Here are **30 one-line problem statements** focusing on **DP Optimization Techniques**:
(Convex Hull Trick, Divide & Conquer DP, Monotonic Queue)

◆ Convex Hull Trick (CHT) - Line/Parabola Optimization (mostly min/max of linear expressions)

1. Given `n` lines, answer `q` queries for minimum y at x.
2. Minimize cost of partitioning array into `k` groups with quadratic group cost.
3. Find the minimum total cost to paint `n` fences where cost is linear per fence.
4. Compute the minimum fuel cost to reach city `n` with fuel price per km.
5. Optimize the DP: `dp[i] = min(dp[j] + (i-j)^2 + c)` for `j < i`.
6. Minimize cost of producing `i` units where cost depends linearly on previous units.
7. Find max profit selling items over days with fixed linear earning model.
8. Optimize cost of merging segments with per-segment linear penalty.
9. Minimize cost to travel cities, with tolls increasing linearly per jump.
10. Optimize `dp[i] = min(dp[j] + A[i]*B[j])` for all `j < i`.

◆ Divide and Conquer DP (D&C Optimization) - Reduces from $O(N^2K)$ to $O(NK \log N)$

11. Partition an array into `k` parts minimizing sum of subarray costs.
12. Minimize total cost of placing `k` postal offices across `n` houses.
13. Optimal k-segmentation of array based on prefix sum-based cost.
14. Divide `n` files into `k` groups minimizing cost function of file group.
15. Optimal placement of `k` schools in `n` towns to minimize total distance.
16. Divide `n` students into `k` groups minimizing squared deviation in scores.
17. Schedule `k` lectures over `n` days minimizing overlap cost.
18. DP for minimizing `dp[i][j] = min(dp[i-1][k] + cost[k+1][j])` over valid `k`.
19. Assign `n` tasks to `k` processors minimizing maximum cost.
20. Split `n` jobs into `k` phases minimizing transition cost between them.

◆ Monotonic Queue Optimization (Sliding Window DP) - Often used in Convex/Concave Monotonic transitions

21. Minimize total cost with windowed DP transitions: `dp[i] = min(dp[j] + cost(j, i))`.
22. Find max sum of length `≤ k` subarray using DP.
23. DP with monotonic constraint: `dp[i] = max(dp[j] + A[i])` with `j in [i-k, i]`.
24. Compute the maximum/minimum score to reach end with `j ≤ k` jumps.
25. Minimize number of steps to reach end using jumping windows.
26. Optimal way to merge stones in ranges of size ≤ `k`.
27. Maximize value by choosing non-overlapping subarrays of fixed size.

28. Minimize energy with bounded jump size `k` (like frog jump problem).
29. Monotonic optimization of `dp[i] = min(dp[j] + C[i]*A[j])` with constraints.
30. Maximize coins collected in a grid path with window constraints on movement.

Let me know if you'd like these 30 problems **categorized by difficulty**, **with editorial/tags**, or **converted into platform-style problems (like Codeforces, AtCoder, LeetCode)**.

You said:

Tree Algorithms Tree DP, Rerooting DP, HLD (Heavy-Light Decomposition)

ChatGPT said:

Here are **30 one-line problem statements** based on **Tree Algorithms**, specifically covering:

- ✅ Tree DP
- ✅ Rerooting DP
- ✅ HLD (Heavy-Light Decomposition)

◆ Tree DP (Bottom-up or Top-down Dynamic Programming on Trees)

1. Find the size of the largest independent set in a tree.
2. Count the number of ways to color a tree with 2 colors such that adjacent nodes have different colors.
3. Find the diameter (longest path) of a tree using DP.
4. Compute the maximum sum of node values such that no two adjacent nodes are selected.
5. For each node, compute the longest path starting at that node.
6. Count the number of paths in the tree where the sum of node values is ≤ K.
7. For each node, compute the size of its subtree.
8. Count number of leaf nodes in the subtree of each node.
9. DP on tree to find maximum matching (non-adjacent edges).
10. Count the number of distinct rooted subtrees in the tree.

