Scaler Companion beta

Overview

In this session, we discussed advanced techniques for handling array-related problems, focusing primarily on **Prefix Sum** and **Carry Forward** approaches. These techniques facilitate efficient calculations over an array, such as finding cumulative sums over specific ranges and handling multiple queries efficiently.

Key Concepts

Prefix Sum Technique

Definition: The Prefix Sum technique is used to compute cumulative sums of a sequence of numbers such that each element at index i in the prefix sum array contains the sum of elements from the beginning of the array to i.

Application

- **Range Sum Queries**: It allows rapid calculations of the sum of any subarray.
- **Example**: Given an array $A[]$, the Prefix Sum Array $P[]$ will have $P[i] = A[0] + A[1] + ... + A[i]$.

Pseudocode for Prefix Sum

```
Function createPrefixSumArray(Array[], size){
    PrefixSum[0] = Array[0]
    for(i = 1; i < size; i++){
        PrefixSum[i] = PrefixSum[i-1] + Array[i];
    }
}
```

- **Time Complexity**: $O(N)$
- **Space Complexity**: $O(N)$

Carry Forward Technique

array or a sequence. This approach is used to optimize time by avoiding redundant calculations.

Analogy

Think of Carry Forward as maintaining a “ledger” where the running totals or counts update dynamically as you progress through the transaction list.

Detailed Examples from Class

Problem 1: Range Sum Query

- **Problem**: For given N elements and multiple Q queries, calculate the sum of elements between indices L and R.
- **Brute Force Solution**: Iterate through the range for every query and calculate the sum.
- **Optimized Approach using Prefix Sum**: Precompute Prefix Sum so that every sum query is equivalent to subtracting two prefix sums.

```
Function querySum(Queries[][], Array[], querySize, size){
     PrefixSumArray = createPrefixSumArray(Array, size)
     for(i = 0 to Queries.length-1){
        L  = Queries[i][0]
        R = Queries[i][1]

        if (L == 0)
            sum = PrefixSumArray[R]
        else
            sum = PrefixSumArray[R] - PrefixSumArray[L-1]

        print(sum)
     }
}
```

- **Time Complexity**: $O(1)$ per query after $O(N)$ precomputation.
- **Space Complexity**: $O(N)$

Problem 2: Count AG Pairs

- **Brute Force Solution**: Two nested loops; outer for “a” positions, inner for “g” positions after “a”.
- **Optimized Approach**: Use Carry Forward to maintain a running count of 'g's as you encounter 'a'.

Explanation

- Traverse the string once and keep a running count of 'g's to the right of any 'a' encountered.

```
For string "abegag", pairs: [0,3], [0,5], [4,5].
```

Miscellaneous Analogies and Insights

- **Cricket Score Analogy**: Understanding the prefix concept through a cricket match score can clarify how cumulative values work.
- **Simplification of Complex Queries**: Prefix Sum reformulates the problem to a simple subtraction which effectively reduces complexity.

Conclusion

Both **Prefix Sum** and **Carry Forward** are powerful tools for boosting efficiency in array-based problem solving, particularly with repetitive operations and numerous queries【6:18†source】【6:6†source】. These techniques demonstrate the value of precomputation and smart iteration in optimizing algorithms.

For further clarification on any doubts, ensure to revisit specific segments in the class where detailed pseudo code and problem-solving strategies were discussed【6:12†source】【6:16†source】.