1184. Distance Between Bus Stops

Easy Array Leetcode Link

Problem Description

The problem presents a circular bus route consisting of `n` stops, numbered from `0` to `n - 1`. Each stop is connected to the next, and the last stop is connected back to the first, forming a circle. The array `distance` contains the distances between each pair of consecutive stops (`i` and `(i + 1) % n`). The bus can travel in both the clockwise and counterclockwise directions.

The task is to calculate the shortest distance between two given stops: the `start` and the `destination`. Since the bus can travel in both directions, we need to compare the travel distances and choose the one that requires the least amount of travel.

Intuition

The solution is based on the premise that the shortest distance between two points in a circle can be in either the clockwise or counter-clockwise direction. Therefore, we need to:

1. Calculate the distance traveling from `start` to `destination` clockwise.
2. Calculate the total distance around the circle to find the distance of the counter-clockwise path, which is the total distance minus the clockwise distance.
3. Finally, return the smaller of the two calculated distances.

To achieve this, an intuitive approach is to:

- Iterate from the `start` stop to the `destination` stop, accumulating the distances between consecutive stops.
- Then, calculate the sum of all distances to capture the complete circuit distance.
- The clockwise distance is the accumulation from start to `destination`.
- The counterclockwise distance can be derived by subtracting the clockwise distance from the total distance.
- After having both distances, we compare them and return the smaller one as the shortest path between the `start` and `destination` stops.

Solution Approach

The implementation of the solution follows these steps:

1. **Initializing Accumulator (`a`) and Circle Size (`n`)**: An accumulator variable `a` is set to 0 to keep track of the distance traveled in the clockwise direction. The variable `n` denotes the size of the distance array, which is equivalent to the number of stops in the circle.

2. **Iterative Computation of Clockwise Distance**: A while loop is used to traverse the bus stops from `start` to `destination`. During each iteration, the distance from the current stop to the next is added to the accumulator `a`. The current stop `start` is updated to the next stop using `(start + 1) % n`. This update statement guarantees that the index remains within the bounds of the array, effectively looping from the last stop back to the first. The loop continues until `start` matches `destination`.

3. **Calculation of Counterclockwise Distance**: Once the clockwise distance is known, the counterclockwise distance is calculated by subtracting the accumulator `a` from the sum of all distances in the array. The built-in `sum()` function computes the total circle distance.

4. **Return Minimum Distance**: Using the built-in `min()` function, the algorithm then returns the smaller value between the accumulated clockwise distance `a` and the counterclockwise distance (`sum(distance) - a`).

This approach uses no complex data structures; it relies on arithmetic and looping to achieve the desired outcome. The pattern involves traversing the array in a circular fashion using modulo arithmetic, which is a common technique in problems involving circular data structures. Additionally, the use of built-in functions for summing and finding the minimum value makes the code concise and efficient.

Example Walkthrough

Let's illustrate the solution approach with a simple example:

Suppose we have 5 bus stops on a circular route with the following distances between consecutive stops: `distance = [3, 10, 1, 5, 8]`. Thus, we have `n = 5` (5 stops).

- The distance from stop 0 to stop 1 is 3.
- The distance from stop 1 to stop 2 is 10.
- The distance from stop 2 to stop 3 is 1.
- The distance from stop 3 to stop 4 is 5.
- The distance from stop 4 back to stop 0 is 8.

Now, let's find the shortest distance between `start = 1` and `destination = 3`.

Following the solution approach:

1. Initializing Accumulator (`a`) and Circle Size (`n`):
   - Set `a = 0`
   - `n` equals the length of the distance array, so `n = 5`.
2. Iterative Computation of Clockwise Distance:
   - Start at `start = 1`. Add to `a` the distance to the next stop (distance[1] = 10).
   - Move to the next stop, `start = (1 + 1) % 5 = 2`. Add to `a` the distance to the next stop (distance[2] = 1).
   - Since we've reached the `destination` stop, we stop accumulating the distance. The clockwise distance `a` is now `10 + 1 = 11`.
3. Calculation of Counterclockwise Distance:
   - Compute the sum of all distances: `sum(distance) = 3 + 10 + 1 + 5 + 8 = 27`.
   - Calculate the counterclockwise distance: `27 - 11 = 16`.
4. Return Minimum Distance:
   - Compare the clockwise and counterclockwise distances and choose the smaller one.
   - Since `11 < 16`, the shortest distance is `11`.

