1899. Merge Triplets to Form Target Triplet

Medium Greedy Array

Problem Description

In this problem, your goal is to determine if it is possible to achieve a specific target triplet by applying an operation on a given list of triplets. Each triplet consists of three integers. The operation you can perform involves selecting any two triplets and updating one of them by taking the maximum value for each position to form a new triplet.

Given:

• An integer array target = [x, y, z], which is the triplet you are trying to create by using the operation on triplets.

A 2D integer array triplets, where triplets[i] = [a_i, b_i, c_i] corresponds to the i-th triplet.

The operation you can use is as follows: Choose two different triplets i and j (0-indexed) and update triplet j to be [max(a_i,

the best candidates for each position of the triplet that we can achieve through the operation.

a_j), max(b_i, b_j), max(c_i, c_j)].

The question asks you to return true if it is possible to have the target triplet [x, y, z] as an element of the triplets array after applying the operation any number of times (including zero times), or false otherwise.

Intuition

To solve this problem, the key idea is to realize that each value in the target triplet [x, y, z] must be obtainable by using the max

For each triplet [a, b, c] in triplets, we check if it is "valid" to use this triplet in the operation towards reaching the target. A valid triplet is one where a is less than or equal to x, b is less than or equal to y, and c is less than or equal to z, meaning that this

operation within the constraints of the given triplets. To figure this out, we can initialize three variables, d, e, and f, to represent

triplet could potentially be used to reach the target without exceeding it. If a triplet is valid, we update our best candidates (d, e, f) to be the maximum values seen so far that do not exceed the corresponding target values. At the end of this process, if our best candidate triplet [d, e, f] is equal to the target triplet, then we know it is possible to achieve the target triplet; otherwise, it is not possible.

The key insight is that we don't need to consider the actual sequence of operations. We only need to find the highest values (up to the target values) obtainable for each element of the triplet. If any required value cannot be met or exceeded, the target cannot be reached.

Solution Approach The solution utilizes a simple but effective approach leveraging the idea of maintainability and upgradability in the context of

triplets with respect to our target. We iterate through all the given triplets, updating our candidate triplet [d, e, f] to the best

Here's the step-by-step breakdown of the solution:

Iterate through each triplet [a, b, c] in given triplets array. For each triplet, check whether it's safe to consider it in the progression towards the target:

Initialize three integers d, e, f to represent the maximum values we can achieve for each element in the target triplet [x, y,

a. To be considerate safe, a must be less than or equal to x, b must be less than or equal to y, and c must be less than or equal to z.

version that could possibly match our target.

z] without ever exceeding those values.

b. If the triplet doesn't conform to these conditions, we discard it as it would take us away from our target by exceeding the intended value in at least one position.

If the triplet [a, b, c] is valid, we take the maximum values between our current candidates [d, e, f] and [a, b, c]. This

- basically simulates the allowed operation but only in the direction of increasing our chances of reaching the target without going past it:
- b. Update e with max(e, b) c. Update f with max(f, c)

starting from [0, 0, 0] to the maximum values that it can reach without exceeding the target values [x, y, z]. If at the end of

this process, we have successfully reached [x, y, z], it means that the operations can indeed reconstruct the target triplet from

a. If they match, it implies that we can indeed form the target by potentially merging the triplets without ever needing to exceed the target values.

a. Update d with max(d, a)

- b. If they don't match, we conclude that it is not possible to reach the target given the operations and constraints defined.
- In this solution, we essentially simulate the process of using the maximum operation to gradually upgrade a fictional triplet

Start iterating through the triplets array:

update our d, e, f values.

constraints, forming the target is not possible.

Solution Implementation

from typing import List

class Solution:

Python

the given array of triplets.

Example Walkthrough

No additional data structures or patterns are required for this approach, as we simply use loop iteration and conditionals to

handle the core logic, and rely on basic variable assignments for state tracking.

After iterating through all triplets, check if the candidate triplet [d, e, f] matches the target:

Consider an example where the triplets array is [[3,4,5], [1,2,3], [2,3,4]] and the target triplet is [2,3,5]. Let's use the solution approach to check if we can achieve the target. 1. We initialize d, e, f to 0 since we start out with a fictional triplet [0, 0, 0].

• First triplet: [3,4,5] • Since 3 is greater than 2 (our x value in target), this triplet cannot contribute to forming the first element of the target. We do not

• All elements are less than or equal to the corresponding target elements. We can consider this entire triplet.

Second triplet: [1,2,3]

• Third triplet: [2,3,4]

 $\blacksquare d = \max(d, 1) \rightarrow d = 1$ \blacksquare e = max(e, 2) \rightarrow e = 2 • $f = max(f, 3) \rightarrow f = 3$

Finally, we compare our candidate [2,3,4] with the target [2,3,5]. Since the third element does not match (4 instead of 5),

Despite the operations we carried out, the target's third value was not met, leading us to the conclusion that under the given

• All elements are less than or equal to the corresponding target elements. We can consider this entire triplet as well.

 $\blacksquare d = \max(d, 2) \rightarrow d = 2$ \blacksquare e = max(e, 3) \rightarrow e = 3

we determine that it is not possible to achieve the target triplet [2,3,5].

Iterate through each triplet in the list of triplets

if a <= target_x and b <= target_y and c <= target_z:</pre>

int targetX = target[0], targetY = target[1], targetZ = target[2];

int maxX = 0, maxY = 0, maxZ = 0;

// Iterate over each triplet in the input list.

// Initialize variables to keep track of the maximum values of the triplets.

