GRCPC 2024 Editorial

A. The Binary Chicken Farm

Author: Manolis Tsamis
Analysis: Matina Nadali

Tags: simulation, brute force **Accepted 28/100**
First Solve Plato's Rave! 54 minutes

Given the small constraints, we can approach the problem using brute force. Our goal is to fill the table eggs$[K][N]$, where eggs$[k][i]$ denotes the egg laid by chicken i on the k-th day. We initialize eggs$[0][i]$ with the egg laid by the i-th chicken on the first day. Then, we iterate over the next $K-1$ days, generating the new eggs that the chickens will lay each day according to the given influence relationships. Specifically:

1. If chicken i is not influenced by another chicken, then

$$\text{eggs}[k][i] = \text{eggs}[k-1][i].$$

2. If chicken i is influenced by chicken j, then

$$\text{eggs}[k][i] = \text{eggs}[k-1][i] \oplus \text{eggs}[k-1][j].$$

Finally, we print the eggs laid by the chickens on the last day, i.e. eggs$[k-1]$.

Observations:

1. Since the length of the binary strings is small ($L \leq 20$), we can conveniently store the eggs as integers and perform the XOR operations directly on them. However, we first need to read the eggs as strings to determine their length so that we can print the eggs with the same length in the output, adding the required number of leading zeros.
2. We notice that the egg that chicken i lays on day k depends only on the egg laid by that chicken and potentially one other chicken on the **previous** day. Therefore, instead of storing the entire $K \times N$ eggs matrix, we can store and update two arrays of length N, corresponding to the eggs laid by the chickens on the previous and current day, respectively. However, this optimization is optional.

Time Complexity: $O(K \times N)$

Solution Link: The Binary Chicken Farm.cpp

B. Bureaucracy

Author: Panagiotis Karelis
Analysis: Panagiotis Megas

Tags: sortings, greedy **Accepted 19/392**
First Solve TAI!FT 26 minutes

Let us start by making the following observations:

1. How many times does each person get reinserted into the back of the queue?
Every time we process a person, the remaining time of their request is reduced by M, $r_i \mathrel{-}= M$. If $r_i \leq 0$, the person leaves; otherwise, they are reinserted at the back of the queue. Thus, each person is processed $\lceil \frac{r_i}{M} \rceil$ times.

2. Does the order change when we reinsert people into the back?
After processing everyone once, each person has either completed their request and left, or remains in the queue. The relative order among the remaining people does not change in subsequent rounds. This is because in each round, the queue is processed from front to back, and anyone who does not meet the exit condition is reinserted at the end in the same order they were processed. In other words in each new round, we encounter the remaining people in the same order as the previous round.

Answer:
Using the above observations, we can compute for each person i the number of times they enter the queue, $q_i = \lceil \frac{r_i}{M} \rceil$. To determine the order in which they complete their requests, we sort the pairs (q_i, i) by q_i, breaking ties by original position i. This ensures that individuals with fewer rounds leave earlier, and if two people have the same q_i, the one who was earlier in the original queue leaves first.
The final sorted order is the answer.

Time Complexity: $O(n \log n)$

Solution Link: Bureaucracy.cpp

C. Fair Split of the Golden Tablet

Author: Panagiotis Karelis
Analysis: Stathis Konstantinou

Tags: binary search **Accepted 5/37**
First Solve Killswitch 138 minutes

You are given a rectangular tablet with a circular hole entirely inside it. The task is to choose a vertical line (an x-coordinate) such that the gold on the left side (rectangle area minus hole area) equals the gold on the right.
Imagine sliding the vertical cut from the left edge of the rectangle to the right. As the cut moves, the amount of gold on the left changes continuously and never decreases. Thus, the gold on the left side of the cut is a continuous, nondecreasing function of x.
Since this function is continuous and monotone, and its values at the left and right edges of the rectangle go from below to above half of the total gold, there must be a unique position $x \in [Sx, Tx]$ where the left-gold equals half the total. This guarantees both existence and uniqueness of the solution, so the solution can be efficiently computed by binary searching on the value of x.

