Nanyang Programming Contest 2025

Stage 3: Welcome AY25/26

Lee Zong Yu, Pu Fanyi, Zhou Xuhang

Nanyang Technological University

30 August 2025

- Easy
  - Problem A: Data - Simulation
  - Problem B: Stack - Stack & Simulation
- Medium
  - Problem C: Contact - Rolling hash or Trie or Binary Search
  - Problem D: Knapsack - Greedy
  - Problem E: Stone - Graph traversal & Map
- Hard
  - Problem F: Polygon - Range DP

Problem A: Data

Problem Author:	Zhou Xuhang
Developement	Zhou Xuhang
Editorial:	Zhou Xuhang

Abridged Problem Statement

Abridged Problem Statement

Use one integer C to compress four small integer. Given q queries including set this four integer and output the current C value.

Initial C is between 0 and MAX_INT.

Bit Operation

- To implement this function, we need to use some basic bit operation.
- For the easiest one, SET_DIRECTION, the thing you want to do is to clear the last 2 bits and do $C = C|newDir$.
- So the whole process would be $C = C\&(11...11100), C| = newDir$
- Similarly, for the *Offset*, the process would be $C = C\&(11..10000000011), C = C|(newOffset << 2)$
- Generally, the set function can be implemented by bitwise Not/OR/AND.

Something to say

- Though in this contest we cannot forbid you from storing four attributes locally and compute the C value once required, you should only use one integer for the whole task if in a tech interview.
- Also you need to be able to decompress such as implementing int GET_ID(int C)...

Problem B: Stack

Problem Author:	Zhou Xuhang
Developement	Zhou Xuhang
Editorial:	Zhou Xuhang

Stack

Abridged Problem Statement

Given a string S. Three consecutive identical char will be removed. Output the final string.

- Subtask1: $N <= 100$
- Subtask2: $N <= 100000$

Subtask 1 & 2

Brute Force

Use a while loop and figure out a group of consecutive identical char. Remove them and continue the loop. The time complexity is $O(N^2)$

Stack

It's a hint in the problem name. Push every char into the stack and once the top 3 of the stack is same, then pop out them. The final string is all char in the stack from the bottom up. The time complexity is $O(N)$.

Proof of correctness

- **Brute Force**: The correctness of brute force is obvious. If there are no char to be removed then stop. It is always correct but may get time limit with strong test data.
- **Stack**: Image the remove process if we push the char one by one. Removal will only happen for the top of the stack. Once these chars are removed from the stack, the previous chars is available to remove. Thus this algorithm works well for case "aabbcccba".

Problem C: Contact

Problem Author:	Lee Zong Yu
Developement	Lee Zong Yu
Editorial:	Lee Zong Yu

Abridged Problem Statement

Abridged Problem Statement

Given n phone strings. For each query, given a query string, output the number of strings with a prefix the same as the query string.

n is the number of strings
q is the number of query strings
$maxlen$ is the maximum length of each string
$sumlen$ is the sum of the length of each string

- Subtask 1: $1 \leq n, q, maxlen, sumlen \leq 100$
- Subtask 2: $n, q \leq 10^5$, $sumlen \leq 2 \cdot 10^6$, $maxlen \leq 1000$
- Subtask 3: $n, q \leq 10^5, maxlen, sumlen \leq 2 \cdot 10^6$

Subtask 1

Complete Search

- For each query string, enumerate all phone numbers and do a check between the phone number and the query string
- To do a check, you can enumerate the index to the length of the query string to check if the string are equal at all indexes.
- Time complexity: $O(n \cdot q \cdot maxlen)$

Subtask 2

- $1 \leq n, q \leq 10^5$
- $1 \leq sumlen \leq 2 \cdot 10^6$
- $1 \leq maxlen \leq 1000$

HashMap

- **Precompute:** For each phones string, you can generate all the possible prefixes. Then you store them as a key in a map/dict (python) with value 1. If there are multiple key that is the same, you just increment the value. Hence, the key is the prefix and the value is the number of times occurs among all phones strings.
- **Query**: For each query, you just need to access the map/dict to get the count.

