Algolympics 2015

Solution Sketches

Problem B: Make Gawa This Program

- Just follow instructions carefully!
- Tests accuracy, not running time.
- General Tips:
  - Code defensively.
  - Think of lots of corner cases.
  - Samples not representative of actual data.
  - Look at constraints, try extremes.

Problem D: Slicing Number Cakes

- Brute force: Try all possible cuts.
- Too slow.

Problem D: Slicing Number Cakes

- Brute force 2: recursively compute best(i, j)
  - best partition of last i digits with exactly j slices.
- $best(i, j) = \max(1 \le k \le i)\ best(k-1, j-1) + N[k..i]$
- best(i, 0) base case, single partition.
- Too slow, still tries everything.

Problem D: Slicing Number Cakes

- Insight: best(i, j) only depends on i and j, not on previous choices beyond digit i.
- **Memoize results** in a 2D table.
- Only need to compute each entry once.
- Running time proportional to:
  - size of table × time to compute each entry.
  - O(digits(N) × digits(N) × S).

Problem D: Slicing Number Cakes

- Alternatively, compute table best(i, j) bottom-up.
  - Also called **dynamic programming**.
- Same running time, but probably slightly faster.
  - O(digits(N) × digits(N) × S).

Problem F: Alien Defense Deux

- Sol. 1: Try all squares, count empty ones.
- Too slow.
  - ~$O(n^5)$

Problem F: Alien Defense Deux

- Sol. 2: For each top-left corner, try all squares in increasing size incrementally.
- Optimizations:
  - Process only “additional” squares as you go.
  - Break on the first non-empty square.
- Faster, but still too slow in the worst case.
  - ~$O(n^4)$

Problem F: Alien Defense Deux

- Sol. 3: Precompute table sums, so each square can be checked non-empty in O(1).
  - Also called **sparse tables**.
- Faster, but still too slow in the worst case.
  - ~$O(n^3)$

Problem F: Alien Defense Deux

- Sol. 4: Binary search the largest nonempty square.
- Now passes! (with reasonable implementation)
  - ~$O(n^2 \log n)$

Problem F: Alien Defense Deux

- Sol. 5: Dynamic programming: f(i, j) = largest nonempty square with top-left corner (i, j).
  - Can you find the recurrence?
- Optimal!
  - $O(n^2)$

Problem C: Rigid Trusses

- Insight 1: “horizontal bars” in each column parallel.
- Insight 2: “vertical bars” in each row parallel.
- Both properties of rhombuses.

Problem C: Rigid Trusses

- Insight 3: If (i_1, j_1), (i_1, j_2), (i_2, j_1) rigid, then (i_2, j_2) also rigid.
- Insight 4: Any rigid cell is either initially rigid or can be shown rigid by repeatedly applying Insight 3.

Problem C: Rigid Trusses

- Insight 5: Insight 3 and 4 equivalent to **connectivity in bipartite graph!**
  - "Insight 3": If (i_1, j_1), (i_1, j_2), (i_2, j_1) connected, then (i_2, j_2) also connected.
  - "Insight 4": Any connected pair can be shown to be connected using "insight 3" (which is just computing "transitive closure")

Problem C: Rigid Trusses

- Solution: Given grid, interpret as “bipartite adjacency matrix”, and simply check if connected.
- Single **BFS/DFS**, easy to code!
  - O(RC) time, optimal

Problem E: N-Fruit Combo

- Each line covers an infinite strip with width 6 in some direction.
- Problem reduces to: What is the minimum width of the given points?

Problem E: N-Fruit Combo

- Insight: The minimum-width strip touches two points on the boundaries.
- Solution: For every pair of points, check their perpendicular bisector, and consider its width-6 strip.
  - The answer is yes if all points are in it, for some pair.
  - $O(n^3)$, passes

Problem E: N-Fruit Combo

- Insight: Width only dependent on **convex hull**
  - Can be computed in O(n log n)
- Problem is now:
  - Given convex polygon, what is maximum width?

