Technical Interview Preparation in Summer (TIPS)

Lecture 5: Post-Coding Techniques & Question Patterns

Admin Stuff

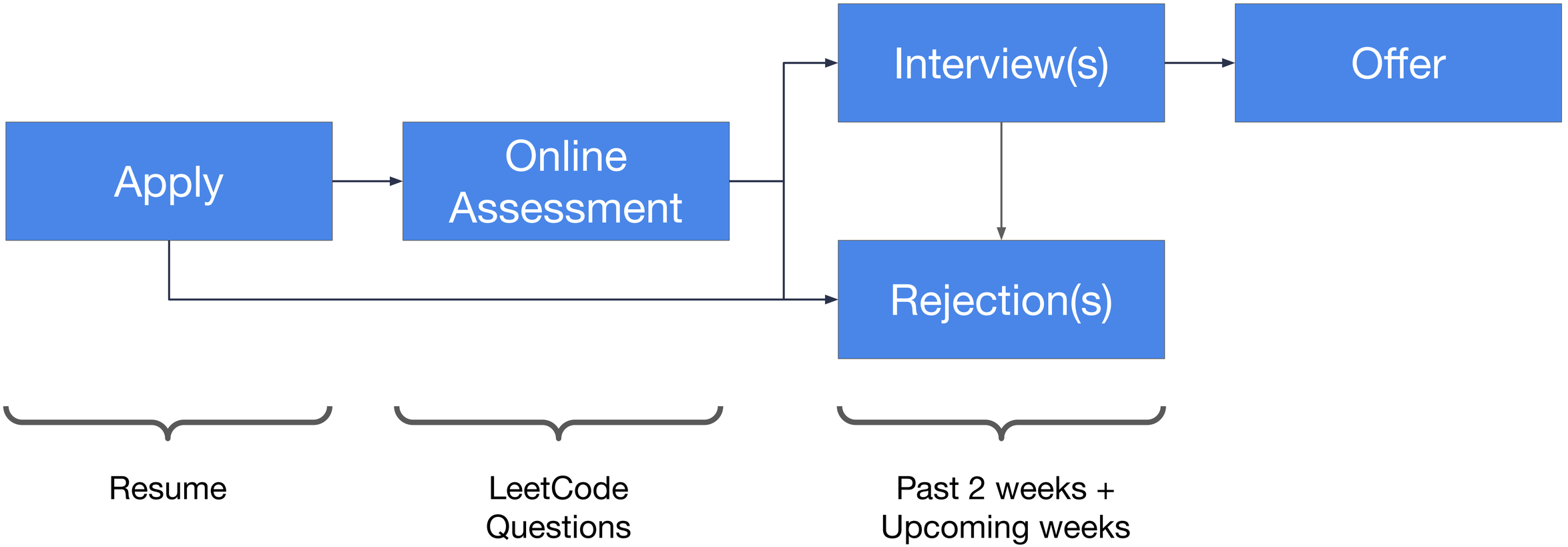
- Mid-course feedback survey will be out tomorrow
  - Responses are anonymous. Do give us some feedback to let us know how we/you are doing.
- Language specific workshops will be conducted next week.

Early Announcements

- Week 8 of TIPS will be on Tuesday 12 July instead of Monday because of Hari Raya Haji
- Week 7 (4 Jul) & Week 8 (12 Jul) - We will be inviting seniors and alumni respectively to share more about internships. These sessions will likely **not be recorded**.

Outline for Tonight

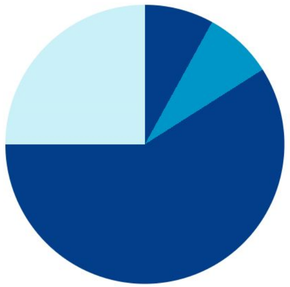
- Handling Mistakes + Testing
- Question Patterns
- Live Demo (Chun Mun & Rohit)


Apply
Online Assessment
Interview(s)
Offer
Rejection(s)
Resume
LeetCode Questions
Past 2 weeks + Upcoming weeks

Outline for Tonight

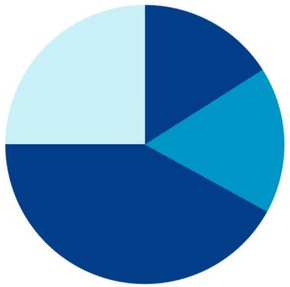
- Handling Mistakes + Testing
- Question Patterns
- Live Demo (Chun Mun & Rohit)

