🔁 Classic & Exchange Argument Based

1. Reorder jobs with deadlines and profits to maximize total profit (1 job/unit time).
2. Select the maximum number of non-overlapping intervals.
3. Cover a line segment with the fewest number of given intervals.
4. Choose k elements from an array to maximize the median.
5. Select items from a list with weight constraints to minimize the largest sum.
6. Divide coins into two piles such that the difference is minimized.
7. Given a set of ropes, connect all ropes with minimum total cost (Huffman-style).
8. Minimize the total waiting time for customers in a queue.
9. Find the lexicographically smallest string after at most k swaps.
10. Choose segments with minimal total length to cover a set of points.

🔒 Greedy + Sorting + Invariants

11. Find the largest number you can form with array digits under sum and digit constraints.
12. Sort people in a queue based on height and number of taller people in front.
13. Minimize the number of platforms needed for train arrival/departure times.
14. Minimize the cost of painting houses under adjacent house color constraints.
15. Distribute candies to children based on ratings (each child must have more than neighbors if rated higher).
16. Choose minimum operations to make an array non-decreasing by only increasing elements.
17. Rearrange characters of a string to avoid adjacent duplicates.
18. Make a string palindrome with minimal insertions or deletions.

🎁 Resource Allocation Greedy

19. Assign tasks to workers with deadlines and durations to maximize completed tasks.
20. Select the minimum number of classrooms for given class intervals.
21. Assign jobs to k machines to minimize the makespan (longest time).
22. Buy the cheapest available product under multiple discount days (with deadlines).
23. Schedule interviews such that maximum number of candidates are interviewed without overlap.
24. Divide n candies among m kids so that the happiest kid is as least happy as possible.

📉 Greedy with Binary Search (Greedy Check)

25. Given n books with pages, assign to m students such that max pages per student is minimized.

26. Distribute water from tanks to cities along a line minimizing max transport cost.
27. Place k gas stations on a highway to minimize the max distance between adjacent ones.
28. Allocate internet bandwidth to users under constraints to maximize minimum allocation.

⛓ Greedy on Graphs / Sequences

29. Given a graph with n nodes, choose edges to maximize the number of connected components.
30. Make an array beautiful by removing as few elements as possible (e.g., alternating parity, increasing/decreasing).

Each of these can be proved correct via **invariants or greedy exchange arguments**:

- **Local optimal choice leads to global optimum**
- **Maintaining a loop invariant (e.g., sortedness, prefix sum validity)**
- **Swapping out suboptimal choices improves or preserves correctness**

Would you like the **Codeforces contest links** or **editorial proof sketches** for some of them?

You said:

Binary Search on Functions With two-pointer/greedy check inside

ChatGPT said:

Here are **30 problem statements** (in one line) that use **Binary Search on Functions** (a.k.a. **Parametric Search**) with a **two-pointer or greedy check inside** — perfect for mastering this common pattern in **Codeforces** and similar contests:

✅ 1. Minimize/Maximize the answer with Greedy Check

1. **Allocate books to students** to minimize the **maximum pages** assigned (greedy: accumulate until limit).
2. **Place k cows in stalls** to maximize the **minimum distance** between any two cows (greedy: place greedily).
3. **Cut wood pieces** to get at least `k` pieces of length ≥ `x` (greedy: count pieces).
4. **Buy minimum-cost bouquet** by selecting k flowers and satisfying distance constraints (2-pointer check).
5. **Select subarrays** such that the maximum sum is at most `x` (greedy + prefix sums).
6. **Place gas stations** to minimize **maximum distance** between any two (greedy: place in largest gaps).

7. **Split array into k parts** to minimize the maximum sum of parts (greedy: partition when sum > mid).
8. **Select minimum radius** for heaters to cover all houses (2-pointer to match heater & house).
9. **Assign jobs to k workers** to minimize the **maximum working time** (greedy/DFS check).
10. **Cut logs of wood** to minimize max size of a piece while making at least `k` pieces (greedy: count).

