2020 Rocky Mountain Regional Programming Contest

Solution Sketches

Credits

- Darko Aleksic
- Darcy Best
- Howard Cheng
- Zachary Friggstad
- Brandon Fuller

A - Stopwatch (31/36)

- For each button press, we flip the state of the timer from stopped to running and vice versa.
- If the button press stopped the display, add the time elapsed to current total.

D - Forced Choice (31/33)

- For each step, look at the choices of cards.
- If the list contains the prediction, respond “KEEP”.
- Otherwise respond “REMOVE”.
- Unnecessary to keep track of the current cards.

- Simple arithmetic can be used to determine which car is boarded by each passenger: $\lfloor \frac{x}{L} \rfloor$, taking care when the passenger stands beyond the train.
- A brute force search for each passenger will also run in time.

G - Distance (17/63)

- Brute force $O(N^2)$ algorithm is too slow.
- First observation: the two dimensions can be solved independently as two one-dimensional problems.
- Second observation: sort the 1D points. As they are processed in increasing order, keep track of the number of points encountered so far.
- As each point is processed, add to the total the product of the number of points to its left and the distance between that point and the previous point.
- Complexity $O(N \log N)$.

B - Arithmetic Decoding (18/32)

- Start with the interval [0,1) and compute $c = a + p_A(b - a)$ as specified.
- If the given real number is less than c, the first letter is 'A'. Otherwise it is 'B'. Update the interval accordingly.
- Repeat until message is decoded.
- Bounds and probabilities are chosen so that the problem can be done using double-precision floating-point numbers.
- It is also possible to encode each of the 2^N possible messages and match the encodings to the given input.

- Straightforward bookkeeping and calculations.

F - Conquest (13/51)

- At each step, keep track of the size of the current army of the Spanning Nation as well as the islands conquered.
- Any adjacent unconquered islands that have smaller army size can be attacked immediately in any order.
- If adjacent unconquered islands are sorted by army size, then we can attack the one with smallest army (if possible).
- If this is not possible then there are no more islands to conquer.
- Use a priority queue to maintain the list of unconquered adjacent islands.
- This is essentially Dijkstra's algorithm. Complexity $O(M \log N)$ using standard binary heaps.

E - Interview Queue (2/32)

- Use a doubly-linked list to maintain the queue.
- Initialize a list of candidates to be removed in the first step.
- For each round, remove the candidates.
- The candidates to be removed next round must be a neighbour of someone removed this round—no need to search the entire list.
- So maintain a pool of neighbours of candidates that were removed last round and only consider them for removal in the upcoming round.
- Can be solved in $O(N)$ time but $O(N \log N)$ time is acceptable (and easier to code).

J - Pegs and Legs (4/22)

- If the disk cannot get stuck: simple dynamic programming: $E(v) := \ell \cdot E(x) + r \cdot E(y)$.
- But it might get stuck. Can show an optimal solution will use the same drop point if the disk gets stuck: but which drop point?
- Let $S(u)$ be the probability the disk eventually gets stuck given it reaches u:

  $$S(u) := (1 - \ell - r) + \ell \cdot S(x) + r \cdot S(y).$$

  If u is leg, let $S(u) := 0$.
- Still, let $E(u)$ be recursively computed as:

  $$E(u) := \ell \cdot E(x) + r \cdot E(y).$$

  If u is a leg, let $E(u)$ simply be its value.

- Let $E_v(u)$ be the expected value for a disk that reached peg u given that we use drop point v if we restart:

$$E_v(u) = (1 - \ell - r) \cdot E_v(v) + \ell \cdot E_v(x) + r \cdot E_v(y).$$

- By induction (on u) and the recursive definitions of $S(u), E(u)$, this easily simplifies:

$$E_v(u) = S(u) \cdot E_v(v) + E(u).$$

- Thus, the expected value from using drop point v satisfies:

$$E_v(v) = S(v) \cdot E_v(v) + E(v).$$

- Output the maximum of $\frac{E(v)}{1-S(v)}$ over all drop points v.

C - Paper Snowflakes (3/14)

- Keep track for each fold: x-coordinate of the fold, the direction (left/right)
- As we process the folds from left to right (increasing x-coordinate), a fold either increases the number of layers by 2 or decreases it by 2.
- The starting point increases the number of layers by 1, and the ending point decreases it by 1.
- Process the combined list of fold points and cut points. As the number of layers changes, keep track of the total length since the last point.
- When a cut point is encountered, output the total length since the last cut point.
- Complexity $O((N+M)\log(N+M))$

H - Antimatter Rain (1/13)

- Sort all droplets and sensors by decreasing y-coordinate.
- Data structure is needed to:
  - maintain the lowest unprocessed drop at each x-coordinate.
  - find the lowest unprocessed drop in the range $[x_1, x_2]$.
- This can be solved using data structures supporting Range Minimum Queries (RMQ).
- Need to "compress" the x-coordinates in the input: we can assume they lie in the range $[0, D + 2 * S)$.

H - Antimatter Rain (1/13)

- Process all drops and sensors from top to bottom.
- If it is a drop, update the RMQ data structure for its x-coordinate.
- If it is a sensor, find the lowest drop in the range. Repeatedly "remove" drops in this range from the RMQ data structure until the next drop in the span (if any) has a higher y-coordinate than the first drop that hit it.
- It is helpful to maintain a stack of unprocessed drops at each x-location. When a drop at an x-coordinate hits a sensor, pop it from the stack and update the RMQ data structure with the next drop in this stack (if any).
- Complexity: $O((D+S)\log(D+S))$