Overview of Techniques



The first 5…

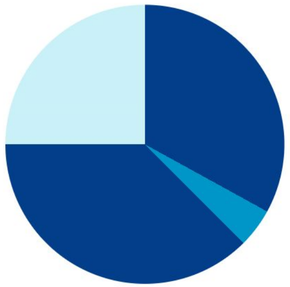
- Proactive Communication
- Designing Your Algorithm



The next 10…

- Maximising Collaborative Editors
- Talking As You’re Coding

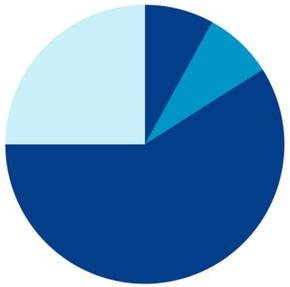
Past 2 weeks



The last 2-3…

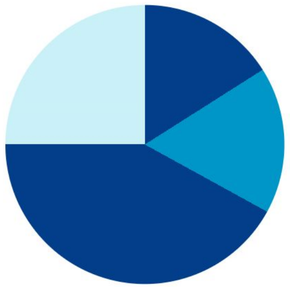
- Handling Mistakes
- Testing Your Code

Overview of Techniques



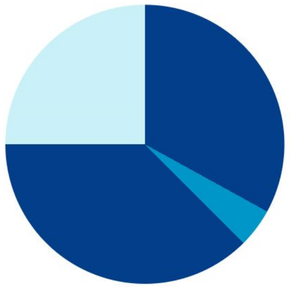
The first 5…

- Proactive Communication
- Designing Your Algorithm



The next 10…

- Maximising Collaborative Editors
- Talking As You’re Coding



The last 2-3…

- Handling Mistakes
- Testing Your Code

This week

Maximising Collaborative Editors (Recap)

1. Leave Comments and Workings
2. Scaffolding
3. Drawing Diagrams

Talking As You're Coding (Recap)

1. See Interviewers as Collaborators
2. Keep Calm and Think Out Loud
3. Ask Questions, Not For Hints
4. Don't Rush!
5. Reiterate the Bounds!

Handling Mistakes: How?

It is normal to make small mistakes in your first draft.

How to quickly check through your code and take care of your mistakes?

Handling Mistakes: After You’re Done Coding

Quickly trace through your code and look for mistakes.

Things to check for:

- Off-by-one errors
- Variable name errors
- Missing braces, tabs, spaces
- Adding vs subtracting, and vice versa
- Min vs max, etc.

Testing Your Code

This needs to be done WITHOUT prompting from the interviewer.

- Testing is a good software engineering practice!
- It also helps you discover potential mistakes.

Ask politely before moving forward to testing …

“I would like to run through a few examples with my code to check its correctness. Is that okay?”

Testing Your Code

- Test your code MANUALLY

- Use examples you have come up with earlier

- Choose **suitable** examples for testing
  - Example size:
    If you're testing with linked lists, maybe ~5 elements would be good. 1-2 would be too little, and >8 will take way too long
  - Edge cases:
    Make sure to test both the normal case and 1-2 edge cases (if available)!

Testing Your Code

- Copy and paste an example into a comment block near your code
  - This is where the examples that you had come up with in the first 5 minutes really help!
- Useful techniques includes:
  - Keeping track of the key variables in your code

```
"""
Example: [...]

left: 0
right: 10
mid: 5

"""
```

Testing Your Code

- Copy and paste an example into a comment block near your code
  - This is where the examples that you had come up with in the first 5 minutes really help!
- Useful techniques includes:
  - Keeping track of the key variables in your code
  - Using pointers / cursors for clarity

It is alright to make mistakes!

For both Handling Your Mistakes and Testing Your Code, simply do the following if you find any mistakes:

- Acknowledge your mistake
- Explain how to fix it
- Quickly fix it afterwards

Moving forward from mistakes

If there are multiple parts or questions for the interview:

- Learn from mistakes made previously
- Try not to make the same mistakes twice

Being able to acknowledge and learn from your mistake is also a trait that interviewers are looking out for!

Handling Mistakes + Testing Your Code

This is one part of the technical interview that may vary A LOT between different companies and/or interviewers.