---

2. Subarray/Window Based Problems

11. **Find smallest subarray size** such that some condition on sum/median/frequency is satisfied.
12. **Minimize max absolute difference** between two arrays using window sliding.
13. **Minimize unfairness** (max - min) of any subarray of size k (greedy over sorted array).
14. **Maximize number of items** that can be picked under a total cost using a window (2-pointer).
15. **Minimize max difference** of heights after k increases or decreases (greedy + sort).
16. **Select items** with cost within a range such that total cost ≤ `mid` (prefix sums + 2-pointer).
17. **Find longest segment** where frequency of any element ≤ k (2-pointer + counting).
18. **Choose maximum subarray** with average ≥ `x` (transform and check prefix sums).

---

3. Greedy Logic with Binary Search on Real/Integer Answers

19. **Find minimal time to paint boards** by k painters (greedy sum ≤ mid).
20. **Assign minimum capacity** to transport packages within D days (greedy shipping).
21. **Find max median** of k-length subsequence (greedy check via prefix sum transform).
22. **Buy maximum number of products** with cost under `x` using 2-pointer or priority queue.
23. **Distribute candies** so that each kid gets at least `x` candies (greedy: count how many get ≥ x).
24. **Allocate tasks** with different durations such that no worker exceeds time limit.
25. **Split rope of length `L`** into k parts of at least `x` length (greedy cut count).

---

4. Monotonicity-based Binary Search + Greedy

26. **Maximize min sweetness** by dividing chocolate bars among friends (greedy: sum and cut).
27. **Determine max number of lectures** that can be scheduled under total time (sort + accumulate).

28. **Find smallest budget cap** such that all departments are funded and total budget ≤ B (greedy sum capped).
29. **Find max group size** such that the difference between max and min ≤ D (2-pointer).
30. **Assign people to rooms** such that no room has people with rating diff > `x` (greedy match).

🔁 Summary of Techniques Used:

- Binary search on answer `x`
- **Greedy check:** Can we do it with this `x`? (prefix sum, segment checking, subarray/window-based, sorting)
- **Two pointers:** Efficient range-based checks in O(n)
- Typical complexity: `O(n log(max_limit))`

Would you like a **template** or **detailed solution sketch** for any of the above patterns (e.g., Cow Placement, Chocolate Division, or Painter Partition)?

You said:

Dynamic Programming (1D/2D/Bitmask) Indexed DP, knapsack, LIS, digit DP

ChatGPT said:

Here are **30 one-line dynamic programming (DP) problem statements** across popular types like **1D, 2D, Indexed DP, Knapsack, Bitmask, LIS, and Digit DP** — ideal for **Codeforces contest preparation**:

✅ 1D/Linear DP (Classic Form)

1. Find the number of ways to climb `n` stairs taking 1 or 2 steps at a time.
2. Compute the **minimum cost** to reach the end of an array with jump costs.
3. Count the number of binary strings of length `n` with no two consecutive 1s.
4. Find the **minimum number of cuts** needed to partition a string into palindromes.
5. Calculate the **minimum cost to paint houses** where no two adjacent houses have the same color.

✅ 2D DP (State on i and j)

6. Count the number of **distinct subsequences** of string `S` equal to `T`.
7. Compute the **length of longest common subsequence (LCS)** of two strings.
8. Find the **minimum number of insertions/deletions** to make two strings equal.
9. Calculate the number of ways to **partition array into two equal sum subsets**.

10. Find the number of ways to **transform one string into another** using insert/delete/replace.

Indexed DP (DP with Index + State)

11. You are given an array; find the **maximum score** by choosing elements under adjacency restrictions.
12. Find max coins you can collect by bursting balloons in optimal order (index DP).
13. Count the number of ways to parenthesize a boolean expression to evaluate to True.
14. You can rob houses but **not adjacent** ones; find max loot (classic House Robber).
15. Given a grid with costs, find the number of **minimum-cost paths** from (0,0) to (n,m).