◆ Rerooting DP (Recomputing DP when rerooting the tree at every node)

11. For each node, compute the sum of distances to all other nodes.
12. For each node, compute the height of the tree when rooted at that node.
13. Find the number of nodes in each subtree and total distance from root to all nodes — for all roots.
14. For every node, compute the maximum distance to any other node in the tree.
15. Rerooting DP to compute longest path passing through each node.
16. For each node, calculate sum of squared distances to all other nodes.

17. Count number of ways to choose `k` nodes such that they form a connected subtree — for every root.
18. Compute minimum cost to guard every node from adjacent attacks using rerooting DP.
19. For each node, find maximum XOR of values in its subtree.
20. Compute for each node the minimum weight of a path passing through it and any two nodes.

◆ Heavy-Light Decomposition (HLD) - Path/Query optimization in trees

21. Answer path queries (e.g., max, min, sum) between any two nodes in $\log^2(N)$.
22. Update values at nodes and query max value along path `u` to `v`.
23. Count number of edges with weight ≥ X on path from `u` to `v`.
24. Support subtree update and path query using HLD.
25. Answer LCA (Lowest Common Ancestor) queries using HLD.
26. Answer k-th ancestor of node `u` efficiently using HLD + Binary Lifting.
27. For given paths, count number of values that occur at least twice.
28. Answer path queries for parity of values (even/odd) between two nodes.
29. Implement dynamic tree edge updates with HLD + segment tree.
30. For each query, find the node on path `u-v` with a specific condition (like first even-valued).

Would you like these problems:

- **? Grouped by platform (Codeforces, AtCoder, etc.)**
- **? With links/editorials/tags**
- **? With implementation tips or patterns**

Just say the word and I'll help you further!

You said:

Advanced Segment Trees Lazy propagation, Merge SegTree, Persistent ST

ChatGPT said:

Here are **30 one-line problem statements** based on **Advanced Segment Trees**, covering:

- ✅ Lazy Propagation
- ✅ Merge Segment Tree
- ✅ Persistent Segment Tree

◆ Lazy Propagation (Range Updates + Range Queries)

1. Update all elements in range `[l, r]` by `x`, and query sum in range `[a, b]`.
2. Flip all bits in range `[l, r]` and count number of 1s in a range.
3. Add `x` to all elements in range `[l, r]` and find max in `[l, r]`.
4. Set all elements in range `[l, r]` to value `x`, then query sum.
5. For a binary array, count number of subarrays with all 1s after multiple flip operations.
6. Range increment and range GCD queries on an integer array.
7. Apply linear function `a*x + b` to range and query sum of squares.
8. Apply addition/multiplication updates and get product of elements in a range.
9. Perform range update on segments and answer whether entire segment is sorted.
10. Add `x` in range `[l, r]` and return count of elements ≥ k in the array.

◆ Merge Segment Tree (Segment Tree with custom merge like map/multiset, useful for frequency, top-k, etc.)

11. For each range `[l, r]`, return the k-th smallest element.
12. Count number of elements less than or equal to `x` in `[l, r]`.
13. Store frequency map in each segment and answer frequency of `x` in range `[l, r]`.
14. Count number of distinct elements in range `[l, r]`.
15. Find majority element (if any) in range `[l, r]`.
16. Find the median in range `[l, r]` using segment tree of multisets.
17. Count number of pairs `(i, j)` in a range with `A[i] + A[j] ≤ X`.
18. Maintain ordered set in each segment and answer number of elements between `a` and `b`.
19. Range frequency query with merge segment tree and coordinate compression.
20. Answer how many times a number `x` appears between `l` and `r`.

◆ Persistent Segment Tree (Versioned Segment Tree — supports rollback, snapshots)

21. Build a versioned array with point updates, and query historical prefix sums.
22. Maintain k versions of an array; for each version, answer kth smallest in range `[l, r]`.
23. Count number of elements ≤ x in a historical version’s subarray.
24. For each update, create a new version of the array and answer range max.
25. Persistent trie using segment tree for max XOR queries at historical time.
26. Rollback to any previous version and answer sum queries.
27. For each update, store previous state and allow querying any older version’s range sum.
28. Perform path queries on persistent tree where each path stores its own version of data.
29. Persistent segment tree to find k-th missing number after deletions.
30. Maintain an array with undo support: allow undoing last k updates and answer max in range.