At TIPS, we are covering the most extensive interview flow, so that you know what to do next at any point of the technical interview.

However, in reality, different interviewers may have different preferences.

- If the interviewer cuts you off when you’re testing your code, adapt accordingly!

Outline for Tonight

- Handling Mistakes + Testing
- Question Patterns
- Live Demo (Chun Mun & Rohit)

Interview Format

- Standalone LeetCode style question(s)
- Long question with multiple parts
  - Easy to difficult
  - Straightforward to higher complexity

The key is to **be prepared** and **be flexible**!!

Array

- One of the most common types of questions.
- O(1) Access & O(n) Search
- Subarray:
  - A range of contiguous values within an array
  - [1, 2, 3, 4, 5] → [2, 3, 4] is a subarray while [1, 3, 5] is not a subarray
- Subsequence:
  - A sequence that can be derived from the given sequence by removing some or no elements without changing the order of the remaining elements
  - [1, 2, 3, 4, 5] → [1, 3, 5] is a subsequence, while [5, 3, 1] is not a subsequence

Array: Prefix Sum

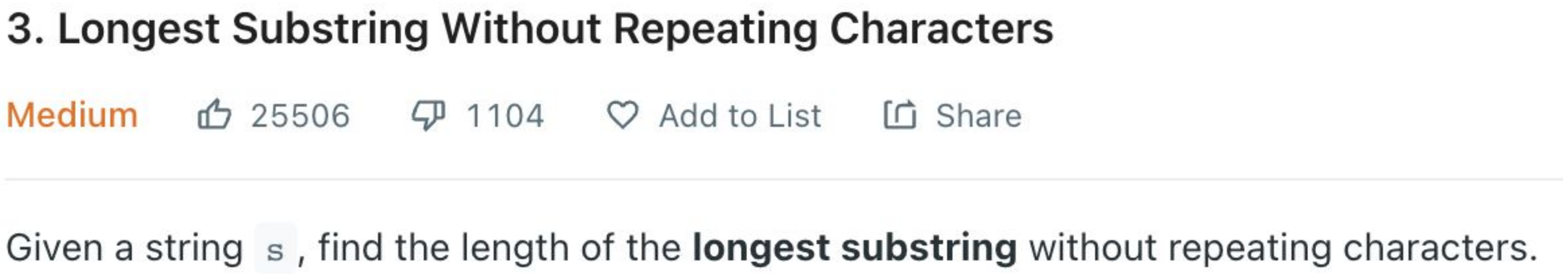
- Keyword: **Subarray Sum** → Consider **Prefix Sum**
- Prefix sum at index i = Sum from index 0 to i
- To find the subarray sum between index i and j
  - prefix_sum[j] - prefix_sum[i-1]
- There are other prefix arrays, e.g. prefix max, but prefix sum is one of the most common ones!

Array: Sliding Window

- Very common in subarray/substring problems
- Two pointers move in the same direction, which ensures each value is only visited at most twice
- Time complexity: O(n)

Array: Sliding Window

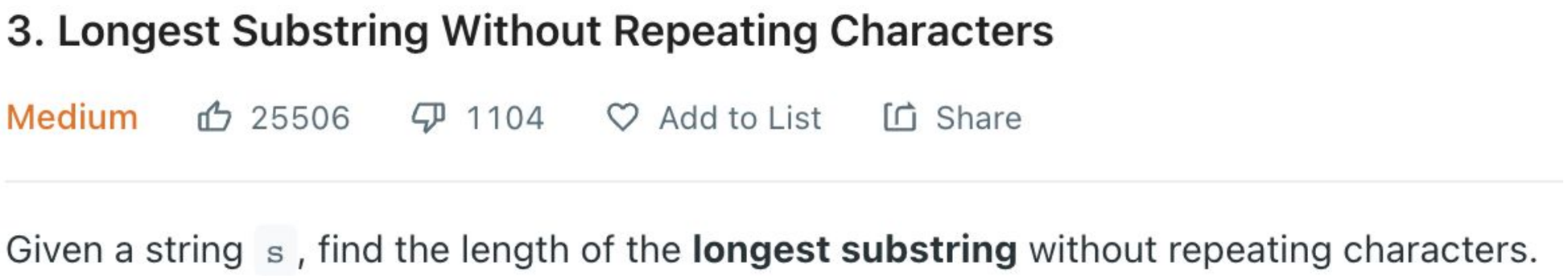
- Very common in subarray/substring problems
- Two pointers move in the same direction, which ensures each value is only visited at most twice
- Time complexity: O(n)