Problem E: N-Fruit Combo

- Idea 1: For each edge, find “farthest point”.
  - $O(n^2)$
- Idea 2: For each edge, find “farthest point” via **ternary search**.
  - $O(n \log n)$
- Idea 3: Use **rotating calipers**.
  - $O(n)$
  - but overall $O(n \log n)$ due to convex hull computation

Problem G: For Science™

- For each node
  - recursively compute set of distinct elements.
  - Take the *set union*.
- Each node x processed in O(size(x) log size(x)).
- Worst case is a tall tree.
  - Overall O(n log n).

Problem G: For Science™

- Insight: when combining two sets S and T, "merge smaller to larger":
  - simply insert each element of the smaller set to the larger set.
  - Now runs in $O(\min(|S|,|T|))$ instead of $O(|S| + |T|)$.
- Requires destroying the copy of the larger set
  - But it's okay since we only need it once.
- What is the complexity?

Problem G: For Science™

- Complexity:
  - Whenever each element is inserted to a new set, the size of the set it is in *at least doubles*.
  - The size of the largest set is n.
  - Can only double at most lg n.
  - Therefore, each element is reinserted ≤ lg n times.
  - Overall work is thus ≤ n lg n.
  - Set insertion is $O(\log n)$, so overall $O(n \log^2 n)$, passes!

Problem A: All About The Base

- $a(a+1)/2 = b^2$ is equivalent to:
- $(2a+1)^2 - 2(2b)^2 = 1$, which are solutions of:
- $x^2 - 2y^2 = 1$.
- This is a **Pell equation**.
  - They have well-known solutions.
  - A whole math theory exists behind them.
  - We recommend reading through it!

Problem A: All About The Base

- The n'th solution (x_n, y_n) can be shown to be:
  - $(x_n + y_n \text{ sqrt}(2)) = (3 + 2 \text{ sqrt}(2))^n$
- Can be computed recursively via:
  - $(x_n + y_n \text{ sqrt}(2)) = (x_{n-1} + y_{n-1} \text{ sqrt}(2))\ (3 + 2 \text{ sqrt}(2))$
  - with base case $(x_0, y_0) = (1, 0)$.
- Again, I suggest reading about Pell's equations to prove them!

Problem H: Algols for Algolympia

- Insight: $a_i + a_j$ is “not too large”.
- Thus, for each s, compute how many pairs (a_i, a_j) have $a_i + a_j = s$.
    - The answer can then be computed in O(n log n) after that. (how?)

Problem H: Algols for Algolympia

- Let c_v = number of i such that $a_i = v$.
- Let d_v = number of (i, j) such that $a_i + a_j = v$.
- Then:
  - $d_v = \text{sum}(r + s = v)\ c_r\ c_s$
  - (almost. Need to handle double counting, but this is the bulk)

Problem H: Algols for Algolympia

- d_v = sum($r + s = v$) $c_r\ c_s$
- This is **polynomial multiplication**!
- Let $C(x) = c_0 + c_1x + c_2x^2 + \ldots$
- Let $D(x) = d_0 + d_1x + d_2x^2 + \ldots$
- Then $D(x) = C(x)^2$

Problem H: Algols for Algolympia

- Fast polynomial multiplication algorithms are known.
  - Karatsuba's algorithm: $O(n^{1.59})$
  - Fast Fourier transform: $O(n \log n)$
- I strongly suggest reading about them!

More Detailed Solutions

- Available at:
  - https://www.overleaf.com/read/wsdfvqvbmhwy

Thank you!

- **Kevin Charles Atienza**
- **Jared Guissmo Asuncion**
- **Karl Ezra Pilario**
- **Tim Joseph Dumol**
- **Payton Robin Yao**
- **Alvin John Burgos**

- **`A`: All About The Base**
  - Asuncion, Atienza
- **`B`: Make Gawa This Program**
  - Asuncion, Atienza
- **`C`: Rigid Trusses**
  - Pilario, Atienza
- **`D`: Slicing Number Cakes**
  - Pilario, Atienza
- **`E`: N-Fruit Combo**
  - Pilario, Atienza
- **`F`: Alien Defense Deux**
  - Atienza, Dumol
- **`G`: For Science™**
  - Atienza, Yao
- **`H`: Algols for Algolympics**
  - Atienza, Burgos