Would you like:

- 🔧 Implementation strategies per category?
- 🔗 Codeforces / AtCoder / LeetCode problem links?
- 🧪 Test cases or templates for any specific type?

Just ask!

You said:

Strong Graph Algorithms Tarjan's (SCC, Bridges), 2-SAT, Dijkstra + Heap, BFS 0-1

ChatGPT said:

Here are **30 one-line problem statements** from **Strong Graph Algorithms**, covering:

- ✅ **Tarjan's Algorithm** (SCCs, Bridges, Articulation Points)
- ✅ **2-SAT**
- ✅ **Dijkstra (with Min-Heap)**
- ✅ **0-1 BFS**

◆ Tarjan's Algorithm – SCCs (Strongly Connected Components)

1. Find the number of strongly connected components in a directed graph.
2. Collapse SCCs into a DAG and find longest path in the DAG.
3. Detect if a directed graph is strongly connected (one SCC).
4. Count the number of SCCs with zero in-degree in the condensation graph.
5. Given a graph, find minimum edges to add to make it strongly connected.
6. For each node, determine its SCC ID and list all nodes in the same SCC.
7. Count total number of cycles in the SCC-decomposed graph.
8. Remove all cycles from a graph by deleting minimum number of edges.
9. Find nodes involved in any cycle using SCC decomposition.
10. Find largest SCC by node count in the graph.

◆ Tarjan's Algorithm – Bridges & Articulation Points

11. Find all bridges (critical connections) in an undirected graph.
12. Count articulation points in an undirected graph.
13. After removing one node, check if the graph remains connected.
14. Given a network, find which single edge failure will disconnect the graph.
15. Number of biconnected components in an undirected graph.
16. Label each edge as a bridge or not in an undirected graph.
17. Check whether a given undirected graph is 2-edge-connected.
18. Check whether a given undirected graph is 2-vertex-connected.
19. For each bridge, calculate the number of nodes separated by its removal.

20. Compress the graph into bridge-tree and answer path queries on it.

◆ 2-SAT (Implication Graph + SCC)

21. Determine if a given 2-SAT instance is satisfiable.
22. Find one valid assignment of variables for satisfiable 2-SAT instance.
23. Count number of satisfying assignments (if small `n`).
24. Convert XOR/OR/AND constraints into 2-SAT and solve.
25. Find minimum number of variables to fix to satisfy the formula.
26. Model friend-enemy relationships as 2-SAT and check for consistency.
27. Solve scheduling constraints using 2-SAT.
28. Given N switches and lights, determine if it's possible to satisfy light configurations.
29. Model graph bipartiteness as 2-SAT instance with parity constraints.
30. Given preferences, assign teams such that no constraints are violated using 2-SAT.

◆ Dijkstra with Min-Heap + BFS 0-1

31. Dijkstra: Find the shortest path from node `1` to node `n` in a weighted graph.
32. Dijkstra: Count number of shortest paths from `s` to `t`.
33. Dijkstra: Find shortest path that avoids a set of forbidden edges.
34. Dijkstra: With edge weights ≤ 1e9 and N ≤ 1e5, compute all-pairs shortest path from K sources.
35. Dijkstra: Add toll tax and optimize path cost with multiple currencies.
36. BFS 0-1: Find shortest path in graph with 0-1 weights using deque.
37. BFS 0-1: Minimum steps to reach target node using teleporters (cost 0) and roads (cost 1).
38. BFS 0-1: Solve shortest binary string transformation using 0/1 cost operations.
39. BFS 0-1: Minimize number of switches needed to reach end of maze with locked doors.
40. BFS 0-1: Determine minimum number of swaps (0-cost and 1-cost) to sort an array using a graph.