Array: Sliding Window

Pseudocode for Longest Substring:

- Two pointers (`start` and `end`) starting from index `0`
- Use a set to keep track of characters between `start` and `end`
- Advance `end`
  - If character at `end` is in the set, advance `start` and remove character from the set
  - Keep track of the longest substring length



Array: Sliding Window

Generalised algorithm pseudocode:

- Two pointers (`start` and `end`)
- Keep track of a current state for elements between `start` and `end`
- Advance `end`
  - If the next element make the state invalid, advance `start` until it's valid again
  - At every step, keep track of any potential result

Array: Sliding Window

- Count Number of Nice Subarrays
- Number of Substrings Containing All Three Characters
- Count Number of Nice Subarrays
- Replace the Substring for Balanced String
- Max Consecutive Ones III
- Binary Subarrays With Sum
- Subarrays with K Different Integers
- Fruit Into Baskets
- Shortest Subarray with Sum at Least K
- Minimum Size Subarray Sum

String

- A sequence of characters
- Very often, string questions are just like array questions
- Know the complexity of concatenating two strings in your chosen language
  - O(m + n)
  - Consider creating an array of strings/characters and joining them together at the end, instead of concatenating every step?
    - E.g. Python's `join` method

String

- Anagram
  - A word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.
  - Possible approach: Store elements in sorted order

- Palindrome
  - A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward as forward, such as madam or racecar.
  - Possible approach: Take a character as the center of the palindrome, and expand

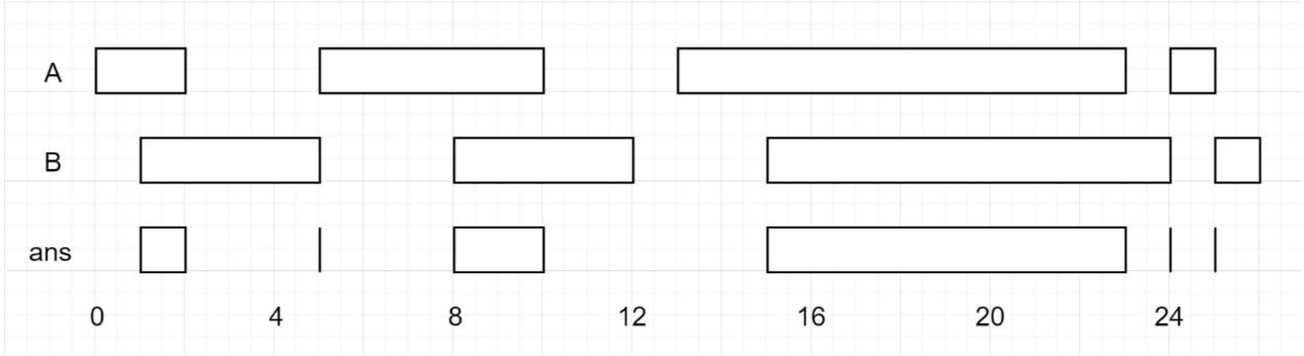
Intervals

- Intervals are special set of array questions
- Merge / Insert / Intersect Intervals …

Intervals

- Intervals are special set of array questions
- Merge / Insert / Intersect Intervals …

- First step is to **sort** all the intervals in **one** list
- Create an empty array for results
- Use min() and max() cleverly!!



Intervals

- Insert Interval
- Interval List Intersections
- Merge Intervals
- Non-overlapping Intervals
- Meeting Rooms I and II (Premium)

Binary Search

- If a question can be solved by linear search, consider binary search
- O(n) → O(logn)

Binary Search

- Sometimes, a binary search question can be hard to identify
  - Often involves a value that will make the final computation a lot easier
  - Use binary search to find this value

Binary Search

- Sometimes, a binary search question can be hard to identify
  - Often involves a value that will make the final computation a lot easier
  - Use binary search to find this value

- Find valid range for the value
  - Starting value of left and right pointers (AKA search range)
- Identify valid condition
- How to update the bounds?

