Competitive Programming Notes

Most important thing is to not tunnel vision (If idea doesn't work after half hour or so try another one) . Draw cases, look for some pattern, and try to reframe the problem to get better understanding of underlying mechanisms.

#pragma GCC optimize("Ofast,no-stack-protector,unroll-loops,fast-math")
vimdiff

tricks

- Try sqrt factor
  - # of distinct numbers is O(sqrtN)
- every 2^i reconstruct static DS (additional logN factor for total construction and query times)
- Divide + Conquer on static query
- **Look for monotonicity**
  - Binary search on answer
  - Two pointers optimization
- Prefix sums (if there are intervals)
- Always split absolute values into 2 cases
  - sweeping offline
  - at worst an additional dimension or log factor for an implicit range tree
  - at best just 2x as slow
- Solve equations for individual variables
- Pigeonhole (https://codeforces.com/contest/1270/submission/99289102)
- If question seems impossible or NP (i.e.^^^) look for something really simple
  - Try making it a graph? to prove possibilities
    - Perfect Matchings sometimes in bipartite graphs
    - Spanning Trees (N edges must create a cycle)
    - DAGs?
  - Might just be flows as well

- Check individual contributions over the course of algorithm (one edge across all trees paths using it)
- Offline / Reversing queries (dynamic connectivity, computational geometry)
- Split question into parts
- Lookup table instead of hash map as const optimization
- Build a segment tree on the active times (instead of deletions) (offline dynamic connectivity in log^2)
- First try brute forcing starting positions (cbarn)
- When is it impossible? (check parity of degree of graph or lower bounds)
- If there are operations, frame as a tree (applying an operation same as drawing an edge from old num -> new num)

Reductions

- Make a DAG
- If multiple colors, try fixing 1 color and solving for only 1 color
- Try solving on simple trees / smaller trees
- Shrink alphabet size (make numbers smaller)
- Think about the frequency tables of each list of elements (if #distinct elements small)

bugs

- Overflow
- Proper Modulo (don’t *mod if using mod inverse)
- Actually sort array
- Division by zero
- Check if map key exists first
- SPAM ASSERTS

Actually save test cases that break your code *.in

ad-hoc

- check invariants
  - Parity
  - Sum
  - Board Colorings
- solve obvious cases first
- Ignore unnecessary info (make sure it's truly unnecessary or red herring)
- look for upper/lower bounds first then try to construct
- make optimization first find complexity after

subarray queries/questions

- Fix left border
- Fix max, check if actual complexity is N^2
- Range tree
- Process Offline, only remembering certain elements
  - apply range tree
  - MO's?
  - Hilbert Order, approx TSP when there is O(fast) movement of subarray borders
- Successfully reducing number modulo X will reduce it by at least factor of 2 (can only happen logarithmic number of times)

dp

- Try adding more to the state
- Always add permutations in order (1,2,3,4)
  - If K elements have been added then try adding K again anyways and add +1 to any element >= K currently in the permutation
- **Try extreme values of individual dimension**
- Consider complementary values
- If it looks like DP, but can't find a case try changing problem a little bit
  - instead of K best lines in a grid for apples (with intersection) split lines so one is always above the other
- Convex Hull Trick
- Divide and Conquer (e.g. the best splitting point is monotonic as the

prefix of consideration increases)
- Binary jumping?? (262144)
- Save memory if really needed
- Matrix expo

tree

- Try flattening with an euler tour
- sqrt decomp based on tree degree?
- Reroot tree
- Diameter
- Try offline
- HLD / CD nuke?
- Binary jumping to precompute answers

graph

- Look at MST
  - Largest edges on cycles can never be in the MST
- Does graph have special shape? (like functional graph)
- DP on DAG
  - Check the topo sort order in general
  - Find SCC's to reduce question to DAG
- Define shortest path states and brute force dijkstras
  - SFPA if desperate (probably never passes, most have at least one anti-heuristic test case)
- BCC
  - makes a tree
- Tree coloring
- DSU on trees (query for #vertices in subtree w/ a property)
- DSU constructs always constructs tree, can flatten this to an array and do queries on it
- Low linking, squeeze more info out of the DFS
-

string

- Bloom filter (check within a string)

- Aho Corasick (search in a static dictionary)
- Polynomial Hashing
- Manacher's algorithm
- Palindormic Tree
- Eer Tree

number theory

- SQRT factors? (Split based on the size of the prime)
- Totient function
- Prime Sieve
- Check divisor tuples
- Num divisors O(sqrtN)
- Goldbach conjecture is always true for reasonable input

combinatorics

- Check OEIS
- Check Wolfram Alpha
- Try Generating Functions + copy paste Kactl's Fast Fourier Transform
- Stirling numbers (PIE)
- advanced PIE (including restrictions)
- Linearity of expectation (and other statistical distributions)

geometry

- convex hull
- triangulation (concave shape -> convex)
- Binary/Ternary searching on segments
- offline is easier

data structures

- Range Trees
  - Segtree (persistent, implicit)
  - BIT
  - Sparse Table
  - Wavelet

- BBST (actually implement these and stop being lazy, c++ stl not that great)
  - set
  - treap
  - splay
  - order-statistics
- DSU / Link-Cut Tree
- Trees
  - Line Tree representation
  - Centroid Decomp Tree
  - Quad tree (geometrical points)
  - Virtual Tree