PATTERNS TO IDENTIFY

IN CODING INTERVIEW QUESTIONS

SAVE AND SHARE

REFER FOR YOUR FUTURE INTERVIEWS

HIMANSHU MAHURI(LINKEDIN) https://www.linkedin.com/in/himanshukumarmahuri

1. Fast and Slow pointers-

How do you identify when to use the Fast and Slow pattern?

- The problem will deal with a loop in a linked list or array
- When you need to know the position of a certain element or the overall length of the linked list.

When should I use it over the Two Pointer method mentioned above?

- There are some cases where you shouldn't use the Two Pointer approach such as in a singly linked list where you can't move in a backwards direction. An example of when to use the Fast and Slow pattern is when you're trying to determine if a linked list is a palindrome.

Problems featuring the fast and slow pointers pattern:

- Linked List Cycle (easy)
- Palindrome Linked List (medium)
- Cycle in a Circular Array (hard)

2. Merge Intervals

How do you identify when to use the Merge Intervals pattern?

- If you're asked to produce a list with only mutually exclusive intervals
- If you hear the term "overlapping intervals".

Merge interval problem patterns:

- Intervals Intersection (medium)
- Maximum CPU Load (hard)

3. Two Pointers or Iterators

Ways to identify when to use the Two Pointer method:

- It will feature problems where you deal with sorted arrays (or Linked Lists) and need to find a set of elements that fulfill certain constraints
- The set of elements in the array is a pair, a triplet, or even a subarray

Here are some problems that feature the Two Pointer pattern:

- Squaring a sorted array (easy)
- Triplets that sum to zero (medium)
- Comparing strings that contain backspaces (medium)

4. SLIDING WINDOW-

Following are some ways you can identify that the given problem might require a sliding window:

- The problem input is a linear data structure such as a linked list, array, or string
- You're asked to find the longest/shortest substring, subarray, or a desired value

Common problems you use the sliding window pattern with:

- Maximum sum subarray of size 'K' (easy)
- Longest substring with 'K' distinct characters (medium)
- String anagrams (hard)

5. Cyclic sort

How do I identify this pattern?

- They will be problems involving a sorted array with numbers in a given range
- If the problem asks you to find the missing/duplicate/smallest number in an sorted/rotated array

Problems featuring cyclic sort pattern:

- Find the Missing Number (easy)
- Find the Smallest Missing Positive Number (medium)

6. Tree BFS

How to identify the Tree BFS pattern:

- If you're asked to traverse a tree in a level-by-level fashion (or level order traversal)

Problems featuring Tree BFS pattern:

Binary Tree Level Order Traversal (easy)

- Zigzag Traversal (medium)\

7. In-place reversal of linked list

How do I identify when to use this pattern:

- If you're asked to reverse a linked list without using extra memory

Problems featuring in-place reversal of linked list pattern:

- Reverse a Sub-list (medium)
- Reverse every K-element Sub-list (medium)

8. Top K elements

How to identify the Top 'K' Elements pattern:

- If you're asked to find the top/smallest/frequent 'K' elements of a given set
- If you're asked to sort an array to find an exact element

Problems featuring Top 'K' Elements pattern:

- Top 'K' Numbers (easy)
- Top 'K' Frequent Numbers (medium).

9. Tree DFS

How to identify the Tree DFS pattern:

- If you're asked to traverse a tree with in-order, preorder, or postorder DFS
- If the problem requires searching for something where the node is closer to a leaf

Problems featuring Tree DFS pattern:

- Sum of Path Numbers (medium)
- All Paths for a Sum (medium)

10. Modified binary search

The patterns looks like this for an ascending order set:

1. First, find the middle of start and end. An easy way to find the middle would be: middle = (start + end) / 2. But this has a good chance of producing an integer overflow so it's recommended that you represent the middle as: middle = start + (end — start) / 2
2. If the key is equal to the number at index middle then return middle
3. If 'key' isn't equal to the index middle:
4. Check if key < arr[middle]. If it is reduce your search to end = middle — 1
5. Check if key > arr[middle]. If it is reduce your search to end = middle + 1

11. Two heaps

Ways to identify the Two Heaps pattern:

- Useful in situations like Priority Queue, Scheduling
- If the problem states that you need to find the smallest/largest/median elements of a set
- Sometimes, useful in problems featuring a binary tree data structure

Problems featuring- Find the Median of a Number Stream (medium)

12. Subsets

How to identify the Subsets pattern:

- Problems where you need to find the combinations or permutations of a given set

Problems featuring Subsets pattern:

- Subsets With Duplicates (easy)
- String Permutations by changing case (medium)

13. K-way Merge

The pattern looks like this:

1. Insert the first element of each array in a Min Heap.
2. After this, take out the smallest (top) element from the heap and add it to the merged list.
3. After removing the smallest element from the heap, insert the next element of the same list into the heap.
4. Repeat steps 2 and 3 to populate the merged list in sorted order.

How to identify the K-way Merge pattern:

- The problem will feature sorted arrays, lists, or a matrix
- If the problem asks you to merge sorted lists, find the smallest element in a sorted list.

Problems featuring the K-way Merge pattern:

- Merge K Sorted Lists (medium)
- K Pairs with Largest Sums (Hard)

14. Topological sort

The pattern works like this:

1. **Initialization**
   a) Store the graph in adjacency lists by using a HashMap
   b) To find all sources, use a HashMap to keep the count of in-degreesBuild the graph and find in-degrees of all vertices
2. **Build the graph from the input and populate the in-degrees HashMap.**
3. **Find all sources**
   a) All vertices with '0' in-degrees will be sources and are stored in a Queue.
4. **Sort**
   a) For each source, do the following things:
   —i) Add it to the sorted list.
   — ii)Get all of its children from the graph.
   — iii)Decrement the in-degree of each child by 1.
   — iv)If a child's in-degree becomes '0', add it to the sources **Queue.**
   b) Repeat (a), until the source Queue is empty.

How to identify the Topological Sort pattern:

- The problem will deal with graphs that have no directed cycles
- If you're asked to update all objects in a sorted order
- If you have a class of objects that follow a particular order

Problems featuring the Topological Sort pattern:

- Task scheduling (medium)
- Minimum height of a tree (hard)

15. PRACTICE MORE AND MORE PROBLEM AND TRY TO IDENTIFY THE PROBLEMS.

SOURCE- INTERNET

HIMANSHU KUMAR (LINKEDIN)

https://www.linkedin.com/in/himanshukumarmahuri