**Report**

**Summary:**

The "RNAFolding" Java program implements an algorithm for finding the optimal secondary structure of RNA sequences using dynamic programming. It employs a 3D array to store intermediate results and a list to collect the base pairs in the optimal structure. The `foldRNA` function takes an RNA sequence, iterates through its elements, and utilizes dynamic programming to calculate the most stable secondary structure, considering energy calculations. The `main` method reads RNA sequences from an input file, determines their optimal structures, and prints the results, including the name, length, and base pairings for each sequence. In summary, the code is a tool for RNA structure prediction, aiding in understanding the structural properties of RNA molecules by determining their optimal secondary structures.

**Data structures used:**

1. **dpArray:**

- `dpArray` is a 3D array used to store intermediate results. It tracks information about the optimal structure for different segments of the RNA sequence. The first dimension represents the starting index, the second dimension represents the ending index, and the third dimension holds two values, including the calculated value and a reference to the previous iteration point. These values help the program determine the optimal structure.

2. **basePairs:**

- `basePairs` is a list of lists used to store the base pairs found in the optimal secondary structure. Each inner list represents a pair of indices in the RNA sequence, indicating which nucleotides form a base pair.

**Summary of methods:**

**1. createBasePairs(i, j):**

- This function aims to create a list of base pairs based on the dynamic programming array.

- It starts by checking if both `i` and `j` are non-negative.

- If a value `t` is found in the dynamic programming array, it means a base pair exists. In this case, two scenarios are considered:

- If `t` equals `j-1`, it means the pair involves the last nucleotide. So, we move on to the previous position `j-1`.

- Otherwise, we identify a pair at positions `t` and `j`. We add this pair to the list and recursively explore the regions to the left and right of these positions.

**2. foldRNA(RNA):**

- This function calculates the optimal secondary structure for a given RNA sequence.

- It starts by determining the length of the RNA sequence and initializing two data structures:

- `dpArray`, a 3D array to store intermediate results and references to previous iterations.

- `basePairs`, an empty list to store the found base pairs in the optimal structure.

- The algorithm sets initial values in `dpArray` for small substrings.

- Then, it iterates through the sequence to find the optimal secondary structure by considering various possibilities for base pair formation and evaluating their energies.

- The optimal structure is determined based on energy calculations and stored in `dpArray`.

**3. Main Function:**

- The `main` function is the entry point of the program and handles reading RNA sequences from an input file.

- It initializes the RNA name and reads lines from the file.

- When it encounters lines starting with "," it extracts the RNA name.

- For non-empty lines that don't start with "," it prints the RNA name, length, and calculates and prints the optimal secondary structure.

- The optimal structures are sorted and displayed, including the nucleotides involved in base pairing and their positions.

- Finally, the total number of base pairs is printed.

**Time and space complexity analysis:**

**Time Complexity analysis:**

- **Dynamic Programming:** The core of the algorithm is dynamic programming. It uses nested loops to iterate over substrings of the RNA sequence. The outer loop goes from 5 to `length`, and the inner loop iterates from 0 to `length - k`, where `k` is the length of the substring. The most time-consuming part is the innermost `for t` loop, which can iterate for up to `length` times in the worst case. Inside this loop, operations like string concatenation and comparisons are typically constant time. This part contributes significantly to the time complexity.- O(n^2)

- **Recursive Function:** The `createBasePairs` function can be called recursively, but the number of recursive calls is limited by the length of the RNA sequence. Each recursive call involves updating the `basePairs` list.

- **Sorting:** After the dynamic programming phase, there's a sorting step using the `Collections.sort` function. This sorting operation has a time complexity of O(n log n), where 'n' is the number of base pairs.

- **Printing Results:** The final step involves printing the base pairs, which takes time proportional to the number of base pairs, or O(n). Printing the count of base pairs is a constant time operation, O(1).

Overall, the time complexity is dominated by dynamic programming and the sorting of base pairs. In big O notation, it can be expressed as O(length^2) for dynamic programming, O(n log n) for sorting, and O(length) for iterating over substrings.

**Space Complexity:**

**- `dpArray`:** The primary contributor to space complexity is the `dpArray`, which is a 3D array of dimensions `[length][length][2]`. It stores intermediate results for each substring, resulting in a space complexity of O(length^2).

**- `basePairs`:** The `basePairs` list stores the base pairs, and in the worst case, it can contain as many pairs as there are possible base pairs in the RNA sequence. Hence, its space complexity is also O(length^2).

**- Recursion:** The recursion in the `createBasePairs` function uses the call stack. The space complexity depends on the maximum recursion depth, which can be O(length) in the worst case.

In summary, the space complexity is primarily determined by `dpArray` and `basePairs`, resulting in a space complexity of O(length^2).