HKOI Mini-Competition (III) Editorial

16 April 2016

A - A Strange Elevator

Setter: Alex Tung

Subtasks 1, 2

- N <= 1
- Sum of terms of arithmetic sequence

- Time complexity: O(1)

Subtasks 3, 5

- N <= 100, S <= 100000
- Dynamic programming

- Time complexity: O(NS)

Subtasks 3, 4

- N = 2
- This actually provides a motivation for the full solution

Subtasks 3, 4

- Let the buttons be +a, +b
- Let s[i] be the lowest reachable floor x, such that x%a = i
- Note that if floor x is reachable, floor (x+k*a) is also reachable for non-negative integers k

Subtasks 3, 4

- How to calculate s[i]?
- s[0] = 0
- s[b%a] = b
- s[(2*b)%a] = 2*b
- …
- Repeat until (r*b)%a = 0

Subtasks 3, 4

- Using s[i], we can calculate the required answer easily (just sum of terms of AS)
- Note that some s[i] may be INF (infinity)

- O(min(a,b))

Full solution

- We want to extend the method of calculation of s [i] to all cases where N >= 2

- This can actually be modelled as a graph problem!

Full solution

- (Assume N >= 2.)
- Let the buttons be $+a_1$, $+a_2$, …, $+a_N$
- Construct a graph with a_1 nodes and $a_1*(N-1)$ edges (no need to construct explicitly)
- Let the nodes be 0, …, $(a_1 - 1)$

Full solution

- Let the nodes be 0, …, (a_1 - 1)
- For each node i and index j (2 <= j <= N), “build” an edge of weight a_j from node i to node (i+a_j)% a_1
- Initially, dist[0] = 0, dist[i] = INF for i > 0
- After running shortest path algorithm, dist[i] is exactly equal to the s[i] we seek

Full solution

- Subtask 6: Inefficient shortest path algorithm
  - e.g. Bellman-Ford, Floyd-Warshall
- Subtask 7: Dijkstra + Priority Queue
  - SPFA also works
- Time complexity: depends on your shortest path algorithm (e.g. $O(|E| \log |V|)$)

Task source

- http://main.edu.pl/en/archive/oi/10/sum

B - RB the Villager

Setter: Theo Leung

Subtask 1

- All the edge has same cost
- We know how many trains we can take using 1 diamond, let it be d
- Let the distance from 1 to i be u_i
- Answer for i is celi(u_i / d)
- Distance of a unit graph can be found via BFS

Subtask 2-3

- The graph is a tree (for Subtask 2 a line)
- The optimal path is the shortest path on a tree
- Which is uniquely determined

Subtask 4

- Modification of general shortest path on graph
- define weight = (diamond, time)
- diamond: the diamond we used
- time: how much time we spent from the moment where the last diamond is used

Subtask 4

- (diamond, time) + weight_of_edge
- = (diamond + 1, weight_of_edge)
  - if time + weight_edge exceeds L
- = (diamond, weight_of_edge)
  - else
- Use dijkstra to achieve (M lg N)

C - Area of Rectangles

Setter: Sampson Lee

Subtask 1

- Inclusion-exclusion principle
- Suppose that a certain area is formed by the overlapping of the set of rectangles R = {R1, R2, R3…}
- There are only 2^N possible such sets
- For each set, compute its area and add to/ subtract from the answer

Subtask 2

- Store all x and y-coordinates
- Divide the map into |x| * |y| pieces by cutting vertically at x coordinates and horizontally at y coordinates
- Define a piece as “black” if some rectangle covers it

Subtask 2

- If a rectangle is (lx, hx, ly, hy)
- Naively color all pieces inside gives a O(N^3) algorithm

Subtask 2

- To speed up coloring, we can split it into two queries using difference array
- Process in increasing y-coordinate
- From ly and so on, there is one more rectangle covering [lx, hx]
- From hy + 1 and so on, there is one less rectangle covering [lx, hx]

Subtask 2

- Therefore, for each piece, the number of rectangles covering it can be calculated
- Time Complexity: O(N^2)

Subtask 3

- The problem can be reformulated into
  - Add 1 or -1 to c[l..r]
  - Sum(width[x] | c[x] > 0)
- This process can be speeded up by segment tree
  - cnt[node] - number of rectangles fully covering the segment
  - ans[node] - width covered in the segment

Subtask 3

- If cnt[node] > 0
  - ans[node] = width[node]
- else
  - ans[node] = ans[left_node] + ans[right_node]

D - AlphaTTT

Setter: Tony Wong

Subtask 1: 3x3

- We do not need to win, we just have to draw
- First move would be the center (4)
- Unless the opponent has a threat, choose the smallest number (that is empty)
- With this strategy, your opponent cannot form two threats (兩頭蛇) in 1 move (why?)

Subtask 2: 4x4

- Similar to 3x3, two optimal players would result in a draw
- This subtask is actually quite hard...

Subtask 3: 3x3x3

- There is no draw scenario, you either win or lose
- First player has a winning strategy
- First move would be the center (13)
- Your next move should be next to your opponent’s, this will create a threat
- You will be able to make a double threat in your third move

Subtask 4: 4x4x4

- Again, first player has a **documented** winning strategy
  - More than 2000 unique board positions (not including reflections and rotations)

Monte-Carlo Tree Search

- How to write a grader that tries its best to take advantage of your suboptimal moves?
- Simple exhaustion is impossible
- Monte-Carlo Tree Search is a technique to limit search space to the best options
  - one of the techniques AlphaGo uses
- Evaluate moves by simulating games using (pseudo-)random moves

Move evaluation

- One way to find the optimal move is to evaluate the board positions for each possible move
- For example, count the number of lines that
  - are dead (have both you and your opponent's pieces)
  - you have 1 piece, 2 pieces, 3 pieces
  - your opponent have 1 piece, 2 pieces, 3 pieces
- Assign weights to each of the case and sum them up. (e.g. you have 1 pieces = +10, your opponent have 1 piece = -100)