# **字符串Hash**

据我的理解，Hash就是一个像函数一样的东西，你放进去一个值，它给你输出来一个值。输出的值就是Hash值。一般Hash值会比原来的值更好储存(更小)或比较。

那字符串Hash就非常好理解了。就是把字符串转换成一个整数的函数。而且要尽量做到使字符串对应唯一的Hash值。

字符串Hash的种类还是有很多种的，不过在信息学竞赛中只会用到一种名为“BKDR Hash”的字符串Hash算法。

它的主要思路是选取恰当的进制，可以把字符串中的字符看成一个大数字中的每一位数字，不过比较字符串和比较大数字的复杂度并没有什么区别(高精数的比较也是O(n)的)，但只要把它对一个数取模，然后认为取模后的结果相等原数就相等，那么就可以在一定的错误率的基础上O(1)进行判断了。

那么我们选择什么进制比较好？

首先不要把任意字符对应到数字0，比如假如把a对应到数字0，那么将不能只从Hash结果上区分ab和b（虽然可以额外判断字符串长度，但不把任意字符对应到数字0更加省事且没有任何副作用），一般而言，把a-z对应到数字1-26比较合适。

关于进制的选择实际上非常自由，大于所有字符对应的数字的最大值，不要含有模数的质因子(那还模什么)，比如一个字符集是a到z的题目，选择27、233、**19260817**都是可以的。

模数的选择（尽量还是要选择质数）：

绝大多数情况下，不要选择一个109级别的数，因为这样随机数据都会有Hash冲突，根据生日悖论，随便找上109−−−√109个串就有大概率出现至少一对Hash 值相等的串（参见BZOJ 3098 Hash Killer II）。

最稳妥的办法是选择两个109级别的质数，只有模这两个数都相等才判断相等，但常数略大，代码相对难写，目前暂时没有办法卡掉这种写法（除了卡时间让它超时）（参见BZOJ 3099 Hash Killer III）。mod1一般取1e9+7，mod2一般取1e9+9，1000000007和1000000009是一对孪生素数

如果能背过或在考场上找出一个1018级别的质数(Miller-Rabin)，也相对靠谱，主要用于前一种担心会超时，后一种担心被卡。

偷懒的写法就是直接使用unsigned long long，不手动进行取模，它溢出时会自动对264进行取模，如果出题人比较良心，这种做法也不会被卡，但这个是完全可以卡的，卡的方法参见BZOJ 3097 Hash Killer I。

用luogu P3370为例。

这是自然溢出hash(100)

C++

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef unsigned long long ull;

ull base=131;

ull a[10010];

char s[10010];

int n,ans=1;

ull hashs(char s[]){

int len=strlen(s);

ull ans=0;

for (int i=0;i<len;i++)

ans=ans\*base+(ull)s[i];

return ans&0x7fffffff;

}

main(){

scanf("%d",&n);

for (int i=1;i<=n;i++){

scanf("%s",s);

a[i]=hashs(s);

}

sort(a+1,a+n+1);

for (int i=2;i<=n;i++)

if (a[i]!=a[i-1])

ans++;

printf("%d\n",ans);

}

这是单模数hash(100)

C++

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef unsigned long long ull;

ull base=131;

ull a[10010];

char s[10010];

int n,ans=1;

ull mod=1000000007;//1e9+7,或1e9+9是一对孪生素数

ull hashs(char s[]){

int len=strlen(s);

ull ans=0;

for (int i=0;i<len;i++)

ans=(ans\*base+(ull)s[i])%mod;

return ans;

}

main(){

scanf("%d",&n);

for (int i=1;i<=n;i++){

scanf("%s",s);

a[i]=hashs(s);

}

sort(a+1,a+n+1);

for (int i=2;i<=n;i++)

if (a[i]!=a[i-1])

ans++;

printf("%d\n",ans);

}

这是双hash(100)

