Nanyang Programming Contest 2025

Stage 1: The First Contest

Lee Zong Yu, Pu Fanyi, Zhou Xuhang

Nanyang Technological University

15 March 2025

- Trivial
  - Problem A: Lucky Seven - Implementation
- Easy
  - Problem G: Tree - Tree Structures
  - Problem F: Subsequence - Sliding Windows
- Medium
  - Problem D: Multiset - Greedy, Sorting
  - Problem B: Distance - Binary Search
- Hard
  - Problem E: Operations - Knapsack DP
  - Problem C: Graph - Dijkstra, Heap, Greedy
- Evil
  - Problem H: Bye Bye - Game Theory

Problem A: Lucky Seven

Problem Source:	CodeChef LUCKYSEVEN
Editorial:	Lee Zong Yu

Lucky Seven

- **Abridged Statement**: Print the 7-th character.
- Just `print(s[6])` because the string stored in most language are in 0-indexed

Problem G: Tree

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

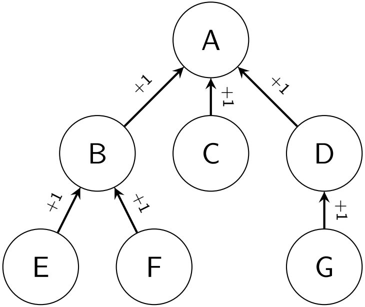
Tree

- **Abridged Statement**: Print the number of nodes with k children for each k.
- To solve this problem, you need to understand what is a rooted tree. When a tree is rooted, each node (except the root) will have exactly 1 parent node (i.e., the node connected to it that is closer to the root) and a number of children nodes.
- Therefore, the number of children is the number of edge connected to it -1 (except for root).

Direct Address Table (Lookup Table)

- Alternatively, you can enumerate the parent from 2 to *n*. The number of time a number appears as parent is the number of children.
- Then, you just need to have two arrays one act as a lookup table of how many children that node have and another act as another lookup table of how many node with that amount of children

Direct Address Table (Lookup Table)



Problem F: Subsequence

Problem Source: Atcoder (ABC 115c)
Editorial: Lee Zong Yu

Subsequence

- Crucial Observation: It is always optimal to choose elements that are neighbouring to each other in the sorted array.
- Explain: Suppose if the it is not the case, there exists an optimal solution where the elements is not contiguous, you can always replace the elements within the gap and make them neighbouring, and this solution is at most the optimal. And since it is optimal, this way is still optimal.

Algorithm

- Sort the array
- Maintain a window of size k. Suppose if the start of the window is placed at position i, and end of the window is placed at position $i + k - 1$. The answer is just $a_{i+k-1} - a_i$.
- with this, you can slide the window from position 0 to position $n - k$ and get the minimum difference.

Problem D: Multiset

Problem Source:	Hackerrank (Matching Sets)
Editorial:	Lee Zong Yu, Pu Fanyi

Possible?

First observation: The answer is -1 if and only if the sum of elements of X and sum of elements in Y is different.

Proof

($\leftarrow$) The operation -1 at some index i and $+1$ at some index j. This means that the sum of elements in the multiset remains unchanged. Therefore, it is impossible to make $X = Y$ if their sum is different.

($\rightarrow$) This proposition means that it is always possible to make them equal if the sum of elements is the same. One can just make all elements in X except the first one to be zero. By this, you can just always -1 from the first index and $+1$ to the index j (for all j) y_j times.

Possible?

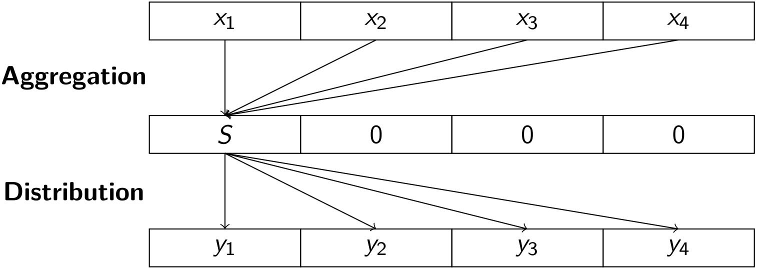


Figure 1: Illustration of the proof in the previous slide

Greedy

To find the minimum operations required, we can adopt a greedy strategy.

- Sort both array in ascending order
- The answer is $\frac{1}{2}\sum_{i=1}^{n}|x_i - y_i|$

Proof.

- From the previous observation, we know that it is possible only if the sum of elements of X and Y are the same.
- Therefore, we can separate the operation -1 and $+1$ independently. That is, one operation is -1 to some index j with cost 0.5 and another operation is $+1$ to some index j with cost 0.5.

