unstop | Share all Chocolates

Problem | Solution | Submissions

Problem Statement

Alice's mom bought n bags of chocolates numbered from 1 to n. The i^{th} bag (1 <= i<= n) has a_i chocolates. Alice wants to share some of the chocolate bags with Bob such that both of them get an equal number of chocolates, but she is not able to divide them in such a way. Now, she wonders if it is even possible to do so. Help Alice find out if the bags can be divided between her and Bob such that both of them get an equal number of chocolates.

Input Format

The first line of the input contains a single integer n – the number of bags that Alice's mom bought.

The second line contains n space-separated integers $a_1, a_2, ..., a_n$ – the number of chocolates in each bag.

Output Format

If it is possible to divided the chocolate bags in two parts with equal chocolates, print "YES" (without quotes). Otherwise, print "NO" (without quotes).

Constraints

1 <= n <= 500

1 <= a_i<= 100

Sample Testcase 0

Testcase Input

Sample Testcase 0

Testcase Input


```
4
1 5 11 5
```

Testcase Output

```
YES
```

Sample Testcase 1

Testcase Input

```
4
1 2 3 5
```

Testcase Output

```
NO
```

Code

```
#include <iostream>
#include <vector>

using namespace std;

bool canDivideChocolatesEqually(int n, vector<int>& chocolates) {
    int total_chocolates = 0;

    for (int i = 0; i < n; i++) {
        total_chocolates += chocolates[i];
    }

    if (total_chocolates % 2 != 0) {
        return false; // If the total number of chocolates is odd, it's impossible to divide them
equally.
    }

    int target_chocolates = total_chocolates / 2;

    // Create a 1D boolean DP array to track which chocolates can be used to reach the
target_chocolates.
    vector<bool> dp(target_chocolates + 1, false);
    dp[0] = true; // Zero chocolates are always possible.

    for (int i = 0; i < n; i++) {
        for (int j = target_chocolates; j >= chocolates[i]; j--) {
            dp[j] = dp[j] || dp[j - chocolates[i]];
        }
    }

    return dp[target_chocolates];
}

int main() {
    int n;
    cin >> n;

    vector<int> chocolates(n);

    for (int i = 0; i < n; i++) {
        cin >> chocolates[i];
    }
```

```
    if (canDivideChocolatesEqually(n, chocolates)) {
        cout << "YES" << endl;
    } else {
        cout << "NO" << endl;
    }

    return 0;
}
```

The dynamic programming approach presented earlier is an efficient way to solve this problem, and it has a time complexity of O(n * target), where `n` is the number of bags and `target` is the total number of chocolates divided by 2. This approach is suitable for the given constraints (n <= 500), and it provides an optimal solution.

An alternative approach to solving this problem is to use a recursive or backtracking approach that explores all possible combinations of bags to check if they can be divided equally. However, this approach can become very inefficient, especially for large inputs, as it may have an exponential time complexity.

The dynamic programming approach is recommended because it provides an efficient solution that can handle the constraints of the problem effectively and avoid unnecessary exploration of all possible bag combinations, which would be less efficient.