<https://blog.csdn.net/weiyuefei/article/details/44004517>

哈希表（Hash table，也叫散列表），是根据关键码值(Key value)而直接进行访问的数据结构。它通过把关键码值映射到表中一个位置来访问记录，有点类似于数组，并且能在O(1)（冲突情况另算）下查找到元素。具体的介绍网上有很详细的描述 ，这里就不再累述了；

下面只是说一下几个关键问题：

#### 1.哈希函数

也叫散列函数，即：根据key,计算出key对应记录的储存位置

position = f(key)

散列函数满足以下的条件：

1、对输入值运算，得到一个固定长度的摘要(Hash Value)；

2、不同的输入值可能对应同样的输出值；

以下的函数都可以认为是一个散列函数：

f(x) = x mod 16; (1)

f(x) = (x 2 + 10) \* x; (2)

f(x) = (x | 0×0000FFFF) XOR (x >> 16); (3)

不过，仅仅满足上面这两条的函数，作为散列函数，还有不足的地方。我们还希望散列函数满足下面几点：

1、散列函数的输出值尽量接近均匀分布；

2、x的微小变化可以使f(x)发生非常大的变化，即所谓“雪崩效应”( [Avalanche effect](http://en.wikipedia.org/wiki/Avalanche_effect) )；

上面两点用数学语言表示，就是：

1， 输出值y的分布函数F(y)=y/m, m为散列函数的最大值。或记为y~U[0, m]

2，|df(x)/dx| >> 1；

从上面两点，大家看看，前面举例的三个散列函数，哪个更好呢？对了，是第三个：

f(x) = (x | 0×0000FFFF) XOR (x >> 16);

它很完美地满足“好的散列函数”的两个附加条件。

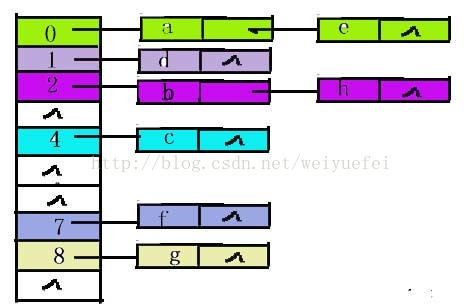
**2、哈希冲突（Hash collision）**

也就是两个不同输入产生了相同输出值的情况。首先，哈希冲突是无法避免的，因此，哈希算法的选择直接决定了哈希冲突发送的概率；同时必须要对哈希冲突进行处理，方法主要有以下几种：  链地址法、 开放地址法等等。

下面就来看看每种方法的具体实现吧；

链地址法：

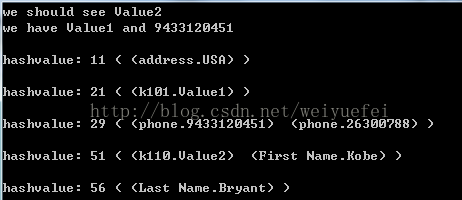
举例说明：设有 8 个元素 { a,b,c,d,e,f,g,h } ，采用某种哈希函数得到的地址分别为： {0 ， 2 ， 4 ， 1 ， 0 ， 8 ， 7 ， 2} ，当哈希表长度为 10 时，采用链地址法解决冲突的；



下面是代码实现，已经测试过：

1. **#include <string.h>**
2. **#include <stdio.h>**
3. **#include <stdlib.h>**
4. **typedef** **struct** node{
5. **char** \*name;*//字段名*
6. **char** \*desc;*//描述*
7. **struct** node \*next;
8. }node;
9. **#define HASHSIZE 100 *//hash表长度***
10. **static** node\* hashtable[HASHSIZE];*//定义一个hash数组，该数组的每个元素是一个hash结点指针,并且由于是全局静态变量,默认初始化为NULL*
12. **unsigned** **int** hash(**char** \*s)
13. {*//哈希函数*
14. **unsigned** **int** h=0;
15. **for**(;\*s;s++)
16. h=\*s+h\*31;*//将整个字符串按照特定关系转化为一个整数，然后对hash长度取余*
17. **return** h%HASHSIZE;
18. }
19. node\* lookup(**char** \*str)
20. {
21. **unsigned** **int** hashvalue = hash(str);
22. node\* np = hashtable[hashvalue];
23. **for**( ; np!=NULL; np = np->next)
24. {*//这里是链地址法解决的冲突,返回的是第一个链表结点*
25. **if**(!strcmp(np->name, str))*//strcmp相等的时候才返回0*
26. **return** np;
27. }
28. **return** NULL;
29. }
30. **char**\* search(**char**\* name)
31. {*//对hash表查找特定元素(元素是字符串）*
32. node\* np=lookup(name);
33. **if**(np==NULL)
34. **return** NULL;
35. **else**
36. **return** np->desc;
37. }
38. node\* malloc\_node(**char**\* name, **char**\* desc)
39. {*//在堆上为结点分配内存，并填充结点*
40. node \*np=(node\*)malloc(**sizeof**(node));
41. **if**(np == NULL)
42. **return** NULL;
43. np->name = name;
44. np->desc = desc;
45. np->next = NULL;
46. **return** np;
47. }
48. **int** insert(**char**\* name, **char**\* desc)
49. {
50. **unsigned** **int** hashvalue;
51. hashvalue = hash(name);
52. *//头插法，不管该hash位置有没有其他结点，直接插入结点*
53. node\* np = malloc\_node(name, desc);
54. **if** (np == NULL) **return** 0;*//分配结点没有成功，则直接返回*
55. np->next = hashtable[hashvalue];
56. hashtable[hashvalue] = np;
57. **return** 1;
58. }
59. */\* A pretty useless but good debugging function,*
60. *which simply displays the hashtable in (key.value) pairs*
61. *\*/*
62. **void** displayHashTable()
63. {*//显示hash表元素（不包括空）*
64. node \*np;
65. **unsigned** **int** hashvalue;
66. **for**(**int** i=0; i < HASHSIZE; ++i)
67. {
68. **if**(hashtable[i] != NULL)
69. {
70. np = hashtable[i];
71. printf("\nhashvalue: %d (", i);
72. **for**(; np != NULL; np=np->next)
73. printf(" (%s.%s) ", np->name, np->desc);
74. printf(")\n");
75. }
76. }
77. }
78. **void** cleanUp()
79. {*//清空hash表*
80. node \*np,\*tmp;
81. **for**(**int** i=0;i < HASHSIZE; ++i)
82. {
83. **if**(hashtable[i] != NULL)
84. {
85. np = hashtable[i];
86. **while**(np != NULL)
87. {
88. tmp = np->next;
89. free(np->name);
90. free(np->desc);
91. free(np);
92. np = tmp;
93. }
94. }
95. }
96. }
97. **int** main()
98. {
99. **char**\* names[]={"First Name","Last Name","address","phone","k101","k110"};
100. **char**\* descs[]={"Kobe","Bryant","USA","26300788","Value1","Value2"};
101. **for**(**int** i=0; i < 6; ++i)
102. insert(names[i], descs[i]);
103. printf("we should see %s\n",search("k110"));
104. insert("phone","9433120451");*//这里计算的hash是冲突的，为了测试冲突情况下的插入*
105. printf("we have %s and %s\n",search("k101"),search("phone"));
106. displayHashTable();
107. cleanUp();
108. **return** 0;
109. }

输出结果：



需要特别注意一下，上面的hash函数计算为29的，用的就是单链表的头插法来解决冲突，不要复杂化了问题！

上面的代码是我自己实现的版本，后面我还会继续增加一些其他更好的版本，Waiting！

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/weiyuefei/article/details/44004517