STS3007 - Advanced Competitive Coding - I

Sieve of Eratosthenes

Goal: Find all prime numbers up to a given limit 'n'.

- **Steps:**
    1. Create a boolean array `b[]` of size `n`, initialized to `true`.
    2. Set `b[0]` and `b[1]` to `false`.
    3. Iterate with `p` from 2, while `p * p <= n`:
        - If `b[p]` is `true`:
            - Mark multiples of `p` (starting from `p * p`) as `false` in `b[]`.
    4. Numbers with `b[number] = true` are prime.

Key Idea: Eliminate multiples of prime numbers to identify primes efficiently.

- Sieve_size - variable to store size of sieve or input n
- b[] - sieve array to mark non prime numbers
- If p is a prime number , next p*p is marked false in the boolean array / sieve array .

Segmented Sieve

- **Goal:** Find primes in a range [m, n] efficiently.
- **Steps:**
    1. **Pre-calculate:** Find primes up to `sqrt(n)` using the standard sieve.
    2. **Divide:** Split [m, n] into segments.
    3. **Process:** For each segment:
        - Create a boolean array `segment[]`.
        - Mark multiples of pre-calculated primes within the segment as `false`.
        - Remaining `true` entries in `segment[]` are primes.

Key Idea: Process smaller segments to reduce memory usage and improve performance.

- m and n are given say 1000 and 2000 , segment size is 100 then number of segments is 10 .
- Used to find prime numbers in a range .

Incremental Sieve

- **Key Idea:** Exploit the fact that all primes > 2 are odd to reduce memory and computation.
- **Implementation:**
    1. Handle 2 separately.
    2. Use an ArrayList to represent only odd numbers.
    3. When marking multiples of a prime, consider only odd multiples.

Benefits:

- Halves memory usage.
- Slightly faster.

Maneuvering

- **Goal:** Find the number of unique paths from top-left to bottom-right in an `m x n` matrix (moving only down or right).
- **Steps:**
    1. Create an `m x n` `dp` matrix initialized to 0.
    2. Set the first row and first column of `dp` to 1.
    3. For each cell `dp[i][j]`, calculate: `dp[i][j] = dp[i-1][j] + dp[i][j-1]`
    4. `dp[m-1][n-1]` holds the total number of paths.

Key Idea: Use dynamic programming to store and reuse intermediate results for efficient path counting.

- m x n matrix is given say 4 x 3
- Find the number of paths to start from top left 0 0 to the destination m-1 n-1 cell .
- Answer would be 10 .Try once
- Create mxn matrix with 0th row as all 1 and 0th column as all 1 , then keep filling the left cells as a sum of 1 top of it and 1 left of it . Repeat this , the final answer is achieved in the m-1 n-1 cell .

Euler's Totient Function (φ(n))

- **Counts:** Positive integers less than or equal to 'n' that are relatively prime to 'n'.

- **Formula:** If $n = p_1\text{^}e_1 * p_2\text{^}e_2 * \ldots * p\ \text{^}e$ (prime factorization), then: $\varphi(n) = n * (1 - 1/p_1) * (1 - 1/p_2) * \ldots * (1 - 1/p_{\square})$
- **Example (n = 140):**
  - $140 = 2^2 * 5 * 7$
  - $\varphi(140) = 140 * (1 - 1/2) * (1 - 1/5) * (1 - 1/7) = 48$

Key Idea: Counts numbers with no common factors with 'n' (except 1).

Chinese Remainder Theorem (CRT)

- **Solves:** Systems of congruences (e.g., $x \equiv a_1 \pmod{m_1}$, $x \equiv a_2 \pmod{m_2}$, ...) where m_1, m_2, ... are pairwise relatively prime.
- **Guarantees:** A unique solution modulo $M = m_1 * m_2 * \ldots * m_{\square}$.
- **Option Elimination:**
  - Eliminate options that don't satisfy any single congruence.
  - Combine congruences to simplify.
  - Calculate a few values to check against options.

Key Idea: Finds a single solution that satisfies multiple divisibility conditions.

Strobogrammatic Numbers

- **Definition:** Appears the same upside down (rotated 180 degrees).
- **Valid Digits:** 0, 1, 6, 8, 9 (6 rotates to 9, and vice versa)
- **Check:**
  1. Ensure only valid digits are used.
  2. Compare digit pairs from both ends; they must form valid pairs (0-0, 1-1, 6-9, 8-8, 9-6).

Key Idea: A number with rotational symmetry when flipped upside down.

- For n= 1 -> 0,1, 8
- For n=2 - 11,69,88,96
- Check if a number is Strobogrammatic
  Say 101 yes | 856 no | 888 yes
- Maybe mixed with prime and asked prime strobogrammatic ,then check if its then yes else no .

Binary Palindrome

- **Definition:** Same binary representation backward and forward.
- **Algorithm:**
    1. Reverse bits.
    2. Compare with original.