Check if the current triplet is valid by comparing it to the target

Update the maximum values seen so far for each position, if applicable

Return True if the maximum values match the target values, otherwise return False

Unpack the values in the current triplet

• $f = max(f, 4) \rightarrow f = 4$ (Note that f is not equal to 5, which is the target's third value)

After iterating through all triplets, our constructed candidate triplet [d, e, f] is [2,3,4].

In this example walk through, we clearly see how the solution methodically processes each triplet in relation to the target and upgrades the candidate triplet [d, e, f] only in ways that are safe and in accordance with our goal of reaching the target.

def mergeTriplets(self, triplets: List[List[int]], target: List[int]) -> bool: # Initialize variables to track the maximum values for each position max_x , max_y , $max_z = 0$, 0, 0 # Unpack target values into separate variables for clarity target_x, target_y, target_z = target

```
Java
class Solution {
    public boolean mergeTriplets(int[][] triplets, int[] target) {
       // Extract the target values for easy reference
```

int targetX = target[0];

int targetY = target[1];

int targetZ = target[2];

for triplet in triplets:

a, b, c = triplet

 $max_x = max(max_x, a)$

 $max_y = max(max_y, b)$

 $max_z = max(max_z, c)$

return [max_x, max_y, max_z] == target

```
// Initialize the max values found in the triplets
       int maxX = 0;
        int maxY = 0;
        int maxZ = 0;
       // Iterate over each triplet in the given array of triplets
        for (int[] triplet : triplets) {
           // Extract the values of the current triplet
            int currentX = triplet[0];
            int currentY = triplet[1];
            int currentZ = triplet[2];
           // Check if the current triplet can potentially contribute to
           // forming the target triplet without exceeding any target value
            if (currentX <= targetX && currentY <= targetY && currentZ <= targetZ) {</pre>
                // Update the maximum values found that do not exceed the target values
                maxX = Math.max(maxX, currentX);
                maxY = Math.max(maxY, currentY);
                maxZ = Math.max(maxZ, currentZ);
        // Return true if the maximum values found match the target values
        return maxX == targetX && maxY == targetY && maxZ == targetZ;
C++
class Solution {
public:
    bool mergeTriplets(vector<vector<int>>& triplets, vector<int>& target) {
        // Extract the target values for comparison.
```

```
for (auto& triplet : triplets) {
           // Extract the triplet values for comparison.
            int currentX = triplet[0], currentY = triplet[1], currentZ = triplet[2];
            // Check if the current triplet's values are less than or equal
           // to the corresponding target values.
            if (currentX <= targetX && currentY <= targetY && currentZ <= targetZ) {</pre>
                // Update the running maxima if the current values are greater.
                maxX = max(maxX, currentX);
                maxY = max(maxY, currentY);
                maxZ = max(maxZ, currentZ);
       // Determine if the largest triplets found match the target triplet.
        return maxX == targetX && maxY == targetY && maxZ == targetZ;
};
TypeScript
/**
 * Checks whether it is possible to form a target triplet from a set of given triplets.
 * @param {number[][]} triplets - An array containing triplets (arrays of three numbers).
 * @param {number[]} target - An array of three integers representing the target triplet.
 * @returns {boolean} - True if the target triplet can be formed, otherwise false.
function mergeTriplets(triplets: number[][], target: number[]): boolean {
    // Destructure the target triplet into the required values.
    const [targetX, targetY, targetZ] = target;
   // Initialize max values for x, y, z to 0.
```

```
let [maxX, maxY, maxZ] = [0, 0, 0];
      // Iterate over each triplet in the given array.
      for (const [tripletX, tripletY, tripletZ] of triplets) {
          // Check if the values of the current triplet are less than or equal to the target values.
          if (tripletX <= targetX && tripletY <= targetY && tripletZ <= targetZ) {</pre>
              // Update max values, if the current value is higher than what was seen before.
              maxX = Math.max(maxX, tripletX);
              maxY = Math.max(maxY, tripletY);
              maxZ = Math.max(maxZ, tripletZ);
      // Return true if the max values for x, y, z match the target triplet exactly.
      return maxX === targetX && maxY === targetY && maxZ === targetZ;
from typing import List
class Solution:
   def mergeTriplets(self, triplets: List[List[int]], target: List[int]) -> bool:
       # Initialize variables to track the maximum values for each position
       max_x, max_y, max_z = 0, 0, 0
       # Unpack target values into separate variables for clarity
        target_x, target_y, target_z = target
       # Iterate through each triplet in the list of triplets
        for triplet in triplets:
           # Unpack the values in the current triplet
            a, b, c = triplet
           # Check if the current triplet is valid by comparing it to the target
            if a <= target_x and b <= target_y and c <= target_z:</pre>
```

Time and Space Complexity The time complexity of the code is O(n), where n is the number of triplets. This is because there is a single loop that iterates over

(comparisons, assignments, and max() function on integers).

return [max_x, max_y, max_z] == target

Update the maximum values seen so far for each position, if applicable $max_x = max(max_x, a)$ $max_y = max(max_y, b)$ $max_z = max(max_z, c)$ # Return True if the maximum values match the target values, otherwise return False

each element in the triplets list once, and within this loop, the operations performed are all constant time operations

The space complexity of the code is 0(1), independent of the number of triplets. The only extra space used is for the variables d, e, and f, which hold the maximum values of the first, second, and third elements of the triplets that can be part of the merge to reach the target. The variables x, y, and z are just references to the elements of the target list and do not count towards additional space.