Função de Hash e Rolling Hash

- A função de hash transforma uma string em um número. O Rolling Hash permite atualizar o hash de uma substring de forma eficiente ao deslizar pela string. Isso é feito removendo o caractere à esquerda e adicionando o caractere à direita.

Matemática do Cálculo do Hash

- O hash de uma string s de comprimento m é calculado como:
- hash(s) = (s[0] * p^(m-1) + s[1] * p^(m-2) + ... + s[m-1] * p^0) % q
- Onde p é uma base (geralmente um número primo) e q é um grande número primo para evitar colisões.

txt = "birthboy"

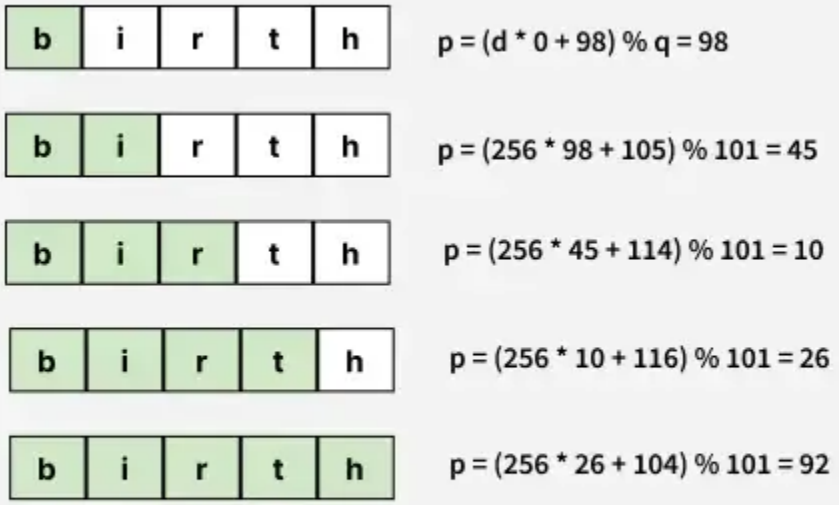
pat = "birth"

d = 256 ASCII base

q = 101 prime modulus

t = 0 hash value of txt

p = 0 hash value of pat



calculation of the hash value of the pat.

txt = "birthboy"

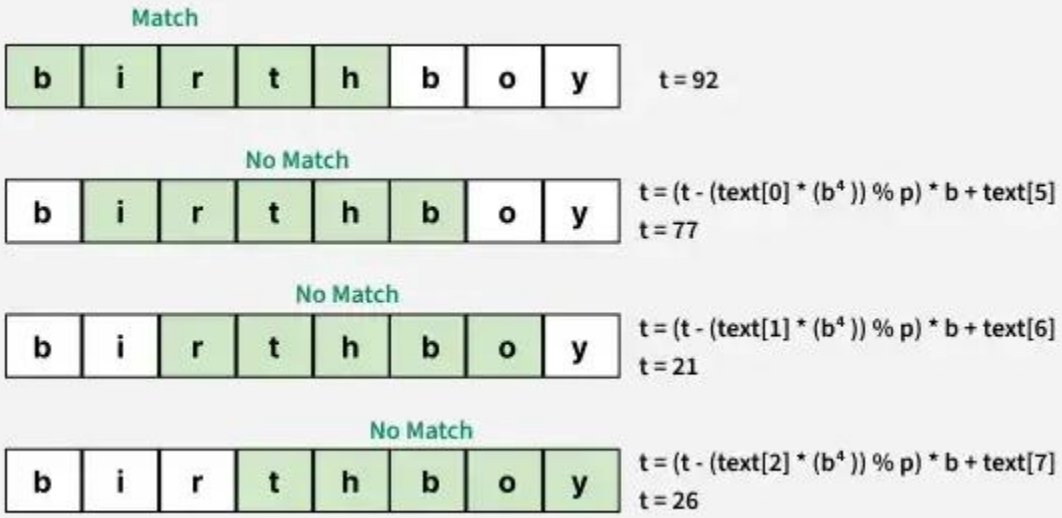
pat = "birth"

d = 256 ASCII base

q = 101 prime modulus

t = 0 hash value of txt

p = 0 hash value of pat



```java
import java.util.ArrayList;

public class RabinKarp {

    // Function to search for all occurrences of 'pat' in 'txt' using Rabin-Karp
    public static ArrayList<Integer> search(String pat, String txt) {

        // Number of characters in the input alphabet (ASCII)
        int d = 256;
        // A prime number for modulo operations to reduce collisions
        int q = 101;
        // Length of the pattern
        int M = pat.length();
        // Length of the text
        int N = txt.length();
        // Hash value for pattern
        int p = 0;
        // Hash value for current window of text
        int t = 0;
        // High-order digit multiplier
        int h = 1;
        // ArrayList to store result indices
        ArrayList<Integer> ans = new ArrayList<>();

        // Precompute h = pow(d, M-1) % q
        for (int i = 0; i < M - 1; i++)
            h = (h * d) % q;

        // Compute initial hash values for pattern and first window of text
        for (int i = 0; i < M; i++) {
            p = (d * p + pat.charAt(i)) % q;
            t = (d * t + txt.charAt(i)) % q;
        }

        // Slide the pattern over text one by one
        for (int i = 0; i <= N - M; i++) {

            // If hash values match, check characters one by one
            if (p == t) {
                boolean match = true;
                for (int j = 0; j < M; j++) {
                    if (txt.charAt(i + j) != pat.charAt(j)) {
                        match = false;
                        break;
                    }
                }
                if (match)
                    ans.add(i + 1); // 1-based indexing
            }

            // Calculate hash value for the next window
            if (i < N - M) {
                t = (d * (t - txt.charAt(i) * h) + txt.charAt(i + M)) % q;

                // Ensure hash value is non-negative
                if (t < 0)
                    t += q;
            }
        }

        return ans;
    }

    public static void main(String[] args) {

        // text and pattern
        String txt = "birthboy";
        String pat = "birth";

        // Search pattern in text
        ArrayList<Integer> res = search(pat, txt);

        // Print result
        for (int index : res)
            System.out.print(index + " ");
        System.out.println();
    }
}
```

Vantagens, Desvantagens e Complexidade

Vantagens:
- Simples e eficiente para padrões curtos.
- Utiliza hashing para acelerar a busca.

Desvantagens:
- Pode ter colisões de hash.
- Menos eficiente para padrões longos.

Complexidade:
- Melhor caso: O(n + m)
- Pior caso: O(nm) devido a colisões de hash.