I Wish I Knew These 12 Algorithms and Their Applications Before the System Design Interview

levelup.gitconnected.com/i-wish-i-knew-these-12-algorithms-and-their-applications-before-the-system-design-interview-5fb7fa8b1177

How to Master Algorithmic Patterns for System Design Interviews?

Photo by frank mckenna on Unsplash

As software engineers, we often find ourselves facing complex system design problems that test our ability to identify and apply the right algorithmic patterns to solve intricate problems. These patterns, such as sliding window, two pointers, two heaps, and many more, are the building blocks of modern software systems. However, understanding how these patterns work and when to use them is a challenge that even experienced engineers struggle with.

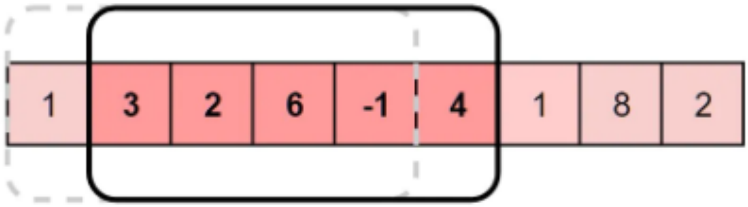
In this blog, I will share my experiences to help you map algorithmic patterns to famous system design questions. We will delve into the intricacies of various coding patterns, explore real-life examples where they can be applied, and reveal how these patterns are the secret sauce behind efficient, scalable, and reliable software systems.

So, whether you're an aspiring software engineer looking to break into the industry or a seasoned professional seeking to expand your skill set, this blog is your ultimate guide to algorithmic patterns and their relevance in system design interviews.

Let's start to level up your design skills.

1. Sliding Window

The sliding window pattern is a technique used for solving problems that involve iterating through an array or a list while keeping track of a "window" of elements. This window either has a fixed size or a variable size that can be expanded or contracted based on specific conditions. The primary goal of the sliding window pattern is to find an optimal solution by minimizing or maximizing a certain property of the window, such as the sum of its elements or the number of unique elements.



Sliding Window Pattern

System Design Examples:

- Video and audio streaming applications, where a sliding window approach is used to buffer and control the playback quality based on network conditions.
- Data compression algorithms, like the Lempel-Ziv-Welch (LZW) algorithm, which use sliding windows to identify repeating patterns for more efficient encoding.
- Analyzing network traffic patterns to detect anomalies or potential DDoS attacks.
- Text search algorithms, such as the Boyer-Moore algorithm, which use a sliding window approach to efficiently search for substrings within a larger text.

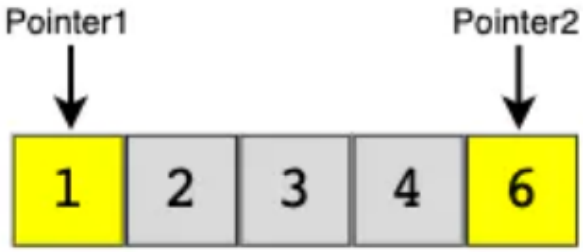
2. Two Pointers

The two-pointer pattern is a technique that involves using two pointers or iterators to traverse a data structure like an array, a list, or a string. This pattern can be used to solve problems that require comparing or processing pairs of elements in the data structure. The two pointers can move in the same or opposite directions, and their movement is often governed by specific conditions or constraints.

System Design Examples:

- Binary search algorithm implementation, where two pointers are used to search for a target value in a sorted array by progressively narrowing the search space.

