😎

Trapping Rain Water 🌧️

Created	@July 28, 2024
Description	**Array, Two Pointers, Dynamic Programming, Stack, Monotonic Stack**
Tags	Arrays

Problem Statement: 🌂

💡 You are given an array of non-negative integers `height` representing the elevation map where the width of each bar is 1⛈️. Compute how much water it can trap after raining.

💡 **Example 1:** 📊

Input: `height = [0,1,0,2,1,0,1,3,2,1,2,1]`

Output: 6

Explanation: The above elevation map (black section) is represented by array `[0,1,0,2,1,0,1,3,2,1,2,1]`. In this case, 6 units of rain water (blue section) are being trapped. 💧

Example 2: 📊

Input: `height = [4,2,0,3,2,5]`

Output: 9

Constraints: 📝

- `n == height.length`
- `1 <= n <= 2 * 10^4`
- `0 <= height[i] <= 10^5`

Code (C++)💻

1. Brute Force 🔨

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

using namespace std;

int trap(vector<int>& height) {
    int n = height.size();
    if (n == 0) {
        return 0; // 🚫 Empty array, no water trapped
    }
    int totalWater = 0;
    for (int i = 1; i < n - 1; i++) { // 🔁 Iterate through the bars
        int leftMax = 0, rightMax = 0;
        for (int j = 0; j <= i; j++) {
            leftMax = max(leftMax, height[j]); // 📈 Find the highest bar to the left
        }
        for (int j = i; j < n; j++) {
            rightMax = max(rightMax, height[j]); // 📈 Find the highest bar to the right
        }
        int currentWater = min(leftMax, rightMax) - height[i]; // 💧 Calculate water trapped at
        if (currentWater > 0) {
            totalWater += currentWater; // 💧 Add to the total
        }
    }
    return totalWater;
}

int main() {
    vector<int> height = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
    cout << "Trapped Water: " << trap(height) << endl;
    return 0;
}
```

2. Two Pointers 🎯

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

using namespace std;

int trap(vector<int>& height) {
    int n = height.size();
    if (n == 0) {
        return 0; // 🚫 Empty array, no water trapped
    }
    int left = 0, right = n - 1;
    int leftMax = 0, rightMax = 0;
    int totalWater = 0;
    while (left < right) { // 🔁 Move pointers until they meet
        if (height[left] < height[right]) {
            if (height[left] >= leftMax) {
                leftMax = height[left]; // 📈 Update left maximum
            } else {
                totalWater += leftMax - height[left]; // 💧 Add water trapped at the left bar
            }
            left++; // ➡️ Move left pointer
        } else {
            if (height[right] >= rightMax) {
                rightMax = height[right]; // 📈 Update right maximum
            } else {
                totalWater += rightMax - height[right]; // 💧 Add water trapped at the right bar
            }
            right--; // ⬅️ Move right pointer
        }
    }
    return totalWater;
}

int main() {
    vector<int> height = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
    cout << "Trapped Water: " << trap(height) << endl;
    return 0;
}
```

3. Dynamic Programming 📈

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

using namespace std;

int trap(vector<int>& height) {
    int n = height.size();
    if (n == 0) {
        return 0; // 🚫 Empty array, no water trapped
    }
    vector<int> leftMax(n, 0); // 📈 Left maximums up to each index
    vector<int> rightMax(n, 0); // 📈 Right maximums up to each index
    leftMax[0] = height[0];
    rightMax[n - 1] = height[n - 1];
    for (int i = 1; i < n; i++) {
        leftMax[i] = max(leftMax[i - 1], height[i]); // 📈 Calculate left maximums
    }
    for (int i = n - 2; i >= 0; i--) {
        rightMax[i] = max(rightMax[i + 1], height[i]); // 📈 Calculate right maximums
    }
    int totalWater = 0;
    for (int i = 1; i < n - 1; i++) {
        int currentWater = min(leftMax[i], rightMax[i]) - height[i]; // 💧 Calculate water trapp
        if (currentWater > 0) {
            totalWater += currentWater; // 💧 Add to the total
        }
    }
    return totalWater;
}