🎒 Knapsack DP

16. 0/1 Knapsack: Max value you can get with weight limit `W`.
17. Count the number of subsets with sum equal to `S`.
18. Subset sum: Can you pick a subset of weights summing exactly to `S`?
19. Partition array into two subsets with **minimum absolute difference** of sums.
20. Bounded Knapsack: Each item has a quantity limit.

Bitmask DP

21. Assign `n` jobs to `n` workers to **minimize total cost** (Assignment problem).
22. Traveling Salesman Problem (TSP): Find the **shortest route** visiting all cities.
23. Given `n` switches and states, **minimize the cost** to turn all off using allowed operations.
24. Count the number of Hamiltonian paths in a graph using DP + Bitmask.
25. Given `n` people and their happiness values for sitting adjacent, **maximize total happiness**.

📊 Longest Increasing Subsequence (LIS) Variants

26. Find the **length of LIS** in an array.
27. Count the number of **distinct LIS** in an array.
28. Find the **minimum number of deletions** to make an array sorted.
29. Compute the LIS where difference between adjacent values ≤ k.
30. Length of LIS in 2D (e.g., envelopes, boxes nesting).

🔢 Digit DP

(used when n is up to 10^18 or string length ≤ 20)

31. Count numbers from 1 to `n` that do **not contain the digit 4**.
32. Count numbers between `a` and `b` whose **digits are strictly increasing**.
33. Count numbers from 1 to `n` where the **sum of digits is divisible by k**.
34. Count how many numbers from `L` to `R` have **at most k non-zero digits**.
35. Count numbers ≤ n where the **digit 3 occurs exactly twice**.

✅ Bonus: Mixed DP Ideas

36. Minimum number of coins to form amount `X` (Unbounded Knapsack).
37. Count total number of arithmetic subsequences in an array (indexed + hash map DP).
38. DP on trees: compute subtree sums and count nodes satisfying constraints.
39. DP on grid with obstacles, count all paths from (0,0) to (n,m).
40. Find maximum path sum in a triangle (classic bottom-up DP).

Would you like me to **group these by Codeforces tags**, or provide **template code** for any of the categories (like Bitmask DP or Digit DP)?

You said:

Bitmask + DP/Greedy Subset sum, state compression

ChatGPT said:

Here are **30 advanced one-line problem statements** involving **Bitmask + DP/Greedy**, especially focused on **subset sums** and **state compression** — key for Codeforces-style contests:

✅ Bitmask + DP (Subset Sum, TSP, Assignment)

1. **Count the number of ways** to divide a set of integers into two subsets of equal sum using bitmask DP.
2. You are given a cost matrix, assign each job to each worker **minimizing total cost** (Assignment Problem).
3. Find the **minimum path cost to visit all cities** in a graph starting and ending at city 0 (TSP).
4. **Maximize profit** by assigning tasks to people with constraints (state: assigned tasks).
5. You have `n` switches and allowed toggle combinations — find **min moves to reach goal state** (bitmask BFS + DP).

6. Partition the set of numbers into `k` subsets such that **max subset sum is minimized** (subset sum DP).
7. You are given a list of subsets; pick some such that **each element appears once** (Set Cover via bitmask).
8. In a grid of lights, each toggle toggles a row and column — **find min steps to turn all off** (bitmask compression).
9. Given a list of strings, find the **minimum number of concatenations** to cover all strings (Superstring via bitmask + overlap DP).
10. Solve 0/1 knapsack with **items ≤ 20** using subset generation and mask-based optimization.

⚙ Bitmask + State Compression (Efficient DP)

11. You are given `n` items and multiple bags — find ways to **pack them without overlaps** using bitmask.
12. Given string `s`, break into palindromic partitions — **min number of partitions** using mask memoization.
13. You are placing knights on a chessboard; ensure no two attack each other — **count valid configurations** using row-by-row DP with bitmask.
14. In a graph with ≤ 20 nodes, **find number of Hamiltonian paths** using mask and last node as state.
15. Find **minimum number of flips** to convert a binary matrix to all 0s with row/col toggle operations (compressed DP).
16. You have a magic square and can only rotate rows/columns — find **min moves** to reach target (bitmask state transitions).
17. You can activate switches in any order — **count ways to turn on all bulbs** with dependency constraints.
18. Solve Sudoku-like problems where each state of a row/box can be represented as bitmask (mask+recursion).
19. For a grid with obstacles and `k` moves, **find number of paths** using bitmask to avoid revisits.
20. Design an escape plan where only certain door combinations open — state is door config as bitmask.