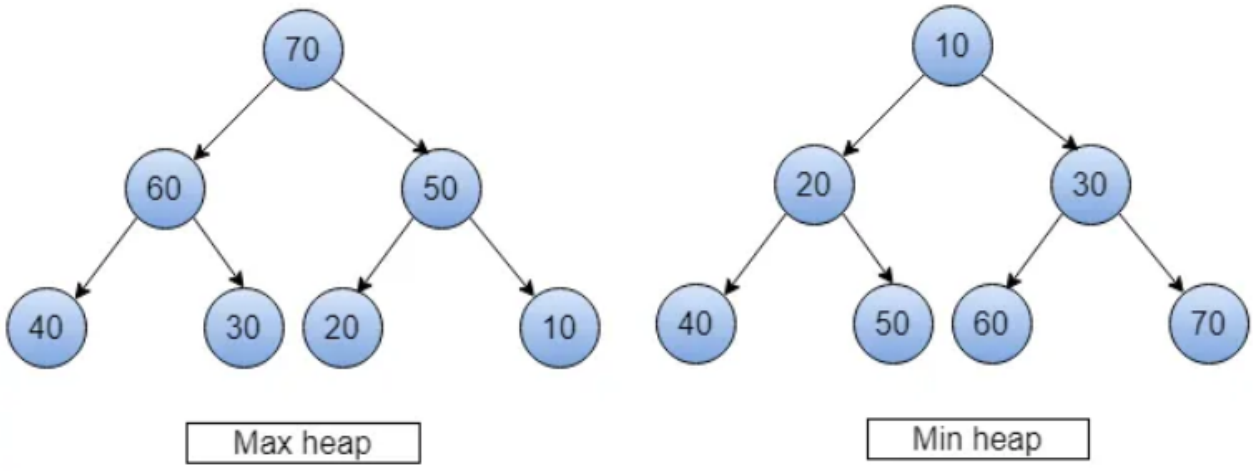
- Comparing DNA sequences in bioinformatics applications, where two pointers are used to align and compare two sequences.
- Merge step in the merge-sort algorithm, where two pointers are used to traverse and merge two sorted arrays into a single sorted array.
- Search algorithms for detecting palindromes in a string, where two pointers are used to compare characters from both ends of the string.



Two Pointer Pattern

3. Two Heaps

The Two-heaps pattern is a technique used for solving problems that involve finding the median or maintaining a balance between two sets of elements. In this pattern, two heaps (a min-heap and a max-heap) are used to divide the dataset into two equal-sized or nearly equal-sized parts. The max-heap stores the smaller half of the elements, while the min-heap stores the larger half. By maintaining this balanced partition, the median or other desired properties can be efficiently computed or maintained.



