Algolympics 2022 Solution Sketch

Problem Discussion

Cisco Ortega

March 26, 2022

Problem D - Dimensional Elevator

Answer: `3*((max(p) - 1) + (n - p.count(1)))`

- We *need* to bring the elevator up to the highest p_i floor that is needed. But we can also drop everyone else off along the way.
- *Everyone* needs to dismount the elevator eventually, except for the people who need to be at floor 1 (who never board the elevator).

These conditions are necessary and sufficient, so this is optimal.

Time Complexity: $\mathcal{O}(n)$ per test case

Problem C - Coin Sort

Intuitive algorithm:

- Find an "inside" subsegment of same-type coins, preferable ones that are the same type as one of the coins in the endpoints—extract it and attach it to that endpoint.
- Repeat until there are no more "inside stacks" (in which case the stack is sorted).

5 10 10 10 5 10 10 5 5 10

Is this correct/optimal? Yes!

Problem C - Coin Sort

Count the number of “contact points”, i.e. indices i such that $coin[i] \neq coin[i+1]$.

- In a sorted stack, there should only be one contact point.

Claim: The algorithm described in the previous slide always decreases the number of contact points by 2, *unless* the stack looks like `5 10 5`, in which case it only decreases by 1.

Thus, the answer is always the number of contact points $\div 2$, rounded down.

Time Complexity: $\mathcal{O}(n)$ per test case.

Problem J - Vintage Enephtys

Note that n is really small.

Just brute force all possible vintage Enephtys and manually compute their C_0, C_1, C_2, C_3, C_4. Count the number of them whose C match the target C.

You can easily iterate over all possible Enephtys using bitmasking (and this probably is a bit faster than recursive backtracking, too).

Time Complexity: $\mathcal{O}(n^2 2^{n^2})$ per test case.

Problem L - Yummy Tea

Simplify the problem!

- Consider only one type of ingredient.

Reframe the problem! For each free real estate on the grid, find the min distance to any `M` square (resp. for other types of ingredients).

- If we can answer this for one ingredient, then we can also precompute the answer for each subset of the ingredients.
- Can be answered by a multi-source BFS.

Time Complexity: Precomputation of $\mathcal{O}(2^x n^2)$, where $x = 5$ in this case is the number of ingredients. Then, $\mathcal{O}(1)$ per query.

Problem H - Specific Gravity

Find a necessary condition: If all rocks have density $< D$, or if all rocks have density $> D$, then it is impossible.

But actually it is sufficient.

- Assume that we have two rocks a and b such that $w_a/v_a \geq D$, and $w_b/v_b \leq D$.
- Suppose we use k_a rocks of type a, and k_b rocks of type b.
- Express the condition algebraically, then do some symbolic manipulations to find a solution for k_a and k_b.

Problem H - Specific Gravity

$$\frac{k_a w_a + k_b w_b}{k_a v_a + k_b v_b} = D,$$

$$k_a w_a + k_b w_b = D k_a v_a + D k_b v_b,$$

$$k_a(w_a - D v_a) = k_b(D v_b - w_b).$$

As a solution, we can take $k_a = D v_b - w_b$ and $k_b = w_a - D v_a$.

But, by our assumption of a and b, these values of k_a and k_b are non-negative! Thus, the condition is sufficient.

Time Complexity: $\mathcal{O}(n)$ per test case

Problem G - Song of Glu Glu

Suppose we have a phrase ending in e'. We want to make it end on e by appending a word to it. What is the max extra power we can get by appending one word? Call this value $best(e', e)$.

$$best(e', e) = \max_{w \text{ such that } w[-1] = e} |e' - w[0]|$$

Problem G - Song of Glu Glu

DP: Let $dp[k][s][e]$ be the max power that can be achieved with k words, where the first letter of the first word is s, and the last letter of the last word is e.

$$dp[k][s][e] = \max_{e'}(dp[k-1][s][e'] + best(e', e)).$$

Question: What should the contents of $dp[1][s][e]$ be, as the base case?
But note that $dp[k]$ *only* depends on the value of $dp[k-1]$. This is a signal that we can use fast matrix exponentiation (or, generally, divide and conquer) to speed up the solution.