C++

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef unsigned long long ull;

ull base=131;

Struct data{

ull x,y;

}a[10010];

char s[10010];

int n,ans=1;

ull mod1=1000000007;

ull mod2=1000000009;

ull hash1(char s[]){

int len=strlen(s);

ull ans=0;

for (int i=0;i<len;i++)

ans=(ans\*base+(ull)s[i])%mod1;

return ans;

}

ull hash2(char s[]){

int len=strlen(s);

ull ans=0;

for (int i=0;i<len;i++)

ans=(ans\*base+(ull)s[i])%mod2;

return ans;

}

bool comp(data a,data b){

return a.x<b.x;

}

main(){

scanf("%d",&n);

for (int i=1;i<=n;i++){

scanf("%s",s);

a[i].x=hash1(s);

a[i].y=hash2(s);

}

sort(a+1,a+n+1,comp);

for (int i=2;i<=n;i++)

if (a[i].x!=a[i-1].x || a[i-1].y!=a[i].y)

ans++;

printf("%d\n",ans);

}

这是只用一个10^18质数的hash(100)

C++

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef unsigned long long ull;

ull base=131;

ull a[10010];

char s[10010];

int n,ans=1;

ull mod=212370440130137957ll;

ull hashs(char s[]){

int len=strlen(s);

ull ans=0;

for (int i=0;i<len;i++)

ans=(ans\*base+(ull)s[i])%mod;

return ans;

}

main(){

scanf("%d",&n);

for (int i=1;i<=n;i++){

scanf("%s",s);

a[i]=hashs(s);

}

sort(a+1,a+n+1);

for (int i=2;i<=n;i++)

if (a[i]!=a[i-1])

ans++;

printf("%d\n",ans);

}

**[字符串匹配(hash算法）](https://www.cnblogs.com/zyf0163/p/4806951.html)**

Hash还有一方面，就是它可以处理子串信息。对于一个字符串，我们可以预处理它1~l的hash值，这样l+1的hash值就可以O(1)的递推出来。

对于一个字符串l~r的子串，我们可以用f[r]−br−l+1\*f[l−1]来求出来，其中b表示进制。

首先我们会想一下二进制数。

对于任意一个二进制数，我们将它化为10进制的数的方法如下（以二进制数1101101为例）：

IMG_256

hash用的也是一样的原理，为每一个前缀（也可以后缀，笔者习惯1 base，所以喜欢用前缀来计算，Hash[i] = Hash[i - 1] \* x + s[i](其中1 < i <= n,Hash[0] = 0)。

 一般地，IMG_257

 而对于l - r区间的hash值，则为：

IMG_258

但是如果n很大呢？那样不是会溢出了吗？

因此我们把hash值储存在unsigned long long里面， 那样溢出时，会自动取余2的64次方，but这样可能会使2个不同串的哈希值相同，但这样的概率极低（不排除你的运气不好）。

因此我们可以通过Hash值来比较两个字符串是否相等。

给出多项式hash的处理：

**typedef** unsigned **long** **long** ull;

**const** **int** N = 100000 + 5;

**const** ull base = 163;

**char** s[N];

ull hash[N];

**void** init(){//处理hash值

    p[0] = 1;

    hash[0] = 0;

**int** n = **strlen**(s + 1);

**for**(**int** i = 1; i <=100000; i ++)p[i] =p[i-1] \* base;

**for**(**int** i = 1; i <= n; i ++)hash[i] = hash[i - 1] \* base + (s[i] - 'a');

}

ull get(**int** l, **int** r, ull g[]){//取出g里l - r里面的字符串的hash值

**return** g[r] - g[l - 1] \* p[r - l + 1];

}

我们来看到题目吧：[传送门](http://acm.hdu.edu.cn/showproblem.php?pid=4300" \t "https://www.cnblogs.com/zyf0163/p/_blank)



题目大意:

