HKOI Mini-Competition (II) Editorial

2 April 2016

A - Unsorting a Sequence

Setter: Alex Tung

Case 1: L = N

If L = N, just check whether the original sequence is increasing

> Yes → Output “No solution”

> No → Output original sequence

Case 2: L > N

Consider the simple case of L = N + 1 first

Since lexicographical order is important, it is “better” to insert a ‘1’ in most cases

Let the sequence be {s[1], s[2], …, s[N]}.

Case 2: L > N

If s is not increasing, answer = {1, s[1], s[2], …, s[N]}

If s is increasing and all ‘1’, answer = {1, 1, …, 2, 1}

If s is increasing and not all ‘1’, answer = {1, 1, …, s[k], 1, s[k+1], …, s[N]}

> s[k] is the first number bigger than 1

Case 2: L > N

If L > N + 1,

> By previous step, we already have a sequence with (N + 1) integers and it is not increasing

> Further insert (L - N - 1) ‘1’s in front of the sequence

Time complexity: O(L)

B - Hyper Knight

Setter: Theo Leung

Observation

All monsters need a_{i,j}/4 cost to be destroyed

except the last row, which need a_{i,j} at least

- precompute this cost, i.e.

ret += a_{i,j} / 4; a_{i,j} = a_{i,j} / 4 * 3;

To minimize the cost, we tend to greedily destroy monsters from behind when possible.

Observation

Notice that:

> Once we reach the last row, we can destroy all monsters with 0 cost (after precomputing the necessary cost)

Minimal cost from first row to last row, which hints shortest path

Subtasks 1-3

Brute force the order of destroying monsters

O((N+M)!), Subtask 1

Brute force the shortest path, using smart way

O((L)!), Subtask 1-2

(L : length of shortest path / 2)

Bellman-ford or Flyod-Warshell or SPFA

O(N^2) to O(N^3), Subtask 1-3

Subtask 4

Dijkstra

O(N lg N), Subtask 1-4

Extra: Directed MST

O(N lg N), Subtask 1-4 (out of syllabus of IOI and even harder contests, very hard to write)

C - Bishop Puzzle

Setter: Alex Tung

Subtasks 1 - 3: N <= 8

Algorithm 1: Naive Exhaustion

> Try all possible configurations

> Time complexity: $O(N^2 * 2^N)$

> Solves subtask 1 (N <= 4)

Subtasks 1 - 3: N <= 8

Observation 1: On the chessboard, light squares and dark squares are independent

> Bishop placed on light (resp. dark) squares does not attack dark (resp. light) squares

> Split the problem into two sub-problems

Subtasks 1 - 3: N <= 8

Algorithm 2: Better Exhaustion

> Search for solutions independently on light squares and dark squares

> Time complexity: $O(N^2 * 2^{N/2})$

> Solves subtasks 1 - 2 (N <= 6)

Subtasks 1 - 3: N <= 8

Algorithm 3: Searching + Pruning

> Cut the search tree to make searching faster

> Time complexity: Hard to analyse

> Solves subtasks 1 - 3 (N <= 8)

> May get full score, but this is not the intended solution

Subtasks 1 - 3: N <= 8

Algorithm 4: Hardcode

> Subtask 1: 5 + 17 = 22 cases

> Subtask 2: 37 cases

> Subtask 3: 65 cases

> Score depends on how determined and careful you are XD

Subtask 4: N <= 100

Use constructive method to build answer

Write $K = K_1 + K_2$. We want to have K_1 light squares and K_2 dark squares attacked

Now, consider light squares only

Subtask 4: N <= 100

Observation 2: A solution exists if and only if (K_1 = 0 or K_1 >= N and K_1 <= $N^2/2$)

“only if” part: a bishop on the board attacks at least N squares

“if” part: by construction

Subtask 4: N <= 100

If $K_1 = 0$, easy!

> From now on, consider $K_1 >= N$ only

For the diagrams in the next three slides:

