各种字符串 Hash 函数比较

常用的字符串 Hash 函数还有 ELFHash，APHash 等等，都是十分简单有效的方法。这些函数使用位运算使得每一个字符都对最后的函数值产生影响。另外还有以 MD5 和 SHA1 为代表的杂凑函数，这些函数几乎不可能找到碰撞。

常用字符串哈希函数有 BKDRHash，APHash，DJBHash，JSHash，RSHash，SDBMHash，PJWHash，ELFHash 等等。对于以上几种哈希函数，我对其进行了一个小小的评测。

Hash 函数	数据 1	数据 2	数据 3	数据 4	数据 1 得分	数据 2 得分	数据 3 得分	数据 4 得分	平均分
BKDRHash	2	0	4774	481	96.55	100	90.95	82.05	92.64
APHash	2	3	4754	493	96.55	88.46	100	51.28	86.28
DJBHash	2	2	4975	474	96.55	92.31	0	100	83.43
JSHash	1	4	4761	506	100	84.62	96.83	17.95	81.94
RSHash	1	0	4861	505	100	100	51.58	20.51	75.96
SDBMHash	3	2	4849	504	93.1	92.31	57.01	23.08	72.41
PJWHash	30	26	4878	513	0	0	43.89	0	21.95
ELFHash	30	26	4878	513	0	0	43.89	0	21.95

其中数据 1 为 100000 个字母和数字组成的随机串哈希冲突个数。数据 2 为 100000 个有意义的英文句子哈希冲突个数。数据 3 为数据 1 的哈希值与 1000003(大素数)求模后存储到线性表中冲突的个数。数据 4 为数据 1 的哈希值与 10000019(更大素数)求模后存储到线性表中冲突的个数。

经过比较，得出以上平均得分。平均数为平方平均数。可以发现，BKDRHash 无论是在实际效果还是编码实现中，效果都是最突出的。APHash 也是较为优秀的算法。DJBHash,JSHash,RSHash 与 SDBMHash 各有千秋。PJWHash 与 ELFHash 效果最差，但得分相似，其算法本质是相似的。

在信息修竞赛中，要本着易于编码调试的原则，个人认为 BKDRHash 是最适合记忆和使用的。

BYVoid 原创，欢迎建议、交流、批评和指正。

附：各种哈希函数的 C 语言程序代码

Hash.c

```
unsigned int SDBMHash(char *str)
{
    unsigned int hash = 0;

    while (*str)
    {
    // equivalent to: hash = 65599*hash + (*str++);
        hash = (*str++) + (hash << 6) + (hash << 16) - hash;
    }
```

```
    return (hash & 0x7FFFFFFF);
}

// RS Hash Function
unsigned int RSHash(char *str)
{
    unsigned int b = 378551;
    unsigned int a = 63689;
    unsigned int hash = 0;

    while (*str)
    {
        hash = hash * a + (*str++);
        a *= b;
    }

    return (hash & 0x7FFFFFFF);
}

// JS Hash Function
unsigned int JSHash(char *str)
{
    unsigned int hash = 1315423911;

    while (*str)
    {
        hash ^= ((hash << 5) + (*str++) + (hash >> 2));
    }

    return (hash & 0x7FFFFFFF);
}

// P. J. Weinberger Hash Function
unsigned int PJWHash(char *str)
{
    unsigned int BitsInUnignedInt = (unsigned int)(sizeof(unsigned int) * 8);
    unsigned int ThreeQuarter = (unsigned int)((BitsInUnignedInt  * 3) /4);
    unsigned int OneEighth    = (unsigned int)(BitsInUnignedInt / 8);
    unsigned int HighBits = (unsigned int)(0xFFFFFFFF) << (BitsInUnignedInt -
OneEighth);
    unsigned int hash                   = 0;
    unsigned int test                   = 0;
```

```
    while (*str)
    {
        hash = (hash << OneEighth) + (*str++);
        if ((test = hash & HighBits) != 0)
        {
            hash = ((hash ^ (test >> ThreeQuarters)) & (~HighBits));
        }
    }

    return (hash & 0x7FFFFFFF);
}

// ELF Hash Function
unsigned int ELFHash(char *str)
{
    unsigned int hash = 0;
    unsigned int x       = 0;

    while (*str)
    {
        hash = (hash << 4) + (*str++);
        if ((x = hash & 0xF0000000L) != 0)
        {
            hash ^= (x >> 24);
            hash &= ~x;
        }
    }

    return (hash & 0x7FFFFFFF);
}

// BKDR Hash Function
unsigned int BKDRHash(char *str)
{
    unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
    unsigned int hash = 0;

    while (*str)
    {
        hash = hash * seed + (*str++);
    }

    return (hash & 0x7FFFFFFF);
}
```

```c
// DJB Hash Function
unsigned int DJBHash(char *str)
{
    unsigned int hash = 5381;

    while (*str)
    {
        hash += (hash << 5) + (*str++);
    }

    return (hash & 0x7FFFFFFF);
}

// AP Hash Function
unsigned int APHash(char *str)
{
    unsigned int hash = 0;
    int i;

    for (i=0; *str; i++)
    {
        if ((i & 1) == 0)
        {
            hash ^= ((hash << 7) ^ (*str++) ^ (hash >> 3));
        }
    else
        {
            hash ^= (~((hash << 11) ^ (*str++) ^ (hash >> 5)));
        }
   }

    return (hash & 0x7FFFFFFF);
}
```

Hash.c