Two Sum using HashMap

Mr. Vikas Kumar
Assistant Professor
Industry Embedded Program

Lesson Plan

Subject/Course	**Competitive Coding**
Lesson Title	**Two Sum using HashMap**

Lesson Objectives
To use a HashMap for fast element retrieval from an array.
To identify two elements whose sum equals the given target.
To return the indices of the matched pair.
To optimize the solution from $O(n^2)$ to $O(n)$ using hashing.

Problem Statement:

Write a program to find two numbers in an array whose sum equals a given target value.

Concept

Why HashMap?

- A **HashMap** stores elements as *(key, value)* pairs.
- We can store each number's **complement (target – num)** while traversing the array.
- Lookup in HashMap is **O(1)** → efficient solution.

Applications:

- Pair matching problems.
- Detecting complement elements in arrays.
- Financial or resource allocation checks (sum validation).

Concept

Brute Force:

- Check every pair using 2 loops
- Time Complexity: $O(n^2)$ ✗

Optimized Approach:

- Use a HashMap to store values and their indices
- Check if target – currentValue exists
- Time Complexity: $O(n)$ ✓

Algorithm/Logic

1. Create a HashMap <Integer, Integer>
2. Loop through array elements
3. For each element:
   - complement = target – current
   - If complement exists in HashMap → return both indices
4. Else insert element and index into HashMap
5. If no pair found → Print "No Match Found"

Visualization

Example: Array = [2, 7, 11, 15], Target = 9

Step	Current Num	Complement (target - num)	Present in Map?	Map State
1	2	7	✗	{2=0}
2	7	2	✓ Found	{2=0, 7=1}

Pair found → Indices [0, 1] → Values (2, 7)

Code Implementation

```java
import java.util.*;

public class Practical19_TwoSum {
    public static int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[]{map.get(complement), i};
            }
            map.put(nums[i], i);
        }
        return new int[]{-1, -1}; // if not found
    }
```

```
16      public static void main(String[] args) {
17          int[] nums = {2, 7, 11, 15};
18          int target = 9;
19          int[] result = twoSum(nums, target);
20
21          System.out.println("Array: " + Arrays.toString(nums));
22          System.out.println("Target: " + target);
23          System.out.println("Indices: " + Arrays.toString(result));
24      }
25  }
```

Output

```
Array: [2, 7, 11, 15]
Target: 9
Indices: [0, 1]
```

Time & Space Complexity

Time Complexity: O(n) — single pass

Space Complexity: O(n) — HashMap storage

Summary

- Efficient method to search pair adding up to target
- Uses HashMap for instant lookup
- Suitable for large inputs
- One-pass solution

Summary

✅ Advantage:

- Eliminates nested loops
- Guaranteed faster performance

📌 Use Cases:

- E-Commerce cart matching
- Financial security validation
- Data search operations

Practice Questions:

1. Two Sum — LeetCode #1

🔗 https://leetcode.com/problems/two-sum/

Concept: Return indices using HashMap

Why Practice: Most frequent interview question

Practice Questions:

2. Implement Queue using Stacks — LeetCode #232

🔗 https://leetcode.com/problems/implement-queue-using-stacks/

Concept: Implement a queue using one or two stacks.

Why Practice: Strengthens your stack understanding by reversing the logic.

Thanks