> Green: Place a bishop

> Yellow: Consider placing a bishop

> Red: Do not place a bishop

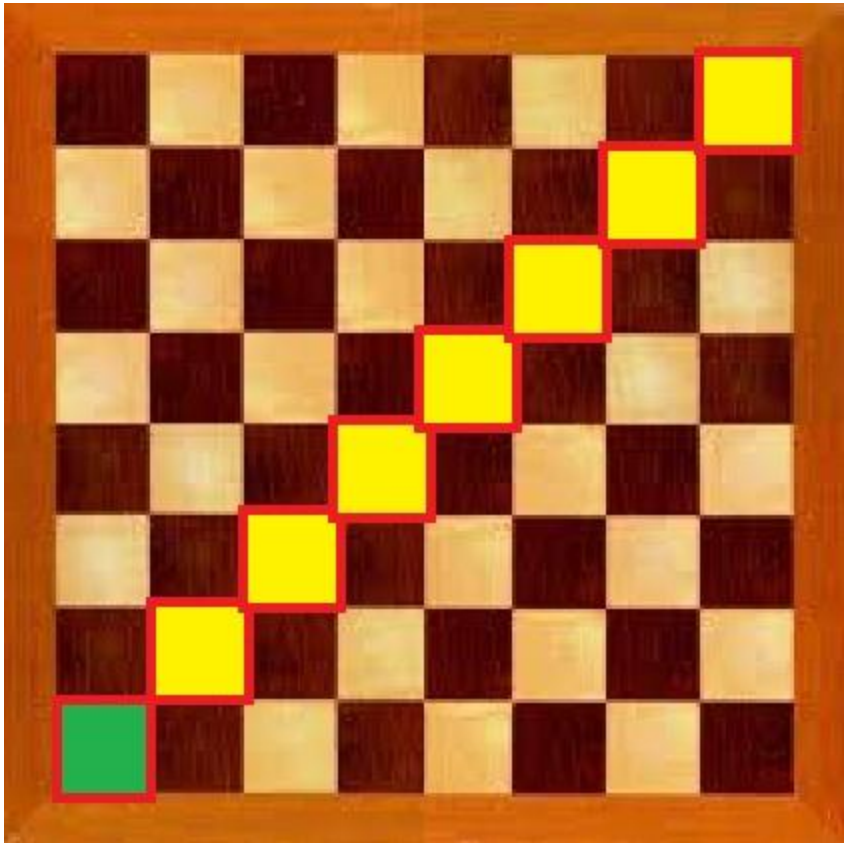
Subtask 4: N <= 100

Construction for even K_1

Selection of yellow squares gives:

2, 4, 6, 6, 4, 2

We can form (K_1 - N) by suitable placement



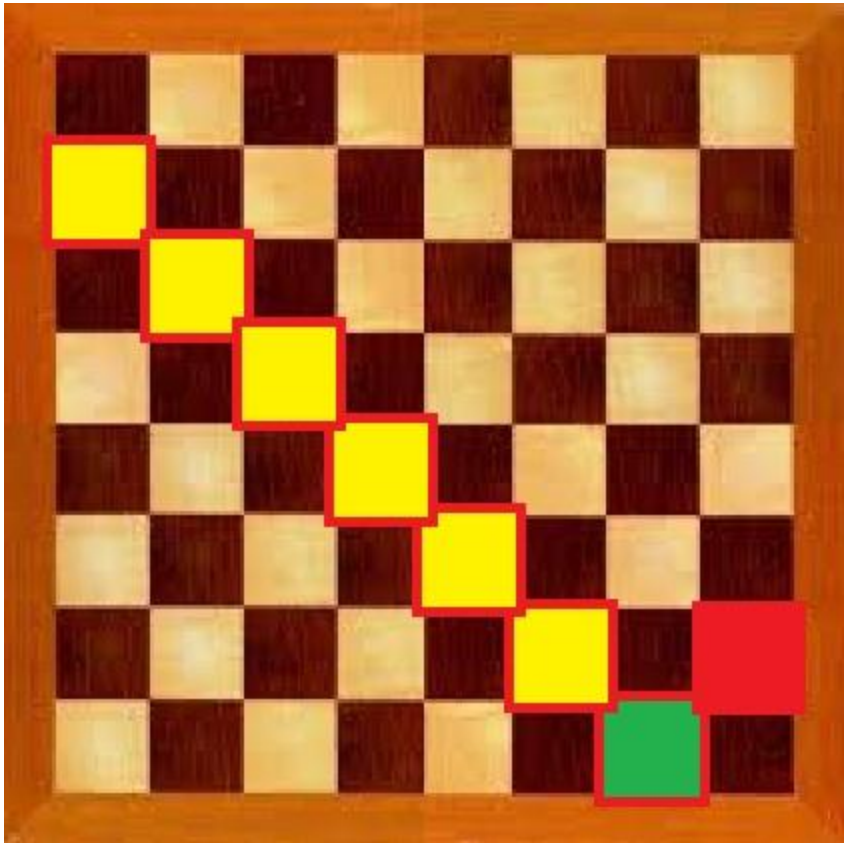
Subtask 4: N <= 100

Construction for odd K_1 and $K_1 < 2N$

Selection of yellow square gives:

1, 3, 5, 7, 5, 3

We can form (K_1 - N) by suitable placement (of one bishop)