int main() {
    vector<int> height = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
    cout << "Trapped Water: " << trap(height) << endl;
    return 0;
}
```

Algorithm😄:

Dry runs for each approach🎉:

Input `height = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]`:

💡 **1. Brute Force** 🔨

1. **Initialization** 🏁**:**
   - `totalWater = 0` 💧
2. **Iteration 1:**
   - `i = 1`, `height[i] = 1`
   - `leftMax = 1`, `rightMax = 3` 📈
   - `currentWater = 0` 💧
   - `totalWater = 0` 💧
3. **Iteration 2:**
   - `i = 2`, `height[i] = 0`
   - `leftMax = 1`, `rightMax = 3` 📈
   - `currentWater = 1` 💧
   - `totalWater = 1` 💧
4. **Iteration 3:**
   - `i = 3`, `height[i] = 2`
   - `leftMax = 2`, `rightMax = 3` 📈
   - `currentWater = 1` 💧
   - `totalWater = 2` 💧
5. **Iteration 4:**
   - `i = 4`, `height[i] = 1`
   - `leftMax = 2`, `rightMax = 3` 📈
   - `currentWater = 1` 💧
   - `totalWater = 3` 💧
6. **Iteration 5:**
   - `i = 5`, `height[i] = 0`
   - `leftMax = 2`, `rightMax = 3` 📈
   - `currentWater = 2` 💧
   - `totalWater = 5` 💧
7. **Iteration 6:**
   - `i = 6`, `height[i] = 1`
   - `leftMax = 2`, `rightMax = 3` 📈
   - `currentWater = 1` 💧
   - `totalWater = 6` 💧

Final Output: `totalWater = 6` 🎉

2. Two Pointers 🎯

1. **Initialization** 🏁**:**
   - `left = 0`, `right = 11` 🎯
   - `leftMax = 0`, `rightMax = 1` 📈
   - `totalWater = 0` 💧

2. **Iteration 1:**
   - `height[left] = 0`, `height[right] = 1`
   - `leftMax = 1` 📈 (Update leftMax)
   - `left = 1` ➡️
3. **Iteration 2:**
   - `height[left] = 1`, `height[right] = 1`
   - `right = 10` ⬅️
4. **Iteration 3:**
   - `height[left] = 1`, `height[right] = 2`
   - `rightMax = 2` 📈 (Update rightMax)
   - `right = 9` ⬅️
5. **Iteration 4:**
   - `height[left] = 1`, `height[right] = 1`
   - `totalWater = 1` 💧 (Add water trapped at left)
   - `left = 2` ➡️
6. **Iteration 5:**
   - `height[left] = 0`, `height[right] = 1`
   - `totalWater = 2` 💧 (Add water trapped at left)
   - `left = 3` ➡️
7. **Iteration 6:**
   - `height[left] = 2`, `height[right] = 1`
   - `leftMax = 2` 📈 (Update leftMax)
   - `left = 4` ➡️
8. **Iteration 7:**
   - `height[left] = 1`, `height[right] = 1`
   - `totalWater = 3` 💧 (Add water trapped at left)
   - `left = 5` ➡️
9. **Iteration 8:**
   - `height[left] = 0`, `height[right] = 1`
   - `totalWater = 5` 💧 (Add water trapped at left)
   - `left = 6` ➡️
10. **Iteration 9:**

- `height[left] = 1`, `height[right] = 1`
- `leftMax = 3` 📈 (Update leftMax)
- `left = 7` ➡️

1. **Iteration 10:**

- `height[left] = 3`, `height[right] = 1`
- `left = 8` ➡️

1. **Iteration 11:**

- `height[left] = 2`, `height[right] = 1`
- `totalWater = 6` 💧 (Add water trapped at left)

- `left = 9` ➡️

1. **Iteration 12:**

- `height[left] = 1`, `height[right] = 1`
- `left = 10` ➡️

Final Output: `totalWater = 6` 🎉

3. Dynamic Programming 📈

1. **Initialization** 🏁**:**
    - `leftMax = [0, 0, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3]` 📈
    - `rightMax = [3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 1, 0]` 📈
    - `totalWater = 0` 💧
2. **Calculate Water Trapped** 💧**:**
    - For each `i` from `1` to `len(height) - 1`:
        - `i = 1`, `currentWater = min(leftMax[1], rightMax[1]) - height[1] = 0 - 1 = -1` (Since it's negative, no water is trapped here.)
        - `i = 2`, `currentWater = min(leftMax[2], rightMax[2]) - height[2] = 1 - 0 = 1`, `totalWater = 1`
        - `i = 3`, `currentWater = min(leftMax[3], rightMax[3]) - height[3] = 1 - 2 = -1` (No water trapped.)
        - `i = 4`, `currentWater = min(leftMax[4], rightMax[4]) - height[4] = 2 - 1 = 1`, `totalWater = 2`
        - `i = 5`, `currentWater = min(leftMax[5], rightMax[5]) - height[5] = 2 - 1 = 1`, `totalWater = 3`
        - `i = 6`, `currentWater = min(leftMax[6], rightMax[6]) - height[6] = 2 - 0 = 2`, `totalWater = 5`
        - `i = 7`, `currentWater = min(leftMax[7], rightMax[7]) - height[7] = 2 - 1 = 1`, `totalWater = 6`
        - `i = 8`, `currentWater = min(leftMax[8], rightMax[8]) - height[8] = 2 - 3 = -1` (No water trapped.)
        - `i = 9`, `currentWater = min(leftMax[9], rightMax[9]) - height[9] = 2 - 2 = 0` (No water trapped.)
        - `i = 10`, `currentWater = min(leftMax[10], rightMax[10]) - height[10] = 2 - 1 = 1`, `totalWater = 7`
        - `i = 11`, `currentWater = min(leftMax[11], rightMax[11]) - height[11] = 2 - 2 = 0` (No water trapped.)

Final Output: `totalWater = 7` 🎉

Java ☕

