1643. Kth Smallest Instructions

Problem Description

Bob starts from cell `(0, 0)` and wants to reach the `destination` at `(row, column)`. He can only go right, denoted with `'H'`, or down, denoted with `'V'`. We need to give Bob a set of instructions that not only gets him to `destination` but also ensures that this set of instructions is the `k`th lexicographically smallest sequence possible.

Bob is picky and wants only the `k`th lexicographically smallest instructions, where `k` is 1-indexed. This means that if all possible correct sequences of instructions are sorted lexicographically, Bob wants the k-th sequence in this sorted order.

The `destination` is given in the form of an integer array `[v, h]` where `v` represents the vertical moves Bob needs to make downwards, and `h` represents the horizontal moves Bob needs to make to the right. In the case that `destination` is `[2, 3]`, he needs to go down twice (`'V'`) and right three times (`'HHH'`). The combinations `'HHHVV'`, `'HVHVH'`, `'VHVVH'`, etc., are different ways to arrange these moves.

Intuition

The solution is based on understanding combinatorics and lexicographical ordering. We base our solution on the facts that:

1. There are `comb(h + v, h)` ways to arrange `h` horizontal moves and `v` vertical moves, with `h + v` being the total number of moves made.
2. A lexicographically smaller sequence starts with the smallest possible character, which is `'H'` in this case.

So the intuition behind the solution is as follows:

- We initialize the moves by calculating the total possible combinations to reach the destination given the current number of horizontal `'H'` and vertical `'V'` moves remaining. This is done using the combinatorial function `comb`.
- At every step, we decide whether to place an `'H'` or a `'V'`. If we place an `'H'`, we reduce the number of horizontal moves, and if we place a `'V'`, we reduce the vertical moves.
- We check if the remaining combinations after placing an `'H'` are less than `k`. If they are, it means that adding an `'H'` would not reach the k-th sequence. In such a case, we must add a `'V'`, decrement `v`, and adjust `k` by subtracting the number of sequences that would've started with an `'H'`.
- If the remaining combinations after placing an `'H'` are more or equal to `k`, it means an `'H'` can be added to the sequence without surpassing the k-th smallest requirement.
- We repeat this process until we run out of either horizontal or vertical moves.
- As `'H'` is lexicographically smaller than `'V'`, we aim to add `'H'` whenever possible, following the rules above to respect the k-th smallest condition.

By iterating through all moves and following this process, we construct the `kth` smallest lexicographical sequence that leads Bob to his destination.

Solution Approach

The implementation is straightforward given our understanding of the intuition behind the problem. Here's a step-by-step explanation of the approach:

1. We begin by identifying the number of vertical and horizontal moves required to reach the destination, `v` and `h` respectively.
2. Define a string list `ans` which will hold the sequence of moves.
3. Iterate through the combined number of moves (`h + v`).
4. If there are no horizontal moves left (`h` is 0), we can only move down. So we add `'i'` to `ans` and continue.
5. Otherwise, calculate the current number of combinations with `x = comb(h + v - 1, h - 1)`. This represents the number of ways to arrange remaining moves if the next move is horizontal.
6. Compare `x` with `k`. If `k` is larger, it means that the k-th sequence is not in the group of sequences that start with an `'H'` followed by all possible combinations of the remaining moves. Therefore, add `'i'` to `ans`, decrement `v`, and subtract `x` from `k` to reflect that we skip all those combinations that start with `'H'`.
7. If `k` is not larger than `x`, it means that the k-th sequence does start with an `'H'` followed by the remaining moves. In this case, add `'H'` to `ans` and decrement `h`.
8. After the loop, `ans` will contain the k-th lexicographically smallest instruction set.

In terms of algorithms, data structures, and patterns, we are using:

- **String/List manipulation**: We use a list `ans` to build the sequence incrementally. The list is then converted to a string at the end.
- **Combinatorics**: By using the combinatorial function `comb`, we calculate the number of ways to arrange the remaining moves.
- **Decision-making under constraints**: At each step, the decision to add `'H'` or `'V'` is made based on the number of combinations remaining and the value of `k`.
- **Greedy approach**: At each step, we take the locally optimal choice (adding `'H'` if possible) to reach the global optimization (k-th smallest sequence) while iterating through the process.

The code given in the solution efficiently performs this approach and keeps track of the sequence to be returned.

Example Walkthrough

Let's illustrate the solution approach with an example where Bob has to reach a `destination` at `[2, 3]` and wants the 3rd lexicographically smallest sequence of instructions. This means `v = 2` for downwards moves and `h = 3` for rightwards moves, and `k = 3`.