Time Complexity: $O(log(T_x - S_x))$

Solution Link: Fair Split of the Golden Tablet.cpp

D. Deciphering Ancient Symbols

Author: Apostolos Giannoulidis
Analysis: Matina Nadali

Tags: brute force **Accepted 10/62**
First Solve Acoustics 29 minutes

First, we notice that we can translate only the parts of the large string that are exact occurrences of the smaller strings. Therefore, we will find all occurrences of each deciphered string in the sentence. This can be done naively in $O(m \times n \times l)$ time by iterating over all possible starting indices of the smaller string and checking whether the following characters of the larger string match the corresponding characters of the smaller string.

After processing all small strings, we know which parts (or indices) of the large string can be translated, and we aim to find the longest continuous segment among them. In other words, we have a boolean array can, where $\text{can}[i] = 1$ if the i-th character can be deciphered, and we need to find the longest segment consisting only of ones in that array. To do this, we keep two variables: ans, representing the length of the longest segment found so far, and cur, representing the length of the current suffix of ones ending at the current index. We loop over the array and update the variables as follows:

1. If $\text{can}[i] = 0$, set $\text{ans} = \max(\text{ans}, \text{cur})$ and $\text{cur} = 0$.
2. If $\text{can}[i] = 1$, increment cur by 1.

Finally, we take the maximum of ans and cur to obtain the length of the longest decipherable segment.

Time Complexity: $O(m \times n \times l)$

Solution Link: Deciphering Ancient Symbols.cpp

Alternative Solution: Alt Deciphering Ancient Symbols.cpp

E. Generation and Transmission Network

Author: Loukas Georgiadis
Analysis: Panagiotis Megas/Matina Nadali

Tags: graphs, MST **Accepted 6/12**
First Solve AC/DC 70 minutes

We can observe that the problem is quite similar to the classic minimum-spanning tree problem in a weighted undirected graph. **What is the main difference?**
The key difference is that in this problem we do not necessarily end up with a tree, but may have multiple sources, resulting in a forest.

Is there a way to transpose this to the classic MST problem?
Instead of working on the original graph that contains self-edges, we can add a node O and replace all self-edges with edges from O to each node.

It is sufficient to run any MST algorithm in the new graph and calculate the cost. That will be our answer.

Why does this work?
You can think of the new added node as the main source, just as the initial problem for every node to get electricity we must either connect it to the main source (build a generator) or connect to another node that has already electricity. At each step, we take the minimum-cost action that connects a new node, which is optimal.

Time Complexity:
Using Prim's algorithm with adjacency matrices: $O(n^2)$.
Using Kruskal's algorithm with sorting: $O(n^2 \log n)$.

Solution Link: Generation and Transmission Network.cpp

F. Anomia

Author: Panagiotis Karelis
Analysis: Stathis Konstantinou

Tags: BFS/DFS **Accepted 11/111**
First Solve: Should I Stay or Should I Code 70 minutes

The problem can be broken down into two phases:

1. Marking of Forbidden Cells
2. Finding if there exists a safe path

1. Marking of Forbidden Cells

We should mark any cell that the Fugitive cannot step into. These include cells that:

- Contain a Building
- Have a police officer
- Can be seen by a police officer

Simply simulate each officer's line of sight, in the appropriate direction, for up to D steps, stopping early if a building or another officer is encountered. Any cell seen this way is marked as forbidden.

2. DFS/BFS For existence of safe path

Once we have marked the forbidden cells, we perform a Depth First Search traversal from the Fugitive's starting location **F**. The Fugitive can only move to adjacent cells that aren't:

- Out of Bounds
- Forbidden

If the Hideout **H** is visited during the traversal, that means there is a safe path, so we output `YES`, otherwise output `NO`.