Problem G - Song of Glu Glu

We can reduce the problem to evaluating the matrix product $A^{m-1}B$, except instead of $+$ and $\times$, these matrix operations use $\max$ and $+$.

- Try replacing $\max$ and $+$ in the previous page's DP recurrence with $+$ and $\times$. It will look like matrix multiplication!

Time Complexity: $\mathcal{O}(n + |A|^3 \lg m)$, where $|A| = 26$ is the size of our alphabet.

Problem E - Mysterious Symbols

Draw an **directed** edge between squares if it is possible to go from one to the other (if the cyclic order of the symbols is respected).

Question is now: Given a directed graph is it possible to visit all of the vertices, only by following the given edges?

This is standard. "Connectivity" in a directed graph is a telegraphed signal for using **strongly connected components**.

Problem E - Mysterious Symbols

Draw an **directed** edge between squares if it is possible to go from one to the other (if the cyclic order of the symbols is respected).

Question is now: Given a directed graph is it possible to visit all of the vertices, only by following the given edges?

Solution:

- Use Tarjan's or Kosaraju's to find the strongly connected components of the graph.
- Condense the graph to its SCCs, turning it into a DAG, which is easier to work with.
- Use DP to get max length path in the DAG, and check that this is equal to the number of SCCs.

Time Complexity: $\mathcal{O}(RC)$ per test case.

Problem K - War of Human Conquest

What if there were only one factory?

- **Idea 1:** Use line sweep. Simulate people entering and exiting their shifts, keeping track of the number of people in the factory at each point in time.
- **Idea 2:** Use a segment tree with range max query and range add updates, to track the number of people in the warehouse at each point in time. Considering someone's shift from L to $R \to +1$ to the body count for times L to R.

Problem K - War of Human Conquest

What if there are **two** factories? Combine both ideas!

- Prime the segment tree with range updates so that it contains the amount of people in the second factory at each point of time.
- Do a line sweep. Simulate people entering and exiting their shifts, keeping track of the number of people in the factory at each point in time.
- When person i enters the first factory, apply -1 to X_i to Y_i in the second factory. When they exit the first factory, apply $+1$ to X_i to Y_i.
  - If we throw the first bomb now and kill this person, then they can't be killed again in the second factory; so, remove them.
  - But once they leave, throwing the bomb won't kill them; so, consider them again in the second factory.

Problem K - War of Human Conquest

What if there are **two** factories? Combine both ideas!

- Prime the segment tree with range updates so that it contains the amount of people in the second factory at each point of time.
- Do a line sweep. Simulate people entering and exiting their shifts, keeping track of the number of people in the factory at each point in time.
- When person i enters the first factory, apply -1 to X_i to Y_i in the second factory. When they exit the first factory, apply $+1$ to X_i to Y_i.
  - If we throw the first bomb now and kill this person, then they can't be killed again in the second factory; so, remove them.
  - But once they leave, throwing the bomb won't kill them; so, consider them again in the second factory.

Problem K - War of Human Conquest

Implementation considerations:

- Coordinate compression
- To handle queries that spill over past midnight, you can consider *two days in a row*, where the second day is a copy of the first one.
  - Slightly more inefficient by a small constant factor, but makes your code so much neater!

Time Complexity: $\mathcal{O}(n \lg n)$.

Problem I - Teleporter Simulation

Main Insight: This scheme creates different *permutations*, and permutations are objects which can be combined and composed in and of themselves.

- You can even make a segment tree of permutations!

