

Hashing文档

id: 519021910861

name: 徐惠东

1. 运行截图

```
C:\Users\14475\Desktop\coding\data_structure> & 'c:\Users\14475\.vscode\extensions\ms-vscode.cpptools-1.3.1\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-1tlkpidb.tnl' '--stdout=Microsoft-MIEngine-Out-ssmojnef.214' '--stderr=Microsoft-MIEngine-Error-c5eu0lus.vxf' '--pid=Microsoft-MIEngine-Pid-ath433fl.50v' '--dbgExe=C:\mingw64\bin\gdb.exe' '--interpreter=mi'
1 -1 1 -1 -1
C:\Users\14475\Desktop\coding\data_structure> & 'c:\Users\14475\.vscode\extensions\ms-vscode.cpptools-1.3.1\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-mxovrm5c.hds' '--stdout=Microsoft-MIEngine-Out-oqnkx1h4.5gb' '--stderr=Microsoft-MIEngine-Error-mnptq2ei.zrp' '--pid=Microsoft-MIEngine-Pid-pmcxigby.ux4' '--dbgExe=C:\mingw64\bin\gdb.exe' '--interpreter=mi'
1 -1 1 -1 -1
C:\Users\14475\Desktop\coding\data_structure>
```

2. 代码说明

- 用结构体定义节点

```
struct node {
    int key; // 键
    int val; // 值
    /* 节点状态
     * 0 --> 空
     * 1 --> 删除
     * 2 --> 有值
     */
    int state;
}
```

- 用stl中数组vector保存元素

```
const int length = 97; // 数组长度，选用质数最佳
vector<node> v;
int hash_function(int key, int val) {
    /* 哈希函数，将key映射到数组空间中 */
    return key % length;
};
```

- put函数

```

/* put伪代码 */
hash_key <-- hash_function(key)
for iter from hash_key to length:
    if iter.state = 0 || 1: iter.key = key, iter.val =
val, iter.state = 2; break;
    elif iter.key = key: iter.val = val; break;
    else: continue
for iter from 0 to hash_key:
    the same as above

```

- get函数：用哈希函数对key进行转换得到hash_key，并根据hash_key在数组中向后进行查找。

```

/* get伪代码 */
hash_key <-- hash_function(key)
for iter from hash_key to length:
    if iter.state = 0: return -1
    elif iter.key = key: return iter.val
    else: continue
for iter from 0 to hash_key:
    the same as above

```

- remove函数

```

/* remove伪代码 */
hash_key <-- hash_function(key)
for iter from hash_key to length:
    if iter.state = 0: return;
    elif iter.key = key: iter.state = 1; return;
    else: continue;

```

3. 时间复杂度分析

1. 分离链表法

假设有N个元素待插入，有M条链表。

1. 最好情况是直接插入（查找）成功，时间复杂度为 $O(1)$ 。
2. 最坏情况是哈希函数选取不当导致所有元素映射到其中一条链表中，时间复杂度为 $O(N)$ 。
3. 平均情况是每条链表有 N/M 个元素，时间复杂度为 $O(N/M)$ 。

2. 线性探测法

1. 最好情况是当哈希表的负载因子严格小于1，则插入和查找操作都可以在 $O(1)$ 内完成。
 2. 最坏情况是 $O(n)$ 。
 3. 平均情况，就负载因子 α ，成功搜索的预期时间为 $O(1 + 1/(1 - \alpha))$ ，而搜索失败的预期时间为 $O(1 + 1/(1 - \alpha)^2)$ 。
3. 布谷哈希法
- 查找和插入算法时间复杂度均为 $O(1)$ 。