Rolling Hashing

Univ: Grover Osvaldo Rodriguez Apaza
Universidad Mayor de San Andrés
Informática

3 de diciembre de 2021

---

1. Descripción

llamado *polynomial rolling hashing function*.
Vamos a definir el hash de una cadena de la siguiente manera:

$$\begin{aligned}\text{hash}(s) &= s[0] \cdot B^0 + s[1] \cdot B^1 + s[2] \cdot B^2 + \cdots + s[n-1] \cdot B^{n-1} \quad \text{mód } m \\ &= \sum_{i=0}^{n-1} s[i] \cdot B^i \quad \text{mód } m\end{aligned}$$

Donde:

s : es una cadena de tamaño n
B : es la base
mod : módulo del hashing

Es mejor utilizar primos para reducir la probabilidad de colision.
la base tiene que ser mas grande que todo el albafeto.

Ejemplo:

$B = 7, a = 1, b = 2, \cdots, z = 26.$

El hashing de $ab = o$

$$a \cdot 7^0 + b \cdot 7^1 = o \cdot 7^0$$
$$15 = 15$$

Existe colisión, es decir para dos cadenas diferentes da el mismo número.

Algunos números primos para la base y modulo.

$$33, 31, 331, 997$$
$$1001265673, 1001864327, 999727999, 1070777777$$

2. Explicación del algoritmo

Para fines prácticos el $hash(s)$, se calculara de la siguiente manera.

Sea:

$$s = "abcab"$$
$$B = 331$$
$$MOD = 10^9 + 7$$
$$a = 97, b = 98, c = 99, \cdots, z = 122.$$

$$s[0] = 97 + 331^0 = 97$$
$$s[1] = 97 \cdot 331^1 + 98 \cdot 331^0 = 32205$$
$$s[2] = 97 \cdot 331^2 + 98 \cdot 331^1 + 99 \cdot 331^0 = 10659954$$
$$s[3] = 97 \cdot 331^3 + 98 \cdot 331^2 + 99 \cdot 331^1 + 97 \cdot 331^0 = 528444850$$
$$s[4] = 97 \cdot 331^4 + 98 \cdot 331^3 + 99 \cdot 331^2 + 97 \cdot 331^1 + 98 \cdot 331^0 = 915244230$$

Donde: $s[4]$ seria el $hash(s)$. todas las operaciónes estan moduladas.
luego se vera porque se implemento de esa manera.

3. Codificación

```cpp
#include <bits/stdc++.h>
typedef long long ll;
ll hash(const vector<int> &v, ll B, ll mod){
    ll hash = 0;
    for(int i = 0; i < v.size(); i++)
```

```
6           hash = (hash * B + v[i]) % mod;
7       return hash;
8   }
```

4. Calcular el hashing de un substring

Problema: Dado un string s y el rango $[L, R]$, calcular el hashing del substring $s[L \cdots R]$.

Para calcular eficientemente el $Hash(s)$, podemos acumular en un vector, luego con operaciones matemáticas se hallara el $hash(L \cdots R)$.

Ahora si aplicaremos de la manera que se programó $hash(s)$.

Como calculamos para cada caracter de la cadena, ahora lo almacenaremos en un vector, de la siguiente manera. Sea $h1$ el vector el cual almacenará el $hash(s)$, en cada posición

$$\begin{aligned} h1[0] &= hash(s[0]) \\ h1[1] &= hash(s[0 \cdots 1]) \\ h1[2] &= hash(s[0 \cdots 2]) \\ &\vdots \\ h1[n-1] &= hash(s[0 \cdots n-1]) \end{aligned}$$

tenga en cuenta la forma que se programo $hash(s)$, servira para hacer consultas sobre substring, de la manera que esta programado, no se necesitara implementar el inverso modular.

A continuación se mostrara las potencias de las bases en cada posicion del vector.

$$\begin{aligned} s &= "abcab" \\ pos &= \ 01234 \end{aligned}$$

$$\begin{aligned} h1[0] &= s_0 \cdot B^0 \\ h1[1] &= s_0 \cdot B^1 + s_1 \cdot B^0 \\ h1[2] &= s_0 \cdot B^2 + s_1 \cdot B^1 + s_2 \cdot B^0 \\ h1[3] &= s_0 \cdot B^3 + s_1 \cdot B^2 + s_2 \cdot B^1 + s_3 \cdot B^0 \\ h1[4] &= s_0 \cdot B^4 + s_1 \cdot B^3 + s_2 \cdot B^2 + s_3 \cdot B^1 + s_4 \cdot B^0 \end{aligned}$$

si queremos obtener el $hash(s)$ del subtring $[3, 4]$

de la forma que se construyo el $hash(s)$, para hallar el $hash[3, 4]$ debe ser:

$$hash[3, 4] = s_3 \cdot B^1 + s_4 \cdot B^0 \tag{1}$$

se puede hacer de la siguiente manera:

como $R = 4$ se usara: $h1[4] = s_0 \cdot B^4 + s_1 \cdot B^3 + s_2 \cdot B^2 + s_3 \cdot B^1 + s_4 \cdot B^0$

como es inclusivo el rango entonces $L = 2$. se necesita

$h1[2] = s_0 \cdot B^2 + s_1 \cdot B^1 + s_2 \cdot B^0$

hay que encontrar una forma de combinar $h1[4]$ y $h1[2]$, para hallar lo esperado.

