Function stack.
- DFS on a graph.

Complexity

Action	Complexity
Pushing	$O(1)$
Popping	$O(1)$
Peeking	$O(1)$
Searching	$O(n)$
Size	$O(1)$

Queue

A linear data structure that is *FILA*: First in - Last out.

- Models a real world queue.
- Two primary operations: *Enqueue and Dequeue*
 - Insert through back, remove from front

Uses

- Modelling real world queues.
- Keep track of x most recent elements.
- Web server management.
- BFS graph traversal.

Complexity

Action	Complexity
Enqueue	$O(1)$
Dequeue	$O(1)$
Peeking	$O(1)$
Contains	$O(n)$
Removal	$O(n)$
isEmpty	$O(1)$

Priority Queues

A *Priority Queue* is an ADT similar to a queue but each item has a priority. This priority determines the order in which elements are removed from the queue. Note: thusly PQs only support comparable data.

What is a heap?

A heap is a DS that satisfies the heap invariant. If A is a parent node of B then A is ordered with respect to B for all nodes A,B in the heap.

- There are two types:
 - *min heaps* where the root is the smallest
 - *max heaps* where the root is the biggest
- Heaps form the canonical underlying DS of a PQ.

Where are PQs used

- Dijkstra's shortest path algorithm
- Anytime you need to fetch the next best or next worst element
- BFS algos
- Minimum Spanning Tree (MST) algos

Complexity

Action	Complexity
Binary Heap Construction	$O(n)$
Polling	$O(\log(n))$
Peeking	$O(1)$
Adding	$O(\log(n))$