2026 UDPC

Official Solutions

UDPC 2026 - Writer / Tester

- ✓ Writer
  - kwoncycle
  - minsung05
  - 99asdfg
  - haruki201sa
  - bnb2011
  - ekwoo
  - ia0102
  - 0917ba
  - leo020630
  - andrew1010
  - queued_q
- ✓ Tester
  - jinhan814
  - jthis
  - pyb1031
  - utilforever

Senior Division	Problem	Difficulty Intention	Writer
A	Venue Selection	**Easy**	bnb2011
B	Ponix and Dalgu 2	**Easy**	0917ba
C	U-Deep-See	**Medium**	leo020630
D	Card Shuffle Game	**Medium**	ia0102
E	Cut the Balloon Strings	**Medium**	queued_q
F	String and String and Queries	**Medium**	leo020630
G	Radio Tower Installation	**Hard**	queued_q
H	UDPC Online Game	**Hard**	99asdfg
I	Ponix Counts Trees	**Hard**	minsung05

Junior Division	Problem	Difficulty Intention	Writer
A	Venue Selection	**Easy**	bnb2011
B	Swap-LIS	**Easy**	0917ba
C	Ponix and Dalgu 2	**Easy**	0917ba
D	U-Deep-See	**Medium**	leo020630
E	Card Shuffle Game	**Medium**	ia0102
F	Let the Wind Carry Me	**Medium**	queued_q
G	Simple and Easy Problem	**Medium**	0917ba
H	UDPC Online Game	**Hard**	99asdfg
I	Dalgu and Yunee's Card Battle	**Hard**	99asdfg

1A/2A. Venue Selection

`math, bruteforcing`

Difficulty Intention – **Easy**

- ✓ Statistics:
  - Junior: 32 submissions, 16 correct, AC rates - 50.000%
- ✓ First Solve:
  - Junior: —**IQ100**—POSTECH, 9 min
- ✓ Writer: bnb2011

1A/2A. Venue Selection

- Suppose that a point on the circular road is located a distance d clockwise from another point.
- The **distance** between two points can be computed as $\min(d, N - d)$.
- Once we fix an arbitrary reference point on the circular road, we can assume that the positions of all schools and the venue are located at integer distances clockwise from that reference point.
- The total number of ways to determine the positions of the schools and the venue is $\mathcal{O}\left(N^4\right)$, and each case can be verified in $\mathcal{O}\left(1\right)$ time.
- Therefore, the problem can be solved in $\mathcal{O}\left(N^4\right)$ time.
- Depending on how the positions are determined, the problem can also be solved in $\mathcal{O}\left(N^3\right)$, $\mathcal{O}\left(N\right)$, or $\mathcal{O}\left(1\right)$ time.

2B. Swap-LIS

Ad-Hoc

Difficulty Intention – **Easy**

- ✓ Statistics:
  - Junior: 40 Submissions, 10 Solved, AC rates - 25.000%
- ✓ First Solve:
  - Junior: **syeyoung**[POSTECH], 21 min
- ✓ Writer: Woojin Lee[0917ba]

2B. Swap-LIS

- ✓ If the given permutation is an increasing sequence, it is impossible to choose any two elements to swap such that the length of the LIS does not decrease.
- ✓ Otherwise, you can simply find an index i where $A_i > A_{i+1}$ and swap A_i and A_{i+1}.

1B/2C. Ponix and Dalgu 2

`game_theory, case_work`

Difficulty Intention – **Easy**

- ✓ Statistics:
  - Junior: 41 Submissions, 10 Solved, AC rates - 24.390%
  - Senior: 53 Submissions, 15 Solved, AC rates - 28.302%
- ✓ First Solve:
  - Junior: **syeyoung**[POSTECH], 42 min
  - Senior: **36ismyfavnumber**[UNIST], 16 min
- ✓ Writer: Woojin Lee `0917ba`

- ✓ If there are two or fewer empty cells remaining, Dalgu wins because either Dalgu's turn never arrives or the board remains symmetric after Dalgu finishes their turn. Therefore, we only need to consider cases with three or more empty cells.
- ✓ We can analyze the problem by dividing it based on the parity of N.

