Problem A. Advanced Monopoly

1 Optimal Strategy

From the profit formula, we observe that each node's contribution is independent. The optimal strategy for both players is to select nodes in descending order of:

$$\text{Priority}(x) = \text{size}[x] - \text{depth}[x] - w_x$$

where:

- size$[x]$: Number of nodes in the subtree rooted at x
- depth$[x]$: Depth of node x in the tree (the root has depth 0)

2 Solution Approach

1. Precompute for each node x:
   - Subtree size size$[x]$
   - Depth depth$[x]$
   - Priority value Priority(x)
2. Sort all nodes by Priority(x) in descending order
3. Alternate selecting nodes between players according to this sorted order
4. Calculate the first player's total profit by summing:

$$\sum_{x \in \text{First player's nodes}} (\text{size}[x] - \text{depth}[x] - w_x)$$

Problem B. Bonnie and the Crumb Critic

3 Problem Description

A cupcake is represented by the number of chocolate chips already placed on it. Initially there are infinitely many cupcakes with 0 chips. During one minute Bonnie can add exactly k chips in total, distributed among cupcakes, and then the Crumb Critic removes one unfinished cupcake. Since the critic wants to delay Bonnie, we may assume he removes a cupcake with maximum current progress.

We need to determine the minimum number of minutes required to make some cupcake reach n chips before it can be removed.

4 Solution Approach

4.1 Initial Observations

- An optimal strategy distributes chocolate chips as evenly as possible among the relevant cupcakes.
- A natural approach is binary search on the answer plus simulation.
- Direct simulation is inefficient for large n.

4.2 Optimized Algorithm

1. **Reverse Calculation**:
   - Work backwards from the target state: one cupcake already has n chips.
   - Compress groups of identical states instead of simulating every cupcake independently.
2. **Special Cases**:
   - There is no solution when $n > 1$ and $k = 1$.
   - For $k = 2$ and $n \geq 2$: the answer is 2^{n-2}, so this case requires special handling because the answer may be very large.
3. **Implementation Notes**:
   - A C++ implementation with 18-bit compression runs in about 800 ms.
   - A Python implementation runs in about 1600 ms.
   - Precomputation for $(n = 70711, k = 3)$ can halve runtime.

5 Complexity Analysis

- The original binary-search approach with direct simulation is too slow.
- The optimized reverse calculation compresses many identical transitions and is fast enough for the constraints.
- Special case handling prevents TLE and avoids unnecessarily expensive big-integer transitions.

6 Key Insight

The distribution process follows a "water-filling" pattern: keeping the relevant cupcakes as balanced as possible minimizes the number of minutes needed before one of them reaches n chips. The reverse calculation models this process without explicitly simulating every minute.

7 Formulas

We work backwards. At the end, there is a single cupcake with n chips. During the reverse process, it is enough to maintain two groups:

- x cupcakes with m chips;
- y cupcakes with $m - 1$ chips.

The cupcake with m chips is the one that will eventually become the finished cupcake.

Going backwards increases the number of cupcakes. While we have fewer than k relevant cupcakes, the transition is simple: in the corresponding forward minute, every relevant cupcake could have received at least one chip.

Suppose we currently have x cupcakes with m chips and y cupcakes with $m-1$ chips. One minute earlier, before the critic removed a cupcake, there were $x + y + 1$ relevant cupcakes. Let

$$q = \left\lfloor \frac{k}{x + y + 1} \right\rfloor, \qquad r = k \bmod (x + y + 1).$$

Every relevant cupcake received q chips, and r of them received one extra chip.

If $x < r$, then every cupcake with m chips received $q+1$ chips in that minute, and the removed cupcake also belonged to this extra group. Therefore

$$m := m - (q+1), \qquad x := x + y - (r - x - 1) + 1, \qquad y := r - x - 1.$$

Otherwise, not all cupcakes with m chips received an extra chip, so

$$m := m - q, \qquad x := x - r + 1, \qquad y := y + r.$$

The $+1$ in both transitions accounts for the cupcake removed by the critic in the forward process.