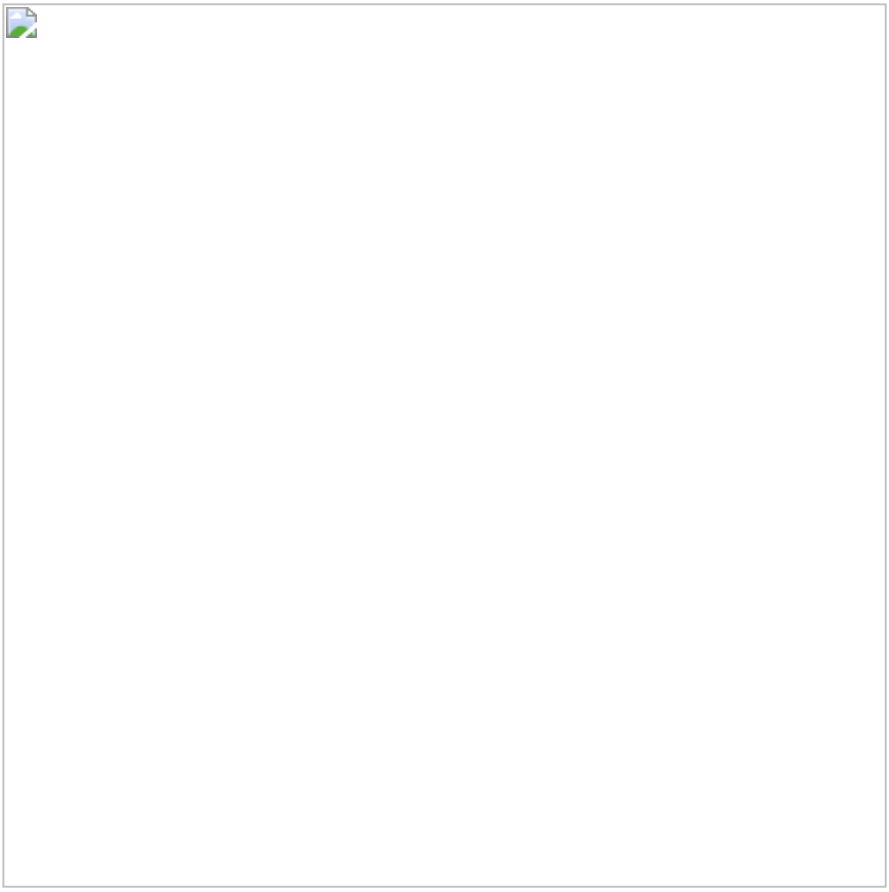
Two Heaps Pattern

System Design Examples:

- Online analytical processing (OLAP) systems, where the two-heaps pattern can be used to maintain percentiles or other summary statistics over a sliding window of data.
- Stream processing systems, where the two-heaps pattern can be used to compute and maintain the median value of a continuous stream of data.
- Resource allocation and scheduling algorithms in operating systems or cloud computing platforms, where the two-heaps pattern can be used to balance workloads between different resources.
- Real-time monitoring and anomaly detection systems, where the two-heaps pattern can be used to maintain summary statistics and identify unusual patterns in the data.

4. Backtracking

Backtracking is a general algorithm for finding all (or some) solutions to a problem that incrementally builds candidates to the solutions, and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot be extended to a valid solution. This technique is often used in constraint satisfaction problems, where the goal is to find a solution that satisfies a set of constraints.



System Design Examples:

- Graph coloring problems, such as assigning frequencies to radio stations in a way that no two adjacent stations share the same frequency.
- Job scheduling and resource allocation problems, where backtracking can be used to find an optimal sequence of tasks that satisfies various constraints, such as deadlines or resource availability.
- The traveling salesman problem, where a salesman has to visit a number of cities and return to the starting city with the shortest possible route. Backtracking can be employed to explore all possible routes and find the most optimal one.
- The Sudoku puzzle, which is a logic-based, combinatorial number-placement puzzle. Backtracking can be used to find a solution by iteratively filling in the cells with appropriate numbers and backtracking when a dead-end is reached.

5. Breadth-First Search (BFS)

Breadth-First Search (BFS) is a graph traversal algorithm that explores all the vertices of a graph in breadth-first order, i.e., it visits all the vertices at the same level before moving on to the next level. BFS uses a queue data structure to keep track of the vertices to be visited and processes them in the order they were added to the queue.

System Design Examples:

- Social networks: BFS can be used to find the shortest path between two users in a social network, which may represent the degree of separation or the number of mutual connections between them.
- Web crawlers: Search engines like Google use BFS-based algorithms to crawl and index web pages, starting from a seed set of URLs and following the hyperlinks in breadth-first order.
- GPS navigation: BFS can be employed in finding the shortest route between two locations on a map, considering various constraints like road types, traffic conditions, and distance.
- Network broadcasting: In computer networks, BFS can be used to implement broadcasting, where a message is sent to all nodes in the network, ensuring that each node receives the message exactly once.