1B/2C. Ponix and Dalgu 2

- When N is even: Dalgu wins if the board is initially symmetric; otherwise, Ponix wins.
  - If the board is initially symmetric, Dalgu can maintain symmetry by placing a stone exactly opposite to where Ponix moved.
  - Otherwise, Ponix can always make a first move that prevents Dalgu from restoring symmetry in the following turn.

1B/2C. Ponix and Dalgu 2

- When N is odd: Dalgu wins if the board is initially symmetric AND the center cell is already occupied; otherwise, Ponix wins.
  - If the board is symmetric and the center is occupied, Dalgu can use the symmetry strategy similar to the even case.
  - If the board is symmetric but the center is empty, Ponix can occupy the center on the first turn, preventing Dalgu from making the board symmetric.
  - If the board is not initially symmetric, Ponix can always make a first move that prevents Dalgu from creating a symmetric state.

- ✓ To summarize, the winning conditions for Dalgu are:
  $(\text{Initial empty cells} \leq 2) \vee \{(\text{Initial board is symmetric}) \wedge (N \text{ is even } \vee (N \text{ is odd } \wedge \text{center is occupied}))\}$.
- ✓ In all other cases, Ponix wins.

1C/2D. U-DeeP-See

`constructive`

Difficulty Intention – **Medium**

- ✓ Statistics:
  - Junior: 13 Submissions, 7 Solved, AC rates - 53.846%
  - Senior: 23 Submissions, 12 Solved, AC rates - 52.174%
- ✓ First Solve:
  - Junior: **튀긴상추**[POSTECH], 40 min
  - Senior: **36ismyfavnumber**[UNIST], 39 min
- ✓ Writer: Leo Jang[`leo020630`]

1C/2D. U-DeeP-See

- If $K = 1$, it is trivially impossible to solve.
- If $K \geq 2$, a solution always exists.
- Let's consider a method to increase the sum of cells in each subgrid by exactly 1.
- Adding 1 to a cell at coordinates $(aK + 1, bK + 1)$ increases the sum of every $K \times K$ subgrid by exactly 1.
- Similarly, subtracting 1 from a cell at coordinates $(cK + 2, dK + 1)$ decreases the sum of every $K \times K$ subgrid by exactly 1.
- By applying both operations simultaneously, the sum of each subgrid remains unchanged.
- There are several other constructive methods to solve this problem as well.

1D/2E. Card Shuffle Game

`permutation_cycle_decomposition`, `sorting`

Difficulty Intention – **Medium**

- ✓ Statistics:
  - Junior: 9 Submissions, 7 Solved, AC rates - 77.778%
  - Senior: 22 Submissions, 11 Solved, AC rates - 50.000%
- ✓ First Solve:
  - Junior: **syeyoung**[POSTECH], 66 min
  - Senior: **36ismyfavnumber**[UNIST], 49 min
- ✓ Writer: Lee YeSeung[`ia0102`]

1D/2E. Card Shuffle Game

- ✓ You can decompose the permutation into cycles via permutation cycle decomposition.
- ✓ For each non visited elements, follow the permutation and append the element into the list. If you step the already visited element, you find one cycle. Repeat it until visit all elements.
- ✓ For the cycle with length L, the alphabet corresponding to p-th element in cycle moves $(p + k) \mod L$-th element in the cycle.
- ✓ With this, for all initial position i, you can determine where it moves after K times of the process by $O(N)$.

1D/2E. Card Shuffle Game

- ✓ Sort the S, and position each alphabet with calcuated resulting position from the front.
- ✓ The time complexity is $O(N)$ when the sorting is counting sort and $O(N \log N)$ if you use the std::sort and other sorting algorithms.