Key: Symmetry in binary using Bit manipulation approach .

Swap Nibbles

- **Nibble:** A group of 4 bits (half a byte).
- **Operation:** Exchange the positions of the two nibbles in a byte.
- **Example:** C3 (hex) -> 3C (hex)

Key Idea: Rearrange bits within a byte.

Booth's Algorithm

- **Multiplies:** Signed binary numbers.
- **Steps:**
    1. Initialize A, Q, Q-1, Count.
    2. Based on Q , Q-1
        - 00/11: ASR A, Q, Q-1
        - 01: A = A + M, ASR
        - 10: A = A - M, ASR
    3. Repeat step 2 until Count = 0.

Key: Groups 1s in the multiplier for efficient multiplication.

- Practice COA concept
- Try solving 5 * -3 (take 4 bits)
- And you are asked to find value of A and Q after 3rd iteration

---

Block Swap Algorithm

- **Rotates:** An array by 'k' positions.
- **Steps:**
    1. **Divide:** Array into blocks A (size k) and B (size n-k).
    2. **Swap:** Blocks (or sub-blocks) until rotated.
    3. **Cases:**
        - `k < n-k`: Swap `A` with last part of `B`, recurse on remaining `B`.

        - `k > n-k`: Swap `A` with first part of `B`, recurse on `A`.
        - `k == n-k`: Swap `A` and `B`.

Key: Efficient rotation by swapping blocks.

- Say array is 1 2 3 4 5
- K=3
- New first element = 4
- Array is now 4 5 1 2 3
- Divided into 2 blocks

Euclid's Algorithm

- **Finds:** GCD of two integers.
- **Steps:**
    1. If `b = 0`, GCD is `a`.
    2. Else, `GCD(a, b) = GCD(b, a mod b)`.

Key: Repeatedly find remainders until 0. Last non-zero remainder is the GCD.

- TC depends on min of a and b
- Say 5 , 3 ans will be 3
- Calc gcd of 5 and 20

Karatsuba Algorithm

- **Multiplies:** Large integers faster.
- **Divide & Conquer:**
    1. Split numbers into halves.
    2. Recursively compute 3 products (`ac`, `bd`, `(a+b)(c+d)`).
    3. Combine using the formula.

Key: Reduces multiplications for speedup

- T(n)= 3 T(n/2) + O(n)
- Uses divide and conquer approach
- Result = ac * 10 ^m + ((a+c)(b+d) -ac -bd))* 10^ m/2 + bd
- Partial expressions are ac*10^m or bd or (a+c)(b+d) -ac -bd))* 10^ m/2

Longest Sequence of 1s after Flipping

- **Finds:** Longest 1s sequence in binary array after flipping at most 'k' 0s.
- **Algorithm:**
    1. Two pointers (`left`, `right`), `zeroCount`, `maxLength`.
    2. Expand window (`right`).
    3. If too many zeros, shrink window (`left`).
    4. Update `maxLength`.

Key: Sliding window to maximize 1s sequence.

- Say 1 0 1 0 1 1 0 0 1 1 1 1 and K=2
- U ll get 1 0 1 0 1 1 1 1 1 1 1 1 , max of 8 consecutive 1s

Maximum Product Subarray

- **Finds:** Subarray with the largest product.
- **Tracks:** `prefix_product`, `suffix_product`, `max_product`.
- **Iterate:** Update values to handle negatives.
- **Example:**
    - Array: 1 0 1 1 -4 -6 99 100
    - Max Product Subarray: [-4, -6, 99, 100]
    - Max Product: 237600

Key: Account for negative numbers to find the maximum product.

Leaders in an Array

- **Finds:** Elements greater than all elements to their right.
- **Rule:** Last element is always a leader.
- **Example:**
    - Array: 1 2 3 9 8 7 0 1 2 8
    - Leaders: 9 8 8
- **Algorithm:** Scan from right to left, keeping track of the current maximum.

Key: Identify elements that dominate those to their right.

Majority Element

- **Finds:** The element appearing more than n/2 times in an array (where n is the array length).
- **Example:**

    - Array: 1 1 2 2 2 2 2 2 2
    - Majority Element: 2 (appears 7 times, which is more than 9/2)
- **Algorithm:** Boyer-Moore Voting Algorithm (efficiently tracks a potential candidate)

Key: Identify the element with the most occurrences.

Lexicographically First Palindrome

- **Creates:** Earliest palindrome in alphabetical order from a string.
- **Condition:** At most one character with odd frequency.
- **Example:**
    1. `aabbccd` -> `abcdcba`
- **Counterexample:**
    1. `aabbccddef` -> No answer (both 'e' and 'f' have odd frequencies)
- **Algorithm:**
    1. Count frequencies.
    2. Place half in the first half (alphabetical order).
    3. Odd character in the middle (if any).
    4. Mirror the first half.

Key: Symmetric arrangement, prioritize alphabetical order.

