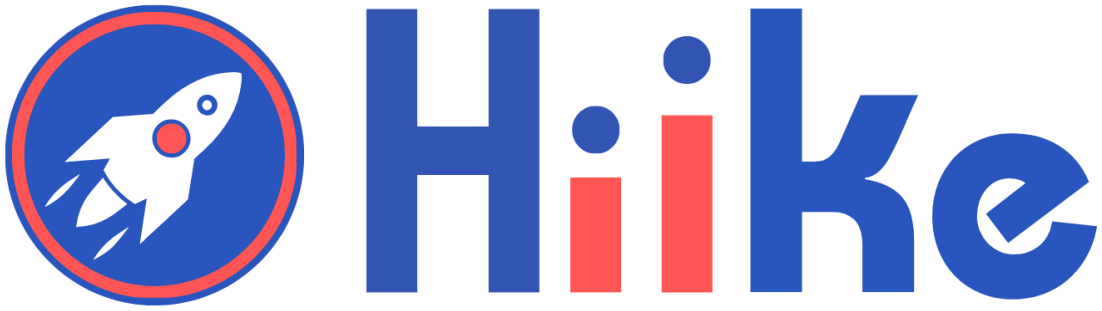
Z Algorithm

(Detailed Notes)



Z Algorithm Overview

The Z algorithm is an efficient linear time string matching algorithm which computes Z-values for a string. A Z-value for a position in the string represents the length of the longest substring starting from that position which is also a prefix of the string. This array is particularly useful for various string processing applications.

How Z Algorithm Works

Constructing the Z-array

- Z[0]: By definition, Z[0] is the length of the entire string.
- For other indices: For index i, if i is outside the current Z-box, start a new Z-box with i as the left boundary and find the right boundary by comparing the substring starting at i with the prefix of the string. Update the Z-box and set Z[i] to the length of the match.
- Inside the Z-box: If i is within the current Z-box bounded by L and R, then Z[i] can be at least min(R-i+1, Z[i-L]). Check further characters to possibly extend R, and adjust Z[i] accordingly.

Algorithm:

```
function Z-Algorithm(S):
    n = length(S)
    Z = array of size n, initialized to 0
    L, R, K = 0, 0, 0

    for i from 1 to n-1:
        if i > R:
            # Case 1: i is outside the current [L, R] segment
            L = R = i
            while R < n and S[R] == S[R - L]:
                R = R + 1
            Z[i] = R - L
            R = R - 1
        else:
            # Case 2: i is within the current [L, R] segment
```

```
            K = i - L
            if Z[K] < R - i + 1:
                Z[i] = Z[K]
            else:
                L = i
                while R < n and S[R] == S[R - L]:
                    R = R + 1
                Z[i] = R - L
                R = R - 1

    return Z
```

Example Problems Solved by Z Algorithm

Example 1: Shortest Palindrome

Problem: Given a string s, you may convert it to a palindrome by adding characters in front of it. Find and return the shortest palindrome you can find by performing this transformation.

Solution Using Z Algorithm:

You can solve this problem by concatenating the string s with a separator and its reverse, then computing the Z-array for this combined string. The logic here is that the longest palindrome prefix of s will correspond to the longest prefix of the reverse of s that matches a prefix of s.

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

using namespace std;

string shortestPalindrome(string s) {
    string reversed_s = s;
    reverse(reversed_s.begin(), reversed_s.end());
    string combined = s + "#" + reversed_s;
    vector<int> Z = computeZ(combined);
    int maxLen = 0;
    for (int i = s.size() + 1; i < Z.size(); i++) {
        if (Z[i] == s.size() - (i - s.size() - 1)) {
            maxLen = max(maxLen, Z[i]);
        }
    }
    return reversed_s.substr(0, s.size() - maxLen) + s;
}

int main() {
    string s = "aacecaaa";
    cout << "Shortest palindrome: " << shortestPalindrome(s) << endl;
    return 0;}
```

Example 2: Sum of Scores of Built Strings

Problem: You are given a string s and need to return the sum of scores for each possible built string from s. Each built string's score is the length of the longest prefix of s which is also a suffix for that built string.

Solution Using Z Algorithm:

Compute the Z-array for the string s, then iterate through the Z-array to compute the sum of scores, where each score is derived from the Z values which describe matching prefix and suffix lengths.

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

using namespace std;

long long sumOfScores(string s) {
    vector<int> Z = computeZ(s);
    long long total = 0;
    for (int i = 0; i < Z.size(); i++) {
        total += Z[i];
    }
    return total;
}

int main() {
    string s = "ababaa";
    cout << "Sum of scores: " << sumOfScores(s) << endl;
    return 0;
}
```

Discussion on Z Algorithm's Utility

The Z algorithm's linear time complexity for building the Z-array makes it highly efficient for problems involving string searches, pattern matching, and analysis involving prefixes and suffixes