Greedy + Bitmask Subset Selection

21. Given a set of people and their skills, pick a **minimum subset covering all skills** (Leetcode: Smallest Sufficient Team).
22. You want to **cover all nodes** using the minimum number of given vertex sets (greedy + bitmask cover).
23. Among all subsets, find one with **sum closest to x**, using bitmask to iterate over 2^n subsets (meet-in-the-middle if n > 30).
24. You are given several conditions in form of bitmasks — **check satisfiability of all**.
25. You have a set of elements with bonuses if picked together — **maximize total value** (greedy subset with mask).

26. Greedily select intervals, with each interval activating a bit — **find smallest subset to activate all bits**.
27. You are scheduling tasks with dependencies — **find lex smallest valid schedule** (topo + bitmask pruning).
28. Given a set of subsets, find the one with **maximum XOR** of selected values (bitmask + greedy).
29. Create subset of array with **no two elements having common set bits** (bitmask + subset check).
30. Pack boxes with weights and constraints — use greedy + mask to **minimize the number of boxes used**.

🔁 Common State Design Patterns:

- `dp[mask]` → best result using this subset
- `dp[mask][last]` → result ending at `last` element in this mask
- `mask & (1 << i)` → check if item `i` is included
- Subset iteration:

  ```
  cpp
  CopyEdit
  for (int sub = mask; sub > 0; sub = (sub - 1) & mask)
  ```

Would you like:

- Bitmask template code (e.g., TSP, Assignment)?
- Python or C++ version?
- Grouped problem tags from Codeforces?

You said:

Graphs: BFS/DFS + Toposort + Components Trees, DAGs, cycles, bridges

ChatGPT said:

Here are **30 graph problem statements** in one line each, focusing on core graph techniques like **BFS/DFS**, **Topological Sort**, **Connected Components**, **Trees**, **DAGs**, **Cycle Detection**, and **Bridges** — perfect for **Codeforces contest prep**:

🔁 BFS / DFS Basic Applications

1. Check whether a given undirected graph is **connected**.
2. Find the **shortest path** from node `s` to node `t` in an unweighted graph (BFS).
3. Count the number of **connected components** in an undirected graph.
4. Given a grid with `0/1`, find the **largest connected region of 1s** (DFS/BFS).
5. Determine if a graph is **bipartite** (colorable using 2 colors).

6. Perform DFS and record **entry/exit times** for all nodes.
7. Given a maze/grid, find the **minimum steps from source to destination** (BFS on grid).
8. Detect a **cycle in an undirected graph** using DFS.
9. Detect a **cycle in a directed graph** using DFS + recursion stack.
10. Given a graph and a starting point, find if there is a **path to a target node**.

📤 Topological Sort + DAG

11. Given a list of tasks with dependencies, **output a valid order** of completion (toposort).
12. Count the number of **valid topological sorts** for a given DAG.
13. Find the **longest path** in a DAG.
14. You’re given prerequisites for courses — is it **possible to finish all**?
15. Given a DAG, find all nodes with **zero indegree**.
16. Given `n` nodes with edges, determine if the graph is a **valid DAG**.
17. You’re given a DAG with edge weights — **find max sum path** from source to sink.
18. Given task durations and dependencies, **compute earliest completion time**.

Connected Components (DFS/BFS/Union-Find)

19. Count the number of **islands** in a 2D grid.
20. Find the **size of the largest connected component** in a graph.
21. Given a friendship graph, find the number of **friend circles** (connected groups).
22. Given `n` users and mutual follow relationships, group them into **influencer clusters** (connected comps).
23. After deleting some edges, **check if the graph remains connected**.
24. You are given a graph with some **disconnected parts** — connect it with **minimum edges**.