$hash[3,4] = h1[4] - h1[2] \cdot B[4-3+1]$

$hash[3,4] = h1[4] - h1[2] \cdot B[2]$

remplzando datos

$$hash[3,4] = (\underbrace{s_0 \cdot B^4 + s_1 \cdot B^3 + s_2 \cdot B^2 + s_3 \cdot B^1 + s_4 \cdot B^0}_{h1[4]}) - (\underbrace{s_0 \cdot B^2 + s_1 \cdot B^1 + s_2 \cdot B^0}_{h1[2]}) \cdot \underbrace{B^2}_{B[2]}$$

repartiendo B^2

$$hash[3,4] = \underbrace{s_0 \cdot B^4 + s_1 \cdot B^3 + s_2 \cdot B^2}_{iguales} + s_3 \cdot B^1 + s_4 \cdot B^0 - \underbrace{(s_0 \cdot B^4 + s_1 \cdot B^3 + s_2 \cdot B^2)}_{iguales}$$

cancelando términos opuestos

$hash[3,4] = s_3 \cdot B^1 + s_4 \cdot B^0$

reemplazando datos

$hash[3,4] = 97 \cdot 331^1 + 98 \cdot 331$

calculando valor

$hash[3,4] = 32205$

Finalemente de logro obtener el mismo resultado de la $ec.(1)$, lo cual nos permite hallar $hash[3,4]$, sin usar inverso modular, solo necesitamos preprocesar todas las potencias de B.

por lo tanto como $s[0,1] = ab \;\; y \;\; s[3,4] = ab$. tambien $hash[0,1] = hash[3,4]$ se logro el resultado esperado.

De manera general se tiene:

$$hash[L,R] = \begin{cases} h1[R] & L = 0 \\ h1[R] - h1[L-1] \cdot B[R-L+1] & L > 0 \end{cases}$$

Nota:

También se puede calcular al revez, es decir desde $[str.size() - 1, 0]$.

5. Código

Implementado con Struct en c++, y con dos hashing uno de $h1[0, n-1]$ y $h2[n-1, 0]$, $BASE[i]$ tiene todas las potencias de la base.

```
1
2  #include<bits/stdc++.h>
3  using namespace std;
4  typedef long long ll;
5  struct Hash{
6      //primos 1001265673, 1001864327, 999727999, 1070777777
7      // 33, 331, 997
8      ll B, MOD;
9      int N_MAX;
10     vector<ll> h1, h2, BASE;
11     Hash(int N, ll Base, ll mod){
12         B = Base;
13         MOD = mod;
14         N_MAX = N;
15         h1.resize(N_MAX);
16         h2.resize(N_MAX);
17         BASE.resize(N_MAX);
18         BASE[0] = 1;
19         for(int i = 1; i < N_MAX; i++)
20             BASE[i] = mul(BASE[i - 1], B);
21     }
22
23     void build(const vector<int> &v){
24         ll hash = 0;
25         for(int i = 0; i < N_MAX; i++){
26             hash = add(mul(hash, B), v[i]);
27             h1[i] = hash;
28         }
29         hash = 0;
30         for(int i = N_MAX - 1; i >= 0; i--){
31             hash = add(mul(hash, B), v[i]);
32             h2[i] = hash;
33         }
34     }
35     ll getLR(int l, int r){
36         if(l == 0)
37             return h1[r];
38         return sub(h1[r], mul(h1[l - 1], BASE[r - l + 1]));
39     }
40
41     ll getRL(int l, int r){
42         if(r == N_MAX - 1)
43             return h2[l];
44         return sub(h2[l], mul(h2[r + 1], BASE[r - l + 1]));
45     }
```

```
46
47      ll add(ll a, ll b){return a + b > MOD ? a + b - MOD : a + b;}
48      ll sub(ll a, ll b){return a - b < 0 ? a - b + MOD : a - b;}
49      ll mul(ll a, ll b){return a * b > MOD ? a * b % MOD : a * b;}
50  };
51
52  int main(){
53      string str = "abcab";
54      Hash H(str.size(), 331, 1e9 + 7);
55      vector<int> v;
56      for(auto i : str)
57          v.push_back(i);
58      H.build(v);
59      cout << H.getLR(0, 1) << '\n';
60      return 0;
61  }
```

6. Complejidad

- Calcular el vector $BASE[i]$ toma tiempo y espacio $O(n)$
- Calcular $h1[i]$ toma tiempo y espacio $O(n)$

Finalmente su contrucción es:
$O(n)$ en Espacio y Tiempo.