Complexity Analysis

Say we have K officers. Simulating their line of sight takes $\mathcal{O}(K \cdot D)$. At a first glance it may seem that at worst case we could have a time complexity of $\mathcal{O}(N \cdot M \cdot D)$, which wouldn't pass under the given constraints, but we can observe that each cell will be visited at most once from every direction (say 2 officers are facing the same direction and have the same cell in their line of sight, one must be behind the other, so his view is obstructed, thus by contradiction its not possible). Thus, marking the forbidden cells takes $\mathcal{O}(4NM) = \mathcal{O}(NM)$ time.
DFS on the grid is also $\mathcal{O}(NM)$

Total time complexity: $\mathcal{O}(NM)$

Solution Link: Anomia.cpp

G. Airport Departures' Optimization

Author: Spitalas Alexandros
Analysis: Panagiotis Megas

Tags: dp, binary search **Accepted 7/49**
First Solve AC/DC 20 minutes

Let us start by making a simplification.
Are all the values given for each flight necessary to calculate the answer?
For each flight i, a reward $b[i]$ and a cost $c[i]$ are given instead of storing both, we can assume that for every plane we will need to pay the cost $c[i]$ and if it departs on time we will be rewarded with $b[i] - c[i]$ instead of just $b[i]$. In this way, we only store for each i $b[i] - c[i]$ and the total sum of $c[i]$.

While not necessary, this transformation simplifies the transitions.

Now back to our problem. **When will a plane depart from the airport?**
Each plane either departs on time or is delayed indefinitely, since delaying does not increase the cost.
Furthermore the only condition for a plane to leave on time is that no other plane left in the last T time units.

Let $dp[i]$ be the maximum reward achievable considering the first i flights. Using the above observations we see that there are 2 possible transitions, we can either delay the ith plane indefinitely and thus we get no extra reward $dp[i] = dp[i-1]$ or we can schedule flight i, then the previous scheduled flight must be the latest one that departs at or before $t[i] - T$. Let's call its index $p[i]$. Then we get the transition $dp[i] = dp[p[i]] + b[i] - c[i]$
Finally, as we want the maximum, $dp[i] = max(dp[i-1], dp[p[i]] + b[i] - c[i])$

The final answer will be $dp[N] + \sum_{i=1}^{n} c[i]$

To calculate $p[i]$ we can either for each plane binary search for the first plane that leaves before $ti - T$, or using the fact that $ti - T$ is increasing use a sweeping algorithm.

Time Complexity: $O(n)$ or $O(n \log n)$

Solution Link: Airport Departures' Optimization.cpp

H. The magical forest of Seih Sou

Author: Panagiotis Karelis
Analysis: Stathis Konstantinou

Tags: DSU **Accepted 2/7**
First Solve GeeksforGreeks 110 minutes

Handling removals and connectivity splits online is hard. But we can process the sequence offline in reverse!

If we remove the withering trees $T_1 \dots T_Q$ and then start adding them back in reverse order $(T_Q, T_{Q-1}, \dots, T_1)$, the graph evolves by only add operations (node additions + their incident edges to already present nodes).
We can maintain connected components with a Disjoint Set Union (DSU). For each DSU component we maintain:

- size[v] = number of nodes in the component,
- a boolean value denoting if the component contains a magical node

A component is magical iff it contains a node from S. When two components are merged, if exactly one of them was magical before the merge, then all nodes of the non-magical component become magical at that union.
When we add a node, we create a 1-node component and then union it with existing neighbors. By summing the sizes of components that change from non-magical to magical during these unions we obtain the number of nodes that regain magic at that step.

Those regained counts in reverse are exactly the numbers that were lost when the nodes were removed in the forward process. So just simulate the reverse process, storing the gains at each step and output them in forward order.

Time complexity: $\mathcal{O}((N + M + Q)\,\alpha(N))$, where α is the inverse Ackermann function.

textbfSolution Link: The magical forest of Seih Sou.cpp