

浙江大学

Data Structure & Database Technique
Project3
Project Report



2020~2021 春夏学期 2021 年 3 月 31 日

Categories

CHAPTER 1: INTRODUCTION	3
CHAPTER 2: ALGORITHM SPECIFICATION.....	3
CHAPTER 3: TESTING RESULTS	4
CHAPTER 4: ANALYSIS AND COMMENTS	6
出现的问题:	7

Project3

Chapter 1: Introduction

Background and Our goals :

散列表 (Hash table, 也叫哈希表), 是根据关键码值(Key value)而直接进行访问的数据结构。也就是说, 它通过把关键码值映射到表中一个位置来访问记录, 以加快查找的速度。这个映射函数叫做散列函数, 存放记录的数组叫做散列表。

给定表 M, 存在函数 $f(key)$, 对任意给定的关键字值 key, 代入函数后若能得到包含该关键字的记录在表中的地址, 则称表 M 为哈希(Hash) 表, 函数 $f(key)$ 为哈希(Hash) 函数。

实现散列表, 要求:

清晰的注释

具有再散列功能

能够处理冲突

具有性能相关的测试

java 中的 hashmap, 当数据数量达到阈值的时候(0.75), 就会发生 rehash, hash 表长度变为原来的二倍, 将原 hash 表数据全部重新计算 hash 地址, 重新分配位置, 达到 rehash 目的。

hash table 的大小是素数时碰撞的概率小 (有证明)。若直接扩大为两倍反而可能性能恶化。另外随着 size 变大, 素数的密度会下降得足够小, 也不至于反复扩容。

Chapter 2: Algorithm Specification

我们处理冲突, 就是采用开放地址, 解决冲突, 尝试 $_{pos}, _{pos+1}, _{pos+2}...$

. 开放寻址法: $H_i = (H(key) + d_i) \text{ MOD } m, i=1, 2, \dots, k (k \leq m-1)$, 其中 $H(key)$ 为散列函数, m 为散列表长, d_i 为增量序列, 可有下列三种取法:

1.1. $d_i=1, 2, 3, \dots, m-1$, 称线性探测再散列;

1.2. $d_i=1^2, -1^2, 2^2, -2^2, (3)^2, \dots, \pm (k)^2, (k \leq m/2)$ 称二次探测再散列;

1.3. d_i 为伪随机数序列, 称伪随机探测再散列。

取关键字或关键字的某个线性函数值为散列地址。即 $H(key)=key$ 或 $H(key) = a \cdot key + b$, 其中 a 和 b 为常数。若其中 $H(key)$ 中已经有值了, 就往下一个找, 直到 $H(key)$ 中没有值了, 就放进去。

我们桶个数 < 最大条目数 / 最大装载因子就扩容到一个素数, 并进行一次 rehash。

再散列, 就是 reload factor > 0.7 时重新散列, 用公式 nextprime 计算 hashtable 的下一个容量, 变大到下一个数, 遍历 hashTable, 备份所有有效的元素, 用公式 nextprime 计算 hashtable 的下一个容量, capacity 变大到公式计算出的值, 然后将之前所有的元素逐一插入。

Chapter 3: Testing Results

```
insert Entry(1,11)...
capacity=3, size=1
key=1, value=11
insert Entry(2,22)...
capacity=3, size=2
key=1, value=11
key=2, value=22
insert Entry(3,33)...
capacity=3, size=3
key=3, value=33
key=1, value=11
key=2, value=22
insert Entry(4,44)...
capacity=41, size=4
key=1, value=101
key=2, value=22
key=3, value=33
key=4, value=44
runtime: 0.002
```

Rehash+ insert 4,44 = 2ms

插入 4 个 rehash 一次 runtime: 0.008s , 8ms

插入 9 个, 9ms

```
wenchi.cpp [1] autobuild.cpp
3   cout << "insert Entry(3,33)..." << endl;
4   pTable->insert(Entry(3,33));
5   pTable->display();
6   cout << "insert Entry(4,44)...";
7   pTable->insert(Entry(4,44));
8   (*pTable)[1]->_value = 101;
9   pTable->display();
10  pTable->insert(Entry(5,55));
11  pTable->insert(Entry(6,66));
12  pTable->insert(Entry(7,77));
13  pTable->insert(Entry(8,88));
14  pTable->insert(Entry(9,99));
15  end_clock();
```

