Techno Main Salt Lake

Name: Shreyash Lal

University Roll: 13030824012

Paper Name: Data Structures And Algorithms

Paper Code: PCCCS301

Dept: CSE (AI & ML)

Sem: 3rd

Stacks: Operations and Applications

Demystifying a Fundamental Data Structure

What is a Stack?

A stack is a fundamental linear data structure that follows a specific order in which operations are performed. Think of it like a pile of plates: you can only add a new plate to the top, and you can only remove the topmost plate.

LIFO Principle

Last In First Out

This means the element added most recently is the first one to be removed.

Simple Visual

Visualizing the stack as a vertical structure helps in understanding its behavior.

Core Stack Operations

Stacks have a limited set of operations, ensuring their integrity and adherence to the LIFO principle.

1

Push

Adds an element to the top of the stack.

`stack.push(item)`

2

Pop

Removes and returns the element from the top of the stack.

`stack.pop()`

3

Peek / Top

Returns the top element without removing it.

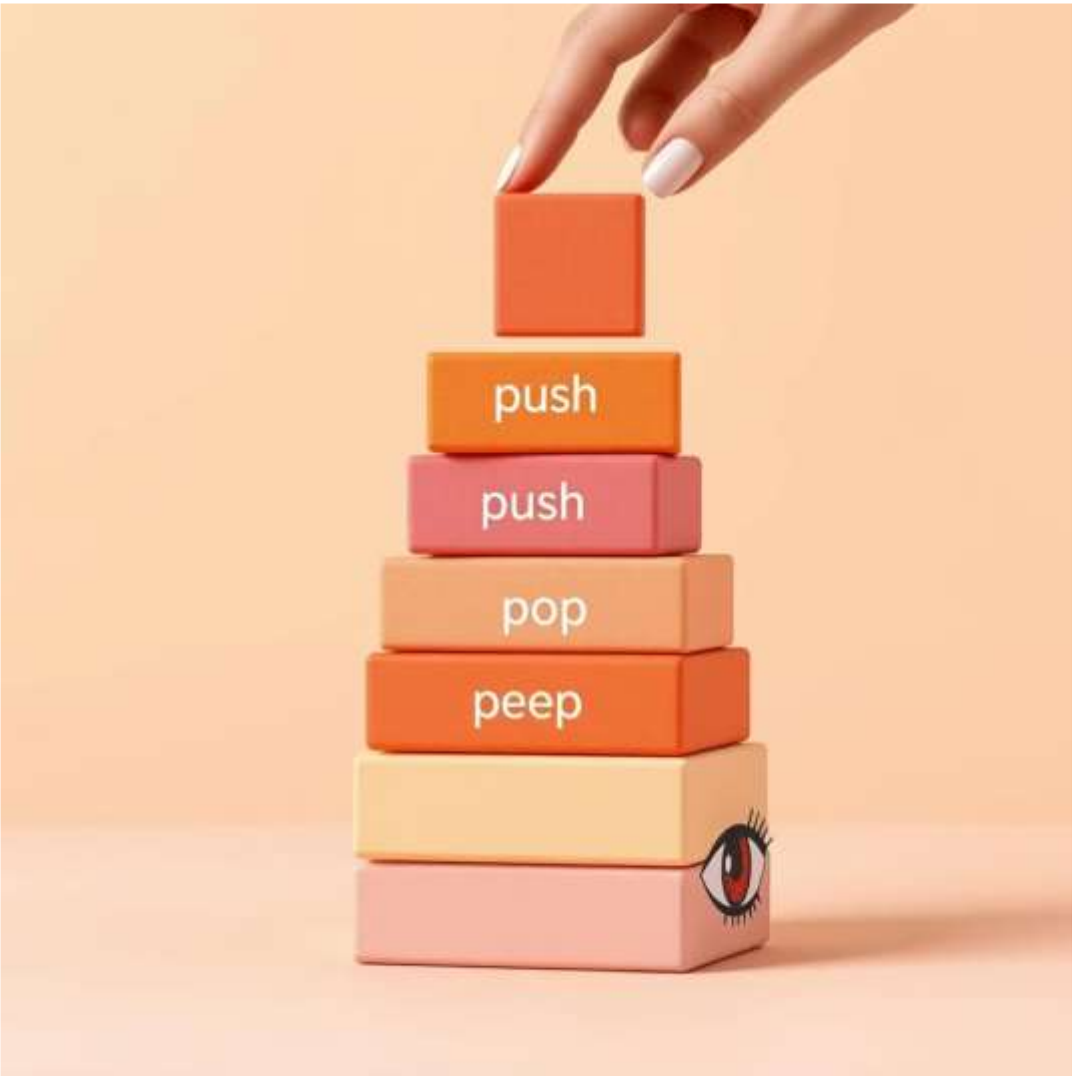
`stack.peek()`

4

isEmpty

Checks if the stack contains any elements (returns true or false).

`stack.isEmpty()`



Implementing Stacks

Stacks can be implemented using various underlying data structures, each with its own performance characteristics.

Using Arrays

A stack can be implemented using a dynamic array. Elements are added and removed from one end (the "top").

- Push: O(1) (amortized)
- Pop: O(1)

Using Linked Lists

Each element is a node, and the "top" of the stack is the head of the linked list.

- Push: O(1)
- Pop: O(1)

Both implementations offer efficient O(1) time complexity for core operations, making stacks a fast choice for many applications.

Real-World Applications

Stacks are integral to many computing processes, from everyday software to complex algorithms.

