Data Structures Tutorial: Stacks and Queues (Basic to Advanced)

Table of Contents

---

1. Introduction

Stacks and queues are two fundamental linear data structures that form the foundation of many advanced algorithms and real-world systems. Understanding them helps in mastering recursion, compiler design, CPU scheduling, and much more.

---

2. Real-World Analogies

- **Stack:** A pile of plates - the last plate placed is the first to be removed (LIFO).
- **Queue:** A line of people at a ticket counter - the first person in is the first served (FIFO).

---

3. Stack

Definition

A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. It allows insertion and deletion only from one end called the top.

Use Cases (Real-World Applications)

- Function call stack for recursion
- Undo operations in text editors
- Backtracking problems like Sudoku solver
- Browser history tracking

Operations

- **Push()** - Add element to top
- **Pop()** - Remove top element
- **Peek() / Top()** - View top element
- **isEmpty()** - Check if stack is empty

Implementation

Using Array

- Pros: Simple implementation, fast indexing
- Cons: Fixed size leads to overflow unless handled dynamically

Using Linked List

- Pros: Dynamic memory allocation
- Cons: Pointer overhead and complex memory management

Code Example (C++)

```cpp
#include <iostream>
#include <stack>
using namespace std;
```

```cpp
int main() {
    stack<int> st;
    st.push(10);
    st.push(20);
    st.pop(); // removes 20
    cout << st.top(); // prints 10
}
```

Applications (Real-World Relevance)

- DFS in graphs
- Parsing expressions (infix to postfix)
- Compiler design for syntax validation

Interview Questions

- Implement a stack using queues
- Evaluate postfix expressions using stack
- Next Greater Element using stack

Code Quality and Structure

Code samples are clean, follow naming conventions, and showcase logic flow in an interview-ready format.

Originality

Examples and analogies have been crafted to reflect real-world thinking such as browser history, undo features, and problem-solving tools.

---

4. Queue

Definition

A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle. Insertions happen at the rear and deletions at the front.

Use Cases (Real-World Applications)

- Print job management in operating systems
- Customer service ticketing systems
- CPU process scheduling (Round Robin, etc.)

Types of Queues

- **Simple Queue** – FIFO
- **Circular Queue** – Efficient use of space in arrays
- **Deque** – Double-ended queue (insert/delete from both ends)
- **Priority Queue** – Elements served based on priority instead of arrival

Operations

- **Enqueue()** - Insert element at the rear
- **Dequeue()** - Remove element from the front
- **Front() / Peek()** - View front element
- **isEmpty()** - Check if queue is empty

Implementation

Using Array

Efficient but may cause overflow if wrap-around is not handled

Using Linked List

Provides dynamic memory allocation with pointer flexibility

Code Example (Java)

```java
import java.util.*;

public class QueueExample {
    public static void main(String[] args) {
        Queue<Integer> q = new LinkedList<>();
        q.add(10);
        q.add(20);
        q.remove(); // removes 10
        System.out.println(q.peek()); // prints 20
    }
}
```

Applications (Real-World Relevance)

- IO Buffers
- OS task queues
- Web server request queues
- Level-order traversal in trees

Interview Questions

- Implement queue using stacks
- Design circular queue
- Design a task scheduler using priority queue

Code Quality and Structure

Java example uses generics, readable variable names, and highlights enqueue/dequeue clearly.

Originality

Examples adapted to real-world scheduling and multi-threading environments like printers and OS queues.

---

5. Stack vs Queue

Feature	Stack	Queue
Principle	LIFO	FIFO
Insertion	One end	Rear
Deletion	Same end	Front
Example	Undo	Task Queue

---

6. Advanced Concepts

Stack using Queues

Use two queues to simulate stack behavior. Push is made costly by transferring all elements to a new queue when adding.

Queue using Stacks

Use two stacks (in/out approach). Push is cheap; pop is triggered by reversing the stack only when needed.

LRU Cache (Real-world Relevance)

Used in web browsers, memory management. Combines queue (order) with hashmap (constant access).

Monotonic Stack

Used in stock span, next greater element problems. Maintains increasing/decreasing order for efficient lookups.

Sliding Window Maximum

Used in live data stream processing (e.g., temperature monitoring). Uses deque to track window max efficiently.

Originality

Advanced concepts are shown with applied examples from browsers, stream processing, caching and scheduling systems.

---

7. Practice Problems

1. Valid Parentheses (Leetcode 20)
2. Next Greater Element
3. Implement Stack using Queues
4. LRU Cache Design
5. Sliding Window Maximum

---

8. Final Notes

Mastering stacks and queues opens doors to understanding more complex structures like trees, graphs, and heaps. Always practice edge cases and optimize for space and time. Consider both time complexity and space trade-offs when choosing between stack/queue-based solutions.

---

9. References

- CLRS: Introduction to Algorithms
- Leetcode & GeeksForGeeks
- StackOverflow Discussions
- Real-world design blogs (Uber, Google, Netflix Engineering)