Algorithm Mental Model

Michael Rao

2022-06-28

Contents

1 Behavioral Mental Model

When asked behavioral questions, try to use examples that can show your soft skills, prioritizing the below qualities.

- Teamwork
- Communication
  - Listen
  - Understanding the audience (e.g. speaking with devs vs business)
  -
- Self-motivated
- Creativity
- Adaptability
- Open-Mindedness
- Confidence

1.1 Questions to ask Interviewer

1.1.1 Google

- Can you tell me what you enjoy the most and the least about working at Google?
- How has your role at Google evolved?
- Given the diverse technical stack in Google, how available are trainings if you want to improve your skill sets?
- Knowing that Google has so many in-house technology, do you think that has any hindrance on your growth as a Software Developer?

2 Algorithm Master List

Below are a list of Algorithms that can be applied to certain scenarios. It is extremely useful to know these algorithms

Algorithm	**Usage**	**Description**
Monotonic Stack/Queue	Range Queries in Array	Extremely useful for min and max tracking in Arrays
Manacher	Finding longest **Palindrome**	Finds palindrome by assuming current index is middle and expanding both ways
Dijkstra	Shortest Path in **Graph**	Greedy BFS Algorithm that works for edges of positive lengths
Bellman-Ford	Shortest Path in **Graph**	BFS Algorithm that works for all lengths
Union Find	Count number of unions in **Graph**	Algorithm that assumes all nodes as its own tree and perform unions by finding and comparing parent nodes
Kadane	Finding Maximum Subarrays	Uses Dynamic Programming and Prefix Sum to find maximum sum in subarray

2.1 Monotonic Stack/Queue

Monotone Stack is a stack that is monotonically increasing/decreasing.

Algorithm 1 Mono-Increasing Stack. Flip < to > for Mono-Decreasing Stack

```
for int i=0;i<array.length();i++ do
    if !stack.isEmpty() && array[i] < stack.getLast() then
        $stack.pollLast()$
        ▷ Perform calculations specific to problem if necessary
    end if
    $stack.offer(array[i])$          ▷ Some problems might require index instead
end for
▷ Perform calculations for rest of the Stack
▷ Handle any edge cases that might be left over
```

2.2 Manacher's Algorithm

2.3 Dijkstra's Algorithm

2.4 Bellman-Ford's Algorithm

Bellman-Ford's Algorithm is a BFS algorithm meant to calculate shortest path in a Directed Graph and can account for negative edge values.

Algorithm 2 Bellman-Ford's Algorithm Generic Steps

```
▷ Create an array of vertices that we will refer to as price that holds cost to get that vertex from source vertex
▷ Set every vertex in price to infinite, meaning not reachable
▷ Set starting vertex in price to 0 since we don't need to pay to start there
▷ Loop through every vertex in our Graph (Unless given a limit on how many nodes can be traversed). The incremental here is typically not used
▷ Create a copy of the price array which we will refer to here as priceCopy since this array might be modified multiple times in our next iteration
▷ Nest our loop by going through the adjacency list. If source vertex is not reachable, we continue; otherwise, check if price[source]+costToDest < priceCopy[dest], if it is, replace priceCopy[dest] with such value
▷ Copy priceCopy back to price after each iteration
▷ Return result
```

2.5 Union Find Algorithm

3 Generic Algorithmic Identification Table

One of the most important things in solving problems is being able to identify what type of problem it is. Below are some general potential algorithms you would want to consider when given a criteria. Note that they might not always work out immediately but it will get you thinking about the solution quicker.

Keywords/Criteria	**Potential Algorithm**	**Sample Problem**
Frequency/Difference/*	Pair, Map, Set	Auxiliary Data Structures, always consider its usage
Sorted List/Array	Binary Search	Find square root of n
Min/Max/Kth value	Max/Min Heap	Find Kth smallest number in unsorted list
Min/Max/Longest Subarray	Sliding Window	Longest Subsequence w/o repeating character
Return **all** solutions that ..	Backtracking	Find all Letter Combination of given phone number

Idea of Prefix Sum is similar to Dynamic Programming where we want to cache what we have already calculated in order to get the next value.