Examples of Precomputations and Query

Phone String:

- 001
  All Prefixes
  - 0
  - 00
  - 001
- 01
  All Prefixes
  - 0
  - 01

Map:

Prefix	Count
0	2
00	1
001	1
01	1

query: 00
By checking the table for the count, the answer is 1

Time Complexity

- **Precompute:** There are $O(sumlen)$ number of prefixes among all n strings, each with length at most $O(maxlen)$. Hence, the complexity to precompute a string is $O(sumlen \cdot maxlen)$. Moreover, if the map is done with hashing, it is expected $O(1)$, so we can ignore it for now.
- **Query:** The time complexity in accessing the map is expected $O(1)$, hence we can ignore them as well. Although there are q query, the number of characters that need to read to access the map is $O(sumlen)$. The time complexity is $O(sumlen)$
- The overall time complexity is $O(sumlen \cdot maxlen)$

Subtask 3

Actually there are three different solutions for this problem!

Continue from Subtask 2

- Realise that when $maxlen \leq 10^5$, the time complexity fails. The main costly task lies in the precomputation.
- Everytime when we map a prefix, we need to map an entire string so it introduces a multiple of $O(maxlen)$ in time complexity
- We can use a **Rolling hash** to progressively hash the string so that we just map a number instead of the entire string.

Solution 1 - Rolling hash

- Click here for the introduction to the data structure
- Given a string s, denote $Hash[i]$ as the hash value of the substring of s prefix at i.
- $Hash[i] = (Hash[i-1] * 11 + s[i] + 1)\%MOD$, $Hash[0] = s[i] + 1$ and MOD is a large prime number
- In order to prevent collision, it is best to set MOD to a prime about 10^{18} such as $2^{64} - 1$ (i.e. the range of integer for an unsigned long long in C++)
- Time complexity: $O(sumlen)$

Solution 2 - Trie

- `https://www.geeksforgeeks.org/dsa/trie-insert-and-search/`
- **Building:** Use may build a Trie to stores all the phone strings. Each time, when you insert a string into the trie, increment the value on each node on the trail by 1.
- **Querying:** For each query string, simply traverse through the trie and print the value on the last node.
- Space complexity: There is at most $O(sumlen)$ amount of nodes and each nodes uses $O(|\Sigma|)$ of space where Σ is the types of characters appears in the string (i.e. '0' - '9'). Hence, the final space complexity is $O(|\Sigma| \cdot sumlen)$.
- Time complexity: $O(sumlen)$ because the algorithm just needs to enumerate through all the characters

Solution 3 - Binary Search

- Sort the phone strings in lexicographical order
- Search the lower bound and the upper bound of the query string in the phone strings. The answer is the difference between the lower bound and upper bound
- Time Complexity: $O(sumlen \cdot \log(n))$ because you need to enumerate through all the query strings character and for each character you need to do binary search on the phone strings.

The implementation might be a little bit difficult. Howevever, it is usual in dealing with strings problems. We explain it in the next slide.

Implementation Details

- Suppose the query string is of len k - the query string is $q_1 q_2 \dots q_k$, q_i is the i-th character of the string.
- We do an enumeration on the query string. We also maintain the index of lower bound and upper bound of the sorted phones strings, says l and r respectively. THe property of l and r will be explained next.
- At i-th step
    - All the phones strings from l to r should has the same prefix at $i - 1$. (You may verify yourself)
    - The character at i of all phones strings from l to r are monotonic.
    - Hence you just binary search the character q_i to reset the shrink the range of l and r
    - By this time, all phones strings from **new** l and r should have same prefix at i

Problem D: Knapsack

Problem Author:	Lee Zong Yu
	CodeForce Problem 1446A
Developement	Lee Zong Yu
Editorial:	Lee Zong Yu

Abridged Problem Statement

Given knapsack with capacity W and n items. What is the minimum number of items chosen such that the total weight is at least half of the capacity.

- Subtask 1: There is at least one item $ceil(C/2)$
- Subtask 2: $n \leq 20$
- Subtask 3: $n, w_i, C \leq 1e4$
- Subtask 4: $n \leq 10^5$, $w_i, C \leq 10^{12}$

Subtask 1 and 2

Subtask 1

- Just choose the item that is at least half of the capacity.
- The answer is 1

Subtask 2

- Brute Force through all the combinations of items.
- Among all the combinations that fulfil the condition, output the length of the smallest combination.