🌳 Trees (DFS, Rerooting, Subtree)

25. Find the **diameter** of a tree (longest path).
26. Compute the **subtree sum or size** for every node.
27. Given a tree, find the **lowest common ancestor (LCA)** of two nodes.
28. You are given a tree and must **find the centroid** (balanced partition point).
29. Reroot the tree and calculate **sum of distances from each node**.
30. Find the **distance between two nodes** in a weighted/unweighted tree.

🔄 Cycle Detection / Bridges / Articulation Points

31. Detect if a directed graph has a **cycle**.
32. Detect if an undirected graph has a **cycle** using DSU (Union-Find).
33. Find all **bridges** in an undirected graph (edges whose removal increases components).
34. Find all **articulation points** in a graph (nodes whose removal disconnects graph).
35. You are given an undirected graph — check if it is a **tree** (connected + no cycles).
36. In a directed graph, find the **strongly connected components (SCC)** (Kosaraju’s or Tarjan’s).
37. Check if a directed graph is **strongly connected** (single SCC).
38. Identify **back edges**, **forward edges**, and **cross edges** using DFS timings.

🔁 More Structural Graph Problems

39. You’re given a graph with weights — **find Minimum Spanning Tree (MST)**.
40. You’re given a list of roads and cities — **find which roads must be built** to connect all.

✨ Optional Add-ons

Let me know if you'd like:

- Visualizations (for bridges, cycles, DAGs)?
- Python or C++ template for DFS/BFS/toposort/bridges?
- Problem tags from real **Codeforces/AtCoder** problems matching each?

These patterns often appear in **Div 2 C/D** and **Div 1 A/B** problems.

You said:

DSU (Disjoint Set Union) Union-Find with path compression

ChatGPT said:

Here are **30 one-line problem statements** involving **DSU (Disjoint Set Union)** aka **Union-Find with Path Compression** — ideal for **Codeforces/CP prep** in graph connectivity, component merging, MSTs, and cycle detection:

🔁 Basic Connectivity & Components

1. Process `n` union queries and answer whether two elements belong to the **same component**.
2. Given friendships among `n` people, find the **number of friend groups**.
3. Merge cities by roads and answer queries: **are two cities connected**?
4. Each operation merges two groups — find **size of group** containing element `x`.

5. After each union operation, **report number of connected components**.

🛤 Cycle Detection (Graph Applications)

6. Given an undirected graph with edges, **detect if adding an edge creates a cycle**.
7. In a forest, add edges and report whether the result is still **acyclic**.
8. For each edge, check if it's a **redundant connection** (already connects connected nodes).
9. Determine whether a set of equality and inequality constraints on elements is **satisfiable**.

□ Union by Rank + Path Compression Efficiency

10. Perform union operations and for each element, return its **root (representative)**.
11. Implement DSU with union by rank and show **logarithmic time behavior**.

🌉 Minimum Spanning Tree (MST) Applications

12. Use **Kruskal's Algorithm** to find the **minimum cost to connect all nodes**.
13. You are given cities and roads — choose a subset to **minimize total repair cost**.
14. Given a weighted graph, build an MST and report **total weight**.
15. Given optional and mandatory edges, **construct MST** including all mandatory ones.
16. Given cities and bridges, find the **cheapest bridge to remove to break the island**.

🌍 Offline Query Processing with DSU

17. Given `n` nodes and union queries, answer whether two nodes were ever in **same component** at any point.
18. Use **DSU on offline queries** to check historical connectivity.
19. Given a dynamic connectivity problem, process **edge deletions using offline DSU**.
20. In a city with changing roads, answer if two buildings are **ever connected**.

🎁 DSU with Extra Metadata

21. Support union and find operations where each group tracks **its size**.
22. After each union, report the **largest size of any group**.
23. Each node has a weight; when merging, maintain **total weight per component**.

24. In a social network, merge users and maintain **maximum friendship score per group**.
25. For each query, return the **sum of values in the connected component**.