1. Bob needs 2 `'i'` moves and 3 `'H'` moves. The possible sequences, sorted lexicographically, are:
    1. `HHHVV`
    2. `HHVHV`
    3. `HHVVH`
    4. `HVHHV`
    5. `HVHVH`
    6. `HVVHH`
    7. `VHHHV`
    8. `VHHVH`
    9. `VHVHH`
    10. `VVHHH`
2. Given the above sequences, we can see that the 3rd sequence is `HHVVH`. We will achieve the same result following our solution approach.
3. Begin with `h = 3`, `v = 2`, and `k = 3`. List `ans` is currently empty.
4. At the starting point (`h + v = 5` moves remaining), we calculate the combinations possible with the first move being `'H'`, which is `comb(4,2) = 6`. This number indicates the count of sequences beginning with `'H'`.
5. Comparing `x` (6) with `k` (3), since `k` is less than `x`, we add `'H'` to `ans` and decrement `h` to 2 (`h + v = 4` moves remaining).
6. We again calculate `x = comb(3, 2) = 3` for the possibility of the next move being `'H'`.
7. `k` equals `x`, so adding an `'H'` is still valid. We add another `'H'` and decrement `h` to 1 (`h + v = 3` moves remaining).
8. Calculate `x = comb(2, 1) = 2`. Since `k` is less than `x` (which is still 3), we cannot add an `'H'` because we need to find the 3rd sequence, and there are only 2 sequences starting with `HHH`. We add `'i'` to `ans`, decrement `v` to 1, and update `k` to `k - x = 1` (`h + v = 2` moves remaining).
9. Now, calculate `x = comb(1, 1) = 1` for the possibility of the next move being `'H'`.
10. Since `k` (1) equals `x` (1), we can add `'H'`. We add `'H'` to `ans`, decrement `h` to 0 (`h + v = 1` move remaining).
11. With only one move remaining and `h` being 0, we add the final `'V'`.
12. Now `ans` holds `HHVVH`, which is the 3rd lexicographically smallest sequence of instructions for Bob to reach `(2, 3)`.

By following each step of our solution approach, we've successfully assembled the 3rd lexicographically smallest set of instructions for Bob's journey to the destination.

Python Solution

```python
from math import comb
from typing import List

class Solution:
    def kthSmallestPath(self, destination: List[int], k: int) -> str:
        vertical, horizontal = destination
        ans = []

        # Loop through the total number of moves (sum of vertical and horizontal moves)
        for _ in range(horizontal + vertical):
            # If there are no more horizontal moves, all remaining moves are vertical
            if horizontal == 0:
                ans.append("V")
            else:
                # Calculate the number of combinations that start with a horizontal move
                combinations_with_h = comb(horizontal + vertical - 1, horizontal - 1)

                # If the kth path is greater than the number of paths starting with 'H',
                # then the path must start with a 'V' move instead.
                if k > combinations_with_h:
                    ans.append("V")
                    vertical -= 1
                    k -= combinations_with_h
                else:
                    # Otherwise, the path starts with a 'H' move
                    ans.append("H")
                    horizontal -= 1

        # Join all moves into a single path string and return it
        return "".join(ans)
```

Java Solution

```java
class Solution {
    public String kthSmallestPath(int[] destination, int k) {
        int verticalSteps = destination[0];
        int horizontalSteps = destination[1];
        int totalSteps = verticalSteps + horizontalSteps;

        // Create a combination lookup table where c[i][j] represents the number
        // of combinations of i elements taken j at a time
        int[][] combinations = new int[totalSteps + 1][horizontalSteps + 1];

        // Base case: there is 1 way to choose 0 items out of i
        combinations[0][0] = 1;
        for (int i = 1; i <= totalSteps; ++i) {
            combinations[i][0] = 1;
            for (int j = 1; j <= horizontalSteps; ++j) {
                // Calculate the number count using Pascal's triangle relationship
                combinations[i][j] = combinations[i - 1][j] + combinations[i - 1][j - 1];
            }
        }

        // Build the path using the combination count to make decisions
        StringBuilder path = new StringBuilder();
        for (int i = totalSteps; i > 0; --i) {
            // If no horizontal steps are left, we must take a vertical step
            if (horizontalSteps == 0) {
                path.append("V");
            } else {
                // Check how many combinations would there be if we took a horizontal step first;
                // this helps determine if kth path starts with 'H' or 'V' given the remaining steps
                int combinationsIfHFirst = combinations[verticalSteps + horizontalSteps - 1][horizontalSteps - 1];
                if (k > combinationsIfHFirst) {
                    // If 'k' is larger than the number of combinations with 'H' first,
                    // the kth path must start with 'V'. We subtract the combinations and reduce the vertical step
                    path.append("V");
                    k -= combinationsIfHFirst;
                    --verticalSteps;
                } else {
                    // Otherwise, we append 'H' to our path and reduce the horizontal step
                    path.append("H");
                    --horizontalSteps;
                }
            }
        }

        // Return the constructed path as a string
        return path.toString();
    }
}
```