Hence, for this example, the shortest distance from stop 1 to stop 3 is `11` units.

Python Solution

```python
class Solution:
    def distanceBetweenBusStops(self, distances: List[int], start: int, destination: int) -> int:
        # Initialize the distance traveled clockwise from 'start' to 'destination'
        clockwise_distance = 0
        # Total number of bus stops
        total_stops = len(distances)

        # Adjust indices if 'start' is greater than 'destination' for a direct path
        if start > destination:
            start, destination = destination, start

        # Calculate the clockwise distance by adding the distances of each stop
        for i in range(start, destination):
            clockwise_distance += distances[i]

        # Calculate the counter-clockwise distance by subtracting the clockwise distance
        # from the total distance around the bus stops
        counter_clockwise_distance = sum(distances) - clockwise_distance

        # Return the minimum of the two distances as the result
        return min(clockwise_distance, counter_clockwise_distance)

# Note: It is important to import List from typing to use the List type hint in the function signature.
from typing import List

```

Java Solution

```java
class Solution {
    public int distanceBetweenBusStops(int[] distances, int start, int destination) {
        // Calculate the total distance of the circular route.
        int totalDistance = 0;
        for (int distance : distances) {
            totalDistance += distance;
        }

        // Initialize the clock-wise distance traveled.
        int clockwiseDistance = 0;

        // Calculate the clock-wise distance from 'start' to 'destination'.
        int currentIndex = start;
        while (currentIndex != destination) {
            clockwiseDistance += distances[currentIndex];
            currentIndex = (currentIndex + 1) % distances.length; // Move to the next stop in a circular manner.
        }

        // Calculate the counter-clockwise distance by subtracting the
        // clock-wise distance from the total distance.
        int counterClockwiseDistance = totalDistance - clockwiseDistance;

        // Return the minimum of the clock-wise and counter-clockwise distances.
        return Math.min(clockwiseDistance, counterClockwiseDistance);
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <numeric>

class Solution {
public:
    int distanceBetweenBusStops(std::vector<int>& distance, int start, int destination) {
        // Calculate the sum of all distances
        int totalDistance = std::accumulate(distance.begin(), distance.end(), 0);

        // Initialize the distance for the first path
        int pathDistance = 0;

        // Calculate the length of the bus route
        int numStops = distance.size();

        // Add up distances in the clockwise direction from start to destination
        while (start != destination) {
            pathDistance += distance[start];

            // Wrap around if we reach the end of the vector
            start = (start + 1) % numStops;
        }

        // Return the minimum of the clockwise distance and the counterclockwise distance
        // since buses can travel in both directions
        return std::min(pathDistance, totalDistance - pathDistance);
    }
};

```

Typescript Solution

```typescript
// Calculates the shortest distance between two bus stops on a circular route
// distance: array representing the distances between each pair of adjacent bus stops
// start: the start bus stop index
// destination: the destination bus stop index
function distanceBetweenBusStops(distance: number[], start: number, destination: number): number {
    // Calculate the total distance of the entire circular route
    const totalDistance = distance.reduce((accumulatedDistance, currentDistance) => accumulatedDistance + currentDistance, 0);

    // Calculate the distance of the path from 'start' to 'destination' moving forward
    let distanceForward = 0;
    const numOfStops = distance.length;

    // Loop through the bus stops from 'start' to 'destination'
    while (start !== destination) {
        // Accumulate the distance of the direct path
        distanceForward += distance[start];
        // Move to the next stop, wrap around to 0 if at the end of the array
        start = (start + 1) % numOfStops;
    }

    // Return the minimum between the direct path and the reverse path
    // The reverse path is the total distance minus the direct path distance
    return Math.min(distanceForward, totalDistance - distanceForward);
}

```

Time and Space Complexity

The time complexity of the code is `O(n)` where `n` is the number of elements in the `distance` list. This results from iterating over the elements starting from the `start` index and stopping once it reaches the `destination`. In the worst case, this could require traversing the entire list.

The space complexity of the code is `O(1)` since it uses a fixed amount of additional space. The variables `a`, `n`, and `start` are the only extra storage used and do not depend on the size of the input list.