Subtask 3

- The problem is a $0-1$ Knapsack problem. Hence, just do the DP where $DP[i]$ is the minimum number of items to achieve total weight of i.
- For each item i, do a reverse enumeration from $C - w_i$ to 0, at index j, set $DP[j + w_i]$ to $DP[j] + 1$ if the resulting is smaller.
- Initially, $DP[0] = 0$ and all other DP is infinity.
- The answer is the minimum from DP[ceil(C/2)] to DP[C].
- The time complexity is $O(n \cdot C)$

Subtask 4

It is a greedy solution.

Greedy Strategy

- Enumerate the items in descresing weight.
- Select the item if it does not cause the knapsack to exceed its capacity. (In fact, we will proof that item can be selected without exceeding the capacity later)
- Once the condition is met (total weight is at least half of the capacity), stop the enumeration.
- The answer is number of items in knapsack.

Why Correct?

Recall $w_i \leq C$. We want to proof that following the greedy strategy (select item until termination when condition met), there is no item that will causes the knapsack to exceed its capacity.

Proof by Contradition

- **Infer from assumption:** Assume that there is such cases and item j is the first item that make the knapsack to exceed its capacity $\rightarrow \sum_{i=1}^{j} w_i > C$ (eq1).
- **Infer from algorithm:** Since the algorithm is not terminated before item j, it implies $\sum_{i=1}^{j-1} w_i < \lceil \frac{C}{2} \rceil$ (eq2)
- Note that items are sorted in descresing of weights, so $w_j < w_{j-1} < \frac{C}{2} + 1$ (From eq2).

Proof continue

Recall all the equations we get

- $\sum_{i=1}^{j} w_i > C$ (eq1)
- $\sum_{i=1}^{j-1} w_i < \lceil \frac{C}{2} \rceil$ (eq2)
- $w_j < \frac{C}{2} + 1$ (eq3)

Proof by Contradition

- Hence, $\sum_{i=1}^{j} w_i = \sum_{i=1}^{j-1} w_i + w_j < \lceil \frac{C}{2} \rceil + \lceil \frac{C}{2} \rceil < C + 1$ (eq2 + eq3).
- It is a contradiction with eq1.

Proof of Optimality

We can refined that greedy strategy

Refined Greedy Strategy

- Enumerate the items in descresing weight.
- Select the item
- Once the condition is met (total weight is at least half of the capacity), stop the enumeration.

Since you choose the largest item available, it is pretty easy to see that this is optimal. A simple proof is just that there is no smaller size item choice that would fulfied the condition (if not it would have been chosen by greedy algorithm)

Problem E: Stone

Problem Author:	Zhou Xuhang
Developement	Lee Zong Yu
Editorial:	Lee Zong Yu

Abridged Problem Statement?

Abridged Problem Statement

Given a 2D grid, and n stones. A stone is removable if there exists some other stone at the same row/column. Find the maximum number of stones removed.

- Subtask 1: It is guaranteed that originally each stone have at least 1 other stone with the same row/column
- Subtask 2: $n, x_i, y_i \leq 1000$
- Subtask 3: $n \leq 1000$, $x_i, y_i \leq 10^9$
- Subtask 4: $n \leq 2 \cdot 10^5$, $x_i, y_i \leq 10^9$

Observation

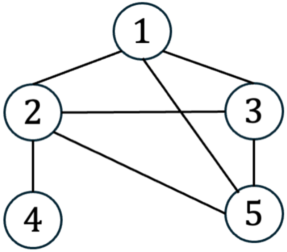
This problem required a crucial observation

Observation

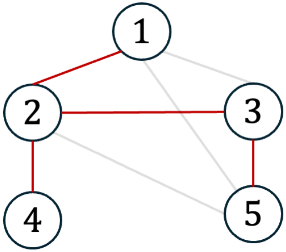
- You may model the problem as a graph.
  - Stone is a vertex
  - There is an edge between two stones if and only if they are in the same row or column.
- Given a connected graph, after you apply **any** DFS traversal, the edges **traversed by DFS** form a tree (i.e. DFS Tree).
- An optimal strategy is always remove the leaf (i.e., the vertex with degree 1 in the DFS tree).
- It is always valid because removing leaf remains the connectivity between all other vertex in a tree.
- With this, only one vertex is left. So it is optimal.