2F. Let the Wind Carry Me

ad_hoc

Difficulty Intention – **Medium**

- ✓ Statistics:
  - Junior: 1 Submissions, 1 Solved, AC rates - 100.000%
- ✓ First Solve:
  - Junior: **syeyoung**[POSTECH], 86 min
- ✓ Writer: Dongkyu Han[queued_q]

2F. Let the Wind Carry Me

- ✓ If we run BFS using the movement rules given in the statement, it takes $O(THW)$, which results in a time limit exceeded.
- ✓ Instead, we can reconstruct the previous position in reverse order starting from the destination.
  - If there is no building in the direction opposite to the wind, then Yunee must have flown in from there.
  - If there is a building in the direction opposite to the wind, then Yunee must have stayed in place while holding onto the building.

2F. Let the Wind Carry Me

- Therefore, if the earliest position obtained by reconstructing previous positions in reverse order is equal to the starting position, the answer is YES. By recording at each step whether Yunee flew with the wind or held onto a building, we can obtain Yunee's plan.
- If we go out of the grid during the reconstruction, or if the earliest reconstructed position is different from the starting position, then the answer is NO.
- The time complexity is $O(HW + T)$.

1F/2F. Gift Distribution

`graph_traversal`

Difficulty Intention – **Medium**

- ✓ Statistics:
  - Junior: 13 Submissions, 5 Solved, AC rates - 38.462%
- ✓ First Solve:
  - Junior: **syeyoung**[POSTECH], 106 min
- ✓ Writer: Woojin Lee[0917ba]

- Let's consider a recurrence relation similar to the one used for finding the LIS:
  - $dp(i)$: The maximum length of a subsequence ending at A_i that satisfies the given conditions.
  - $dp(i) = \max_{1 \le j < i,\, |A_i - A_j| \neq 1} dp(j) + 1$
- We can calculate these DP values for all i in $O(N^2)$ time complexity, and the maximum among them will be the answer.
- However, since $N \le 500\,000$, optimization is required. How should we proceed?

- ✓ In fact, we don't need to examine every j such that $1 \leq j < i$ to calculate $dp(i)$.
- ✓ Since there are at most two values of A_j that satisfy $|A_i - A_j| = 1$, it is sufficient to check only the top three $(A_j, dp(j))$ pairs with the largest $dp(j)$ values at each step.
- ✓ By storing $(A_j, dp(j))$ pairs in a data structure like `std::map` and pruning it to keep the size at most 3 in each iteration, we can solve the problem in $O(N)$ time complexity.
- ✓ There are also $O(N \log N)$ solutions available using data structures such as `std::multiset` or segment trees.

1E. Cut the Balloon Strings

`dijkstra`

Difficulty Intention – **Medium**

- ✓ Statistics:
  - Senior: 17 Submissions, 7 Solved, AC rates - 41.176%
- ✓ First Solve:
  - Senior: **36ismyfavnumber**[UNIST], 72 min
- ✓ Writer: Dongkyu Han[queued_q]

1E. Cut the Balloon Strings

- The height of a balloon is equal to the length of the shortest path from the ring to that balloon along the strings.
- First, run Dijkstra's algorithm once to compute the shortest distance from the ring to each balloon. Let the distance to balloon u be denoted by d_u.
- Now we need to find tight strings supporting each balloon. When a string of length w connects balloon u and balloon or ring v, if $d_u = d_v + w$, then that string is holding balloon u tight.

1E. Cut the Balloon Strings

- For each balloon u, it is enough to keep only the longest tight string among the strings connected to it from below.
- Since exactly one downward string remains for each balloon, we cut $M - N$ strings and keep N strings. These remaining strings form a shortest path tree.
- The time complexity is $O(M \log N)$.