□ Advanced DSU Variants

26. **DSU with rollback**: Undo the last `k` union operations.
27. Maintain connected components over time using **persistent DSU** (retroactive queries).
28. In a bipartite graph, use DSU to **check bipartiteness** after unions.
29. You are given parity constraints (even/odd); use **DSU with parity** to check contradictions.
30. Use **DSU on trees** (a.k.a. DSU on subtree) to solve subtree queries efficiently.

🛠 Common DSU Template (C++)

```
cpp
CopyEdit
vector<int> parent(n), size(n, 1);
for (int i = 0; i < n; ++i) parent[i] = i;

int find(int x) {
    if (parent[x] != x) parent[x] = find(parent[x]); // path compression
    return parent[x];
}

void unite(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return;
    if (size[a] < size[b]) swap(a, b); // union by size
    parent[b] = a;
    size[a] += size[b];
}
```

Would you like:

- Real **Codeforces problems** for each?
- **Weighted DSU**, **DSU with rollback**, or **DSU on tree** templates?
- Or conversion into Python?

You said:

Prefix Sums + Segment Trees/Fenwick Trees Range sum, range queries

ChatGPT said:

Here are **30 one-line problem statements** focusing on **Prefix Sums**, **Segment Trees**, and **Fenwick Trees (Binary Indexed Trees)** — all centered around **range sum / range query** problems, ideal for **Codeforces and CP preparation**:

✓ Prefix Sums (Fast Precomputation + Static Queries)

1. Given an array, answer multiple **range sum queries** in O(1) using prefix sums.
2. Find the number of **subarrays whose sum is equal to K** using prefix sum + hashing.
3. For a binary string, find the number of **equal 0-1 subarrays** using prefix difference mapping.
4. Precompute prefix sum and return **sum of even elements** in given ranges.
5. Answer queries of the form: **count of elements ≥ x in a subarray** using prefix count arrays.
6. Given a grid of numbers, **answer sum queries for any subrectangle** using 2D prefix sum.
7. Count the number of **subarrays with sum divisible by K** using prefix mod count.
8. For each prefix, return **max subarray sum ending at that point**.
9. Count the number of **balanced subarrays** (e.g., for brackets or sign patterns).
10. Check if a subarray is **palindromic using prefix hash** (rolling hash + reverse).

✓ Fenwick Tree / Binary Indexed Tree (Dynamic Point Update, Range Query)

11. Given `n` elements, perform **point updates** and **range sum queries** in log(n).
12. Answer queries of type: **add x to index i**, then find **prefix sum up to index j**.
13. For a permutation, compute the **number of inversions** using Fenwick Tree.
14. Count how many elements less than `x` appeared before position `i` (online frequency count).
15. For each position, compute **how many times it was updated** using difference array + BIT.
16. Process queries of form: **add to a range, query single index** using range update BIT.
17. Answer **kth order statistics** (kth smallest element in prefix) using BIT + coordinate compression.
18. Build a Fenwick Tree to **track frequency of values** and answer rank/count queries.
19. For a game leaderboard, **update score and find rank** dynamically.
20. Implement a **2D BIT** to perform point updates and submatrix sum queries.

✓ Segment Tree (Static/Range + Dynamic Updates)

21. Build a segment tree for **range minimum queries (RMQ)**.
22. Modify elements and answer **range sum queries**.
23. Modify elements and answer **range maximum queries**.
24. Implement **lazy propagation** to support range updates and range queries efficiently.
25. Given an array, find **first index from left with sum ≥ k** using segment tree.
26. Maintain array and answer queries of form: **what is the minimum in subarray [l, r]?**

27. Solve the **dynamic RMQ**: update elements and query min in given range.
28. Build a segment tree that **counts number of even and odd elements** in range.
29. Segment Tree Beats: range assignment + conditional max queries (advanced).
30. Maintain **GCD/LCM in a range** using segment tree.

