**SIMATS ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**CHENNAI-602105**

**Equally Balanced Array Split with Same Average Ussing**

**Dynamic Programming**

**CAPSTONE REPORT**

**Submitted to**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**In partial fulfilment for the award of the degree of**

**BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING**

**BY**

**T RAJESH(192211633)**

**K.MADHAN KUMAR(192211186)**

**SK.SHARUKH KHAN(192211800)**

**Supervisor**

**Dr JOY PRIYANKA**

**Problem Statement**

Given an integer array **nums**, the task is to divide its elements into two separate arrays, denoted as A and B. Each array should be non-empty, and the condition is that the average of elements in array A should equal the average of elements in array B. In other words, after partitioning, both arrays should have the same arithmetic mean. The goal is to determine whether it is possible to achieve such a partitioning. If it is possible, the function should return true; otherwise, it should return false. It's important to note that the average of an array is calculated by summing all its elements and dividing the sum by the length of the array.

**Proposed Design Work**

**1.Identifying the Key Components**

**Input Array (nums):** This is the initial integer array that we need to split into two arrays A and B. It forms the basis of our problem and serves as the input for our solution.

* **Arrays A and B:** These are the two arrays into which we will move elements from the input array nums. Both A and B need to be non-empty, and their averages should be equal. We will devise a strategy to determine how elements are distributed between these two arrays.
* **Average Calculation:** The average of an array is calculated as the sum of all elements divided by the length of the array. Ensuring that the averages of arrays A and B are equal is the primary objective of our solution.
* **Feasibility Check:** We need to determine whether it's possible to achieve equal averages for arrays A and B. This involves devising algorithms or strategies to check the feasibility of such a split.
* **Algorithm Design:** We'll need to design algorithms to partition the input array into two arrays A and B in a way that maintains equal averages. This could involve techniques such as dynamic programming, backtracking, or other optimization strategies.
* **Return Value:** The final output of our solution will be a boolean value indicating whether it's possible to split the input array nums into arrays A and B with equal averages. This return value will be based on the feasibility check performed by our algorithm.

By identifying and understanding these key components, we can formulate a comprehensive plan to solve the problem effectively. Each component plays a crucial role in the design and implementation of our solution.

**2.Functionality**

1. **Check Initial Conditions**: First, check if the input array **nums** is empty or has only one element. If so, it's not possible to split the array, so return **false**.
2. **Calculate Total Sum**: Calculate the sum of all elements in the array **nums**. This sum will be useful for calculating averages.
3. **Iterate Through Possible Lengths of Array A**: Start iterating from 1 to the length of **nums**/2 (since the length of array A cannot exceed half the length of the original array). For each length **lenA**, calculate the sum of all combinations of elements in **nums** of length **lenA**.
4. **Check If Averages Match**: For each sum calculated in step 3, check if it's possible to split the remaining elements (those not in A) such that the average of both arrays matches. This can be done recursively.
5. **Return Result**: If at any point during the iteration, you find a split that satisfies the conditions, return **true**. If no split satisfies the conditions, return **false**.

**3.Architectural Design:**

1. **Input Handling Module**: This module will handle the intake of the integer array **nums** as input to the system. It should include validation checks to ensure that **nums** is a valid array of integers.
2. **Algorithmic Core**: This component houses the algorithm responsible for partitioning the input array into two subsets with equal averages. It will involve exploring various approaches such as backtracking, dynamic programming, or a combination of both to find an optimal solution. The algorithm should efficiently search for valid partition combinations.
3. **Data Structures**: Efficient data structures play a crucial role in optimizing the algorithm's performance. Depending on the chosen approach, data structures such as arrays, lists, sets, or maps may be utilized to store and manipulate subsets and intermediate results.
4. **Optimization Techniques**: To enhance the efficiency of the algorithm, optimization techniques such as memoization, pruning, or early termination criteria can be incorporated. These techniques help reduce redundant computations and improve the overall runtime of the solution.
5. **Output Generation**: Once a valid partitioning is found, this module will generate the output indicating whether it is possible to achieve two subsets with equal averages. The output could be a boolean value (**true** or **false**) representing success or failure in achieving the desired split.
6. **User Interface (Optional)**: Depending on the deployment scenario, a user interface component may be included to interact with the system. This could be a command-line interface, a web-based interface, or an API endpoint for integration with other systems.
7. **Testing and Validation**: A comprehensive testing strategy should be devised to validate the correctness and robustness of the solution. This includes unit tests to verify individual components, as well as integration tests to ensure seamless interaction between modules.