1F. String and String and Queries

dp

Difficulty Intention – **Medium**

- ✓ Statistics:
  - Senior: 7 Submissions, 1 Solved, AC rates - 14.286%
- ✓ First Solve:
  - Senior: **outfinity**[UNIST], 176 min
- ✓ Writer: Leo Jang[leo020630]

1F. String and String and Queries

- How can we find the length of the Longest Common Substring (LCS) of two strings?
- While there is a well-known $O(N \log N)$ method using Suffix Array and LCP Array, it can also be solved using a simpler approach.
- Let's define $dp(i, j)$ as the maximum length of a common substring ending at the i-th character of A and the j-th character of B.
- Specifically, it can be computed as: $dp(i,j) = \begin{cases} dp(i-1, j-1) + 1 & \text{if } A_i = B_j \\ 0 & \text{otherwise} \end{cases}$
- Thus, the problem can be solved in $O(|A||B|)$.

- ✓ Let's apply this logic to the current problem.
- ✓ If we assume all $x = 1$, we can directly use the DP definition described above.
- ✓ Now, let's adapt this for cases where $x > 1$.
- ✓ First, represent each string as a sequence of blocks of consecutive identical characters.
- ✓ For example, string A consists of X_1^A instances of character C_1^A, X_2^A instances of C_2^A, and so on, up to X_N^A instances of C_N^A.

- With this definition, $dp(i, j)$ can be defined as the maximum length of a common substring ending at the last character of the i-th block of A and the j-th block of B.
- Formally: $dp(i,j) = \begin{cases} dp(i-1, j-1) + X_i^A & \text{if } C_i^A = C_j^B \text{ and } X_i^A = X_j^B \\ 0 & \text{otherwise (Special cases apply)} \end{cases}$
- Note: Since the answer might not end exactly at the block boundary, if $C_i^A = C_j^B$, we must consider $dp(i-1, j-1) + \min(X_i^A, X_j^B)$ as a candidate for the answer for all (i, j) pairs.
- By updating the DP table while accounting for the addition of consecutive characters, we can solve the problem in $O(Q^2)$.

1G. Radio Tower Installation

`greedy`, `stack`

Difficulty Intention – **Hard**

- ✓ Statistics:
  - Senior: 3 Submissions, 0 Solved, AC rates - 0.000%
- ✓ First Solve:
  - Senior: -
- ✓ Writer: Dongkyu Han`queued_q`

1G. Radio Tower Installation

- We can solve the problem with a greedy strategy that places the radio towers in order from 1 onward, pushing each one as far to the left as possible.

1G. Radio Tower Installation

- ✓ Suppose we have already installed towers up to $i-1$, and now it is time to install radio tower i.
- ✓ For two already installed radio towers $k < j$, if tower k has lower signal priority $(p_k < p_j)$, then tower k does not need to be considered.
  - If $p_i > p_k$, then $x_k + R_k \leq x_j < x_i$, so radio tower k does not cause radio interference with radio tower i.
  - If $p_i < p_k$, then $x_k + R_k \leq x_j \leq x_i - L_i$, so the radio interference ranges of towers k and i do not overlap.

1G. Radio Tower Installation

- ✓ Let's say that radio tower k is hidden by radio tower j when $k < j$ and $p_k < p_j$.
- ✓ When installing radio tower i, it is enough to consider only those previously installed radio towers that are "not hidden by another tower."
  - Let this set of towers be S_i.
  - In S_i, the signal priorities of the towers decrease as we move to the right.

- On the other hand, for any two towers $k < i$, the following holds. Let us call this the minimum distance inequality.

$$x_i \geq x_k + \min(R_k, L_i)$$

- If we find the smallest x_i satisfying the minimum distance inequality for every $k \in S_i$, that will be the optimal installation position.

- ✓ Let us inspect the towers $k \in S_i$ from right to left.
- ✓ If $R_k \leq L_i$,
  - when we set $p_k < p_i$, the condition for no radio interference $x_k + R_k \leq x_i$ matches the minimum distance inequality.
  - In other words, this does not interfere with our goal of minimizing x_i.
  - Therefore, it is safe to assign radio tower i a higher signal priority and move on.
  - Repeating this process, we maintain the maximum lower bound $x_k + R_k$ on x_i.

