NEERC 2011 Problem Review

A. ASCII Area

- Scan the picture top-to-bottom, left-to right
- Count the number of '\' and '/':
  - If even, we're outside the polygon
  - If odd, we're inside the polygon
- Area =
  (number of '/' and '\') / 2 +
  number of '.' that are inside

B. Binary Encoding

- Find the smallest n, such that: $m <= 2^n$
- Now $k = 2^n - m$, is the number of "unused" codes compared to binary encoding
  - k is exactly the max number of codes with n−1 bits
- So, to get the answer write
  - For i in [0, k−1] write binary encoding of i with n−1 bits
  - For i in [k, m−1] write binary encoding of (i + k) with n bits

C. Caption

- Precompute the following costs:
  - e[i,j] – the cost of placing i-th letter of new text starting from horizontal position j on the caption
  - f[i,j] – the cost of leaving a range of horizontal positions [i,j-1] on the caption empty
- Now use dynamic programming:
  - Define subproblem c[i,j] – the optimal placing of i letters from new text so that the last i-th letter is placed onto horizontal position j.
  - c[i,j] = min for s in $[s_{min}, s_{max}]$ of
    c[i-1,j-s-k] + f[j-s,j] + e[i,j]
- Answer is min c[len(new_text),j] + f[j+k,n]

D. Dictionary Size

- Build a two tries:
  - all words in a dictionary (trie of prefixes)
  - all words in a dictionary in reverse order (trie of suffixes in reverse order)
- Use the second trie to count the number of suffixes starting with letters a to z and the total number of suffixes
- Using the first trie analyze all prefixes:
  - +count the number of suffixes for all letters that do not constitute the continuation of suffixes
  - +1 all suffixes that are in the dictionary (words)

E. Eve

- Analyze which matrilineal family each individual belongs to (the information about fathers should be ignored)
- A family is either sequenced (at least one individual is) or assign it some unique negative id
- Now analyze the set of families of alive individuals
  - Two different positive family ids -> NO
  - Just one family alive -> YES
  - Otherwise -> POSSIBLY

F. Flights

- Create a data structure with a “skyline” of parabolas (list of intervals)
- Build trivial skyline for each missile
- Recursively merge those skylines to produce a binary tree – interval tree by time, so that log(n) skylines needs to be analyzed for any time range
- For each node in the time interval tree, build a space interval tree, so that in log(n) a maximum in any space range can be found.
- Now, each query can be answered in $\log^2(n)$

G. GCD Guessing Game

- The hardest number to guess is 1
  - All answers are 1. All other possible numbers have to be eliminated by questioning
- For each prime number in [2,n] range we can ask it, to eliminate all numbers divisible by it
- But we can do better
  - For n=6 we can ask 6=2*3 and 5.
- So we need to group primes into the fewest number of groups, with a product <= n
- Greedy algorithm will do just fine
  - Just group 2 with the largest prime so that their product <= n, etc.

H. Huzita Axiom 6

- For a point and a line, define a family of possible folds that place this point onto this line parameterized by some real t.
  - Write an equation in the form a(t)*x+b(t)*y+c(t)=0
  - Where a(t) and b(t) are linear in t, c(t) is quadratic.
- For two families we need to find t_1 and t_2, so that lines are the same
  - The normals (a_1,b_1) and (a_2,b_2) are collinear
  - Any point from the first line lies on the second.
- Solving this system for t_1 gives a cubic equation for t_1.
  - Take care of degenerate cases and solve it using binary search
  - Resulting t gives a fold.

I. Interactive Permutation Guessing

- Pick a permutation *p* and a number *i*
  - Now try all possible positions for *i* in *p*
  - On some of them the longest common subsequence has the length *k* on others *k* – *1*
  - Any of the positions that gives an answer *k* has the following property: *i* is a part of any common subsequence of length *k*
- Solution: For all numbers from 1 to *n* try all their positions and pick the one with max longest common subsequence
  - By the above properly we get a common subsequence that contains all *i* from 1 to *n*

J. Journey

- For each pair (F_i, d), where d defines one of 4 directions, recursively find:
  - Direction after executing F_i
  - (dx, dy) – position shift after executing F_i
  - max x+y, max –x+y, max –x–y, max x–y
  - Use memoization
  - Use arbitrary precision numbers (max answer = 10^{200})
- Track infinite recursion, when we attempt to compute (F_i, d) that is already being computed:
  - Collapse all current (dx, dy) on stack
  - If they total to (0, 0) – the answer is finite
  - If they total to something else – the answer is Infinity.

K. Kingdom roadmap

- The graph is a tree. We shall connect each leaf to some other leaf, so that there are no bridges.
- Hang the tree by non-leaf node and recursively solve on subtrees:
  - Connect leaves in a subtree passing 1 or 2 leaves to the parent level
  - In each subtree connect pairs of dangling leaves, leaving 1 or 2 to return to the parent level
  - On the root level, connect two remaining leaves, or connect one to the root

L. Lanes

- Model left–to–right traffic assuming t = m
  - Now move t to t–1 (reverse lane earlier)
  - Having one more queued car at time moment t–1, find the next free time slot (maintain a list of those), thus update the model
- Model right–to–left traffic assuming t = 1
  - Move t to t+1 (reverse lane later)
  - Update the model in a similar way
- Having found the total queue time for left–to–right and right–to–left traffic for each t, now find the earliest optimal time t