**UI Design**

**1.Layout Design**

1. **Header Section**:
   * Include the title of the problem: "Split Array With Same Average".
   * Possibly add a brief description or instruction for the user.
2. **Input Section**:
   * Provide an input field where the user can enter the integer array **nums**.
   * Optionally, include a button to submit the array.
3. **Output Section**:
   * Display the result of the computation (true/false) indicating whether it's possible to split the array with the same average.
   * If the result is true, you can provide additional information on how the split was achieved.
4. **Flexible Layout**:
   * Ensure that the UI is responsive, adjusting to different screen sizes and orientations.
   * Utilize responsive design techniques to adapt elements fluidly.
5. **Color Selection**:
   * Choose colors that are visually pleasing and provide good contrast for readability.
   * Use a combination of light and dark colors for background and text to enhance readability.
   * Consider using colors to highlight important elements such as the result or user inputs.
6. **User-Friendly Design**:
   * Arrange elements in an intuitive manner for easy understanding and interaction.
   * Use clear labels and instructions to guide the user through the process.
   * Provide feedback to the user after they input the array, indicating whether the computation is possible or not.

**2.Feasible Elements Used**

In order to split the given integer array nums into two arrays A and B such that the average of both arrays is equal, a crucial observation lies in understanding that both arrays need to have the same average, not necessarily the same sum. This implies that the sum of elements in each array may vary as long as their averages remain identical. Thus, the problem transforms into a search for subsets of nums whose sums can achieve equal averages.

To implement this, we can explore a depth-first search (DFS) approach or dynamic programming to systematically examine different subsets of nums and calculate their sums. During this process, we keep track of the sums and their corresponding counts for each subset. By ensuring that both arrays A and B have the same average, we effectively distribute the elements of nums into these two arrays.

Furthermore, this problem necessitates a careful consideration of edge cases, such as ensuring that both arrays are non-empty and that the division of elements between A and B results in equal averages. Additionally, we must account for scenarios where no feasible split exists, which occurs when it's impossible to achieve equal averages due to the nature of the input array.

In summary, solving the "Split Array With Same Average" problem involves exploring subsets of nums to distribute its elements into two arrays while maintaining equal averages. This process entails meticulous calculation and management of sums to ensure the feasibility of such a split.

**3. Elements Function**

The "split array with same average" problem revolves around distributing elements from a given integer array, **nums**, into two separate arrays, A and B. The objective is to ensure that both arrays have non-zero length and share the same average value. To achieve this, an "Elements" function could be devised. This function iterates through various combinations of elements from the input array, assessing whether it's possible to create two arrays with identical averages. This involves exploring different subsets of elements and checking if the sum of their elements divided by their lengths equals the average. The function would return true if such a distribution is feasible and false otherwise. Essentially, it involves a search for a partition of the array that satisfies the required conditions, leveraging mathematical properties related to averages and subsets.

**CODE**

**:** class Solution {

public:

    bool splitArraySameAverage(vector<int>& A) {

        if(A.size()<=1) {

            return false;

        }

        int n = A.size();

        int Sum = 0;

        for(auto x: A) Sum += x;

        int dp[Sum+1];

        memset(dp, 0, sizeof dp);

        dp[A[0]] = 2;

        for(int i=1; i<n; i++) {

            for(int s=Sum-A[i]; s>=0; s--) {

                if(dp[s])

                    dp[s+A[i]] |= (dp[s]<<1);

            }

            dp[A[i]] |= 2;

        }

        for(int len=1; len<n; len++) {

            if( (Sum\*len)%n == 0 && ((1<<len) & dp[Sum\*len/n])) {

                return true;

            }

        }

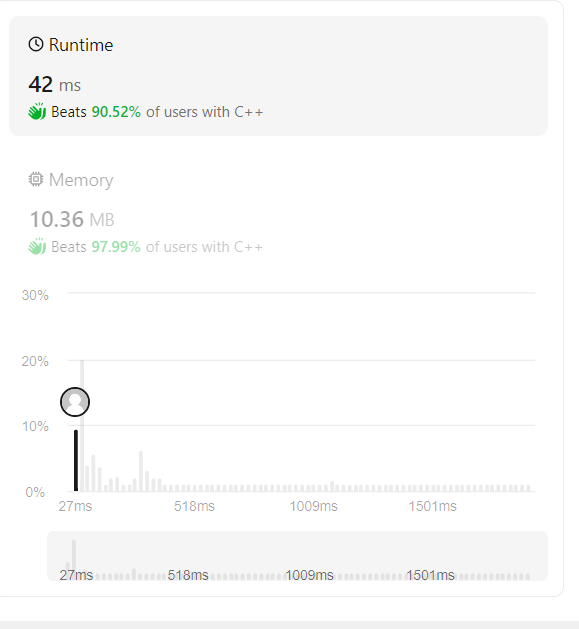
        return false;

    }

};

**Output**

[1,2,3,4,5,6,7,8]

****

**CONCLUSION:**

1. **Feasibility**: It's not always possible to split the array into two parts with the same average. Some arrays cannot be partitioned in a way that satisfies the condition.
2. **Complexity**: Finding a solution involves exploring various partitioning possibilities. It requires calculating averages and considering all combinations, which can be computationally expensive for large arrays.
3. **Algorithmic Approach**: A common approach to solving this problem involves dynamic programming or backtracking techniques to explore all possible partitions efficiently.
4. **Optimization**: It's essential to optimize the solution to handle large arrays efficiently. This can include techniques like memoization or pruning the search space to avoid unnecessary computations.