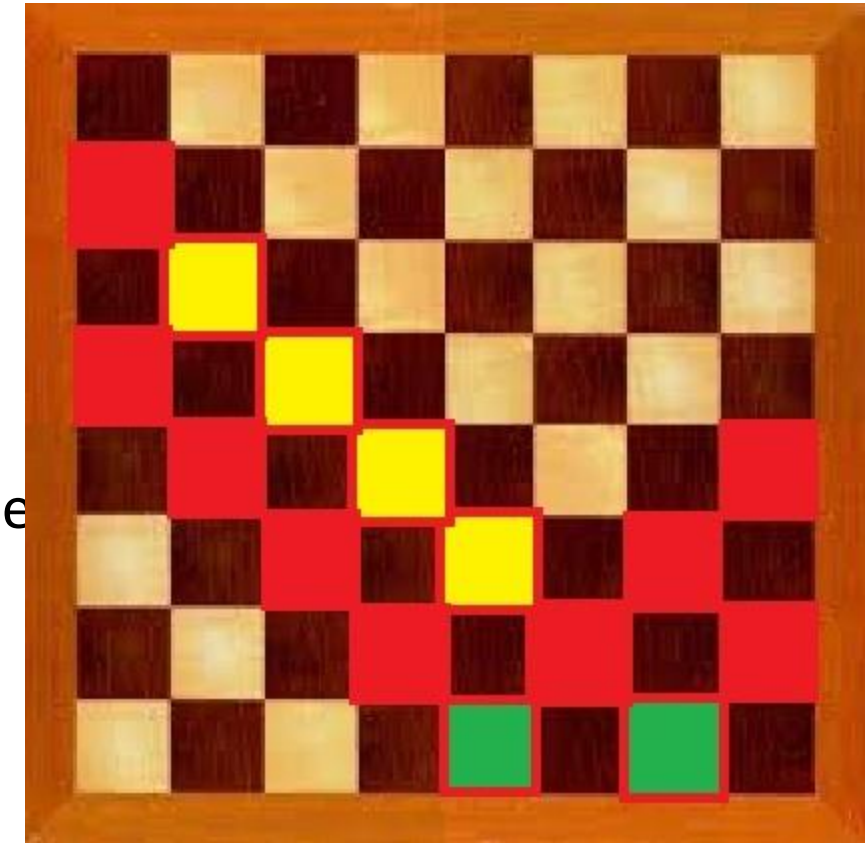
Subtask 4: N <= 100

Construction for odd K_1 and $K_1 > 2N$

Selection of yellow squares gives:

2, 4, 6, 4

We can form (K_1 - (2N - 1)) by suitable placement



Subtask 4: N <= 100

After forming the patterns on both parts, combine them and print the answer

Time complexity: $O(N^2)$

D - Blendoku

Setter: Tony Wong

General Approach

- Write a function that checks whether a configuration is legal. It is always useful
  - Used numbers match the provided ones
  - All segments are arithmetic sequences
- Write the simplest solution to verify the correctness of the above function
- Improve the solution
  - Do not allow “Wrong Answer” to appear, you should only see “Time Limit Exceeded”

Try all permutations

- Put all the numbers into an array a of size e
- Sort the array
- Assign an “ID” to each each empty cell
- Create permutations using array i, and assign
  - cell[i] = a[i] for i = 0 .. e - 1
  - C++ users can simply use next_permutation(a, a + e)

Try all permutations

- Complexity analysis:
  - Number of permutations = E!
  - Checking the correctness of the configuration: O (NM) worst case
  - Overall complexity: O(E! NM)
- Expected score: 22 (ID: 122878)
  - Subtask 1 and Subtask 2 Case 1

Optimization 1 – Pre Solve

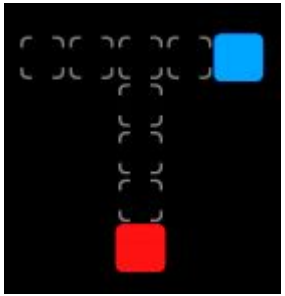
- Once a segment has two known tiles, we can deduce the other tiles
- For example, ? 3 ? ? 9, the segment can only be 1 3 5 7 9
- Solve all segments of this kind first, followed by exhaustion
- Expected score (this optimization only): 30 (122984)
  - Additional: Subtask 2 Case 3 and 10
- Although there is not much increase in score, this “solver” function would be extremely useful for optimizations later

Optimization 2 – Can Fill

- For each number, we can know whether it is part of an arithmetic sequence beforehand
- For example, if the only numbers are 4 5 6 7, then we can say
  - 4, 7: length 4, position 1
  - 5, 6: length 4, position 2
  - Note that length x position y implies both length x – 1 position y and length x – 1 position y – 1 (recursively)
- When performing exhaustion, try only the possible numbers for the cell
- Expected score (this optimization only): 26 (123703)

Optimization 3 – Key Points

- In fact, we do not need to exhaust all cells
- Try to identify a subset of cells that, when these cells are fixed, the other cells can be deduced using “solver” function
- All empty cells can be the only key point for the puzzle on the right
- Expected score (this optimization only): 82 (123709)



Optimization 4 – Cut Tree

- During exhaustion, determine whether if the current configuration is legal, ignoring unfilled cells
  - Check if there is any sequence of non-integer common difference
  - Check if any cell would be out of range / duplicated
- Expected score (this optimization only):
  70 (123711)

Mixing the optimizations

Pre Solve	0	1	0	0	0	1	1	1	0	0	0	1	1	1	0	1
Can Fill	0	0	1	0	0	1	0	0	1	1	0	1	1	0	1	1
Key Points	0	0	0	1	0	0	1	0	1	0	1	1	0	1	1	1
Cut Tree	0	0	0	0	1	0	0	1	0	1	1	0	1	1	1	1
Score	**12**	**26**	**30**	**82**	**70**	**30**	**82**	**70**	**86**	**74**	**3ms**	**86**	**74**	**3ms**	**1ms**	**1ms**
ID	122 878	122 984	123 703	123 709	123 711	123 713	123 715	123 716	123 717	123 718	123 719	123 722	123 723	123 724	123 725	123 702