□ Summary of Use Cases:

Task Type	Best Tool
Static prefix sum queries	Prefix sum (O(1) per query)
Point update + range sum	Fenwick Tree / Segment Tree
Range update + point query	Fenwick Tree (difference array)
Range update + range query	Lazy Segment Tree
K-th order stat / freq	Fenwick Tree + compression
2D grid range sum	2D prefix sum or 2D BIT

Would you like:

- C++ / Python templates for Fenwick Tree or Segment Tree?
- Real **Codeforces problems** matched to these types?
- A practice sheet or grouped roadmap by difficulty?

You said:

Math/Combinatorics + Inclusion- Exclusion Modular arithmetic, nCr mod p, totient, primes

ChatGPT said:

Here are **30 one-line problem statements** involving **Math, Combinatorics, Modular Arithmetic, Inclusion-Exclusion, nCr mod p, Euler's Totient, Primes, etc.** — perfect for **Codeforces contests and olympiad-level prep**:

🔢 Modular Arithmetic + Number Theory

1. Compute **(a × b) mod p** for large `a` and `b` (e.g., up to 1e18).
2. Solve **modular exponentiation**: find (a^b) mod m efficiently.
3. Given a and m, compute the **modular inverse of a modulo m**.
4. Compute **nCr modulo p** (for large n, using Fermat's Little Theorem).
5. Compute the **product of factorials mod p**.

6. Find the **smallest number `x`** such that `a^x ≡ b (mod m)` (Discrete Log - Baby-step Giant-step).
7. Calculate **n! mod p** where `n < p`, but very large.
8. Count the number of **integers ≤ n that are coprime to k** using Euler's Totient Function φ(k).
9. Check if `a^b` is divisible by `c` using prime factorizations and exponent comparison.
10. Evaluate **binomial coefficients** modulo composite numbers using Lucas’s Theorem.

□ Combinatorics + nCr + DP + Modulo

11. Count the number of ways to **distribute k identical balls into n distinct boxes**.
12. Count the number of **derangements (permutations with no fixed point)** modulo p.
13. Given n, compute the number of **binary strings of length n with no consecutive 1s**.
14. Find the number of **ways to choose k elements** from an array such that their sum is divisible by m.
15. Compute number of **combinations of r non-negative integers summing to n**.
16. Count ways to **place k non-attacking rooks on an n×n chessboard**.
17. Given a number `n`, compute the number of ways to **partition it into at most k parts**.
18. Count permutations of n elements with **exactly k inversions** (using DP or EGF).
19. Find number of **lattice paths** from (0,0) to (n,m) avoiding some obstacles.
20. Compute number of **ways to choose a subset with GCD = 1** from array A.

⚖ Inclusion-Exclusion Principle

21. Count the number of integers ≤ N divisible by **at least one** of a given set of primes.
22. Count the number of strings of length n **without forbidden substrings** using IEP or Aho-Corasick + DP.
23. Count permutations of n items **with some constraints** using IEP (e.g., fixed points, colors).
24. Count numbers between L and R which are **not divisible by any element** of a given set.
25. Count subsets of array where sum is divisible by m **excluding all-zero sum** (IEP).
26. Find number of integers ≤ N that are **not coprime** to any of a given list using IEP on φ.
27. Count integers ≤ N **not divisible by any square number** > 1 (Moebius function + IEP).
28. Count numbers ≤ N whose decimal digits **do not contain a forbidden digit**.
29. Count integers ≤ N **coprime to both a and b** using inclusion-exclusion and gcd/lcm.
30. Count ways to color n objects with k colors such that **at least one color is unused**.

🔁 Common Math/Comb Techniques:

Concept	Use
Modular Inverse (Fermat/Lucas)	nCr mod p, inverse under mod
Inclusion-Exclusion	Overlapping sets, constraints
Euler's Totient ϕ(n)	Count coprimes, RSA, modular reductions
Sieve of Eratosthenes	Fast prime generation
Moebius Function μ(n)	IEP with multiplicative functions
Factorial + Inverse Fact Array	Fast binomial coefficient

Would you like:

- Code templates for `nCr mod p`, modular inverse, Euler φ?
- A C++/Python snippet pack?
- A Codeforces problemset mapping these techniques by topic and difficulty?