When the number of relevant cupcakes is greater than k, one forward minute cannot increase every relevant cupcake by one chip. We divide the process into groups of size k. In each group, all cupcakes that survive receive one chip, and one more cupcake is the one removed by the critic. Going backwards, this means grouping by $k-1$. Let

$$q = \left\lfloor \frac{x}{k-1} \right\rfloor, \qquad r = x \bmod (k-1).$$

After q reverse steps, m decreases by 1. Also, y is always less than k.

If $r = 0$, then all cupcakes with m chips belong to complete groups, and no cupcake with $m-1$ chips is used inside them. Thus

$$m := m - 1, \qquad x := q \cdot k + y, \qquad y := 0.$$

If $r \neq 0$, then m also decreases by 1. The $(q+1)$-th group consists of one removed cupcake with $m-1$ chips, r surviving cupcakes with $m-1$ chips, and $k-r-1$ cupcakes with $m-2$ chips. The remaining $y-(k-r-1)$ cupcakes with $m-1$ chips stay unchanged. Therefore the new state is

$$x := q \cdot k + 1 + r + y - (k - r - 1), \qquad y := k - r - 1.$$

Every forward minute removes exactly one unfinished cupcake, so the total number of minutes is the number of removed cupcakes reconstructed by the reverse process.

Problem C. Clockwork Concert in D

8 Problem Statement

Given n numbers $a_1, a_2, \ldots, a_n$, for each pair (i, j) where $1 \leq i < j \leq n$, we:

- Remove a_i and a_j
- Add $a_i + a_j$ to the sequence
- Compute the LCM of the resulting sequence

Find the sum of all such LCMs modulo 998 244 353.

Constraints: $n \leq 5 \times 10^5$, $1 \leq a_i \leq 10^6$.

9 Definitions

For prime p and positive integer A:

- $v_p(A)$: Exponent of p in A's prime factorization ($p^{v_p(A)} \mid A$ but $p^{v_p(A)+1} \nmid A$)
- $M(p) = \max\{v_p(a_1), \cdots, v_p(a_n)\}$
- $m(p)$: Second largest value in $\{v_p(a_i)\}$ (equal to $M(p)$ if maximum appears multiple times)

The original LCM is $\prod_{p \in \text{Prime}} p^{M(p)}$.

10 Key Theorem

When replacing a_i and a_j with $a_i + a_j$, the change in p-exponent for the LCM is:

$$\Delta M_p(a_i, a_j) = [M(p) = v_p(a_i)](m(p) - M(p)) + [M(p) = v_p(a_j)](m(p) - M(p)) + \max(v_p(a_i + a_j) - M(p), 0)$$

11 Proof Sketch

Define indicators:

- $f_i = [M(p) = v_p(a_i)]$
- $f_j = [M(p) = v_p(a_j)]$
- $f_k = [v_p(a_i + a_j) \geq M(p)]$
- $f_l = [m(p) = M(p)]$

Case analysis shows the formula holds for all combinations of (f_i, f_j).

12 Solution

Define functions:

$$f(i) = \prod_{p \in \text{Prime}} p^{[M(p) = v_p(i)](m(p) - M(p))}$$

$$g(j) = \prod_{p \in \text{Prime}} p^{\max(v_p(j) - M(p), 0)}$$

The answer is:

$$\sum_{1 \leq i < j \leq n} f(a_i) f(a_j) g(a_i + a_j)$$

13 Implementation

Using generating functions:

- Let $F(x) = \sum_{i=1}^{n} f(a_i) x^{a_i}$
- Compute $F^2(x) \cdot G(x)$ where $G(x) = \sum_{i \geq 0} g(i) x^i$ (here $\cdot$ is a dot product)
- Remove diagonal terms where $i = j$

Complexity: $O((n + \max a_i) \log(\max a_i))$ using efficient prime factorization.

