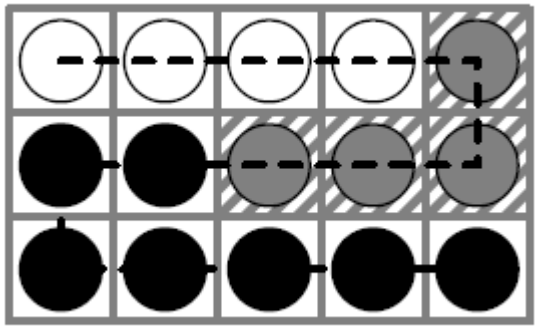
NEERC 2012
Problem Review

© Roman Elizarov

A. Addictive Bubbles

- The easiest problem in the contest
- One possible solution is to layout the bubbles in the following way (row by row, switching between left-to-right and right-to-left):



B. Blind Problem Solving

- The first phase of the solution is to get bit string A to all zeroes by accepting only when weight decrease
  - Accept as the first action to know what weight is in A
- The second phase is to get bit string A to all ones and learn all w_i in the process by accepting only when weight increase
  - Then solve knapsack by exhaustive search of all 2^n bit strings and find the answer bit string
- The third phase is to get bit string B to the answer bit string by accepting only when its gets closer to solution
- The probability that this solution will not finish in 1000 turns is negligible

C. Caravan Robbers

- Sort all intervals by a_i (they also get ordered by b_i automatically). It is easy to prove that the new intervals must follow in the same order
  - The solution is the minimum of $(b_j - a_i)/(j - i+1)$ for all pairs of i and j when $i \leq j$, but this solution is $O(n^2)$ and will not fit into the time limit.
- The correct solution is to use binary search to find the answer length and then find the closest simple irreducible fraction p/q to this answer

D. Disjoint Regular Expressions

- Parse each regular expression with recursive descent and create non-deterministic final automata with ϵ-transitions
- One approach
  - Get rid of all ϵ-transitions in automaton by finding their transitive closure. Now we have two NFA. A with n_A states and B with n_B states
  - Intersect these two NFAs. Build NFA with $n_A \times n_B$ states and transition from on state-pair to the other when both A and B have the the transition on the same letter
  - Now use BFS on intersection NFA to find the answer
- Alternatively, leave ϵ-transitions, do 0,1-BFS and take special care of empty strings

E. Exact Measurement

- Find x mod 10. Notice, that if the result is non-zero, you have to open some boxes with 1 ng weights, because all other boxes have weights that are multiple of 10 ng
- The correct solution is to greedily open the heaviest 1 ng boxes until there is enough 1 ng weights to cover x mod 10
  - Then combine remaining 1 ng boxes and 10 ng boxes and use the same greedy algorithm for x mod 100, taking into account the boxes that were already opened so far
  - Repeat for all powers of 10

F. Folding Snake Cube

- Each Snake Cube puzzle can be represented by a sequence of straight segments ‘S’ and turns ‘T’
- The first phase of the solution is to analyze plain configuration from the input and covert it into a sequence of 25 ‘S’ and ‘T’
- The second phase is to use exhaustive search with back-tracking to fold it into a $3 \times 3 \times 3$ cube

G. Great Deceiver

- Convert n into a k-radix notation:
  "$a_m a_{m-1} \ldots a_6 a_5 a_4 a_3 a_2 a_1 a_0$"
- Now the problem is to count how many numbers up to k have zero digits for odd powers of k, that is, have the following form (assuming m is even):
  "$b_m 0 \ldots b_6 0\ b_4 0\ b_2 0\ b_0$"
  - There is one zero that Baron has to remember
  - Add k-1 numbers with exactly 1 digit
  - Add (k-1)k numbers with exactly 3 digits, etc …
  - When m is even, add $(b_m - 1)k^{m/2}$ numbers with exactly m digits but with first digit less than b_m
    - Now for the numbers "b_m0xxxx" if $a_{m-1} > 0$ add $k^{m/2}$ and return, otherwise add $b_{m-2}\ k^{m/2-2}$
    - And continue to look for the numbers "$b_m 0 b_{m-2}$0xxxx", etc

H. Hyperdrome

- Notice, that a Hyperdrome string is a string that has at most one letter that occurs an odd number of times in it
- Scan the string from left to right and maintain a bit string that for each letters counts whether it occurs an odd (1) or an even (0) number of times
  - Count the number of occurrences of each bit string in a hash map (tree map will do, too)
    - For each bit string check how many times it was previously seen (it corresponds to the number of even-length Hyperdromes that end here) and add to the result
    - For each bit string and for all possible letters, flip the corresponding bit and check how many times the resulting bit string was previously see (it corresponds to the number of odd-length Hyperdromes that end here) and add to the result

I. Identification of Protein

- Represent each peak as $\text{Weight}_P p_i + \text{Wieght}_Q q_i$
  - Let M_P and M_Q be p_i and q_i of the largest peak
  - Peaks that cannot be represented this way are obviously noise; peaks whose p_i exceed M_P or q_i exceed M_Q, too
- The solution idea:
  - Build the protein from two ends: its suffix and prefix adding one amino acid at both ends at a time
  - Count the number of all peaks in (p_i, q_i) item of $M_P \times M_Q$ matrix, and also count its suffix in $(M_P - p_i, M_Q - q_i)$
    - But peaks with $2(p_i + q_i) = M_P + M_Q$ should be counted once
  - Implement $O(n^3)$ dynamic programming where for each length of protein's prefix and suffix, number of Ps in prefix, and number of Ps in suffix solve the problem of explaining the maximum number of peaks
    - Count the case when prefix and suffix are the same only once
  - Then find the maximum number of peaks that can be explained, separated taking care for case of odd and even protein length
  - Then figure out what protein it corresponds to

J. Jumping Around

- The solution always exists when a, b, and c are at least 3 and the solution is constructive
- For each c mod 3 case get rid of all c-type tickets that make +/−3 jumps, jumping right, left, and right again with them:
  - Case 0: Need two a-type tickets
  - Case 1: Need one a-type and two b-type tickets
  - Case 2: Need two a-type tickets
- Jump right with a-type tickets leaving exactly one a-type ticket
- The get rid of all b-type tickets jumping right and the left and using one remaining a-type ticket

K. Kingdom Reunion

- The hardest problem ever given on NEERC
- The solution is straightforward but requires a lot of careful programming
  - Use sweeping line algorithm to check each polyline for self-intersections and self-touches to make sure they are all polygons
    - $O(n^2)$ check of all pairs is too slow
  - Use similar sweeping line algorithm to check for intersection between first and second polygons
  - Use similar sweeping line algorithm with all three polygons to check that union of first and second polygons is equal to the third one
    - There are many alternative ways to make this check

L. Labyrinth of the Minotaur

- The key fact to the solution:
  - If one traverses the path from the entrance to the lair using left-hand rule and using right-hand rule, then an obstacle blocks all paths from the entrance to the lair if and only if it blocks both left-hand and right-hand paths
- Precompute the number of blocked cells and the number of cells visited by each path in each rectangular (1, 1)…(x, y) region
  - This way, one can find the corresponding number in each rectangular $(x_1, y_1)…(x_2, y_2)$ region in O(1) time
- For each (x, y) use binary search to find the smallest square that blocks both left and right path; check that the resulting square is not blocked by any obstacles