2019 Mini Competition 0 Editorial

2019-02-14

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

M1901 Zero and Heart Model

M1901 Solution

Area of each semicircle = $(x / 4)^2 * \pi / 2$

Area of the triangle = $x * y / 2$

Therefore, total area = $x * x * \pi / 16 + x * y / 2$

Remember to use floating point division instead of integral division ;)

Otherwise, you could only get scores from Subtask 1

M1902 Zero and Scheduling Problem

M1902 Solution

First, write a function that converts string hh:mm into number of minutes past midnight (0-1439)

For example 07:30 -> 7x60+30 = 450

Then create an boolean array of size 1440, one cell for each minute.
If a[i] true, it means either Aice or Bob (or both) is not free at minute i.

So for Alice, we mark a[i] = true for each minute that her activities cover.
Repeat for Bob.

Then, we just need to find the longest continuous "false" value from the array

M1903 Zero and Love Locks

M1903 Problem Statement

Given a rooted tree of N nodes.

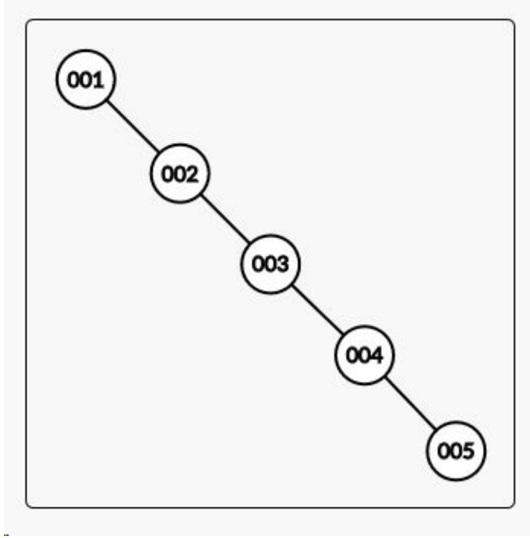
Node i has password 0 <= p[i] <= 999

Given Q operations in order.

Operation i removes all nodes with p[j] = u[i]

Output the number of nodes removed.

M1903 Subtask 1



In subtask 1, the tree is a chain

We can store the small lock number of each password, and the number of the last lock removed (initial = N + 1)

For example, there are 10 locks,
Locks 4, 7, 10 have password 123 and
Lock 6 has password 456
Lock 9 has password 789 (ignore other locks)

Operation 1 unlocks password 789. Number of locks removed = 11 - 9 = 2
Operation 2 unlocks password 123. Number of locks removed = 9 - 4 = 5
Operation 3 unlocks password 456. Number of locks removed = 0 (because 6 > 4)

M1903 Solution

We store two information for each node:

int p[i] = the password and bool d[i] = whether it's removed (initially false)

We also store an index of password to node numbers

int[] roots[j] = nodes that has password j

For example, if nodes 4, 7, 10 have password 123, we store roots[123] = {4, 7, 10}

For operation i, go through the roots[u[i]] and perform Depth-first Search from those nodes, marking nodes as removed and at same time count them.

Note: When you see a removed node, do not traverse into them.

M1903 Solution

Worst case is O(nq)

All 10^6 nodes have the same password, and in all 10^5 operations we remove locks of that password.

Observe that once you remove all locks with password x, the answer for every subsequent operation that remove the same password will remove no locks.

Although input Q = 100000, there can at most be 1000 effective operations.

We can create an boolean array unlocked[1000] that store whether a password has appeared already. If seen before, output 0 immediately.

Time complexity is O(n + q) because each lock can be dfs-ed at most twice. (once for its removal and once for the parent traversing into it)

M1904 Zero and Speed Dating

M1904 Trivial Solution

- We can simulate the elimination process **N-1** times *//O(N)*
  - For each time, just search through the whole array
    - Finding the adjacent pair with smallest difference
  - Remove the smaller element by shifting all elements after that to left *//O(N)*
    - `[2, 7, 6, 1, 4]`
    - `[2, 7, X, 1, 4]`
    - `[2, 7, 1, 4]`
- Overall Time Complexity: $O(N^2)$ Expected Score: 10 out of 20

M1904 Better Solution

- Storing with normal arrays is time-consuming
  - As each removal takes up to O(N) in worst case
- We can use cyclic doubly linked list instead!!!
  - Removal takes O(1)
- How to find minimum difference among all?
  - Using min-heap
  - When we remove some numbers, two differences to be deleted, one to be inserted
  - Insertion is easy
  - Deletion can be done using another min-heap (as mentioned in DS(II) lecture)
  - Deletion can also be done using boolean arrays to maintain `deleted` status of numbers
- Overall Time Complexity: O(N logN)   Expected Score: 20 out of 20

M1904 Faster Implementation

- Using `std::map` or `std::set` to maintain deleted numbers
  - They are too slow
  - Their implementation are having higher constant
  - Expecting 15 ~ 18 out of 20
- Better approach
  - Using another heap (`std::priority_queue`)
  - Using `bool` array
  - Expecting 20 out of 20
- Remember to use 64-bit integers as the range of `a[i]` is large