Problem D. Delicious Apples

14 Problem Description

Given a tree with weighted nodes, support the following operations:

- Insert a new node on an existing edge
- Add a new leaf node
- Range addition on a chain (path between two nodes)
- Query the number of nodes $\geq k$ on a chain

15 Solution Approach

15.1 Key Ideas

- Randomly select s key points
- Build a virtual tree from these key points where:
  - Maximum distance from any node to the nearest key point: $O(n/s)$
  - Chain segments between key points and their LCA have an average length of $O(n/s)$
- Maintain an ordered array for each chain segment

15.2 Operations

- **Chain Extraction**:
  - Find LCA using a method similar to heavy-light decomposition
  - Handle scattered parts with brute force
- **Range Addition**:
  - For complete segments: apply lazy tags
  - For scattered parts: brute force modification
- **Query**:
  - For complete segments: binary search on ordered arrays
  - For scattered parts: brute force counting

15.3 Complexity Analysis

- Single operation complexity: $O(s \log n + n/s)$
- Optimal $s = \sqrt{n/\log n}$ yields total complexity $O(m\sqrt{n \log n})$
- For insertions:
  - Periodic reconstruction every k operations
  - Additional $O(k)$ cost for scattered parts
  - Reconstruction cost: $O(n \log n)$
  - Total complexity remains $O(m\sqrt{n \log n})$ when $k = \sqrt{n \log n}$

16 Implementation Notes

- Edge insertion and leaf addition can be handled directly
- Dynamic maintenance similar to block-linked lists may be needed for frequent insertions
- Alternative approach: periodic full reconstructions to maintain balance

Problem E. Easy Game For Two Novices

17 Problem Statement

Given a card game with:

- n card types, with exactly 2 cards per type (total $2n$ cards)
- Two players (First and Second) each start with n cards
- Players alternate turns playing cards onto a stack
- When two identical cards appear in the stack:
  - Both cards and all cards between them are moved to the current player's hand
- Player loses when they have no cards to play

Additional constraints:

- Second player follows a fixed queue strategy (plays front card, adds won cards to back)
- First player can play any card optimally
- Find the minimal number of moves for First to guarantee victory
- $n \leq 3 \cdot 10^5$

18 Key Analysis

18.1 Lower Bound

The minimal move count is at least n, requiring that the Second player never gains cards.

18.2 Critical Observations

- If Second holds the pair of the bottom stack card, they will eventually claim it
- If First holds the pair, they control the stack by clearing it at will
- The "trump card" (stack pair) should be used as late as possible

18.3 Strategy with Pairs

When First has at least one pair initially:

1. Play one card from the pair initially
2. Clear the stack when Second could gain cards
3. Otherwise play any non-bottom card

This ensures:

- First maintains stack control after clearing
- First always has non-bottom cards to play

18.4 No Initial Pairs Case

When First has all n distinct cards:

- Minimum moves: $n + 2$
- Example strategy against queue $1, 2, \ldots, n$:
  - Play sequence: $n, 1, 2, \ldots, n - 1$
  - Leaves Second with only a pair of n cards

19 Complexity Analysis

The optimal strategy requires:

- $O(n)$ time for pair checking
- $O(1)$ per move decision
- Overall $O(n)$ complexity

Problem F. Fuel For The Lighthouse

20 Problem Description

Given N positive integers $a_1, \ldots, a_N$ and a limit L, two players take turns performing operations:

- Each operation randomly selects one a_i and adds it to the player's current total (initially 0)
- After both players complete their turns, the player with the larger total $\leq L$ wins
- If totals are equal, the first player wins

Compute the probability of the first player winning.
Constraints: $1 \leq N, a_i \leq 2000$, $1 \leq L \leq 10^9$.

21 Key Insight

The second player can optimize their strategy based on the first player's final total X.

22 Solution Approach

22.1 Second Player's Strategy

For a given X, the second player with current total Y:

- If $Y \leq X$: Must draw again
- If $Y > X$: Should stop

The second player maximizes $P(X < Y \leq L|X) = \sum_{y=X+1}^{L} g_X(y)$, where $g_X(y)$ is the probability of ending at $Y = y$.

