Minimum Subset Sum

minimum set sum explain all the methods and code in c++

The **Minimum Subset Sum Difference** problem involves dividing a set of integers into two subsets such that the absolute difference between their sums is minimized. Here's a breakdown of the problem, methods to solve it, and the corresponding C++ implementations.

Problem Explanation

Given a set of integers arr with size n, the goal is to partition it into two subsets $S1$ and $S2$ such that the absolute difference between their sums, $|S1\_sum - S2\_sum|$, is minimized.

Approach to Solve the Problem

The problem can be solved using the following methods:

1. **Recursive Approach**
2. **Recursive + Memoization (Top-Down Dynamic Programming)**
3. **Dynamic Programming (Bottom-Up Approach)**

---

1. Recursive Approach

Idea:

- Explore all possible subsets.
- Calculate the sum of each subset and compute the difference.
- Return the minimum difference.

Algorithm:

1. Use recursion to calculate all possible subset sums.
2. Find the subset with a minimum difference between the two partitions.

Code:

```cpp


#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

int minSubsetSumDiff(int index, int sum1, int sum2, vector<int>& arr) {
```

```
    if (index == arr.size()) {
        return abs(sum1 - sum2);
    }

    // Include current element in subset S1
    int includeInS1 = minSubsetSumDiff(index + 1, sum1 + arr[index], sum2, arr);

    // Include current element in subset S2
    int includeInS2 = minSubsetSumDiff(index + 1, sum1, sum2 + arr[index], arr);

    return min(includeInS1, includeInS2);
}

int main() {
    vector<int> arr = {1, 6, 11, 5};
    cout << "Minimum Subset Sum Difference: " << minSubsetSumDiff(0, 0, 0, arr) << endl;
    return 0;
}
```

---

2. Recursive + Memoization

Idea:

- Use a 2D DP table to store results of subproblems to avoid redundant calculations.

Code:

cpp

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

int minSubsetSumDiff(int index, int sum1, int totalSum, vector<int>& arr, vector<vector<int>
>& dp) {
    if (index == arr.size()) {
        return abs((totalSum - sum1) - sum1);
    }
    if (dp[index][sum1] != -1) {
        return dp[index][sum1];
    }

    // Include current element in subset S1
    int includeInS1 = minSubsetSumDiff(index + 1, sum1 + arr[index], totalSum, arr, dp);

    // Exclude current element from subset S1
    int excludeFromS1 = minSubsetSumDiff(index + 1, sum1, totalSum, arr, dp);

    return dp[index][sum1] = min(includeInS1, excludeFromS1);
}

int main() {
    vector<int> arr = {1, 6, 11, 5};
    int totalSum = 0;
    for (int num : arr) {
        totalSum += num;
    }

    vector<vector<int>> dp(arr.size(), vector<int>(totalSum + 1, -1));
    cout << "Minimum Subset Sum Difference: " << minSubsetSumDiff(0, 0, totalSum, arr, dp) <
< endl;
    return 0;
}
```

3. Dynamic Programming (Bottom-Up Approach)

Idea:

- Use the **subset sum problem** logic.
- Compute all possible sums of subsets using a DP table.
- Minimize the difference between $S1$ and $S2$.

Algorithm:

1. Calculate the total sum of the array.
2. Use a DP array to determine which subset sums are possible.
3. Find the subset sum closest to $\text{totalSum}/2$.

Code:

cpp

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

int minSubsetSumDiff(vector<int>& arr) {
    int n = arr.size();
    int totalSum = 0;
    for (int num : arr) {
        totalSum += num;
    }

    vector<vector<bool>> dp(n + 1, vector<bool>(totalSum / 2 + 1, false));

    // Initialize DP table
    for (int i = 0; i <= n; i++) {
        dp[i][0] = true;
    }

    // Fill DP table
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= totalSum / 2; j++) {
            if (arr[i - 1] <= j) {
                dp[i][j] = dp[i - 1][j] || dp[i - 1][j - arr[i - 1]];
            } else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }

    // Find the closest subset sum to totalSum / 2
    int minDiff = INT_MAX;
    for (int j = totalSum / 2; j >= 0; j--) {
        if (dp[n][j]) {
            minDiff = totalSum - 2 * j;
            break;
        }
    }

    return minDiff;
}

int main() {
    vector<int> arr = {1, 6, 11, 5};
    cout << "Minimum Subset Sum Difference: " << minSubsetSumDiff(arr) << endl;
```

```
    return 0;
}
```

Comparison of Methods

Method	Time Complexity	Space Complexity
Recursive	$O(2^n)$	$O(n)$
Recursive + Memoization	$O(n \times \text{sum})$	$O(n \times \text{sum})$
Dynamic Programming	$O(n \times \text{sum})$	$O(n \times \text{sum})$

Conclusion

The **Dynamic Programming (Bottom-Up)** approach is the most efficient and commonly used method for solving the Minimum Subset Sum Difference problem due to its lower time complexity and ease of implementation.