4 Sliding Window

4.1 Identifying Sliding Window Problems

4.2 Sliding Window Template

4.3 Practice Problems

4.3.1 Buying and Selling Stock

5 Binary Trees

5.1 Identifying Binary Tree Problems

Identifying Binary Tree Problems is trivial since it is always immediately given to you as a Binary Tree.

5.2 Binary Tree Template

One template type is as below:

1. Consider all possible traversals for Trees
    - **DFS**
    - **BFS**
    - **Preorder Traversal** (Root -> Left SubTree -> Right SubTree)
    - **Inorder Traversal** (Left SubTree -> Root -> Right SubTree)
        - Inorder on BST always produce an ascending sorted list
    - **Postorder Traversal** (Left SubTree -> Right SubTree -> Root)
2. Find one or more base cases
3. Perform validation/calculation on current node if necessary
4. Call the same function on the left subtree
5. Call the same function on the right subtree
6. Join the results from steps 2 and 3

6 Greedy

Greedy Algorithm is a myopic algorithm that processes the input one piece at a time with no apparent look ahead. In layman's terms, it is an algorithm that doesn't look ahead and tries maximize the immediate situation as much as possible.

6.1 Identifying Dynamic Programming Problems

6.2 Greedy Template

1. Use a simple rule to select a request i (e.g. pick smallest values, shortest paths, etc)
2. Reject all requests incompatible with i
3. Repeat until all requests are processed

6.3 Interval Scheduling

Given a set amount of resources (time in this case) and requests where the requests have a start time of $s(i)$ and end time of $f(i)$. We want to maximize the **number of requests fulfilled**.

6.3.1 Greedy Interval Scheduling Template

Following the Greedy Template, we first look for a **simple rule** to follow.

1. **rule:** Scan for earliest finish time (e.g. $min(f(i)$ for all i where i is request)
2. Reject all requests incompatible with i
3. Repeat until all requests are processed

Claim: Given a list of intervals L, greedy algorithm with earliest finish time product k^* intervals where k^* is maximum

- Induction on k^*:
  - **Base case**: $k^* = 1$, any interval works

6.3.2 Practice Problems

Maximum Subarray

Jump Game

Jump Game II

Gas Station / University Career Fair

7 Dynamic Programming

7.1 Identifying Dynamic Programming Problems

Dynamic Programming is the algorithmic model of solving a problem consisting of overlapping sub-problems by means of capturing the solutions of duplicated sub-problems by either **Memoization** or **Tabulation**

7.2 Memoization

7.3 Memoization Template

1. Work out an initial solution
   (a) Visualize the problem by drawing out a Decision Tree
   (b) Solve trivial base cases
   (c) Solve rest of the problem using recursion
2. Make the initial solution more efficient
   (a) Add a Memo Object, typically a O(1) insert and accessor (e.g. Array, Map)
   (b) Add a base case for the memo values
   (c) Store return values into the memo object
   (d) Add a return using the memo object

7.3.1 Fibonacci

7.3.2 Grid Traveler

7.4 Tabulation

7.5 Tabulation Template

1. Visualize the problem as a 1D or 2D table
2. Initialize the table with default values
3. Initialize the base cases of the table
4. Iterate and fill out rest of the table

8 BackTracking

8.1 Identifying Backtracking Problems

8.2 Backtracking Template

Backtracking is the algorithm used to generate every possibility in a given scenario.

1. Draw out decisions we can performs as a Decision Tree
    - e.g. Choose or not Choose to use arr[i]
2. Find out the failure paths if any (e.g. N-Queens)

9 String Algorithm

9.1 Boyer-Moore String Matching

9.2 Aho-Corasick String Searching