插入 28 个, 9ms

```
D:\curriculum_design to git\curriculum
key=1, value=11
key=2, value=22
insert Entry(3,33)...
capacity=3, size=3
key=3, value=33
key=1, value=11
key=2, value=22
insert Entry(4,44)...
capacity=41, size=4
key=1, value=101
key=2, value=22
key=3, value=33
key=4, value=44
runtime: 0.009
```

```

9 pTable->insert(Entry(24,264));
0 pTable->insert(Entry(25,275));
1 pTable->insert(Entry(26,286));
2 pTable->insert(Entry(27,297));
3 pTable->insert(Entry(28,308));
4     end=clock();
5     ret=double(end-begin)/CLOCKS_PER_SEC;
6     cout<<"runtime: " <<ret<<endl;
7     cout << endl << "delete Entry(1,11)..." <<
8     pTable->remove(Entry(1,11));
9     pTable->display();
0     cout << "delete Entry(2,22)..." << endl;
1     pTable->remove(Entry(2,22));
2     pTable->display();

```

```

key=2, value=22
insert Entry(3, 33)...
capacity=3, size=3
key=3, value=33
key=1, value=11
key=2, value=22
insert Entry(4, 44)...
capacity=4, size=4
key=1, value=101
key=2, value=22
key=3, value=33
key=4, value=44

delete Entry(1, 11)...
capacity=4, size=3
key=2, value=22
key=3, value=33
key=4, value=44
delete Entry(2, 22)...
capacity=4, size=2
key=3, value=33
key=4, value=44
delete Entry(3, 33)...
capacity=4, size=1
key=4, value=44
delete Entry(3, 33)...
capacity=4, size=1
key=4, value=44
runtime: 0.017

```

插入 4 个, 删除 3 个, 17ms

执行一百次空的 remove, 10ms

The screenshot shows a C++ IDE with a file named `testhash.cpp`. The code defines a hash table with a bucket array `pTable` and an `Entry` struct. It performs a series of insert and remove operations. The output window shows the runtime of these operations, including capacity and size changes, and the time taken for each operation.

```

146 pTable->remove(Entry(3,33));
147 pTable->remove(Entry(3,33));
148 pTable->remove(Entry(3,33));
149 pTable->remove(Entry(3,33));
150 pTable->remove(Entry(3,33));
151 pTable->remove(Entry(3,33));
152 pTable->remove(Entry(3,33));
153 pTable->remove(Entry(3,33));
154 pTable->remove(Entry(3,33));
155     end=clock();
156     ret=double(end-begin)/CLOCKS_PER_SEC;
157     cout<<"runtime: " <<ret<<endl;
158     cout << endl << "delete Entry(1,11)";
159     pTable->remove(Entry(1,11));

pTable->insert(Entry(20,220));
pTable->insert(Entry(21,231));
pTable->insert(Entry(22,242));
pTable->insert(Entry(23,253));
pTable->insert(Entry(24,264));
pTable->insert(Entry(25,275));
pTable->insert(Entry(26,286));
pTable->insert(Entry(27,297));
pTable->insert(Entry(28,308));
pTable->insert(Entry(25,275));
pTable->insert(Entry(26,286));
pTable->insert(Entry(27,297));
pTable->insert(Entry(28,308));
    
```

Output window content:

```

D:\curriculum_design to git\curricu
capacity=3, size=1
key=1, value=11
insert Entry(2, 22)...
capacity=3, size=2
key=1, value=11
key=2, value=22
insert Entry(3, 33)...
capacity=3, size=3
key=3, value=33
key=1, value=11
key=2, value=22
insert Entry(4, 44)...
capacity=41, size=4
key=1, value=101
key=2, value=22
key=3, value=33
key=4, value=44
runtime: 0.01

delete Entry(1, 11)...
capacity=41, size=26
key=2, value=22
    
```

100 次空的 insert, 8ms.

Chapter 4: Analysis and Comments

time complexities are $O(1)$ because use hash function to insert.

第一步: `key.hashfunction()`, 时间复杂度 $O(1)$ 。

第二步: 找到桶以后, 判断桶里是否有元素, 如果没有, 直接 new 一个 entey 节点插入到数组中。时间复杂度 $O(1)$ 。

space complexities of the algorithms

All space complexities are $O(n)$,rehash function needs to store all old values in the vector.

未来期待的改进:

优化_pos 函数, 减少_pos 私有变量的耦合,现在在各个函数中耦合比较严重.

出现的问题:

问题 1 我写了 100 个插入的测试,但是没有输出,

```
cout<<"runtime:  "<<ret<<endl;
pTable->display();
```

都没东西,为什么?

排查方法: 我把 99 个注释了, 正常输出.

36 个就不行, 不知道为啥. 21 个也是可以的.29 个也可以, 30 个就不行.