Let me know if you want:

- 📚 Tutorials or visual explanations
- 🔗 Problem links from platforms (Codeforces, CSES, AtCoder, etc.)
- □ How to choose the right algorithm among these for a given problem

I'm happy to help further!

You said:

Bitmask + DP / Graphs Held-Karp, Bitmask on Graph

ChatGPT said:

Here are **30 one-line problem statements** based on **Bitmask + DP / Graphs**, including:

- ✅ **Held-Karp Algorithm** (TSP with Bitmask DP)
- ✅ **Bitmask DP on Graphs** (subset DP, state compression, paths/covering cycles, etc.)

◆ Held-Karp Algorithm (Traveling Salesman Problem with Bitmask + DP)

1. Find the minimum cost to visit all cities exactly once and return to the start.
2. Given a symmetric cost matrix, solve TSP starting and ending at node 0.
3. TSP where visiting certain cities is optional; find minimum cost covering at least `k`.
4. Compute minimum TSP path visiting all capital cities in a country.
5. TSP where some cities can only be visited after others (partial ordering).
6. Find the number of distinct TSP tours with minimum cost.
7. TSP with time windows — arrive at each city within a given range.
8. Given N keys in different cities, find minimum cost path collecting all keys.
9. Solve asymmetric TSP with directed weighted edges using Held-Karp.
10. Find minimum TSP path while skipping at most one city.

◆ Bitmask DP on Graphs – Subset-Based State Compression

11. Count number of Hamiltonian paths from node 0 in a directed graph.
12. Count number of ways to visit all nodes exactly once from a given start.
13. Bitmask DP to find minimum cost to connect a subset of nodes.
14. DP: Find shortest path covering all special nodes using bitmask state.
15. Bitmask DP for counting number of perfect matchings in a bipartite graph.
16. Count number of independent sets in a graph using bitmask DP.
17. Compute longest path in DAG visiting a subset of nodes using bitmask.
18. Bitmask DP to count number of ways to partition graph nodes into cliques.
19. Given costs between sets of nodes, find min-cost way to assign them disjointly.
20. Bitmask DP to calculate minimum vertex cover in a bipartite graph.

◆ Bitmask + Graph Variants

21. Given an N×N grid, find minimum moves to visit all marked cells using BFS + bitmask.
22. Solve shortest path with multiple states (position + items collected).
23. Compute all-pairs shortest paths among k special nodes using bitmask over visited set.
24. Given directed edges and portals, use bitmask to track teleportation states.
25. Bitmask + BFS: Collect all coins in a maze with minimal steps.

26. Bitmask DP: Maximize points while visiting cities with limited energy.
27. Minimum number of steps to visit all switches in a graph-controlled room system.
28. Bitmask + DP: Minimum lights needed to cover all nodes in a tree.
29. Bitmask to represent all on/off configurations and find minimum toggles.
30. Given N machines, compute the optimal path to turn them all on with cost constraints.

Want me to:

- 📎 Provide actual problem links from Codeforces / AtCoder / CSES?
- ✍ Explain a few of these with state transitions or recursion + memoization?
- ⚙ Share template code for Held-Karp or Bitmask BFS?

Just say the word!

You said:

Mathematics Modular Inverse, CRT, Euler's theorem, Number Theoretic Transforms (NTT)

ChatGPT said:

Here are **30 one-line problem statements** from advanced **Mathematics topics**, including:

- ✅ Modular Inverse
- ✅ Chinese Remainder Theorem (CRT)
- ✅ Euler's Theorem
- ✅ Number Theoretic Transform (NTT)

◆ Modular Inverse

1. Given `a` and `m`, find `a⁻¹ mod m` (modular inverse), or report if it doesn't exist.
2. Compute `(A / B) % M` under modulo where `M` is prime.
3. Precompute modular inverses of all numbers up to `n` modulo a prime `p`.
4. Given factorials modulo `p`, compute `C(n, r) % p` using modular inverse.
5. Solve `a * x ≡ b (mod m)` using modular inverse.
6. Find inverse of product of array elements under modulo prime.
7. Given array `A`, compute sum of `A[i] / B[i]` modulo `p` efficiently.
8. For each query, compute modular inverse of `x mod p` for large number of queries.
9. Use modular inverse in Fermat's Little Theorem to compute `a^b / c^d % p`.
10. Compute inverse of a polynomial modulo a prime.