Binary Search

```python
def solve_a_binary_search_question(arr: List[int]) -> int:
    # identify the search range
    # min_val, max_val = ...
    l, r = min_val, max_val

    while l < r:
        # be mindful of integer overflow (language dependent)
        mid = (l + r) // 2

        if is_valid(mid):
            l = mid + 1
        else:
            r = mid

    return l

def is_valid(val: int, arr: List[int]) -> bool:
    # check if val aka mid is valid
    # normally involve iteration of arr
```

```python
def solve_a_binary_search_question(arr: List[int]) -> int:
    # identify the search range
    # min_val, max_val = ...
    l, r = min_val, max_val

    while l < r:
        # be mindful of integer overflow (language dependent)
        mid = (l + r + 1) // 2

        if is_valid(mid):
            l = mid
        else:
            r = mid - 1

    return l

def is_valid(val: int, arr: List[int]) -> bool:
    # check if val aka mid is valid
    # normally involve iteration of arr
```

Binary Search

- Minimum Number of Days to Make m Bouquets
- Find the Smallest Divisor Given a Threshold
- Divide Chocolate
- Capacity To Ship Packages In N Days
- Koko Eating Bananas
- Minimize Max Distance to Gas Station
- Split Array Largest Sum

Coursemology forum post on Binary Search

Tree

- Revise CS2040(S)

- Be mindful of the type of Tree
  - Binary Tree
  - Binary Search Tree
  - Complete Binary Tree
  - Balanced Binary Tree

- Three types of traversals
  - In-order
  - Pre-order
  - Post-order

Graph

- Revise CS2040(S)
- Make sure to familiarise yourself with your language's libraries!

- DFS
  - Be familiar with both the recursive implementation and the stack implementation
- BFS
  - Often implemented using a deque
- Topo Sort

Graph

```
def dfs(matrix):
  # Check for an empty matrix/graph.
  if not matrix:
    return []

  rows, cols = len(matrix), len(matrix[0])
  visited = set()
  directions = ((0, 1), (0, -1), (1, 0), (-1, 0))

  def traverse(i, j):
    if (i, j) in visited:
      return

    visited.add((i, j))
    # Traverse neighbors.
    for direction in directions:
      next_i, next_j = i + direction[0], j + direction[1]
      if 0 <= next_i < rows and 0 <= next_j < cols:
        # Add in question-specific checks, where relevant.
        traverse(next_i, next_j)

  for i in range(rows):
    for j in range(cols):
      traverse(i, j)
```

```
from collections import deque

def bfs(matrix):
  # Check for an empty matrix/graph.
  if not matrix:
    return []

  rows, cols = len(matrix), len(matrix[0])
  visited = set()
  directions = ((0, 1), (0, -1), (1, 0), (-1, 0))

  def traverse(i, j):
    queue = deque([(i, j)])
    while queue:
      curr_i, curr_j = queue.popleft()
      if (curr_i, curr_j) not in visited:
        visited.add((curr_i, curr_j))
        # Traverse neighbors.
        for direction in directions:
          next_i, next_j = curr_i + direction[0], curr_j + direction[1]
          if 0 <= next_i < rows and 0 <= next_j < cols:
            # Add in question-specific checks, where relevant.
            queue.append((next_i, next_j))

  for i in range(rows):
    for j in range(cols):
      traverse(i, j)
```

Graph

```python
def graph_topo_sort(num_nodes, edges):
    from collections import deque
    nodes, order, queue = {}, [], deque()
    for node_id in range(num_nodes):
        nodes[node_id] = { 'in': 0, 'out': set() }
    for node_id, pre_id in edges:
        nodes[node_id]['in'] += 1
        nodes[pre_id]['out'].add(node_id)
    for node_id in nodes.keys():
        if nodes[node_id]['in'] == 0:
            queue.append(node_id)
    while len(queue):
        node_id = queue.pop()
        for outgoing_id in nodes[node_id]['out']:
            nodes[outgoing_id]['in'] -= 1
            if nodes[outgoing_id]['in'] == 0:
                queue.append(outgoing_id)
        order.append(node_id)
    return order if len(order) == num_nodes else None

print(graph_topo_sort(4, [[0, 1], [0, 2], [2, 1], [3, 0]]))
# [1, 2, 0, 3]
```

DFS vs Backtracking

- Backtracking
  - restore previous state of visited node, by making visited = false, after exploring current path whereas
- DFS
  - the state of the node remains same after a path is explored so that it will not be explored again.