6. Depth-First Search (DFS)

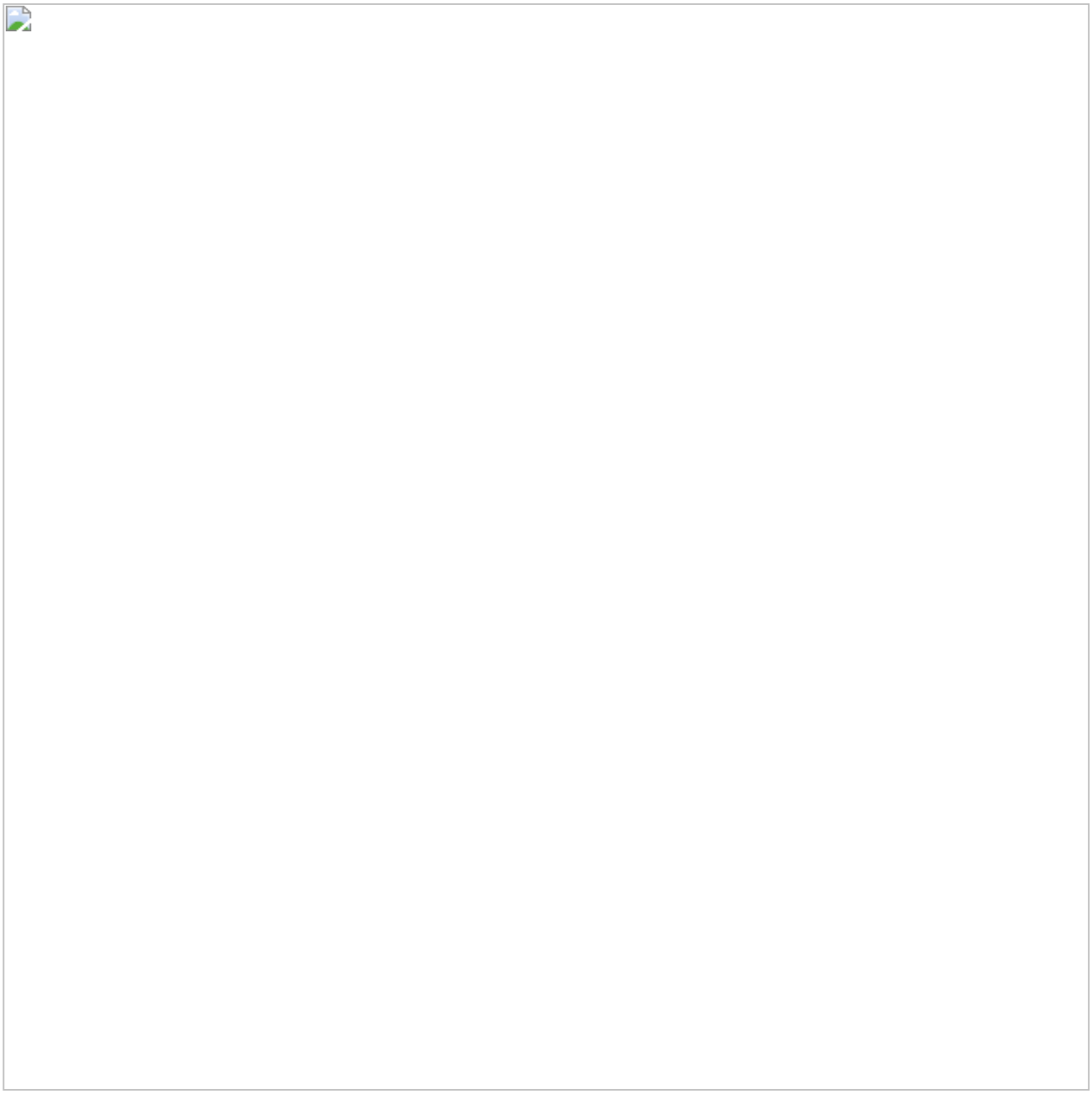
Depth-First Search (DFS) is another graph traversal algorithm that explores the vertices of a graph in depth-first order. It visits a vertex and then recursively visits all its adjacent vertices before backtracking. DFS uses a stack data structure, either explicitly or implicitly through recursion, to keep track of the vertices being visited.