☐

Proof (Cont.)

Proof.

The problem is reduced to finding a **one-to-one mapping** between set X and set Y.

- If x_i is map with y_j, the cost is $0.5 \cdot |y_j - x_i|$.
- The cost of the mapping is $0.5 \cdot \sum_{\text{all mapping}} |y_j - x_i|$.

The optimal strategy is to choose the smallest x with the smallest y. You may refer to some similar problems in Geekforgeek (Assign mice to holes). The proof is by case discussion. Please see ▸ here

☐

Problem B: Distance

Problem Source:	USACO 2005 Feb. Gold
Development:	Lee Zong Yu
Editorial:	Lee Zong Yu

Decision Version of the problem

Given n positions (x),

Abridged Statement (Optimization Problem)

choose k positions such that the minimum distance between any of the two k positions are maximized.

Decision Version

Given a distance D, is it possible to select k positions, such that the distance between any two positions is at least D (i.e. the distance between neighbouring positions is at least D).

Greedy for answering Decision version

Greedy

- Sort the positions, set the current latest position as negative infinity ($-\infty$)
- Enumerate the array, if the current position is at least current latest position $+D$, then select the current position and set it to the current latest position.
- If there is at least k position chosen, then the answer is YES.

Proof of Correctness of Greedy

- The greedy algorithm is essentially implying given D if the maximum number of positions you can choose with the aforementioned constraint **is greater than or equal to** k, then the answer is YES.
- Instead of giving a full proof of correctness, we show a reduction that reduces the problem into a classical greedy problem where the correctness of the solution is well-known and proven.
- For each of the position x_i, we create an interval $[x_i, x_i + D)$. Then the problem is reduced to finding the maximum number of non-overlapping intervals (which is known as Activity Selection ▸ Geeksforgeeks).
- The proof of correctness of the problem can be found at Section 15.1 in the book introduction of algorithm ▸ Book

(Cont.) Proof of Reduction

- The optimal strategy for the Activity Selection problem is to sort the interval by end time and apply the same algorithm as our greedy algorithm.
- Since in this specially constructed input, the duration of all intervals are the same, therefore, the order of intervals sort by end time is same as the order sort by start time (which is the position).
- Therefore, the reduction is correct and thus, our greedy algorithm is correct.

Binary Search the Answer (BSTA)

Complete Search / Brute Force

Enumerate all the D from 1 to 10^9 and select the last D that return YES because you know that it is the maximum possible

Monotonicity

A sequence is monotonic if and only if the sequence is non-decreasing or non-increasing.

Binary Search

There exists a answer A such that $\forall D < A$, the decision version is YES and $\forall D \geq A$, the decision version is NO. The answer to the optimization problem is $D - 1$. Therefore, the sequence of answers produced by decision problems is monotonic. Therefore, you can binary search the answer.

Problem E: Operations

Problem Source:	USACO 2015 Dec. Gold
Development:	Pu Fanyi
Editorial:	Pu Fanyi

Revisit Knapsack DP

Given n types of items, item of type i has weight w_i and value v_i. You have a bag with capacity C. You have to choose the items such that the sum of items chosen is at most C and the sum of values should be maximised.

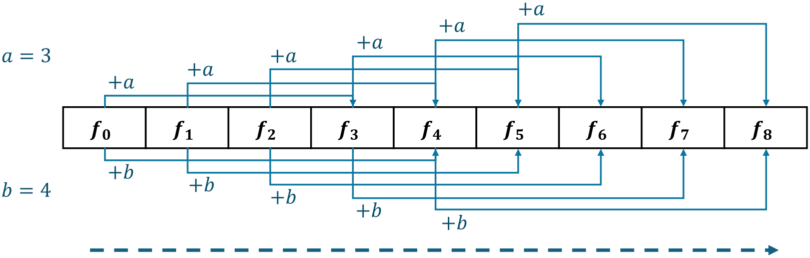
Knapsack Versions

- **[Unbounded Version]** There are infinite numbers of items of each type
- **[0-1 Version]** Each type of item can only be chosen at most once.