◆ Chinese Remainder Theorem (CRT)

11. Solve system of congruences: `x ≡ a1 (mod m1), x ≡ a2 (mod m2), ...`.
12. Given remainders mod 3, 5, 7, find smallest `x` satisfying all.
13. For given `k` congruences with coprime moduli, compute x modulo `M = m1*m2*...*mk`.
14. Solve x such that `x ≡ a mod m` and `x ≡ b mod n`, where `m` and `n` are not coprime.
15. Apply CRT to combine results of large modulo operations.
16. Combine multiple polynomial results under different moduli using CRT.
17. Solve simultaneous congruences with overflow-safe version of CRT.
18. Find smallest positive integer satisfying multiple time remainders.
19. Count how many numbers less than N satisfy given set of congruences.
20. Use CRT to reconstruct number modulo `MOD = 1e9+7` from mod1, mod2, mod3.

◆ Euler's Theorem

21. Compute `a^b mod m` where `a` and `m` are coprime using Euler's theorem.
22. Compute $a^{-1} \bmod m$ using Euler's theorem if `gcd(a, m) = 1`.
23. Precompute `φ(n)` for all `n ≤ 10^6` and use in queries.
24. Solve `a^b mod m` where `b` is large (up to 10^9) and `m` is not necessarily prime.
25. Reduce large powers using Euler's totient: `a^b mod m` when `b > m`.
26. Euler's Theorem based fast exponentiation modulo non-primes.
27. Compute sum of divisors of `n` using φ function.
28. Count numbers ≤ n that are coprime to n using Euler's totient.
29. Solve `x^k ≡ a mod m` using Euler's theorem and discrete log tricks.
30. Compute Carmichael's function and use it to find tighter modulus reduction than φ(n).

◆ Number Theoretic Transform (NTT)

31. Multiply two polynomials modulo a prime using NTT.
32. Compute convolution of arrays `A` and `B` modulo `998244353`.
33. Implement NTT-based multiplication for degree `10^5` polynomials.
34. Use NTT to count number of ways to reach a sum using dice throws.
35. Multiply binary strings by treating them as polynomials (modulo prime).
36. Perform range convolution efficiently using precomputed NTT.
37. Use NTT to compute `(f(x))^k mod x^n` in O(n log n).
38. Compute inverse of a polynomial using NTT + Newton-Raphson.
39. Use NTT to find product of polynomials with coefficients under 10^9.
40. Combine NTT + CRT to do arbitrary modulo polynomial multiplication.

Let me know if you'd like:

- □ Templates for modular inverse (Fermat / Extended Euclidean), CRT, NTT
- 🔗 Competitive problems from Codeforces, AtCoder, CSES using these topics
- 🗒 Study plan to master each topic step by step

I'm happy to assist!

You said:

Constructive Algorithms Greedy with strong observations, output-building under constraints

ChatGPT said:

Here are **30 one-line problem statements** on **Constructive Algorithms**, including:

- ✅ Greedy with Strong Observations
- ✅ Output-Building under Constraints

◆ Greedy with Strong Observations

1. Given an array, rearrange it so that no prefix sum is zero.
2. Construct the lexicographically smallest permutation with given number of inversions.
3. Given string `s`, convert it to palindrome by inserting minimum characters.
4. Rearrange array so that adjacent elements have GCD = 1.
5. Create a binary string with equal number of `01` and `10` substrings.
6. Given `n`, construct a tree with `k` leaves or output "Impossible".
7. Given frequency of digits, form the smallest number divisible by 3.
8. Given array, select elements to form a strictly increasing subsequence of length `k`.
9. Construct a graph with `n` nodes and `m` edges such that the diameter is exactly `d`.
10. Construct permutation `p` such that for all `i`, `|p[i] - i| ≠ k`.