- Pure DFS is a variant of backtracking in which state of visited nodes are not restored, and this variant is only useful for problems related to searching (reachability, etc) and not for problems involving pattern finding, for which we need to use the usual backtracking tree pruning algorithm.

Backtracking

- Subsets & Subsets II
- Permutations & Permutations II
- Combinations
- Combination Sum I & II & III
- Generate Parentheses
- Word Search

More on Question Patterns

- Technical Interview Handbook

- https://www.educative.io/courses/grokking-the-coding-interview
- https://seanprashad.com/leetcode-patterns/

Learn From Your Mistakes!

Revised	Name	Tags	Status	Learning points	Date	Url
☐	Merge k Sorted Lists	Hard Heap LinkedList Blind	Didn't Solve	python heap (priority, object)	August 24, 2021	https://leetcode.com/problems/merge-k-sorted-lists/
☐	Top K Frequent Elements	Medium Heap Blind	Solved Can be improved	sort python dictionary gives an array of sorted **keys** `sorted(dict, key=dict.get, reverse=True)[:k]` or `heapq.nlargest(k, dict.keys(), key=dict.get)`	August 24, 2021	https://leetcode.com/problems/top-k-frequent-elements/
☐	Find Median from Data Stream	Hard Heap Blind	Time limit exceeded	minheap and maxheap implement maxheap using negative numbers. heap[0] to find min/max in O(1)	August 25, 2021	https://leetcode.com/problems/find-median-from-data-stream/
☐	Clone Graph	Medium Graph Blind	Solved	dfs or bfs. use hashmap to keep track if already cloned nodes	August 23, 2021	https://leetcode.com/problems/clone-graph/
☐	Course Schedule	Medium Graph Blind	Didn't Solve	- cycle detection - use set for adjacency list - dfs: use color method to distinguish unvisited, currently recursing and visited - bfs: topo sort, check topo sort length == total length	August 23, 2021	https://leetcode.com/problems/course-schedule/
☐	Pacific Atlantic Water Flow	Medium Graph Blind	Didn't Solve	2 dfs from 2 corner of the matrix → DAG (visited == locations reached)	August 23, 2021	https://leetcode.com/problems/pacific-atlantic-water-flow/
☐	Number of Islands	Medium Graph Blind	Solved	save space using original grid to track the visited cell	August 24, 2021	https://leetcode.com/problems/number-of-islands/
☐	Longest Consecutive Sequence	Medium Graph Blind	Didn't Solve	list to set conversion : O(n) not really graph, make use of set O(1) look up	August 24, 2021	https://leetcode.com/problems/longest-consecutive-sequence/
☐	Alien Dictionary	Hard Graph Blind	Solved Do again	- create graph: take note of invalid cases - keep track of all vertices/letters - topological sort (kahn's algo) - detect non-DAG from topo sort result	August 25, 2021	https://leetcode.com/problems/alien-dictionary/
☐	Graph Valid Tree	Medium Graph Blind	Didn't Solve	method 1: tree property (n-1 edges and connected) method 2: union find (1 connected graph → one union)	August 26, 2021	https://leetcode.com/problems/graph-valid-tree/
☐	Number of Connected Components in an Undirected Graph	Medium Graph Blind	Solved	method 1: dfs, keep track of visited, count components method 2: disjoint set union find	August 27, 2021	https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph/
☐	Insert Interval	Medium Interval Blind	Took too long	update `newInterval` making use of `min` and `max`	August 27, 2021	https://leetcode.com/problems/insert-interval/
☐	Interval List Intersections	Medium Interval Facebook	Solved		August 21, 2021	https://leetcode.com/problems/interval-list-intersections/
☐	Merge Intervals	Medium Interval Facebook Blind	Solved	method 1: modify last added item in result array method 2: add interval to result when end < current start	August 21, 2021	https://leetcode.com/problems/merge-intervals/
☐	Non-overlapping Intervals	Medium Interval Blind	Solved Do again	greedy: if two intervals overlap, keep the one with smaller endpoints, as it will always overlap with less intervals	August 27, 2021	https://leetcode.com/problems/non-overlapping-intervals/

Demo With Chun Mun and Rohit!

Alumni Interviewer: Soon Chun Mun

Chun Mun is a graduate from SoC Computer Science in 2016 and has been working at Google ever since.

He was previously the Head Tutor of CS1010S in AY13/14 Semester 1, and was VP & Project Team Lead at CVWO.

QnA