Stanford CS161 Knapsack

Plus Only

- Let us first consider the case without $x \leftarrow \left\lfloor \frac{x}{2} \right\rfloor$.
- In this case, a very simple idea is that we can modify the Knapsack problem so that each item's value and weight are equal.
- Let f_i denote the maximum value when the total weight is i.
- We have $f_i = \max\{f_{i-a} + a, f_{i-b} + b\}$



```
for i from 0 to n: $f_i \leftarrow \max\{f_{i-a} + a, f_{i-b} + b\}$
```

Plus Only

- One observation is that

$$f_i = \begin{cases} i & \text{if there exists a way to reach weight } i \\ 0 & \text{otherwise} \end{cases}$$

- Therefore, we actually have a more concise way to define the DP array:

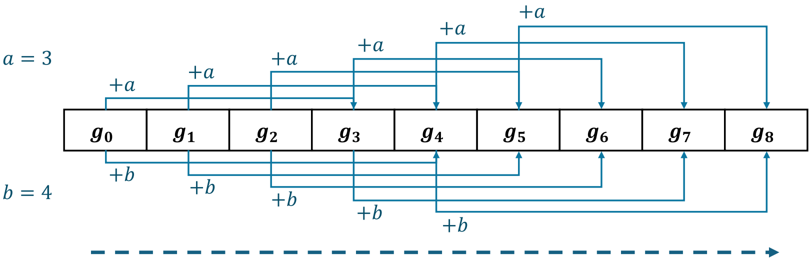
$$g_i = \begin{cases} 1 & \text{if there exists a way to reach weight } i \\ 0 & \text{otherwise} \end{cases}$$

Plus Only

- Considering $a, b > 0$, we have

$$g_i = \begin{cases} g_{i-a} \vee g_{i-b} & i > 0 \\ 1 & i = 0 \\ 0 & i < 0 \end{cases}$$

- In other words, as long as $i - a$ can be achieved or $i - b$ can be achieved, i can also be achieved by adding a or b.



`for i from 0 to n:` $g_i \leftarrow g_{i-a} \vee g_{i-b}$

Plus Only

- Then, lets continue thinking. Suppose we already have an array g', where g'_i indicates that $x = i$ can definitely be achieved in some way (for example, through a division by 2 operation). Can we obtain the complete g function using $+a$ and $+b$?
- The answer is yes, we can slightly modify the equation

$$g_i = \begin{cases} g'_i \vee g_{i-a} \vee g_{i-b} & i > 0 \\ 1 & i = 0 \\ 0 & i < 0 \end{cases}$$

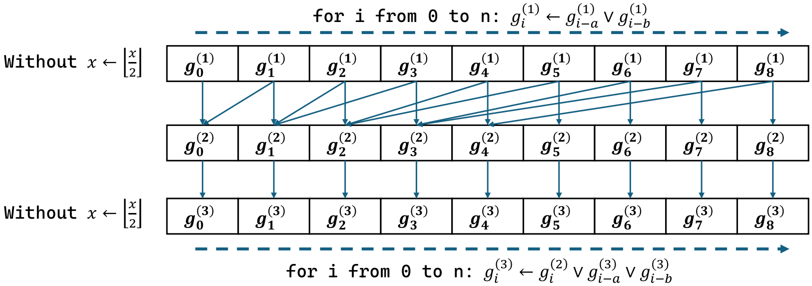
Divided by 2

- We consider the case of $x \leftarrow \left\lfloor \frac{x}{2} \right\rfloor$.
- Assume we have already obtained the answer without the division by two operation, $g^{(1)}$. We can obtain the final answer with the last operation being division by two, $g^{(2)}$, in the following way:

$$g_i^{(2)} = g_i^{(1)} \vee g_{2i}^{(1)} \vee g_{2i+1}^{(1)}$$

- Finally, based on $g_i^{(2)}$, we can obtain the final $g_i^{(3)}$ in the same way as $g_i^{(1)}$.

Final Solution



`for i from 0 to n:` $g_i^{(1)} \leftarrow g_{i-a}^{(1)} \vee g_{i-b}^{(1)}$

`Without` $x \leftarrow \left\lfloor \frac{x}{2} \right\rfloor$

`Without` $x \leftarrow \left\lfloor \frac{x}{2} \right\rfloor$

`for i from 0 to n:` $g_i^{(3)} \leftarrow g_i^{(2)} \vee g_{i-a}^{(3)} \vee g_{i-b}^{(3)}$

Problem C: Graph

Problem Source: Atcoder (ABC 305d)
Editorial: Lee Zong Yu

TLE Solution

Brute Force Solution

For every guard, we apply an Breath First Search (BFS) traversal. After the guard walks through an edge, reduce its stamina by 1. Then, stop the traversal when the stamina is 0. A node is said to be guarded if at least one of the guards reach it. Therefore, just maintain the state of whether the node is guarded and update everytime when a guard reach the node.