```
import java.util.Arrays;

public class TrappingRainWater {

    // 1. Brute Force 🔨
    public static int trap1(int[] height) {
        int n = height.length;
        if (n == 0) {
            return 0; // 🚫 Empty array, no water trapped
        }
        int totalWater = 0;
        for (int i = 1; i < n - 1; i++) { // 🔁 Iterate through the bars
            int leftMax = 0, rightMax = 0;
            for (int j = 0; j <= i; j++) {
                leftMax = Math.max(leftMax, height[j]); // 📈 Find the highest bar to the left
            }
            for (int j = i; j < n; j++) {
                rightMax = Math.max(rightMax, height[j]); // 📈 Find the highest bar to the rig
            }
            int currentWater = Math.min(leftMax, rightMax) - height[i]; // 💧 Calculate water t
            if (currentWater > 0) {
                totalWater += currentWater; // 💧 Add to the total
            }
        }
        return totalWater;
    }

    // 2. Two Pointers 🎯
    public static int trap2(int[] height) {
        int n = height.length;
        if (n == 0) {
            return 0; // 🚫 Empty array, no water trapped
        }
        int left = 0, right = n - 1;
        int leftMax = 0, rightMax = 0;
        int totalWater = 0;
        while (left < right) { // 🔁 Move pointers until they meet
            if (height[left] < height[right]) {
                if (height[left] >= leftMax) {
                    leftMax = height[left]; // 📈 Update left maximum
                } else {
                    totalWater += leftMax - height[left]; // 💧 Add water trapped at the left ba
                }
                left++; // ➡️ Move left pointer
            } else {
                if (height[right] >= rightMax) {
                    rightMax = height[right]; // 📈 Update right maximum
                } else {
                    totalWater += rightMax - height[right]; // 💧 Add water trapped at the righ
                }
                right--; // ⬅️ Move right pointer
            }
        }
        return totalWater;
    }