Example

Examples of a DFS tree generated from a graph



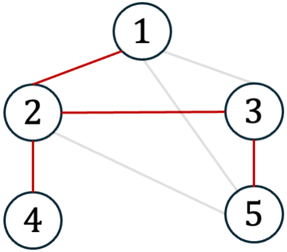
Original Graph

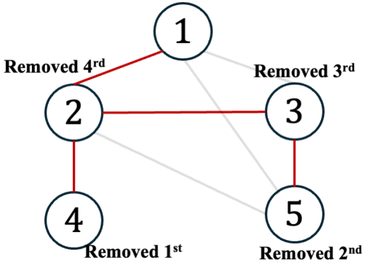


DFS Tree

Example (Cont.)

Examples of the removal of vertex from a DFS Tree





DFS Tree

Order of Removal

Subtask 1

It is guaranteed that originally each stone has at least 1 other stone with the same row/column.

- This subtask is intended for you to make the aforementioned observation.
- The given constraint means that the graph constructed is **connected**
- Hence, only one stone cannot be taken. The answer is $n - 1$

Subtask 2,3,4

- With the previous observation, you may realise that a connected component will have exactly 1 stone is not removable. Here for meaning of connected component
- Hence, the answer is just n minus the number of connected components.
- Alternatively, you may simply do a DFS on the graph and remove the stone in the order of **Post-traversal order** except the last vertex (The vertex you first visited) which is not removable.
- The different subtask means different implementation skills.

Subtask 2

You may do a DFS directly on the grid. That is, you do not explicitly construct a graph.

- In your DFS, in determining which stone is in the same row and same column as your current visiting stone, you may do an enumeration on the row or column since the xy-coordinate value is at most 1000.
- Time complexity: $O(n \cdot xy_{max})$, where xy_{max} is the maximum value of the xy coordinate.

Subtask 3

Explicity construct the graph as told.

- Since $n \leq 1000$, you may construct a graph by comparing the coordinates of each pair of stones.
- That is, build n vertex, where the i-th vertex represents stone i.
- There is an edge between vertex i and vertex j $(i \neq j)$ if stone i and j are in the same row/column.
- Then, do the dfs/bfs traversal on the graph to find the connected components.
- Time complexity is $O(n^2)$ because there is at most $O(n^2)$ number of edges.

Subtask 4

This subtask extends the solution of **Subtask 3**.

Crucial Optimization

Suppose there are k stones in a row, you do not need to build edges between any two stones (In this case $\frac{k\cdot(k-1)}{2}$ edges). You just need to build an edge $k-1$ edges that connect all k stones

Subtask 4 (Implementation)

To implement the optimization,

- maintain a list of stones in each row/column.
- It can be done by a map/dict. The key is the row or column, and the value is an array/list of stones Id.
- Two choices in connecting the stones

Choice 1	connect two stones neighbouring in the list
Choice 2	Create a virtual node representing the row/column and connect every stone in the row/column to the virtual node.

- Then do a DFS/BFS on the graph generated

Time Complexity

- Each stone will appear twice (in the corresponding row or column).
- Hence, the number of nodes is n and the number of edges is at most $2n$.
- The map is expected $O(1)$.
- The time complexity is $O(n)$

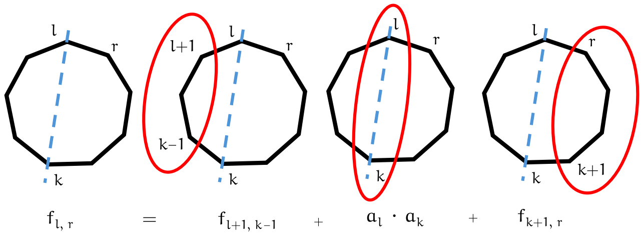
Problem F: Polygon

Problem Author:	Lee Zong Yu
	Codeforce 2074G
Developement	Pu Fanyi
Editorial:	Pu Fanyi