Maximum Equilibrium Sum

- **Finds:** Maximum sum at an index where the sum of elements to the left equals the sum of elements to the right.
- **Example:**
    - Array: -7 1 5 2 -4 3 0
    - Equilibrium Indices: 3 (sum = 1 on both sides)
    - Max Equilibrium Sum: 1
- **Algorithm:** Calculate prefix sums and suffix sums, then find the maximum sum where they are equal.

Key: Balance sums on both sides of an index to find the maximum equilibrium sum.

Maximum Sum Hourglass

Goal: Find the hourglass with the biggest sum in a matrix.

- An hour glass has 7 elements arranged 3 in top row, 1 in middle, 3 in bottom.
- No of hour glasses in a R x C matrix is (R-2) x (C-2)
- Min matrix size is 3 x 3
- Sum = arr[i][j] +   i j+1.   + i j+2.   + i+1 j+1.   + i+2 j.   + i+2 j+1.   + i+2 j+2

Selection Sort

- **Sorts:** By finding the minimum and swapping.
- **Example:**
    - Array: 5 2 8 1
        4. Find min (1), swap with 5 -> 1 2 8 5
        5. Find min (2), already in place -> 1 2 8 5
        6. Find min (5), swap with 8 -> 1 2 5 8
        7. Find min (8), already in place -> 1 2 5 8
- **Key:** Repeatedly select the smallest.

Quick Sort

- **Sorts:** By partitioning around a pivot.
- **Example:**
    - Array: 1 2 9 7 10 6 7 0 -1 100, Pivot: 7
    - Partitioned: 1 2 0 -1 6 7 7 9 10 100
    - Recursively sort left and right partitions.
- **Key:** Divide and conquer using partitions.

`Arrays.sort(arr)`

- **Use this:** For a fast, built-in sort.

Weighted Substring

- **Calculates:** String weight, finds max sum of 'k' consecutive weights.
- **Weights:** a=1, b=2, ... z=26 (customizable).
- **Example:**
    1. String: "abbccdeaghba"
    2. Weights: "09852731640985273164098527"
    3. String Weight: 54
- **Algorithm:**
    1. Map weights.
    2. Calculate string weight.

3. Find max k-sum (sliding window).

Key: Assign weights, sum them, find heaviest substring.

Move Hyphens

- **Transforms:** A string with hyphens scattered throughout into a string with all hyphens at the beginning, preserving the order of other characters.
- **Example:**
  - Input: "this-is-a-string-with-hyphens"
  - Output: "---thisisasstringwithhyphens"

Josephus Problem

- **Simulates:** A circle of people where every k-th person is eliminated until one remains.
- **Input:** `N` (number of people), `k` (elimination step)
- **Example:** `N = 14`, `k = 2` -> Answer: 13 (if starting position is 1) or 12 (if starting position is 0)
- **Note:** If both 12 and 13 are options, 13 is usually preferred (assuming 1-based indexing).

Activity Selection

- **Finds:** The maximum number of non-overlapping activities that can be scheduled.
- **Input:** `S[]` (start times), `F[]` (finish times)
- **Example:** `S[] = {10, 20, 30}`, `F[] = {25, 22, 50}` -> Answer: {0, 2} (activities 1 and 3)

N-Queens

- **Places:** N chess queens on an N×N board so no two queens threaten each other.
- **Minimum N:** 1, 4 (for solutions with more than one queen)
- **Queen Movement:** Horizontal, vertical, diagonal

Permutations and Combinations

- **nPr:** Number of ways to arrange 'r' items from a set of 'n' (order matters).
- **nCr:** Number of ways to choose 'r' items from a set of 'n' (order doesn't matter).

MST (Minimum Spanning Tree) - Kruskal's Algorithm

- **Finds:** A tree that connects all vertices in a graph with the minimum total edge weight.
- **Kruskal's:** Greedy algorithm that adds edges in ascending order of weight (while avoiding cycles).

Graph Coloring - Chromatic Number

- **Assigns:** Colors to vertices so no adjacent vertices have the same color.
- **Chromatic Number:** Minimum number of colors needed.

Huffman Coding

- **Compresses:** Data by assigning variable-length codes to characters based on their frequency.
- **Uses:** Prefix codes (no code is a prefix of another) for efficient decoding.

Hamiltonian Cycle/Path

- **Cycle:** A path that visits every vertex in a graph exactly once and returns to the starting vertex.
- **Path:** Visits every vertex exactly once (doesn't have to return to the start).

Warnsdorff's Rule

- **Heuristic:** For finding a knight's tour on a chessboard.
- **Rule:** The knight always moves to the square with the fewest onward moves.
- **Minimum N:** 5 (for a solvable knight's tour)

Also, refer to this for time and space complexity, plus some practice on Huffman Coding, Kruskal's Algorithm, and more

https://drive.google.com/file/d/1sG0zENrNUDXP4DF0MvxFYqG_uRrnL54j/view