Stacks and Queues

1 Stacks

A **stack** is a Last-In-First-Out (LIFO) data structure. Elements are inserted and removed from the same end, called the *top*.

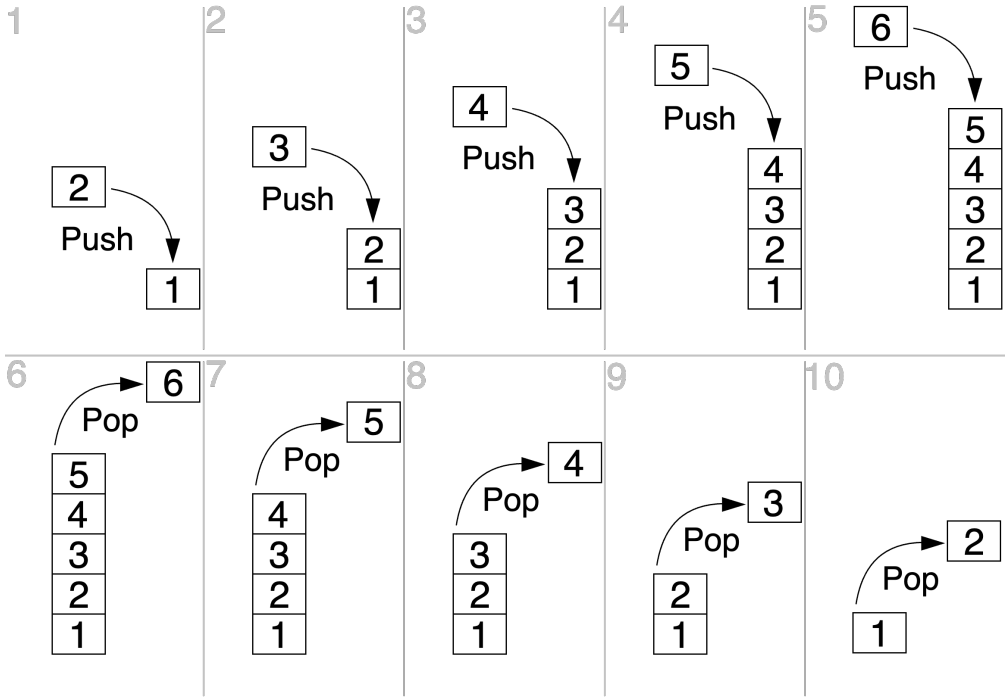

Figure 1: Stack operation (from Wikimedia)

Linked List Implementation: Each node points to the next element. Insertion and deletion happen at the head of the list.

```
struct Node {
  int data;
  Node* next;
};

class Stack {
  Node* top;
public:
  Stack() : top(nullptr) {}
  void push(int x) {
    Node* n = new Node{x, top};
    top = n;
  }
  void pop() {
    if (top) {
      Node* temp = top;
```

```
      top = top->next;
      delete temp;
    }
  }
  int peek() const { return top->data; }
  bool empty() const { return top == nullptr; }
};
```

STL: `std::stack` is implemented using `std::deque` by default. It supports `push()`, `pop()`, `top()`, and `empty()`.

- **Read:** $\mathcal{O}(1)$ for top element
- **Insert:** $\mathcal{O}(1)$ at head
- **Delete:** $\mathcal{O}(1)$ at head
- **Update:** $\mathcal{O}(1)$ if on top; $\mathcal{O}(n)$ otherwise

2 Queues

A **queue** is a First-In-First-Out (FIFO) data structure. Insertion occurs at the *rear*, deletion at the *front*.

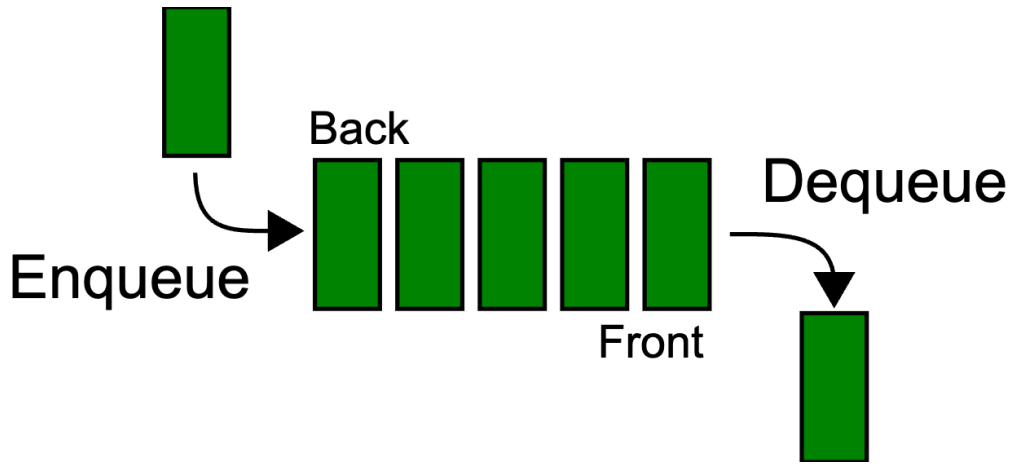

Figure 2: Queue operation (from Wikimedia)

Linked List Implementation: We maintain both front and rear pointers.

```
struct Node {
  int data;
  Node* next;
};

class Queue {
  Node* front;
  Node* rear;
public:
  Queue() : front(nullptr), rear(nullptr) {}
  void enqueue(int x) {
    Node* n = new Node{x, nullptr};
    if (rear) rear->next = n;
    else front = n;
    rear = n;
  }
  void dequeue() {
    if (front) {
      Node* temp = front;
```

```
      front = front->next;
      if (!front) rear = nullptr;
      delete temp;
    }
  }
  int peek() const { return front->data; }
  bool empty() const { return front == nullptr; }
};
```

STL: `std::queue` uses `std::deque` internally. `std::deque` (double-ended queue) supports insertion/removal at both ends in $\mathcal{O}(1)$ time.

- **Read:** $\mathcal{O}(1)$ at front
- **Insert:** $\mathcal{O}(1)$ at rear
- **Delete:** $\mathcal{O}(1)$ at front
- **Update:** $\mathcal{O}(n)$

3 Note on `std::deque`

A `deque` (double-ended queue) allows fast insertions/removals at both front and back. STL uses it as the default underlying container for both `std::stack` and `std::queue` for performance.

4 Practice problem

Leetcode Problem 20: *Valid paranthesis*
`https://leetcode.com/problems/valid-parentheses/description/`