字符串哈希Hash

一.定义

字符串哈希： **把不同的字符串映射成不同的整数**

哈希函数：定义一个把字符串映射到整数的函数f，这个函数称为是 $Hash$ **函数。**

通常我们采用的是多项式 Hash 的方法

对于一个长度为n的字符串s来说，定义如下的 Hash 函数，将字符串映射成一个base进制的数：

$$f(s)=\sum_{i=1}^{n}s[i]*base^{n-i}(mod\ \ M)$$

例如：字符串ABC , 其哈希值为：$A*base^2+B*base^1+C$【类似进制转换】

哈希算法：通过哈希函数将字符串转换为能够用变量表示的数，通过哈希算法转换到的值，称之为哈希值。哈希值可以实现快速查找和匹配，这个值会非常大，所以会使用**取余**的方法去得到具体的值。

二.主要功用

哈希函数可以方便地帮我们判断两个字符串是否相等。

补充解释：

通过哈希函数要比直接逐字符比较方法($==$或$strcmp$)效率更高

- **时间复杂度优化**
  - 哈希比较：计算字符串哈希值的时间复杂度为$O(n)$，但比较哈希值仅需$O(1)$。若提前预计算并存储哈希值（如前缀哈希），比较子串时复杂度可降至$O(1)$。
  - 直接比较：逐字符比较的时间复杂度为$O(n)$，n为字符串长度。
- **哈希值的特性**
  - 哈希值固定长度，无论原字符串多长，比较哈希值均为整数比对，效率远高于变长字符串的逐字节比较。
  - 哈希冲突概率低时，不同字符串的哈希值几乎不会相同，可快速排除不匹配的情况

三.性质

具体来说，哈希函数最重要的性质可以概括为下面两条：

性质1：

在 $Hash$ 函数值不一样的时候，两个字符串一定不一样

性质2：

在 $Hash$ 函数值一样的时候，两个字符串不一定一样（但有大概率一样，且我们当然希望它们总是一样的）

我们将 $Hash$ 函数值一样但原字符串不一样的现象称为**哈希碰撞**

解决哈希碰撞的方法：

巧妙设置$base$和M的值，保证$base$和M互质

$base$通常设置质数为131 或 13331 或 798

M通常设置为大整数2^{64}，为了扩大模数，我们在$C++$中我们会使用$unsigned\ \ long\ \ long$来定义Hash函数的结果，一方面无符号长整型不包含负数，而且如果超过会自动溢出，类似于取余，所以**只要开了$unsigned\ \ long\ \ long$我们就不需要去取余了**

四.理论实现

通常我们采用的是多项式 $Hash$ 的方法，将字符串中的每一个字母都看作是一个数字(例:从$a-z$,视为$1-26$);
将字符串视为是一个 $BASE$ 进制的数。

这里的 $Hash$ 函数有以下2种方案：
方案1：

$$f(s)=\sum_{i=1}^{n} s[i]*base^{n-i}$$

字符串ABC , 其哈希值为：$A*base^2+B*base^1+C$

方案2：

$$f(s)=\sum_{i=1}^{n} s[i]*base^{i-1}$$

字符串ABC , 其哈希值为：$A+B*base^1+C*base^2$

显然，上面这两种哈希函数的定义函数都是可行的，但二者在之后会讲到的计算子串哈希值时所用的计算式是不同的，因此千万注意 **不要弄混了这两种不同的 $Hash$ 方式。**

由于前者的 $Hash$ 定义计算更简便、使用人数更多、且可以类比为一个 进制数来帮助理解，所以下面所将要讨论的都是**使用方法1**来定义的 $Hash$ 函数。

五.具体实现

推导一下具体的求Hash值的流程，如下图：

字符串	哈希值
A	A
AB	$A*base+B$
ABC	$A*base^2+B*base+C$
$ABCD$	$A*base^3+B*base^2+C*base+D$
...	......

很清楚的看到：

$A=A$

$AB=A*base+B$

$ABC=AB*base+C$

$ABCD=ABC*base+D$

观察这像之前学过的那个知识点？？？？

前缀和！！！！！

综上所述：
求一个字符串的哈希值相当于求前缀和

推导出公式为：

$$\text{前缀和}\ \ sum[i]\ =\ sum[i-1]*base\ +\ s[i]$$

也可以直接用变量去储存最终的Hash值

$$\text{前缀和}\ \ sum\ =\ sum*base\ +\ s[i]$$

同理可以推导出：

求一个字符串的子串的哈希值相当于求区间和

例如 字符串$ABCDE$ 获得第3项到第4项的子串即 CD

大家可以思考一下：可以直接 $ABCD$ - AB 得到 CD吗？

明显不行：
$ABCD: A*base^3+B*base^2+C*base+D$

$AB: A*base+B$

我们需要让AB的哈希值 * $base^2$，再让$ABCD$的哈希值去做减法

同理如果要得到第2项到第4项的子串，即 BCD 呢?

$ABCD: A*base^3+B*base^2+C*base+D$

$A: A$

我们需要让A的哈希值 * $base^3$，再让$ABCD$的哈希值去做减法

推导出公式如下：

$$\text{区间和}\ \ sum[L-\ R]\ =\ sum[R]\ -\ sum[L-1]*base^{R-L+1}$$

另外，目前$base$的指数次方如果用pow函数会出现精度丢失导致WA，我们需要一个储存指数的数组pb做辅助

```
unsigned long long pb[100005];
// pb[i]即代表第i项的base的指数次方值
pb[0] = 1;
for(int i=1;i<=n;i++){
    pb[i] = pb[i-1] * base;
}
```

六.代码实现

1.只是单纯求字符串的Hash值

```
#define ull unsigned long long
ull base = 131 ;
string s;
inline ull get(string s){
    ull sum = 0;
    for(int i=0;i<s.length();i++){
        sum = sum*131 + s[i];
    }
    return sum
}

```

2.涉及到了子串相关问题

```
#include<bits/stdc++.h>
using namespace std;
#define ull unsigned long long
ull base = 131 ,  sum[10005] , pb[10005];
string s;
signed main() {

    // 预处理
    s = "!" + s;
    pb[0] = 1;
    for(int i=1;i<s.length();i++){
        sum[i] = sum[i-1]*base + s[i];
        pb[i] = pb[i-1] * base;
    }

    // 求子串Hash值 L-R
    ull t = sum[R] - sum[L-1]*pb[R-L+1];


    return 0;
}
```