◆ Output-Building under Constraints

11. Given a number `n`, construct a binary string of length `n` with no two consecutive 1s.
12. Construct an array of size `n` such that sum = `S` and GCD of adjacent elements is 1.
13. Build a string of length `n` using at most `k` characters such that no substring of length ≥ 3 is palindrome.
14. Construct a matrix with given row and column sums.
15. Rearrange array so that adjacent differences alternate between positive and negative.
16. Given a set of numbers, divide them into two groups such that their sums are equal.
17. Generate `n` integers such that no two subsets have same XOR.
18. Given a graph with `n` nodes, construct a spanning tree with even number of leaf nodes.
19. Create a password of length `n` using `k` different characters that avoids a forbidden substring.
20. Design a circular seating arrangement such that no adjacent persons are enemies.

◆ More Greedy + Construction Problems

21. Fill a grid with numbers so that each row and column is strictly increasing.
22. Construct array of `n` integers such that LCM of all pairs is same.
23. Create a string such that it has exactly `k` palindromic substrings.
24. Given a permutation, perform minimum adjacent swaps to sort it (output the swap operations).
25. Build a string `t` such that `s` is a subsequence of `t` and `t` has minimum length.
26. Construct a polygon with integer sides of length `a1, a2, ..., an` such that it's non-degenerate.
27. Build a tree with minimum height and given degree constraints.
28. Rearrange digits of `n` to form the smallest number divisible by `m`.
29. Build an `n x n` matrix with 0s and 1s such that each row/column has exactly `k` ones.
30. Create a path through all cells of an `n x m` grid such that no adjacent move is in the same direction as previous.

Would you like:

- 🛠 Code templates for popular constructive patterns (e.g. matrix filling, permutation building)?
- 🔗 Real contest problems grouped by each category?
- □ Explanation on how to "spot" constructive patterns during contests?

Let me know how you'd like to continue!

You said:

DSU on Tree, MO's Algorithm Offline queries, subtree answers

ChatGPT said:

Here are **30 one-line problem statements** on advanced techniques:

- ✅ **DSU on Tree** (a.k.a. Small-to-Large, subtree answers)
- ✅ **Mo's Algorithm** (sqrt decomposition for offline queries)
- ✅ Covers **Offline Queries**, **Subtree Queries**, and **Heavy-Light Queries**

◆ DSU on Tree – Small-to-Large for Subtree Answers

1. For each node, count the number of distinct colors in its subtree.
2. For each node, find the most frequent value in its subtree.
3. For each node, calculate how many nodes in its subtree have a greater value.
4. For each node, compute the sum of values in its subtree.
5. Find the color with the maximum frequency in each node's subtree.
6. Count number of colors that appear exactly once in each node's subtree.

7. For each node, compute XOR of all values in its subtree.
8. Count how many nodes in each subtree have the same depth.
9. For each node, find how many nodes in its subtree are divisible by a given number `k`.
10. Given a label for each node, count how many times it appears in the node's subtree.

◆ Mo’s Algorithm – Offline Range Query Optimization (sqrt decomposition)

11. Given an array, answer offline queries of form: count of distinct elements in range `[L, R]`.
12. For each query `[L, R]`, find frequency of most frequent element.
13. For each query, determine if any number appears more than half the time in `[L, R]`.
14. Find sum of squares of frequencies of elements in a range.
15. Count how many elements in `[L, R]` are less than or equal to `X`.
16. Count number of pairs `(i, j)` in `[L, R]` such that `A[i] = A[j]`.
17. Find the number of subarrays within `[L, R]` with XOR = 0.
18. For each query `[L, R]`, determine how many primes are in that range.
19. Find number of unique elements whose frequency is at least `k` in `[L, R]`.
20. Compute the mode (most frequent value) in each range query `[L, R]`.