System Design Examples:

- Maze solving: DFS can be used to find a path from the starting point to the goal in a maze by exploring all possible paths in a depth-first manner and backtracking when a dead-end is encountered.
- Dependency resolution: In build systems and package managers, DFS can be used to resolve dependencies between components or packages, ensuring that each dependency is built or installed before the dependent component.
- Topological sorting: DFS can be employed to perform topological sorting of a directed acyclic graph (DAG), which is useful in scheduling tasks with precedence constraints, such as project management or build systems.
- Finding connected components: In image processing, DFS can be used to identify connected components or regions of an image with similar properties, such as color or intensity.

7. Topological Sort

Topological Sort is an algorithm for linearly ordering the vertices of a directed acyclic graph (DAG) such that for every directed edge (u, v), vertex u comes before vertex v in the ordering. Topological sorting is possible if and only if the graph has no directed cycles.



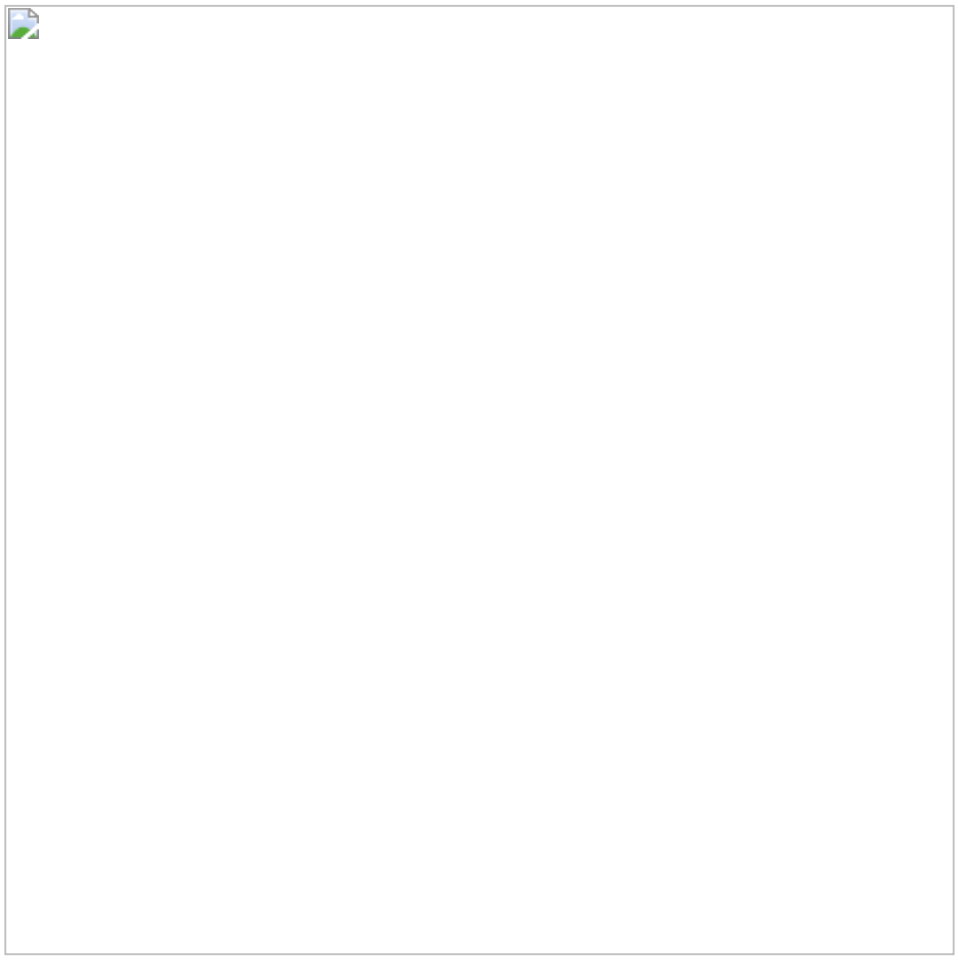
Topological Sort Patterb

System Design Examples:

- Project scheduling: Topological Sort can be used to determine the order in which tasks in a project should be executed, taking into account their dependencies.
- Course prerequisites: In an academic curriculum, Topological Sort can be employed to find a valid sequence of courses to be taken, considering their prerequisites.
- Build systems: In software build systems, Topological Sort can be utilized to determine the correct order of compilation for a set of files with dependencies.
- Package managers: Package managers can use Topological Sort to install packages and their dependencies in the correct order.