$O(n^4)$ Solution

- If we cut a single edge and consider the regular polygon as a polyline of n points, then we can think of using some form of Range DP to solve this problem.
- $f_{l,r}$ denotes the optimal answer we can obtain when considering only the points from l to r.
- We can consider point l connect with point k, then

$$f^*_{l,r} = \max_{l<k<r} \left\{ f_{l+1,k-1} + f_{k+1,r} + a_l \cdot a_k \right\}, f_{l,r} = \max \left\{ f^*_{l,r}, f_{l+1,r} \right\}$$

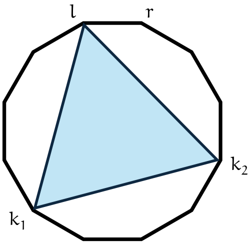


$O(n^4)$ Solution

Now let's consider the case of adding one more point. We see that this is essentially just replacing k with two numbers, k_1 and k_2. Then we can get an $O(n^4)$ solution, which can pass 80% of the test cases:

$$f^*_{l,r} = \max_{l<k_1<k_2\leq r} \left\{ f_{l+1,k_1-1} + f_{k_1+1,k_2-1} + f_{k_2+1,r-1} + a_l \cdot a_{k_1} \cdot a_{k_2} \right\}$$

$$f_{l,r} = \max \left\{ f^*_{l,r}, f_{l+1,r} \right\}$$

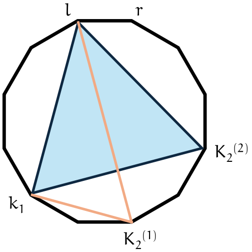


$O(n^3)$ Solution

- A simple idea is: can we just enumerate one of the points between k_1, k_2? The answer is yes.
- Why? We need to make an observation: for $l_1 \leq l_2$, we have $f_{l_1,r} \geq f_{l_2,r}$. This is because we can simply choose not to select any points in the range $l_1 \sim l_2 - 1$, which reduces the problem to $l_2 \sim r$.

$O(n^3)$ Solution

That is to say, for $k_2^{(1)} \leq k_2^{(2)}$, when we enumerate $k_2 = k_2^{(2)}$, we can fully include the cases where the triangle $\langle l, k_1, k_2^{(1)} \rangle$ is formed. As long as we restrict the rightmost subproblem to be $k_2^{(2)} + 1 \sim r$, we can cover all triangles $\langle l, k_1, k_2^{(1)} \rangle$, without forcing l and $k_2^{(2)}$ to be connected by an edge.



$O(n^3)$ Solution

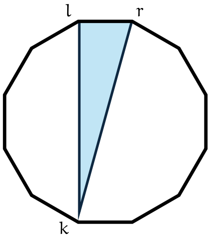
Similarly, we can also allow l to move. We may arbitrarily discard some points in the range $l \sim l' - 1$, so that the triangle becomes $\langle l', k_1, k_2^{(1)} \rangle$.
In this way, the subproblems are formed, and by enumerating k_2 we can simply decompose the problem into $l \sim k_2$ and $k_2 + 1 \sim r$.
This works because the actual triangle $\langle l, k_1, k_2^{(1)} \rangle$ is completely free, only needing to satisfy $l < k_1 < k_2^{(1)} \leq k_2^{(2)}$. In other words, this is exactly $f_{l,k_2^{(2)}}$:

$$f_{l,r} = \max_{l \leq k \leq r} \{ f_{l,k} + f_{k+1,r} \}$$

$O(n^3)$ Solution

- The solution is already nearly perfect, but it seems that something is still missing.
- When $k = r$, we need to handle it separately. If l and r are connected, we need to find a point k and divide the problem into subproblems:

$$\max_{l<k<r} \{f_{l+1,k-1} + f_{k+1,r-1} + a_l \cdot a_k \cdot a_r\}$$



$O(n^3)$ Solution

So, the final answer is:

$$f_{l,r} = \max\left\{ \max_{l\le k<r}\left(f_{l,k} + f_{k+1,r}\right), \right.$$
$$\left. \max_{l<k<r}\left(f_{l+1,k-1} + f_{k+1,r-1} + a_l \cdot a_k \cdot a_r\right) \right\}$$