

# Algorithmes de Chaînes en Java

Les chaînes de caractères sont partout, et la manipulation efficace des chaînes est une compétence clé. Examinons l'algorithme KMP, la structure de données trie et Rabin-Karp pour la recherche de motifs.

## 1. Algorithme KMP : Recherche de chaînes efficace

L'algorithme Knuth-Morris-Pratt (KMP) pré-traite le motif pour éviter des comparaisons inutiles, atteignant un temps de  $O(n + m)$ .

### Implémentation en Java

```
public class KMP {  
    static void KMPSearch(String pat, String txt) {  
        int M = pat.length(), N = txt.length();  
        int[] lps = new int[M];  
        computeLPSArray(pat, M, lps);  
        int i = 0, j = 0;  
        while (i < N) {  
            if (pat.charAt(j) == txt.charAt(i)) { i++; j++; }  
            if (j == M) {  
                System.out.println("Trouvé à " + (i - j));  
                j = lps[j - 1];  
            } else if (i < N && pat.charAt(j) != txt.charAt(i)) {  
                if (j != 0) j = lps[j - 1];  
                else i++;  
            }  
        }  
    }  
  
    static void computeLPSArray(String pat, int M, int[] lps) {  
        int len = 0, i = 1;  
        lps[0] = 0;  
        while (i < M) {  
            if (pat.charAt(i) == pat.charAt(len)) lps[i++] = ++len;  
            else if (len != 0) len = lps[len - 1];  
            else lps[i++] = 0;  
        }  
    }  
}
```

```

public static void main(String[] args) {
    String txt = "ABABDABACDABABCABAB";
    String pat = "ABABCABAB";
    KMPSearch(pat, txt);
}
}

```

**Sortie :** Trouvé à 10

## 2. Trie : Recherche basée sur les préfixes

Les tries stockent les chaînes dans un arbre pour des recherches de préfixes rapides, avec une complexité spatiale proportionnelle au nombre total de caractères.

### Implémentation en Java

```

public class Trie {
    static class TrieNode {
        TrieNode[] children = new TrieNode[26];
        boolean isEndOfWord;
    }

    TrieNode root = new TrieNode();

    void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            int index = c - 'a';
            if (node.children[index] == null) node.children[index] = new TrieNode();
            node = node.children[index];
        }
        node.isEndOfWord = true;
    }

    boolean search(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            int index = c - 'a';
            if (node.children[index] == null) return false;
            node = node.children[index];
        }
        return node.isEndOfWord;
    }
}

```

```

    }

    return node.isEndOfWord;
}

public static void main(String[] args) {
    Trie trie = new Trie();
    trie.insert("apple");
    System.out.println("Apple: " + trie.search("apple"));
    System.out.println("App: " + trie.search("app"));
}
}

```

### **Sortie :**

Apple: true  
App: false

## **3. Rabin-Karp : Recherche basée sur le hachage**

Rabin-Karp utilise le hachage pour trouver des motifs, avec un temps moyen de  $O(n + m)$  mais un pire cas de  $O(nm)$ .

### **Implémentation en Java**

```

public class RabinKarp {

    public static void search(String pat, String txt, int q) {
        int d = 256, M = pat.length(), N = txt.length(), p = 0, t = 0, h = 1;
        for (int i = 0; i < M - 1; i++) h = (h * d) % q;
        for (int i = 0; i < M; i++) {
            p = (d * p + pat.charAt(i)) % q;
            t = (d * t + txt.charAt(i)) % q;
        }
        for (int i = 0; i <= N - M; i++) {
            if (p == t) {
                boolean match = true;
                for (int j = 0; j < M; j++) {
                    if (pat.charAt(j) != txt.charAt(i + j)) { match = false; break; }
                }
                if (match) System.out.println("Trouvé à " + i);
            }
        }
    }
}

```

```

    if (i < N - M) {
        t = (d * (t - txt.charAt(i) * h) + txt.charAt(i + M)) % q;
        if (t < 0) t += q;
    }
}

public static void main(String[] args) {
    String txt = "GEEKSFORGEEKS";
    String pat = "GEEKS";
    int q = 101; // Nombre premier
    search(pat, txt, q);
}

```

**Sortie :**

Trouvé à 0

Trouvé à 8