22.2 Probability Distribution

Define $\pi^{(X)}$ as the distribution when the second player stops at $Y \geq X$:

- Initial $\pi^{(0)} = (1, 0, \cdots)$
- Recurrence: Transfer $\pi_X^{(X)}$ to $\pi_{X+a_i}^{(X+1)}$
- Only non-zero for $X \leq y \leq X + \max a_i - 1$

22.3 Polynomial Representation

The distribution can be computed via polynomial modulo:

$$z^L \mod \left(z^{\max a_i} - \sum_{j=1}^{n} \frac{1}{n} z^{\max a_i - a_j} \right)$$

This gives $\pi^{(L-\max a_i+1)}$ for $y \in [L - \max a_i + 1, L]$.

22.4 First Player's Winning Probability

Define $f(x)$ as the first player's winning probability with current total x:

$$f(x) = \begin{cases} P(X < Y \leq L|X = x) & \text{if stopping} \\ \frac{1}{n} \sum_{i=1}^{n} f(x + a_i) & \text{if continuing} \end{cases}$$

Boundary: $f(x) = 0$ for $x > L$.

23 Complexity Analysis

- Polynomial modulo computation: $O((\max a_i)^2 \log L)$
- Requires careful implementation to avoid TLE
- Single polynomial modulo sufficient (computing twice may timeout)

Problem G. Get Them Equal!

24 Problem Description

Given a sequence of length $4n$ where each integer from 1 to n appears exactly 4 times, determine if it can be partitioned into two equal subsequences. If possible, output the partition scheme.

25 Key Properties

- If a solution exists, there must exist one where for every prefix:
  - The length assigned to subsequence 0 $\geq$ length assigned to subsequence 1
- The partition depends on how pairs of identical numbers are matched

26 Solution Approach

26.1 Feasibility Condition

A solution exists if and only if no two pairs of identical numbers have a containment relationship.

26.2 Pairing Patterns

For each number appearing at positions 1,2,3,4 (in order), there are two valid pairing options:

- Pair (1,3) and (2,4)
- Pair (1,2) and (3,4)

The pairing (1,4) and (2,3) is invalid.

26.3 Algorithm

- Model as a 2-SAT problem with:
  - Variables representing pairing choices
  - Constraints preventing incompatible selections
- The constraints form a 2D partial order relation
- Optimize using Chairman Tree (Persistent Segment Tree) for graph construction
- Time complexity: $O(n \log n)$

Problem H. Hard? NP-Hard!

27 Problem Description

Given:

- n types of items where type i has volume v_i and value c_i
- q queries, each specifying a knapsack capacity x

- Select items (with unlimited copies) such that:
  - Total volume exactly equals x
  - Total value is maximized

Constraints:

- $1 \le n \le 50$
- $1 \le v_i \le 10^5$, $1 \le c_i \le 10^6$
- $1 \le q \le 10^5$
- $10^{11} \le x \le 10^{12}$

28 Solution Approach

28.1 Key Insight

For very large x, the optimal solution consists of:

- Mostly selecting the item with maximum $\frac{c_i}{v_i}$ ratio (denoted as (c_0, v_0))
- Using other items only for the residual volume ($x \bmod v_0$)

28.2 Algorithm

1. Compute the complete knapsack DP array f_i for non-optimal items
2. Define:
$$g_i = \max_{k \ge 0} f_{i+kv_0} - kc_0$$
where g_i represents the optimal residual solution for volume $i \bmod v_0$
3. The DP transition when adding item (c_i, v_i):
$$g_{(j+v_i) \bmod v_0} \leftarrow \max\left(g_{(j+v_i) \bmod v_0}, g_j + c_i - \left\lfloor \frac{j+v_i}{v_0} \right\rfloor c_0\right)$$
4. Handle cyclic transitions by processing each cycle twice

28.3 Complexity

- Preprocessing: $O(n \max v_i)$
- Query: $O(1)$ per query
- Total: $O(q + n \max v_i)$

29 Correctness Proof

- Each residual solution uses $\le v_0$ items
- Maximum residual volume: $\max v_i^2 = 10^{10} \ll V_{min} = 10^{11}$
- Ensures the residual solution always fits within x

Problem I. Invisible Jewels and Judge J

30 Problem Description

Two players A and B simultaneously select integers:

- Player A chooses $x \in \{0, 1, \ldots, n\}$
- Player B chooses $y \in \{0, 1, \ldots, n\}$

Payoff matrix for Player A:

$$\text{Payoff}(x, y) = \begin{cases} x & \text{if } x > y \\ 0 & \text{if } x = y \\ y/2 & \text{if } x < y \end{cases}$$

31 Solution Approach

31.1 Game Decomposition

The game can be decomposed into n 2×2 zero-sum subgames, each deciding whether to select the maximum value n.