8. Merge Intervals

Merge Intervals is an algorithm that, given a collection of intervals, merges any overlapping intervals into a single interval. The problem can be solved by first sorting the intervals based on their starting points, and then iterating through the sorted intervals and merging them if they overlap.



Merge Interval Pattern

System Design Examples:

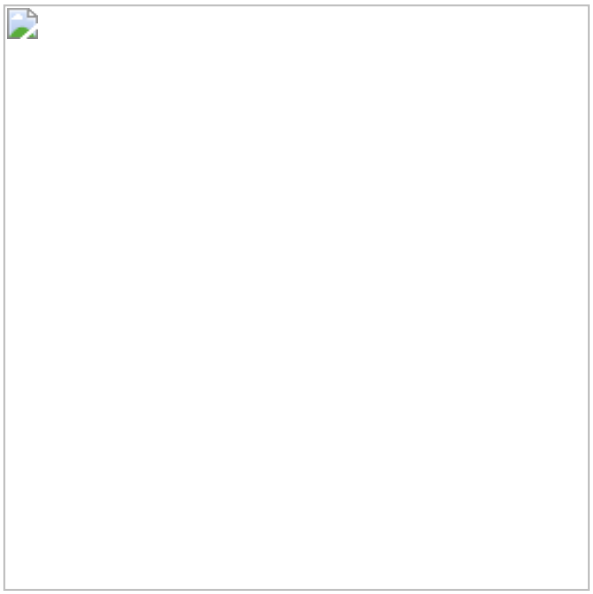
- Calendar scheduling: Merge Intervals can be used to find free time slots in a busy calendar by merging overlapping appointments or events.
- Resource allocation: In resource allocation problems, Merge Intervals can be employed to determine periods of high demand or overlapping usage.
- Network traffic analysis: Merge Intervals can be utilized to analyze network traffic data and identify periods of high traffic or congestion.
- Genome data processing: In bioinformatics, Merge Intervals can help identify overlapping gene sequences or analyze genetic data.

9. Trie (Prefix Tree)

A Trie, also known as a Prefix Tree, is a tree-like data structure that stores a dynamic set of strings. It is particularly useful for efficient searching, insertion, and deletion of strings in a large dataset. Each node in a Trie represents a character, and the path from the root to a node represents the corresponding prefix.

System Design Examples:

- Autocomplete suggestions: Tries are used in search engines and text editors to provide autocomplete suggestions based on the user's input.
- Spell checkers: Trie data structures can be employed in spell checkers to efficiently store and search a dictionary of words.
- IP routing: In computer networks, Tries can be used for IP routing to quickly find the longest matching prefix in the routing table.
- Bioinformatics: Tries can be utilized in bioinformatics applications for searching and matching DNA sequences.



Trie

10. Union-Find (Disjoint Set)

Union-Find, also known as Disjoint Set, is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets. It supports two main operations: Union, which merges two subsets into one, and Find, which determines the subset that an element belongs to.

System Design Examples:

- Network connectivity: Union-Find can be used to efficiently check and maintain connectivity between nodes in a network.
- Image segmentation: In image processing, Union-Find can be employed for segmenting an image into distinct regions based on similarity criteria.
- Kruskal's algorithm: Union-Find is a key component of Kruskal's algorithm for finding the minimum spanning tree of a connected, undirected graph.
- Percolation simulation: Union-Find can be utilized in simulating percolation in porous materials to determine whether a path exists between two points.