You said:

Sliding Window + Monotonic Queues Max/min in window, optimal subarray

ChatGPT said:

Here are **30 one-line problem statements** focused on **Sliding Window + Monotonic Queues**, great for Codeforces-style preparation:

✅ Sliding Window + Monotonic Queue Problem Statements

1. Find the maximum in every subarray of size `k`.
2. Find the minimum in every subarray of size `k`.
3. Count subarrays where max - min ≤ `k` using monotonic deque.
4. Longest subarray where max ≤ 2×min.
5. Maximum sum of subarray of size `k`.
6. Minimum number of swaps to bring all ≤ `k` elements together.
7. Longest subarray with at most `k` distinct integers.
8. Longest subarray with sum ≤ `k`.
9. Count number of subarrays with sum exactly `k`.
10. Minimum length subarray with sum ≥ `k`.
11. Longest subarray where difference between max and min is ≤ `t`.
12. Longest subarray where frequency of any number is ≤ `k`.
13. Smallest window with all characters of another string.
14. Minimum window substring with exactly `k` distinct characters.
15. Longest substring where frequency of most frequent character × `k` ≤ window size.
16. Check if any subarray of size `k` has a sum divisible by `m`.

17. Find number of good subarrays where max = min.
18. Count number of subarrays with unique elements.
19. Minimum swaps to group all 1s together.
20. Find k-th smallest element in every subarray of size `n`.
21. Find the number of subarrays where max < threshold.
22. Find number of continuous subarrays with strictly increasing elements.
23. Find maximum product in subarray of size `k`.
24. Count subarrays with sum multiple of `k`.
25. Count subarrays with sum in range `[L, R]`.
26. Check if there exists a subarray of size at least `k` with average ≥ `t`.
27. Sliding window median for every window of size `k`.
28. Maximum number of vowels in any substring of size `k`.
29. Find the shortest subarray with sum strictly greater than `k`.
30. Count of subarrays where max appears before min in window of size `k`.

Let me know if you want detailed explanations or solutions for any of these.

You said:

Trie / Bit Trie (optional) XOR problems, string masks

ChatGPT said:

Here are **30 one-line problem statements** focused on **Tries and Bit Tries**, useful for solving **XOR problems and string mask problems**, especially in contests:

✅ Trie / Bit Trie Problem Statements

◆ Binary/Bit Trie (XOR-focused)

1. Maximum XOR of any two elements in the array.
2. Maximum XOR of subarray.
3. Maximum XOR for each prefix of an array.
4. Find a pair with given XOR in array.
5. Count number of pairs with XOR less than `k`.
6. Find maximum XOR of a number with elements in the array.
7. Number of subarrays with XOR equal to `k`.
8. Longest subarray with given XOR value.
9. Count number of pairs (i, j) such that `A[i] ^ A[j] = K`.
10. Minimum XOR of any two numbers in the array.
11. For each query, return XOR of prefix until index `i`.
12. For every prefix of array, return maximum XOR with a fixed number.
13. Find the minimum number of elements to remove to make XOR of array ≤ `k`.
14. Construct a binary trie and return the path of maximum XOR.
15. Count number of distinct XORs among all subarrays.

◆ **String Trie (Prefix-focused)**

16. Insert strings into Trie and find count of unique prefixes.
17. Longest common prefix from a list of strings.
18. Auto-complete suggestions using Trie.
19. Count how many words share the same prefix.
20. Find if any two words in a list form a palindrome pair.
21. Implement prefix search with wildcard `?` or `*`.
22. Check if string can be segmented into dictionary words (Word Break).
23. Given a string `s`, find the shortest unique prefix of each word in dictionary.
24. Given queries of prefix search, answer efficiently using Trie.
25. Count the number of words that appear as prefix of another word.
26. Check if a word exists in Trie and return number of times it appears.
27. Lexicographically smallest string using Trie traversal.
28. Number of string pairs with a common prefix of length ≥ `k`.
29. Build a Trie of binary representations of integers and find the deepest common prefix.
30. Replace words in a sentence using dictionary (Trie-based filter).