	0	1	2	3
0	0	0.5	1	1.5
1	1	0	1	1.5
2	2	2	0	1.5
3	3	3	3	0

31.2 Nash Equilibrium

For each 2×2 subgame with strategies:

	Option 1	Option 2
Player A	x	y
Player B	z	w

Let $(p, q = 1 - p)$ be Player A's mixed strategy probabilities. By Nash equilibrium:

$$px + qz = py + qw$$

Solving this equation gives the optimal mixed strategy for both players.

32 General Solution

- The complete solution can be built either bottom-up or top-down
- Each 2×2 subgame has a unique solution
- Optimal strategies form a probability distribution over $\{0, 1, \ldots, n\}$

Details

1) The 2×2 subgame at level j

At level j each player either:

- **H (High)**: play j, or
- **L (Low)**: "fall back" and play optimally in $\{0, \ldots, j-1\}$.

Let $V := V_{j-1}$ be Player A's value in the smaller game.
The 2×2 payoff matrix to Player A is:

	B:L	B:H
A:L	V	$\frac{j}{2}$
A:H	j	0

Let $p = \Pr(\text{A plays H})$ and $q = \Pr(\text{B plays H})$.

2) Indifference conditions

A indifferent between L and H: Payoff if A plays H:

$$j(1-q).$$

Payoff if A plays L:

$$V(1-q) + \frac{j}{2}q.$$

Setting equal:

$$j(1-q) = V(1-q) + \frac{j}{2}q \quad \Rightarrow \quad q = \frac{j-V}{\frac{3j}{2}-V}. \tag{$\star$}$$

B indifferent between columns L and H: Column L payoff to A:

$$pj + (1-p)V.$$

Column H payoff to A:

$$(1-p)\frac{j}{2}.$$

Setting equal:

$$pj + (1-p)V = (1-p)\frac{j}{2} \quad \Rightarrow \quad p = \frac{\frac{j}{2}-V}{\frac{3j}{2}-V}. \tag{$\star\star$}$$

Thus:

$$p_j = \frac{\frac{j}{2}-V_{j-1}}{\frac{3j}{2}-V_{j-1}}, \qquad q_j = \frac{j-V_{j-1}}{\frac{3j}{2}-V_{j-1}}.$$

3) Value recursion

At equilibrium, A's expected payoff equals either action's payoff when indifferent. Using "A plays H":

$$V_j = j(1-q) = j\left(1 - \frac{j-V_{j-1}}{\frac{3j}{2}-V_{j-1}}\right) = \frac{\frac{j^2}{2}}{\frac{3j}{2}-V_{j-1}} = \frac{j^2}{3j-2V_{j-1}}.$$

With $V_0 = 0$.

4) Solving the recursion

Claim:

$$V_j = \frac{j(j+1)}{2(j+2)}.$$

Proof by induction: Base $j = 0$: $V_0 = 0$ holds.
Assume $V_{j-1} = \frac{(j-1)j}{2(j+1)}$. Then:

$$V_j = \frac{j^2}{3j - 2V_{j-1}} = \frac{j^2}{3j - \frac{j(j-1)}{j+1}} = \frac{j}{\frac{3j - \frac{j(j-1)}{j+1}}{j}} = \frac{j}{\frac{3 - \frac{j-1}{j+1}}{1}} = \frac{j(j+1)}{2(j+2)}.$$

Thus the formula holds for all j.