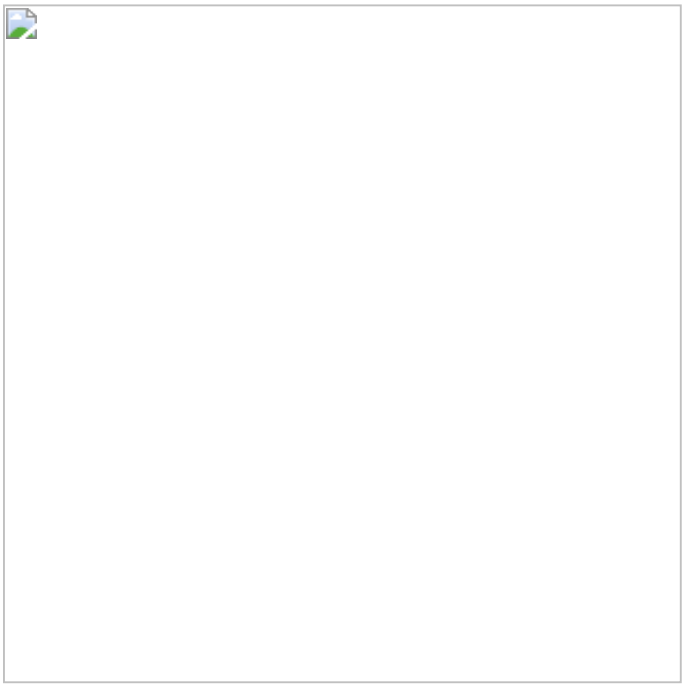
11. Flood Fill

Flood Fill is an algorithm used to determine the area connected to a given node in a multi-dimensional array, typically employed in image processing and computer graphics. The algorithm starts at a given node, replaces the target color with a new color, and recursively processes its neighboring nodes. The process continues until all connected nodes with the target color have been visited and updated.

System Design Examples:

- Image editing tools: Flood Fill is commonly used in image editing tools for the "bucket fill" or "paint fill" feature, which allows users to fill an area with a specific color.

