Problem-Specific Implementation Templates

Template 1: Array/List Problems

Problem Type: Find Two Numbers that Sum to Target

Pseudocode:

```
FUNCTION twoSum(array, target):
    CREATE empty hashmap
    FOR each element in array:
        complement = target - element
        IF complement exists in hashmap:
            RETURN [complement_index, current_index]
        ADD element to hashmap with its index
    RETURN empty result
```

C++ Implementation:

cpp

```cpp
#include <vector>
#include <unordered_map>
using namespace std;

vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, int> numMap;

    for (int i = 0; i < nums.size(); i++) {
        int complement = target - nums[i];

        if (numMap.find(complement) != numMap.end()) {
            return {numMap[complement], i};
        }

        numMap[nums[i]] = i;
    }

    return {}; // No solution found
}
```

Java Implementation:

java

```java
import java.util.*;

public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> numMap = new HashMap<>();

    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i];

        if (numMap.containsKey(complement)) {
            return new int[]{numMap.get(complement), i};
        }

        numMap.put(nums[i], i);
    }

    return new int[0]; // No solution found
}
```

Python Implementation:

python

```python
def two_sum(nums, target):
    num_map = {}

    for i, num in enumerate(nums):
        complement = target - num

        if complement in num_map:
            return [num_map[complement], i]

        num_map[num] = i

    return []  # No solution found
```

Ruby Implementation:

ruby

```ruby
def two_sum(nums, target)
  num_map = {}

  nums.each_with_index do |num, i|
    complement = target - num

    if num_map.key?(complement)
      return [num_map[complement], i]
    end

    num_map[num] = i
  end

  []  # No solution found
end
```

Template 2: String Problems

Problem Type: Check if String is Palindrome

Pseudocode:

```
FUNCTION isPalindrome(string):
    left = 0
    right = string.length - 1

    WHILE left < right:
        IF string[left] != string[right]:
            RETURN false
        left++
        right--

    RETURN true
```

Implementation Pattern:

cpp

```cpp
// C++
bool isPalindrome(string s) {
    int left = 0, right = s.length() - 1;

    while (left < right) {
        // Skip non-alphanumeric characters
        while (left < right && !isalnum(s[left])) left++;
        while (left < right && !isalnum(s[right])) right--;

        if (tolower(s[left]) != tolower(s[right])) {
            return false;
        }

        left++;
        right--;
    }

    return true;
}
```

Template 3: Linked List Problems

Problem Type: Reverse Linked List

Pseudocode:

```
FUNCTION reverseList(head):
    previous = null
    current = head

    WHILE current is not null:
        next = current.next
        current.next = previous
        previous = current
        current = next

    RETURN previous
```

Implementation Pattern:

python

```python
# Python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def reverse_list(head):
    prev = None
    current = head

    while current:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node

    return prev
```

Template 4: Binary Tree Problems

Problem Type: Tree Traversal (Inorder)

Pseudocode:

```
FUNCTION inorderTraversal(root):
    IF root is null:
        RETURN empty list

    result = []
    result.addAll(inorderTraversal(root.left))
    result.add(root.val)
    result.addAll(inorderTraversal(root.right))

    RETURN result
```

Implementation Pattern:

java

```java
// Java
class TreeNode {
    int val;
    TreeNode left
```