5) Simplifying p_j and q_j

With $V_{j-1} = \frac{(j-1)j}{2(j+1)}$, compute the common denominator:

$$\frac{3j}{2} - V_{j-1} = \frac{3j}{2} - \frac{j(j-1)}{2(j+1)} = \frac{j(j+2)}{j+1}.$$

Numerators:

$$\frac{j}{2} - V_{j-1} = \frac{j}{2} - \frac{j(j-1)}{2(j+1)} = \frac{j}{j+1},$$

$$j - V_{j-1} = j - \frac{j(j-1)}{2(j+1)} = \frac{j(j+3)}{2(j+1)}.$$

Therefore:

$$p_j = \frac{\frac{j}{j+1}}{\frac{j(j+2)}{j+1}} = \frac{1}{j+2},$$

$$q_j = \frac{\frac{j(j+3)}{2(j+1)}}{\frac{j(j+2)}{j+1}} = \frac{j+3}{2(j+2)}.$$

Conclusion: Once V_{j-1} is in closed form, the mixing probabilities p_j and q_j simplify neatly to:

$$\boxed{p_j = \frac{1}{j+2}, \quad q_j = \frac{j+3}{2(j+2)}}.$$

Problem J. Juggling With Sequences

33 Problem Description

Given a sequence specification where each integer i ($1 \le i \le n$) appears exactly a_i times, construct a permutation of this sequence that maximizes the sum of prefix majority elements.

Constraints: $1 \le a_i, n \le 10^5$.

34 Solution Approach

34.1 Greedy Intuition

The optimal strategy prioritizes:

- Maximizing the number of prefixes where n is the majority
- Then maximizing prefixes where $n-1$ is majority, and so on

34.2 Algorithm

1. First place all a_n copies of n
2. For other numbers i, place $\min(a_n, a_i)$ copies
3. Next place all a_{n-1} copies of $n-1$
4. For other numbers i, place $\min(\max(a_n, a_{n-1}), a_i)$ copies
5. Repeat this process for decreasing values

34.3 Optimization

The naive approach is $O(n^2)$. Instead:

- Insert numbers in rounds: first all 1st occurrences, then 2nd occurrences, etc.
- For each round k, the contribution is max(remaining counts) $\times$ $|\{i : a_i \geq k\}|$
- Maintain counts using a set for $O(\max a_i + n)$ complexity

35 Correctness Proof

The greedy approach achieves the theoretical maximum because:

- For any i, the number of prefixes with majority $\geq i$ is maximized
- The round-based construction matches this upper bound

Problem K. Kinky Chase Festival

36 Problem Description

Players can repeatedly challenge a tower, aiming to achieve exactly i points ($1 \leq i \leq M$) through multiple attempts. Each challenge involves:

- Possible outcomes per floor: failure, success, or success with high honors
- Option to exit after completing a floor
- Penalty mechanism:
  - Failed exit: accumulated points multiplied by penalty coefficient
  - Voluntary exit or full completion: direct accumulation of all points

Constraints: $1 \leq n \leq 6$ (max floors), $1 \leq m \leq 10000$ (target points).

37 Solution Approach

37.1 Dynamic Programming Formulation

Define $f(i, S)$ as the probability of achieving exactly i points from state S:

$$f(i,S) = \begin{cases} f(i - \text{Fail}(S), S_0) & \text{if failure occurs} \\ f(i - \text{Success}(S), S_0) & \text{if voluntary/successful exit} \\ f(i, S') & \text{if continuing challenge} \end{cases}$$

where $\text{Fail}(S)$ and $\text{Success}(S)$ represent points earned in each case.

37.2 Computational Challenges

- Self-loops occur when $\text{Fail}(S) = 0$
- Dependency structure allows sequential computation by i

37.3 Numerical Solution

- **Binary Search Approach**:
  - For each $f(i, S_0)$, perform binary search on probability values
  - Terminate when reaching precision $\epsilon = 10^{-18}$ (long double recommended)
- Complexity: $O(m2^n \log 1/\epsilon)$
- Alternative iterative methods also possible

38 Implementation Notes

- Sample 2 contains critical self-loop cases for testing
- Precision below 10^{-6} may fail due to strengthened test data
- Runtime: ≤ 0.5s for proper implementation (author-tested)

Acknowledgement

The problem background was adapted after the original game's service termination announcement during problem preparation.