Advanced String Algorithms in DSA: KMP, Z-Algorithm & Rabin-Karp

Subtitle: Learn efficient string matching techniques for pattern searching and substring problems.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Advanced string algorithms like **KMP, Z-Algorithm, and Rabin-Karp** allow **fast pattern matching** in strings, reducing naive O(n*m) searches to O(n + m) or expected O(n) time. These are essential for **competitive programming, text search, and DNA sequence analysis**.

Step 1: Knuth-Morris-Pratt (KMP) Algorithm

- Finds occurrences of **pattern P in text T** efficiently
- Uses **Longest Prefix Suffix (LPS) array** to avoid rechecking characters
- **Time Complexity:** O(n + m)

```
function buildLPS(pat){
 let m = pat.length, lps = Array(m).fill(0), len=0, i=1;
 while(i<m){
  if(pat[i]===pat[len]){
   len++; lps[i]=len; i++;
  } else {
   if(len!==0) len = lps[len-1];
   else { lps[i]=0; i++; }
  }
 }
 return lps;
}


function KMP(text, pat){
```

```
  let n=text.length, m=pat.length, lps=buildLPS(pat);
  let i=0,j=0, res=[];
  while(i<n){
    if(text[i]===pat[j]){ i++; j++; }
    if(j===m){ res.push(i-j); j=lps[j-1]; }
    else if(i<n && text[i]!==pat[j]){
      if(j!==0) j=lps[j-1];
      else i++;
    }
  }
  return res;
}

// Example usage
let text="ababcababc", pat="ab";
console.log(KMP(text, pat)); // [0,2,5,7]
```

Step 2: Z-Algorithm

- Computes **Z-array** where Z[i] = length of longest substring starting at i that is also a prefix
- **Used for pattern matching and substring queries**
- **Time Complexity:** O(n)

```
function buildZ(s){
  let n=s.length, Z=Array(n).fill(0), l=0, r=0;
  for(let i=1;i<n;i++){
    if(i<=r) Z[i] = Math.min(r-i+1, Z[i-l]);
    while(i+Z[i]<n && s[Z[i]]===s[i+Z[i]]) Z[i]++;
    if(i+Z[i]-1>r){ l=i; r=i+Z[i]-1; }
  }
  return Z;
```

```
}


// Example usage
let pattern="ab";
let combined = pattern+"$"+"ababcab";
let Z = buildZ(combined);
console.log(Z);
```

- Occurrences of pattern correspond to **Z[i] === pattern.length**

Step 3: Rabin-Karp Algorithm

- Uses **rolling hash** to find patterns efficiently
- Compare hash of pattern with **hash of text substrings**
- Expected **O(n + m)** for single pattern, O(n*m) worst-case with collisions

```
function rabinKarp(text, pat){
 let n=text.length, m=pat.length, base=256, mod=101;
 let pHash=0, tHash=0, h=1, res=[];

 for(let i=0;i<m-1;i++) h=(h*base)%mod;
 for(let i=0;i<m;i++){
  pHash = (base*pHash + pat.charCodeAt(i)) % mod;
  tHash = (base*tHash + text.charCodeAt(i)) % mod;
 }

 for(let i=0;i<=n-m;i++){
  if(pHash===tHash){
   if(text.slice(i,i+m)===pat) res.push(i);
  }
  if(i<n-m){
   tHash = (base*(tHash - text.charCodeAt(i)*h) + text.charCodeAt(i+m)) % mod;
```

```
      if(tHash<0) tHash += mod;
    }
  }
  return res;
}

// Example usage
console.log(rabinKarp("ababcababc","ab")); // [0,2,5,7]
```

Step 4: Applications

1. **Pattern matching in text editors**
2. **Substring search in DNA sequences**
3. **Detect plagiarism using Rabin-Karp**
4. **Solve competitive programming string problems efficiently**

Step 5: Mini Exercises

1. Use **KMP** to find multiple pattern occurrences
2. Use **Z-Algorithm** to compute **prefix matches for a string**
3. Implement **Rabin-Karp** for multiple pattern detection
4. Solve **longest repeating substring** problem

Step 6: Common Mistakes

- Forgetting **0-based indexing** in arrays
- Miscalculating **LPS or Z values** → wrong matches
- Ignoring **hash collisions** in Rabin-Karp
- Using naive string comparison when rolling hash is required

Step 7: Practice Plan for Advanced String Algorithms

Day 1:

- KMP and LPS array → 3–5 problems

Day 2:

- Z-Algorithm pattern queries → 3–5 problems

Day 3:

- Rabin-Karp and mixed string problems → 3–5 problems

Key Takeaways

- KMP, Z-Algorithm, and Rabin-Karp provide **efficient string matching**
- Essential for **text search, substring analysis, and competitive programming**
- Practice **construction, pattern search, and hashing techniques**
- Builds mastery over **advanced string algorithms**

Visit haas.dev for more guides, problem sets, and interview preparation resources on advanced string algorithms.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app