Problem I - Teleporter Simulation

- Naive will get you TLE, worst case 50k bridges and 50k queries. You need to process all swaps until some row Y.
- Bridges represent swaps, and groups of bridges are permutations.
- One way is through segment tree. Assume that all bridges are known beforehand, and are activated/deactivated based on commands.
- For each node in the segment tree, store the effective permutation and how may bridges passed. Permutations can be composed. Similarly, bridges passed can be computed

Time Complexity: $\mathcal{O}(nm \lg m)$

Problem A - Sushiimon

Let $R[k] = r_1 + r_2 + \cdots + r_k$, and define $B[k]$ similarly. Also, let $dp[k]$ be the optimal answer to get to research level k. Then, we have the following recurrence:

$$dp[i] = \min_{0 \le j \le i} (B[j] + R[i-j]),$$

(considering only the j such that both j and $i - j$ lie in $[0, n]$).

With DP, the complexity is $\mathcal{O}(n^2)$.

Problem A - Sushiimon

There exists a well-known trick call **Divide and Conquer Optimization** for DP.

- Consider the j that makes $dp[i]$ optimal; call it $A[i]$, as in this is the **argument** that makes the min optimal.
- Because r is non-decreasing, we can show that as i increases, $A[i]$ is also non-decreasing!
- Leveraging this insight is the basis for Divide and Conquer Optimization.

Problem A - Sushiimon

No time to explain this beautiful technique in a way you'll appreciate!

- You can read the tutorial of "Ciel and Gondolas" (Codeforces) or "Guardian of the Lunatics" (Hackerrank)
- There was also a paper describing that technique in a cleaner way.
  - I can't find it at the moment, but I swear it exists.
  - To follow na lang sa Discord!

Time Complexity: $\mathcal{O}(n \lg n)$.

Problem B - Alien Chess

Common pattern in combinatorial game theory: Suppose that whatever move my opponent makes, I *always* have a “response” move. Then, I win, because my opponent will always be the one to run out of moves first.

Problem B - Alien Chess

Main insight: find a **maximum matching** of squares.

If a perfect matching exists, let your opponent choose the starting position of the knight.

- Whenever your opponent moves the knight somewhere, your response is to move the knight to the matched square.
- Since you always have a valid response to any move your opponent makes, you are guaranteed to win.

Problem B - Alien Chess

If no perfect matching exists, then take any maximum matching. You should select the starting position of the knight, and place it on any unmatched square.

- Your opponent's move always moves it to a matched square.
- From here, do the same strategy: wherever the opponent moves the knight, you respond by moving it to its matched square.
- The opponent's moves always move the knight to a matched square, because if not, then we would have an augmenting path and the matching wouldn't have been maximal.

Problem B - Alien Chess

But finding a maximum matching is easy because the knight-graph is bipartite!

Time complexity: $\mathcal{O}(S\sqrt{S})$ using Hopcroft-Karp to find a maximum matching, where $S = 64$ in our case. But because S is small, generic max flow algorithms (even Ford Fulkerson) will pass for this problem.

Problem F - End of the Universe

Represent each GC with a bitmask `mask`, where the ith bit is 1 if the ith immortal is in this GC, and 0 if they are not.

Let f_{mask} be the **count** of the number of GCs that have this bitset of mask.

Let $\text{exact}[\text{mask}]$ count the number of sequences which ultimately end with *exactly* the immortals in mask being invited. If we have this for all mask, then we can use DP to compute for the answers involving don't-care people.

Problem F - End of the Universe

If $k = 2$, then we want to compute,

$$\text{exact}[\text{mask}] = \sum_{i \text{ or } j = \text{mask}} f_i \times f_j,$$

For general k, we want to compute,

$$\text{exact}[\text{mask}] = \sum_{\text{OR } i_t = \text{mask}} f_{i_1} \times f_{i_2} \times \cdots \times f_{i_k}.$$

for all values of mask.

This looks a lot like polynomial multiplication!

Problem F - End of the Universe

Solution: Do something similar to the Fast Walsh-Hadamard Transform, except modify the magic matrix so that it performs an `OR` convolution instead of an `XOR` convolution.

Once you've computed exact[mask] for all values of mask, you can do DP to handle the do-not-cares, in order to answer each query.

Time Complexity: Precomputation of $\mathcal{O}(mn + n2^n \log k + 3^n)$, then $\mathcal{O}(1)$ per query.