    // 3. Dynamic Programming 📈
    public static int trap3(int[] height) {
        int n = height.length;
        if (n == 0) {
            return 0; // 🚫 Empty array, no water trapped
        }
        int[] leftMax = new int[n]; // 📈 Left maximums up to each index
        int[] rightMax = new int[n]; // 📈 Right maximums up to each index
        leftMax[0] = height[0];
        rightMax[n - 1] = height[n - 1];
        for (int i = 1; i < n; i++) {
            leftMax[i] = Math.max(leftMax[i - 1], height[i]); // 📈 Calculate left maximums
        }
        for (int i = n - 2; i >= 0; i--) {
            rightMax[i] = Math.max(rightMax[i + 1], height[i]); // 📈 Calculate right maximums
        }
        int totalWater = 0;
        for (int i = 1; i < n - 1; i++) {
            int currentWater = Math.min(leftMax[i], rightMax[i]) - height[i]; // 💧 Calculate wa
            if (currentWater > 0) {
                totalWater += currentWater; // 💧 Add to the total
            }
        }
        return totalWater;
    }

    public static void main(String[] args) {
        int[] height = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
        System.out.println("Approach 1: " + trap1(height));
        System.out.println("Approach 2: " + trap2(height));
        System.out.println("Approach 3: " + trap3(height));
    }
}
```

Python 🐍

```
def trap1(height):
    """
    Approach 1: Brute Force 🔨
    """
    n = len(height)
    if n == 0:
        return 0  # 🚫 Empty array, no water trapped
    totalWater = 0
    for i in range(1, n - 1):  # 🔁 Iterate through the bars
        leftMax = 0
        rightMax = 0
        for j in range(i + 1):
            leftMax = max(leftMax, height[j])  # 📈 Find the highest bar to the left
        for j in range(i, n):
            rightMax = max(rightMax, height[j])  # 📈 Find the highest bar to the right
        currentWater = min(leftMax, rightMax) - height[i]  # 💧 Calculate water trapped at curre
        if currentWater > 0:
            totalWater += currentWater  # 💧 Add to the total
    return totalWater

def trap2(height):
    """
    Approach 2: Two Pointers 🎯
    """
    n = len(height)
    if n == 0:
        return 0  # 🚫 Empty array, no water trapped
    left = 0
    right = n - 1
    leftMax = 0
    rightMax = 0
    totalWater = 0
    while left < right:  # 🔁 Move pointers until they meet
        if height[left] < height[right]:
            if height[left] >= leftMax:
                leftMax = height[left]  # 📈 Update left maximum
            else:
                totalWater += leftMax - height[left]  # 💧 Add water trapped at the left bar
            left += 1  # ➡️ Move left pointer
        else:
            if height[right] >= rightMax:
                rightMax = height[right]  # 📈 Update right maximum
            else:
                totalWater += rightMax - height[right]  # 💧 Add water trapped at the right bar
            right -= 1  # ⬅️ Move right pointer
    return totalWater

def trap3(height):
    """
    Approach 3: Dynamic Programming 📈
    """
    n = len(height)
    if n == 0:
        return 0  # 🚫 Empty array, no water trapped
    leftMax = [0] * n  # 📈 Left maximums up to each index
    rightMax = [0] * n  # 📈 Right maximums up to each index
    leftMax[0] = height[0]
    rightMax[n - 1] = height[n - 1]
    for i in range(1, n):
        leftMax[i] = max(leftMax[i - 1], height[i])  # 📈 Calculate left maximums
    for i in range(n - 2, -1, -1):
        rightMax[i] = max(rightMax[i + 1], height[i])  # 📈 Calculate right maximums
    totalWater = 0
    for i in range(1, n - 1):
        currentWater = min(leftMax[i], rightMax[i]) - height[i]  # 💧 Calculate water trapped a
        if currentWater > 0:
            totalWater += currentWater  # 💧 Add to the total
    return totalWater

if __name__ == "__main__":
    height = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]
    print("Approach 1:", trap1(height))
    print("Approach 2:", trap2(height))
    print("Approach 3:", trap3(height))
```