C++ Solution

```cpp
class Solution {
public:
    // Function to find the k-th smallest path in a grid given the destination.
    string kthSmallestPath(vector<int>& destination, int k) {
        int v = destination[0]; // Vertical moves required
        int h = destination[1]; // Horizontal moves required
        int n = h + v; // Total number of moves

        // Initialize combinatorial lookup table to store the binomial coefficients
        vector<vector<int>> comb(n + 1, vector<int>(h + 1, 0)); // Fill the array with zeroes
        comb[0][0] = 1;

        // Populate the table with binomial coefficients using Pascal's Triangle
        for (int i = 1; i <= n; ++i) {
            comb[i][0] = 1;
            for (int j = 1; j <= h; ++j) {
                comb[i][j] = comb[i - 1][j] + comb[i - 1][j - 1];
            }
        }

        string ans; // String to hold the result

        // Construct the lexicographically k-th smallest path
        for (int i = 0; i < n; ++i) {
            if (h == 0) { // If no more horizontal moves, then move vertical
                ans.push_back('V');
            } else {
                // Find the number of sequences starting with 'H'
                int countHStart = comb[v + h - 1][h - 1];

                // If k is greater than countHStart, then the current move must be 'V'
                if (k > countHStart) {
                    ans.push_back('V');
                    --v; // One less vertical move needed
                    k -= countHStart; // Reduce k by the number of sequences counted
                } else {
                    // If k is within the range, the current move is 'H'
                    ans.push_back('H');
                    --h; // One less horizontal move needed
                }
            }
        }

        return ans; // Return the constructed path
    }
};
```

Typescript Solution

```typescript
function kthSmallestPath(destination: number[], k: number): string {
    let verticalMoves = destination[0]; // Vertical moves required
    let horizontalMoves = destination[1]; // Horizontal moves required
    let totalMoves = verticalMoves + horizontalMoves; // Total moves required

    // Initialize combinatorial lookup table to store the binomial coefficients
    const comb: number[][] = Array(totalMoves + 1).fill(0).map(() => Array(horizontalMoves + 1).fill(0));
    comb[0][0] = 1;

    // Populate the table with binomial coefficients using Pascal's Triangle
    for (let i = 1; i <= totalMoves; ++i) {
        comb[i][0] = 1;
        for (let j = 1; j <= horizontalMoves; ++j) {
            comb[i][j] = comb[i - 1][j] + comb[i - 1][j - 1];
        }
    }

    let path = ""; // String to hold the result

    // Construct the lexicographically k-th smallest path
    for (let i = 0; i < totalMoves; ++i) {
        if (horizontalMoves === 0) { // If no more horizontal moves, then move vertically
            path += 'V';
            continue;
        }

        // Find the number of sequences starting with 'H'
        let countHStart = comb[verticalMoves + horizontalMoves - 1][horizontalMoves - 1];

        // If k is greater than countHStart, then the current move must be 'V'
        if (k > countHStart) {
            path += 'V';
            --verticalMoves; // One less vertical move needed
            k -= countHStart; // Reduce k by the number of sequences counted
        } else {
            // If k is within the range, the current move is 'H'
            path += 'H';
            horizontalMoves--; // One less horizontal move needed
        }
    }

    return path; // Return the constructed path
}

// Test the function with a sample input
let destinationSample = [2, 3];
let kSample = 3;
console.log(kthSmallestPath(destinationSample, kSample)); // Outputs the k-th smallest path in string format
```

Time and Space Complexity

Time Complexity

The main operation of this algorithm is conducted within a loop that iterates `h + v` times, where `h` and `v` are the numbers of horizontal and vertical steps to reach the destination, respectively.

Within this loop, the time-critical step is the computation of combinations using `comb(h + v - 1, h - 1)` to decide whether to add `'H'` or `'V'` to the path. The `comb` function's complexity can vary depending on the implementation, but generally, it is computed using factorials which can be iterative. However, since Python's `comb` function uses an efficient algorithm presumably based on Pascal's triangle or multiplicative formula, we can assume it runs in O(r) or O(n-r) time complexity, where `n` is the first argument and `r` is the second argument to `comb`.

In the worst-case scenario, we are looking at:

- A time complexity of O(h * min(h, v)) for computing the combination when `v >= h`, because `comb(h + v - 1, h - 1)` simplifies to `comb(h + v - 1, v)` when `v < h`. Since the loop runs `h + v` times and the most time-consuming operation is O(min(h, v)), the overall time complexity would be O(h + v) * min(h, v)).

Space Complexity

The space complexity is determined by the storage used for the `ans` list. Since the `ans` list will contain at most `(h + v)` characters, the space complexity is O(h + v).