是有一份文件，前面是密文，后面是原文，但那个人接到这个文件后不知道中间从哪里开始是原文，所以你要帮忙还原一下，如果后面原文比密文少，你就将它补全, 第一行是密文转换格式，例如第二个样例表示将q翻译成a,w翻译成b。

思路：

我们只要先把密文都翻译成明文，然后去比较原来的字符串的后缀和翻译之后的字符串前缀的最长匹配长度就行（注：最长匹配的长度不能超过原长的一半）

hash水题（附AC代码）：

 #include <cstdio>

#include <cstring>

#include <algorithm>

**using** **namespace** std;

**typedef** unsigned **long** **long** ull;

**const** **int** N = 100000 + 5;

**const** ull base = 163;

ull Hash1[N], Hash2[N], p[N];

**char** s[N], t[30], r[N];

**int** T;

**int** c[30];

**void** init(){

    p[0] = 1;

**for**(**int** i = 1; i <=100000; i ++)p[i] =p[i-1] \* base;

}

ull get(**int** l, **int** r, ull g[]){

**return** g[r] - g[l - 1]\*p[r - l + 1];

}

**void** work(){

**for**(**int** i = 0; i < 26; i ++) c[t[i] - 'a'] = i;

    //puts(r+1);

**int** n = **strlen**(s + 1);

    Hash1[0] = Hash2[0] = 0;

**for**(**int** i = 1; i <= n; i ++){

        Hash1[i] = Hash1[i - 1] \* base + (s[i] - 'a');

        Hash2[i] = Hash2[i - 1] \* base + (c[s[i] - 'a']);

    }

**int** ans = n;

**for**(**int** i = n; i < n \* 2; i ++){

**if**(i & 1) **continue**;

**int** tmp = i / 2;

**int** len =n - tmp;

        ull s1 = get(1, len, Hash2);

        ull s2 = get(n - len + 1, n, Hash1);

**if**(s1 == s2){

            ans = tmp;

**break**;

        }

        //printf("%llu %llu\n", s1, s2);

    }

    //printf("ans = %d\n", ans);

**for**(**int** i = 1; i <= ans; i ++)**printf**("%c", s[i]);

**for**(**int** i = 1; i <= ans; i ++)**printf**("%c", c[s[i]-'a'] + 'a');

**puts**("");

}

**int** main(){

**scanf**("%d", &T);

    init();

**while**(T--){

**scanf**("%s%s", t, s + 1);

        work();

    }

**return** 0;

}

[cogs1570](http://www.cogs.pro/cogs/problem/problem.php?pid=1570" \t "https://www.cnblogs.com/Slager-Z/p/_blank)

【题目描述】

法国作家乔治·佩雷克(Georges Perec，1936-1982)曾经写过一本书，《敏感字母》（La disparition），全篇没有一个字母‘e’。他是乌力波小组（Oulipo Group）的一员。下面是他书中的一段话：

Tout avait Pair normal, mais tout s’affirmait faux. Tout avait Fair normal, d’abord, puis surgissait l’inhumain, l’affolant. Il aurait voulu savoir où s’articulait l’association qui l’unissait au roman : stir son tapis, assaillant à tout instant son imagination, l’intuition d’un tabou, la vision d’un mal obscur, d’un quoi vacant, d’un non-dit : la vision, l’avision d’un oubli commandant tout, où s’abolissait la raison : tout avait l’air normal mais…

佩雷克很可能在下面的比赛中得到高分（当然，也有可能是低分）。在这个比赛中，人们被要求针对一个主题写出甚至是意味深长的文章，并且让一个给定的“单词”出现次数尽量少。我们的任务是给评委会编写一个程序来数单词出现了几次，用以得出参赛者最终的排名。参赛者经常会写一长串废话，例如500000个连续的‘T’。并且他们不用空格。