The time complexity is $O(k \cdot (n + m))$

Optimization

- **Crucial Observation**: For any node u, among all the guards that can reach node u, the guard with the **most** stamina left can reach all the nodes that other guards can reach.
- You only need to process the guard with the **most** stamina left.

Revisit Dijkstra's Algorithm

- **Problem**: Given a weighted graph, find the shortest path from a node u to a node v.
- **Algorithm**: At every iteration, you maintain a set of nodes that is visited, and you select the node that is **not** in the set of visited nodes and **shortest** from u to the node. Then, you include the node in your visited set. It can be done by maintaining a **Min Heap (Priority Queue)** that stores the unvisited node directly connected to the visited nodes.
- For the correctness of Dijkstra algorithm, please refer to your notes in SC2001.

Similar ideas

- **Algorithm**: At every iteration, maintain the set of nodes that is visited, then choose the node connected to the visited node with the largest guard's stamina. At the beginning of the iteration, the position of the guards is assumed to be connected to the visited nodes (empty set).
- You may implement the algorithm similar to dijkstra but with a **Max Heap** (Priority Queue).
- The proof of correctness is similar to Dijkstra.

Alternative Explanation

We have an alternative intuitive explanation of the solution.

- At every iteration, we choose the guard with the maximum stamina. Let's denote it as h and it is at node u. Then, we create a new graph with all the nodes connected to u having a new guard with stamina $h-1$. Then, you remove the node u.
- The solution for the new graph with node u is equivalent to the solution of the old graph. That is if S_{new} is the solution of the new graph, $S_{new} \bigcup \{u\} \equiv S_{old}$, where S_{old} is the solution for the old graph.
- The iteration ends when there are no nodes left.

Therefore to implement the idea, you just need to maintain a **max heap** and pop element at every iteration.

Illustration

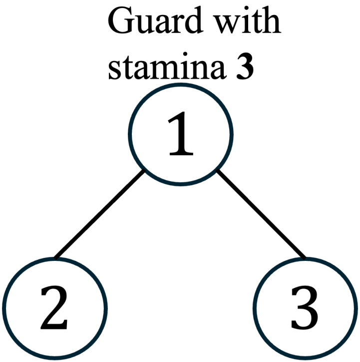


Figure 2: The graph at an iteration before removal of node

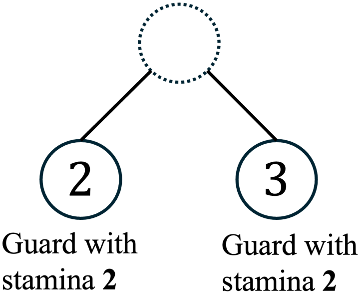


Figure 3: The graph after the removal of node

Problem H: Bye Bye

Problem Source: AtCoder (AGC 023d)
Editorial: Pu Fanyi

Solution

- Let's consider the people living in x_1 and x_n, that is, the residents of the leftmost and rightmost apartments.
- WLOG, we assume $p_1 \geq p_n$. Let's think about what the person living in x_n is considering.
- First, they will realize that they will be the last one to be dropped off no matter what. Even if there are a large number of people at x_{n-1}, causing the vehicle to keep moving right, once the vehicle reaches x_{n-1}, all the residents from x_1 to x_{n-2} will unanimously vote to go left. This is because by doing so, the bus will continue moving left, and except for the person at x_n, it will head straight in the direction that everyone else desires.

Solution

- At this point, the clever NTU students will realize that the residents of x_n unanimously hope that the residents of x_1 get home as soon as possible. This is because once the residents of x_1 have reached home, the bus will only move to the right – no one would choose to go left and send the bus into an uninhabited area.
- In other words, they have accepted their fate of being the last to arrive. However, once the bus reaches x_1, they only need to wait for a duration of $x_n - x_1$ to get home. Since their fate is fixed once the bus reaches x_1, all they need to do is strive to reach x_1 as quickly as possible.

Solution

- So, can we think that the residents of x_n first let themselves "temporarily stay" at x_1, and then we recursively solve the subproblem? Once the subproblem is solved, we can then direct the bus toward x_n.
- In other words, the original problem is $\langle (x_1, \ldots, x_n), (p_1, \ldots, p_n) \rangle$. Now, it becomes $\langle (x_1, \ldots, x_{n-1}), (p_1 + p_n, p_2, \ldots, p_{n-1}) \rangle$. After solving this subproblem, we add the time to reach x_n to the final answer, which will give the solution to the original problem.
- Similarly, if $p_1 < p_n$, then the residents of x_1 will do everything they can to make the bus reach x_n quickly.