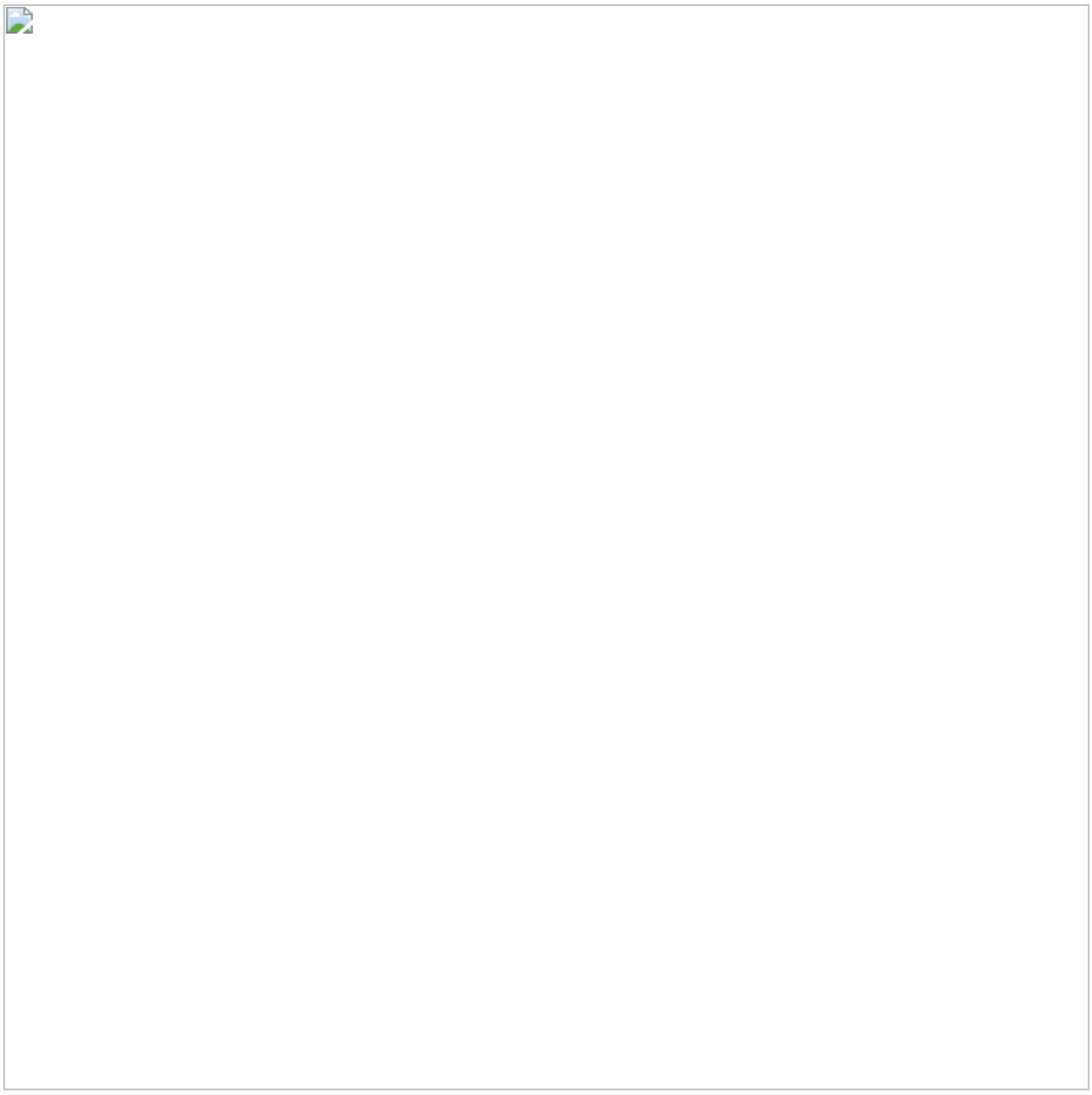
- Game development: In game development, Flood Fill can be employed to fill a region with a particular texture or to determine the boundaries of a region for gameplay purposes.
- Map generation: Flood Fill can be used in map generation algorithms to identify and fill connected regions or create islands in procedural terrain generation.
- Maze solving: Flood Fill can be utilized in maze-solving algorithms to explore and mark visited areas, allowing the solver to find a path to the exit.



Flood Fill Algorithm

12. Segment Tree

A Segment Tree is a tree-based data structure designed for efficient processing of range queries and updates on an array. It is particularly useful for problems that involve querying and updating array elements within specific ranges. Each node in the segment tree represents a range of elements in the array, and the tree is built in a hierarchical manner, with the leaves representing individual elements and the internal nodes representing the union of ranges of their children.



Segment Tree

System Design Examples:

- Range sum queries: Segment Trees can be used to efficiently calculate the sum of elements within a specified range in an array, such as the total income for a specific period in a financial application.
- Range minimum/maximum queries: In applications that require finding the minimum or maximum value within a range, Segment Trees can be employed for efficient querying and updating, such as in stock market analysis for identifying the highest or lowest stock prices within a time frame.
- Image processing: Segment Trees can be utilized in image processing tasks that involve processing and updating pixel values within specific regions, such as applying filters or adjusting brightness and contrast.
- Computational geometry: Segment Trees can be used in computational geometry problems, such as finding the intersection of line segments or calculating the area of overlapping rectangles within a specified range.

Conclusion

Mastering these patterns is crucial for every software engineer looking to excel in their career. By understanding the inner workings of various coding patterns such as sliding window, two pointers, two heaps, and many more, you can tackle complex problems with confidence and demonstrate your expertise during interviews.

Remember, practice is the key to success. Continuously refine your understanding of these patterns and their applications to hone your problem-solving skills. As you grow more adept at identifying and utilizing the appropriate algorithmic patterns, you'll not only improve your chances of acing system design interviews but also elevate your career as a software engineer.

Coding Interviews are getting harder to pass. To prepare for coding interviews, you will need weeks, if not months of…