因此我们想要尽快找到一个单词出现的频数，即一个给定的字符串在文章中出现了几次。更加正式地，给出字母表{'A','B','C',...,'Z'}和两个仅有字母表中字母组成的有限字符串：单词W和文章T，找到W在T中出现的次数。这里“出现”意味着W中所有的连续字符都必须对应T中的连续字符。T中出现的两个W可能会部分重叠。

【输入格式】

输入包含多组数据。

输入文件的第一行有一个整数，代表数据组数。接下来是这些数据，以如下格式给出：

第一行是单词W，一个由{'A','B','C',...,'Z'}中字母组成的字符串，保证1<=|W|<=10000（|W|代表字符串W的长度）

第二行是文章T，一个由{'A','B','C',...,'Z'}中字母组成的字符串，保证|W|<=|T|<=1000000。

【输出格式】

对每组数据输出一行一个整数，即W在T中出现的次数。

【样例输入】

3  
BAPC  
BAPC  
AZA  
AZAZAZA  
VERDI  
AVERDXIVYERDIAN

【样例输出】

1  
3  
0

代码

C++

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef unsigned long long ull;

ull base=131;

ull po[100010],hs[100010\*100];

char s1[100010],s2[100010\*100];

int n,ans=1,T;

ull geth(int l,int r){

return (ull)hs[r]-po[r-l+1]\*hs[l-1];

}

main(){

freopen("oulipo.in","r",stdin);

freopen("oulipo.out","w",stdout);

po[0]=1;

for (int i=1;i<=10010-5;i++)

po[i]=po[i-1]\*base;

scanf("%d",&T);

while(T--){

scanf("%s%s",s1+1,s2+1);

int l1=strlen(s1+1),l2=strlen(s2+1);

ull a1=0,ans=0;

for (int i=1;i<=l1;i++)

a1=a1\*base+(ull)s1[i];

for (int i=1;i<=l2;i++)

hs[i]=hs[i-1]\*base+s2[i];

for (int i=1;i+l1-1<=l2;i++)

if (a1==geth(i,i+l1-1))

ans++;

printf("%d\n",ans);

}

}

写到这里突然发现hash好像可以暴力水过很多字符串算法。。

1、kmp

问题：给两个字符串S1，S2，求S2是否是S1的子串，并求S2在S1中出现的次数

把S2 Hash出来，在S1里找所有长度为|S2|的子串，Hash比较。效率O(|S1|)O(|S1|)

2、AC自动机

问题：给N个单词串，和一个文章串，求每个单词串是否是文章串的子串，并求每个单词在文章中出现的次数。

把每一个单词hash成整数，再把文章的每一个子串hash成整数，接下来只需要进行整数上的查找即可。

复杂度：O(|A|2+|S|)

用AC自动机可以做到O(|A|+|S|)O(|A|+|S|)的复杂度,|S||S|是单词串总长，|A||A|是文章长度

3、后缀数组

问题：给两个字符串S1,S2，求它们的最长公共子串的长度。

将S1的每一个子串都hash成一个整数，将S2的每一个子串都hash成一个整数

两堆整数，相同的配对，并且找到所表示的字符串长度最大的即可。

复杂度：O(|S1|2+|S2|2)

用后缀数组可以优化到O(|S|log|S|)

4、马拉车

问题：给一个字符串S，求S的最长回文子串。

先求子串长度位奇数的，再求偶数的。枚举回文子串的中心位置，然后二分子串的长度，直到找到一个该位置的最长回文子串，不断维护长度最大值即可。

复杂度：O(|S|log|S|)

用manacher可以做到O(|S|)的复杂度

5、扩展kmp

问题：给一个字符串S，求S的每个后缀与S的最长公共前缀

枚举每一个后缀的起始位置，二分长度，求出每个后缀与S的最长公共前缀。

复杂度：O(|S|log|S|)

用extend-kmp可以做到O(|S|)的复杂度

后记

hash真是一种优雅的暴力。

因为字符串特殊的性质，我们可以二分得处理它，一般都有单调性。