◆ Tree + Mo’s Algorithm (Euler Tour + Mo on Trees)

21. For each query, find number of distinct labels in the subtree of node `u`.
22. For each query, check if a certain label appears in the subtree of node `u`.
23. For a rooted tree, process offline queries to count values with frequency > 1 in subtree.
24. For each node, answer whether its subtree contains at least `k` distinct values.
25. Given queries on node-subtrees, compute number of nodes with values in given range.
26. Count number of even-valued nodes in subtree of given node `u`.
27. For each subtree, find max element in subtree using offline queries.
28. Find number of paths passing through a node which contain at least one duplicate.
29. Perform Mo's algorithm over DFS-in/out times to solve subtree path queries.
30. Combine Mo’s and Euler tour to compute XOR of values in path or subtree queries.

Would you like:

- □ Explanation of **when to use DSU-on-tree vs Mo's on Tree**
- 📎 Real problems using these (Codeforces, CSES, AtCoder)
- 🛠 Template code for Mo’s Algorithm on array/tree and DSU on Tree?

Let me know, and I’ll set you up!

You said:

Game Theory Basics Grundy numbers, Sprague-Grundy Theorem

ChatGPT said:

Here are **30 one-line problem statements** based on **Game Theory Basics**, including:

- ✅ **Grundy Numbers**
- ✅ **Sprague-Grundy Theorem**
- ✅ **Nim Game Variants**

◆ Grundy Numbers – Single Pile / Game Position Evaluation

1. Given `n` stones, each player can remove 1, 3, or 4 stones per turn — who wins?
2. Compute Grundy number for game where moves are subtracting perfect squares from a pile.
3. Determine winner for a pile where allowed moves are any divisor of the current number.
4. Find Grundy numbers for a graph-based game where a move goes to a smaller node.
5. Given `n`, determine winning player if each move halves the number (if even).
6. Determine Grundy numbers for piles where allowed moves are `[1, 3, 5, 7]`.
7. Find winner of the game where each player reduces the pile to a divisor less than itself.
8. Grundy numbers where state `n` has transitions to `n-1`, `n/2`, and `n-3`.
9. Construct Grundy number table for small pile sizes with given move set.
10. Compute Grundy values of positions in a stone game on a graph.

◆ Sprague-Grundy Theorem – Multiple Independent Games

11. You are given multiple piles; player can remove `1`, `3`, or `4` stones — who wins?
12. Given `k` independent piles, determine winner by XORing Grundy numbers.
13. Given multiple nim piles, determine if the first player has a winning strategy.
14. Combine several distinct games (e.g., subtraction games) and compute combined Grundy XOR.
15. Given several game graphs, decide winner using Sprague-Grundy.
16. Each node in a tree is a game; determine winner over combined tree-nodes.
17. Combine Grundy values from independent coin-path games on different grids.
18. Given multiple independent games with same rules, compute total Grundy XOR.
19. Determine winner when combining different types of pile games.
20. Multiple heaps with different move sets; apply SG theorem to determine the winner.

◆ Nim Game and Variants – Classic & Tweaks

21. Standard Nim: Given piles, each player removes any number of stones — who wins?

22. Misère Nim: Same as Nim, but the last move loses — determine the winner.
23. Nim Sum Problem: Given XOR of all piles, determine if current player has winning move.
24. Modified Nim: Players can only remove ≤ `k` stones per move.
25. Chocolate Bar Game: Remove rows/columns; reduce to Nim and solve.
26. Circle Nim: Circular arrangement of piles, no two adjacent can be played consecutively.
27. Wythoff's Game: Move either left/down or diagonally on a board — who wins?
28. Grundy number for a 1D board game with blocked cells.
29. Square-tile game: Place 1×1 or 1×2 tiles in a strip of `n` cells — who wins?
30. Game on binary strings: Flip bits from `1` to `0` in allowed positions — determine winner.

Would you like:

- 📑 Grundy function implementation templates
- 💡 How to convert problems into impartial games for Sprague-Grundy?
- □ Test cases and example walkthroughs for tricky positions?
- 🔗 Real problems from Codeforces / HackerRank / CSES using Grundy numbers?

Let me know!