- ✓ Then, when we first encounter a tower with $R_k > L_i$,
  - when we set $p_k > p_i$, the condition for having no radio interference $x_k + L_i \leq x_i$ matches the minimum distance inequality.
  - In other words, to avoid interfering with the goal of minimizing x_i, we should assign a lower signal priority.
  - For every tower further to the left $k' < k$, we also automatically have $p_{k'} > p_k > p_i$ and $x_{k'} + L_i < x_k + L_i \leq x_i$, so no radio interference occurs.
- ✓ Therefore, if we set x_i to the maximum among the lower bounds maintained so far, we can install the tower at the optimal position.

1G. Radio Tower Installation

- By maintaining S_i with a stack and popping towers whose signal priority is lower than p_i, we can carry out the above process while also computing S_{i+1}.
- On the other hand, when assigning the signal priority p_i, there is an issue that it may need to be inserted between the priorities of two previously installed towers.
- If we simply shift signal priority values by 1, it results in a time limit exceeded.
- This issue can be resolved by not assigning a tower's signal priority immediately when it is installed, but only when it is popped from the stack. Whenever a pop occurs, we assign the signal priority to be the same as the total number of pops performed so far.
- The overall time complexity is $O(N)$.

1H/2H. UDPC Online Game

`trees`, `disjoint_set`, `smaller_to_larger`, `dp_tree`

Difficulty Intention – **Hard**

- ✓ Statistics:
  - Junior: 2 Submissions, 0 Solved, AC rates - 0.000%
  - Senior: 2 Submissions, 0 Solved, AC rates - 0.000%
- ✓ First Solve:
  - Junior: -
  - Senior: -
- ✓ Writer: Dohoon Kim[99asdfg]

1H/2H. UDPC Online Game

- ✓ Let's construct a tree where node 0 is the root, each dungeon is a vertex, and a prerequisite dungeon is the parent of the subsequent dungeon.
- ✓ For dungeons without any prerequisites, we can simply set node 0 as their parent to ensure the structure is a single tree rather than a forest.
- ✓ Since it is optimal for each player to clear a boss dungeon exactly once, the best strategy is for each player to pick one target boss dungeon and head straight to clear it.
- ✓ Naturally, any boss dungeon at a depth greater than K cannot be cleared by anyone within K days, so we can disregard those dungeons entirely.

1H/2H. UDPC Online Game

- If a boss dungeon at depth x has a capacity of y, then a maximum of $y(K - x + 1)$ players can obtain titles over $K - x + 1$ days.
- For two boss dungeons at different depths, if the shallower one can accommodate more people, it is optimal to assign as many players as possible there; otherwise, remaining players should be sent to the deeper one.
- Using Tree DP, we can implement a strategy that maximizes assignments to the deepest boss dungeons first. If a bottleneck occurs at a common prerequisite (normal dungeon), we remove players starting from those assigned to the deepest boss dungeons.

- ✓ A naive implementation of this logic results in $O(N^2)$ complexity, so optimization is required.
- ✓ Perform Tree DP starting from the deepest nodes. Use a Disjoint Set Union (DSU) with the "smaller-to-larger" technique to store $\{$boss depth, assigned players$\}$ and the total number of players assigned up to the current node.
- ✓ By using data structures like a treeset (or a priority queue), we can efficiently remove players from the deepest nodes when the total count exceeds the current node's capacity. This optimizes the Tree DP to approximately $O(N \log N)$.
- ✓ Finally, based on the results remaining at node 0, calculate the sum of $(\text{assigned players}) \times (K - \text{depth} + 1)$ and output it as the answer.