M1905 Zero and His Dice

M1905 Problem Statement

- Throw N dice
- Find the probabilities $P(\text{sum}\,\%\,5 == i)$ for $i = 1, 2, 3, 4, 0$ respectively
- Output the answer as $PQ^{-1}\,\%\,1000000007$ for $\frac{P}{Q}$
  - See Mathematics in OI (I)

M1905 Subtask 1

$N \leq 10^6$

- Dynamic programming
- Let $dp_{i,j} = P($sum of i throws % $5 == j)$
- Then the answers are $dp_{N,j}$ for $j = 0, 1, 2, 3, 4$

M1905 Subtask 1

$N \leq 10^6$

- $dp_{i,j} = P(\text{sum of } i \text{ throws} \,\%\, 5 == j)$
  We have the following transition formula:

$$dp_{i+1,j} = \sum_{k=0}^{4} \sum_{l=1}^{6} \begin{cases} \frac{1}{6} dp_{i,k} & (k+l) \,\%\, 5 = j \\ 0 & \text{otherwise} \end{cases}$$

$$= \sum_{k=0}^{4} \begin{cases} \frac{1}{6} dp_{i,k} & (k+1) \,\%\, 5 \neq j \\ \frac{2}{6} dp_{i,k} & (k+1) \,\%\, 5 = j \end{cases}$$

- Time complexity: $O(NK^2)$ where $K = 5$

M1905 Subtask 1.5

$N \le 10^{18}$

- Observations
  - $dp_{i+1,j}$ is directed affected by only $dp_{i,k}$, i.e. a function of $dp_{i,k}$
  - The contribution to $dp_{i+1,j}$ by $dp_{i,k}$

$$\begin{cases} \frac{1}{6} & (k+1)\,\%\,5 \neq j \\ \frac{2}{6} & (k+1)\,\%\,5 = j \end{cases}$$

    is independent of i
  - Then $dp_{i+2,j}$ can be easily written as a function of $dp_{i,k}$ as well

M1905 Subtask 1.5

$N \leq 10^{18}$

- Divide and conquer (idea of fast exponentiation)
  - See Mathematics in OI (I)
- Let $dp_{i+l,j} = \sum_{k=0}^{4} con_{l,j,k} dp_{i,k}$
- Base case: $con_{1,j,k} = \begin{cases} \frac{1}{6} & (k+1) \,\%\, 5 \neq j \\ \frac{2}{6} & (k+1) \,\%\, 5 = j \end{cases}$

M1905 Subtask 1.5

$N \leq 10^{18}$

- Transition formula: $con_{l_1+l_2,j,k} = \sum_{m=0}^{4} con_{l_1,j,m} con_{l_2,m,k}$
- For applying the idea of fast exponentiation (recursive version), we have

$$con_{2l,j,k} = \sum_{m=0}^{4} con_{l,j,m} con_{l,m,k}$$

$$con_{l+1,j,k} = \sum_{m=0}^{4} con_{l,j,m} con_{1,m,k}$$

- Time complexity: O($\log NK^3$) where $K = 5$

Advanced content (feel free to skip)

M1905 Subtask 1.5

$N \le 10^{18}$

- Let $dp_i = \begin{pmatrix} dp_{i,0} \\ dp_{i,1} \\ dp_{i,2} \\ dp_{i,3} \\ dp_{i,4} \end{pmatrix}$ and $con = \begin{pmatrix} con_{1,0,0} & con_{1,0,1} & \cdots & con_{1,0,4} \\ con_{1,1,0} & con_{1,1,1} & \cdots & con_{1,1,4} \\ \vdots & \vdots & \ddots & \vdots \\ con_{1,4,0} & con_{1,4,1} & \cdots & con_{1,4,4} \end{pmatrix}$

- $dp_{i+1} = (con)(dp_i), dp_{i+2} = (con)^2(dp_i), \ldots$
  $dp_N = (con)^N(dp_0)$

- Fast exponentiation can be applied to the transition matrix con directly

- Time complexity: O($\log NK^3$) where $K = 5$ (same as before)

M1905 Subtask 2

$N \leq 10^{500}$

- Required to handle / 2 and % 2 for large number
  - / 2 here means integer division

- % 2 is easy
  - Time complexity: O(1)

- / 2 can be done by processing digit by digit
  - Time complexity: O$(\log_{10} N)$

- Overall time complexity: O$(\log N(\log N + K^3))$ where $K = 5$

M1905 Subtask 3

$N \leq 10^{1000000}$

- Fast exponentiation is based on the binary representation of N
- Change fast exponentiation for decimal representation
  - since / 10 and % 10 is super easy for N
- For example, $3^{246} = [(3^2)^{10} \times 3^4]^{10} \times 3^6$
  The same idea can be applied to the algorithms for subtask 1.5
- Overall time complexity: O($\log NK^3$) where $K = 5$
  - possibly $\texttt{TLE}$

M1905 Subtask 3

$N \leq 10^{1000000}$

- Observation
- Only 2 distinct values in answer (once for one value, 4 times for the another) for each N
  - Derive the formula for dynamic programming with 2 as the second dimension (instead of 5)
  - Note that the index of the outstanding value varies with N
- Sum of all values in answer is $1 \pmod{1000000007}$
  - Further reduce the second dimension to 1, i.e. only one dimension left
- Time complexity: $O(\log N)$