Expression Evaluation

Used in compilers and interpreters for evaluating arithmetic expressions (e.g., converting infix to postfix notation).

Undo/Redo Functionality

Many applications use stacks to store previous states, allowing users to undo or redo actions.

Backtracking

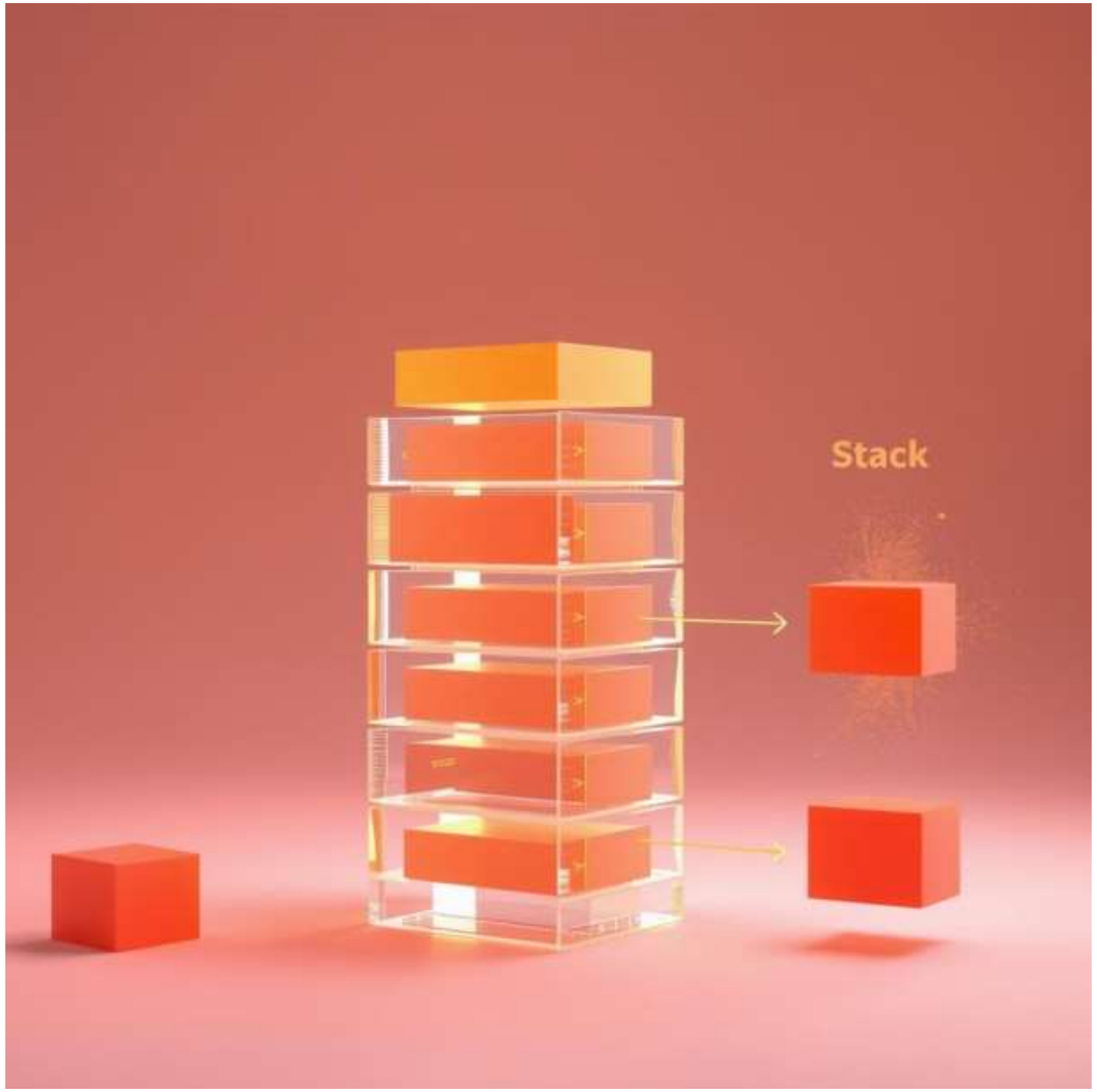
Algorithms that explore multiple paths (like solving mazes or finding solutions in games) use stacks to keep track of previous choices.

Recursion

Recursive function calls implicitly use a call stack to manage function contexts and return addresses.

Problem: Valid Parentheses

How do we check if a string of parentheses is valid (e.g., `"()[]{}"` is valid, but `"([)]"` is not)?



The Stack Solution

1. Iterate through the string.
2. If an opening bracket (`(, [, {`) is found, push it onto the stack.
3. If a closing bracket (`), ], }`) is found:
   - Check if the stack is empty or if the top element doesn't match the corresponding opening bracket.
   - If they match, pop the top element.
   - Otherwise, the string is invalid.
4. After iterating, if the stack is empty, the string is valid.

Key Takeaways

Stacks are intuitive yet powerful. Understanding their behavior is crucial for efficient algorithm design.

1. LIFO is the defining principle.
2. Push, Pop, and Peek are core operations.
3. Efficiently implemented with arrays or linked lists.
4. Versatile in programming and real-world applications.

Further Reading & Resources

- Textbooks:
  - Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
  - Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in Java* (6th ed.). Wiley.
- Online Resources:
  - GeeksforGeeks - Stack Data Structure
  - Programiz - Stack Data Structure
- Research Papers:
  - Knuth, D. E. (1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms* (3rd ed.). Addison-Wesley Professional.

Thank You!