Problem Solving Techniques

[1] Data Structure-Based

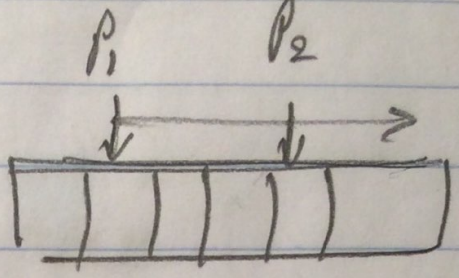
- Strings / arrays
- Matrix
- Linked List
- Binary Tree
- Graphs
- Heaps

- Mathematics
- Geometry

[2] Independent techniques

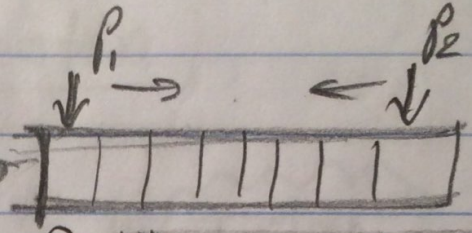
- Dynamic Programming
- Divide & Conquer
- Brute Force
- Greedy Algorithms
- Search Problems (AI)

[1] Arrays



① Sliding window

② Multiple Pointers
- many from start
- one start, one end



firstMin, secondMin, ...

③ Sorting trick (what if arr is sorted)
- merge sort
- quick sort

④ Data structure to keep track of Data
- stacks / queues
- Maps / sets
- tree
- heaps

⑤ Divide & Conquer

⑥ Definition of problem
- ex: definition of median of 2 sorted arrays

⑦ Search Problem
- word ladder

⑧ Binary Search

① Easier version of the problem to solve yours
- 2sum closest → 2sum, 2sum II
- intersection of 2 sorted arrays → what if they have same length

② Start from end

③ Juggle (no extra space)

④ Dynamic Programming

Jump pointers
arr[arr[i]]

Arrays

(5) for Dp[i]
accumulation trick

(1) No extra space

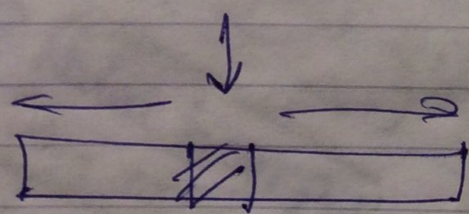
→ use 2 variables (if small storage)

→ use -ve mark of
more variables and arr[i] ∈ [1-n]

(2) Good idea (scan from left / right)
→ on your left → scan once from left
→ on your right → scan once from right

(3) Bucket sort
→ Sorting is required (bottleneck)
→ ranges are limited

(4) fanning-out trick



→ Strings

* scan both from left
or both from right

Scanned with CamScanner

[2] Matrix

smart traversal

① Traversal

```
for row = 0 to m
    for col = 0 to n
```

② Search DFS

→ with Dynamic Programming

→ use BFS for shortest path

③ Bounding index

→ left, right, top, Bottom

④ Spread matrix into array

```
K = row * n + col
col = K % n
row = K // n
```

⑤ Accumulation Trick

→ sum of col/row values up to this cell

⑥ Aux space

→ use some row/col as aux space

→ use every cell as aux space

- ex: DFS use "X" as your flag instead of extra set
- ex: change to known value (game of life)
- ex: shift operation to hold many flags
- Juggling → Rotate image

Extras

① optimize direction moves

- dx = [0, 0, -1, 1]
- dy = [1, -1, 0, 0]

② Reduce to our problem

- using accum trick

③ Disjoint sets / Trie

- implement as arrays

[3] Linked Lists

① Slow/fast Pointer

② Fake head
→ Very common for solutions

③ Pointer Manipulation
→ Reverse List, delete specific

④ No Extra space
→ chain to existing list (keep order correct)
ex: Copy with random pointer

[13] Binary Trees

[4] Must-Know Traversals

- inorder, preorder, postorder
- Level-order
- Morris Traversal

---

① Top-down Solution:
→ iterative: level-order ↓
→ Recursive: DFS

---

② Bottom-up Solution (Recursive) ↑
→ Base Case
→ Definition of problem

~~Recursive~~ ~~Iterative stack~~

~~Traversal with use stack~~

---

③ Traversal
→ mostly inorder

---

left limit & right-limit validation trick

track of BFS using one queue only counter to keep track of level

definition of problem

stack

Dynamic Programming

[1] Brute-Force — Top-down solution

→ find brute-force, top-down solution

[2] Dynamic Programming

→ identify problem:

① Recursive solution (brute-force)
② overlapping subproblem
→ (arriving at i,j many times)

[3] Formulate Problem — Definition of Problem

→ F(n) = ∅ (F(n-1))
→ identify subproblems

[4] DP solution

→ Top-down & memoization
→ Bottom-up?

Accumulation trick

$$dp[i] = max \begin{cases} dp[i-1] \\ arr[i] + dp[i-2] \end{cases}$$

5 Graphs

directed
undirected

adjacency matrix
adjacency list

1 Traversal
- BFS O(V+E)
- DFS O(V+E)

2 Shortest Path (Single source)
directed & undirected
→ Unweighted (BFS)
→ weighted
directed + Undirected
* Dijkstra O(E+VlogV)
+ve weights
* Bellmanford O(VE)

directed can handle -ve weights

3 Minimum Spanning Tree
* Prim (V+ElogE)
* Kruskal (ElogE)

4 Disjoint Sets
* makeset O(1)
* union O(logk)
* findset O(logk)

① Cycle
- undirected (BFS/DFS/disjoint)
- directed DFS

② Topological Sort
- DFS

③ Critical
- Edges (Bridges)
- Nodes (Articulation)

④ Strongly Connected Components