2I. Dalgu and Yunee's Card Battle

`greedy`
Difficulty Intention – **Hard**

- Statistics:
  - Junior: 7 Submissions, 0 Solved, AC rates - 0.000%
- First Solve:
  - Junior: -
- Writer: Dohoon Kim[99asdfg]

2I. Dalgu and Yunee's Card Battle

- Since the goal is to find the maximum number of rounds Dalgu can win, we can solve the problem by assuming Dalgu already knows which cards Yunee will play in each round.
- Now, let's categorize Yunee's cards into Even and Odd sets, and assume Dalgu assigns his cards as "Even-counters" and "Odd-counters" respectively.
- When assigning cards to counter **Even cards**, it is always optimal to assign larger numbers as Odd-counters and smaller numbers as Even-counters.
- Conversely, when countering **Odd cards**, it is always optimal to assign larger numbers as Even-counters and smaller numbers as Odd-counters.

- ✓ In other words, if the number of even cards Dalgu received is E, choosing the number of even cards e_1 to be used as Odd-counters naturally determines the number of even cards $e_2 = E - e_1$ to be used as Even-counters.
- ✓ Furthermore, since the number of odd cards is $O = N - E$, and the remaining Odd/Even-counters to be assigned are already fixed, determining just one value (e_1) automatically decides how to assign all cards against the opponent's cards.

2I. Dalgu and Yunee's Card Battle

- ✓ Let's calculate the number of winning rounds when the Odd-counter set contains x odd cards and y even cards.
- ✓ First, if we assume all cards in the Odd-counter set are odd, it reduces to a problem of maximizing the count of $b_i < a_i$ given an ascending sequence a.
- ✓ This is equivalent to assigning the largest x numbers from sequence b to the front and finding x such that $b_k > a_k$ for $k > x$. This can be solved in $O(N)$ or $O(N \log N)$.

- ✓ Now, if we replace one odd card with one even card in the Odd-counter set, it is optimal to remove the largest odd card and insert the largest even card.
- ✓ At this time, it is optimal to place the inserted even card in the position of the largest odd card smaller than it, which can be found in $O(\log N)$ using binary search.
- ✓ By applying the same process to the Even-counter set, we can precalculate the win counts for each case in $O(N \log N)$. Then, for each e_1, the calculation takes $O(1)$, solving the problem in a total complexity of $O(N \log N)$.
- ✓ With additional observations, other approaches such as Two Pointers or Network Flow can also be used.

11. Ponix Counts Trees

`combinatorics`

Difficulty Intention – **Hard**

- ✓ Statistics:
  - Senior: 1 Submissions, 0 Solved, AC rates - 0.000%
- ✓ First Solve:
  - Senior: -
- ✓ Writer: Minsung Kwak`minsung05`

- ✓ By observing the tree after contracting all vertices of degree 2, leaving only vertices of degree 1 or 3, we can confirm that $A_1 = A_3 + 2$ is a necessary and sufficient condition for the tree to exist.
- ✓ Let n_0, n_1, n_2 be the number of nodes with 0, 1, and 2 children, respectively.
- ✓ Since the empty left/right child positions can be multiplied by 2^{n_1} at the end, we can assume that nodes are always placed in the left child position first when arranging them.

1I. Ponix Counts Trees

- Now, let's represent the sequence of arranging nodes with 0, 1, or 2 children as a single string.
- By listing the number of children for each node based on a preorder traversal, we can construct a unique string for each tree.
- Since the root must be the first character of the string, the first character must be either 1 or 2.
- Also, since the traversal must end at a leaf, the last character must be 0.
- Finally, for all $i < N$, the count of 0s in the prefix $[1, i]$ must always be less than or equal to the count of 2s in $[1, i]$. The number of such cases can be calculated using Catalan numbers.

- By separately calculating cases where the root is included in n_1 and cases where it is in n_2, the total number of cases is as follows:

$$\frac{2^{A_2+1} \times (N-2)!}{(A_1-1)!A_2!A_3!} + \frac{2^{A_2} \times (N-2)!}{A_1!(A_2-1)!A_3!}$$

- Be careful of overflow, as the values can be very large.