3221225477 (0xC0000005): 访问越界, 一般是读或写了野指针指向的内存

```

❗ ser1[0] = error: use of undeclared identifier 'ser1'
❗ ser1[0] = error: use of undeclared identifier 'ser1'
01 ret = (double) -9.2559631349317831E+61
✓ 01 pTable = (HashTable * | 0x169b770) 0x0169b770
01 _primeN = (int) 2
> 01 _pTable = (Entry ** | 0x16a0728) 0x016a0728
01 _size = (int) 28
01 _capacity = (int) 41
01 _pos = (int) 28
01 end = (long long) -3689348814741910324
01 begin = (long long) 0

```

$41 * 0.7 = 28.7$ 所以 29 个会出错. 下一步变成了 43

```
int HashTable::hashFunction(const KeyType& key) { //一种简单的哈希函数 ,
为了未来改进封装一下.

    return key%_capacity;
}
```

我怀疑是这里 key 没有取值,是一个地址

我一开始用的是

```
find(const KeyType& key) {
```

中修改 pos, 然后后面直接用这个 pos 插入.

```

_pos = (_pos+i)%_capacity;
...
_pTable[_pos] = new Entry(e); //当前位置插入 e

```

我觉得可能 return 一个下标更好?

Indirection requires pointer operand ('KeyType' (aka 'int') invalid

解决方法:

问题 2 Process finished with exit code -1073741819 (0xC0000005)

[问题 3](#) 函数。。。。已有主体”。

应该先在头文件中声明函数，然后在.cpp 中实现，这样调用函数时才不会提示说“函数。。。。已有主体”。

我就是多复制了一遍，然后 test.cpp 应该

```
#include "hashtable.hpp"
```

而不是 include cpp 文件

[问题 4 Exception 0xc0000005 encountered at address 0x961d54: Access violation reading location 0xcdcdcdcd](#)

ptr 是一个指点，但你似乎从来没有初始化它。换句话说，没有指向任何东西。

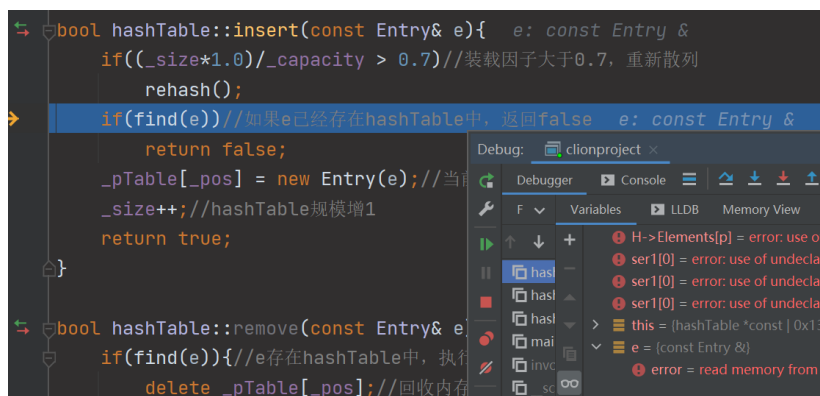
当您尝试使用此指针时，你得到未定义的行为。你实际上很幸运，应用程序崩溃，因为任何其他可能发生。它本来可以似乎工作。要直接解决此问题，您需要初始化。

Key 为什么会没有东西呢？

在这里是正确的



Rehash 后 e 没有了,为什么？



我明白了，


```
for(int i = 0; i < sizeOld; i++) //将之前所有的元素逐一插入
    insert(*pTableOld[i]);
```

这里会逐个插入,调用 insert, 第一个会是一个空指针传入 insert.

```
pTableOld[i] = new Entry(*_pTable[i]);
```

没有存进去, 为啥?

那 4 为啥不会出 bug? 4 的第一个是 3,33 传入 insert 没有出错,但是其实应该是第一个 1,11,.

第二个是 1,11,第三个是 2,22, 然后 I=3 ,sizeold = 3

为什么第一个是 3,33?

然后 30 插入的时候,需要 rehash

```
if(_pTable[i] != NULL)
    pTableOld[i] = new Entry(*_pTable[i]);
delete []_pTable;
_capacity = nextPrime(); //hashTable增容到下一个公式计算出的数
_pTable = new Entry*[_capacity];
for(int i = 0; i < _capacity; i++)
    _pTable[i] = NULL;
int sizeOld = _size; sizeOld: 3
_size = 0;
for(int i = 0; i < sizeOld; i++) //将之前所有的元素逐一插入 sizeOld: 3
    insert(*pTableOld[i]); pTableOld: 0x0080f930
```

在备份后, 插回去就报错了

```
for(int i = 0; i < _capacity; i++) //遍历 hashTable, 备份所有有效的元素
    if(_pTable[i] != NULL)
        pTableOld[i] = new Entry(*_pTable[i]);
```

这里没有备份进去,但是我不知道为什么.