朱奕新
19307090029

Lab 5 实验报告

数据结构
2021 年 10 月 15 日

1 中文字符串哈希函数

1. 由于以姓名作为关键字，就必须要实现将中文转化为整数。办法是每个字转化为 Unicode 编码，然后转化为按适当的基数符号表示的整数。

2. 其他卫星数据，和姓名放在同一个列表内。姓名是列表第一个元素，以它为关键字参与判断。同一组数据“同进同退”，作为散列表的一个元素。

3. python 中的 ord() 函数，可以取一个字符的 Unicode 值。

4. 经查阅资料得知，基本汉字与汉字扩展包 A 的 Unicode 值都在十进制 100000 以下，故可以选取 100000 为基数，产生中文字符串对应的整数值（字符串较长时可能会较大），详情请见算法 1.

5. 转化为对应整数值以后，就可以使用开放寻址法中的双重散列法和线性探查法的散列函数，对应到散列表中的下标了，详情请见算法 2 和算法 3.

算法 1: 中文字符串哈希函数

```
def chineseStringHash(s):
        result = 0
        for i in range(0, len(s)):
                result *= 100000
                result += (ord(s[i]))
        return result
```

算法 2: 双重散列法的散列函数

```
def h1(k, m):
        return k % m

def h2(k, m):
        return 1 + k % (m-1)

def double(k, i, m):
        return (h1(k, m)+i*(h2(k, m))) % m
```

算法 3: 线性探查法的散列函数

```
def h(k, m):
```

```
2         return k % m
3
4 def linear(k, i, m):
5         return (h(k, m)+i) % m
```

2 开放寻址散列表的实现

1. HashTableDouble 类的几个主要函数核心代码如下。insert(key) 函数可以依照关键字插入一份数据。insrtList(keys) 函数可以插入一个数据表。search(key) 函数可以依照关键字查找，并打印它及其卫星数据。searchList(keys) 函数可以依照关键字查找一整个数据表。delete(key) 函数可以依照关键字找到并删除一份数据。

2. HashTableLinear 类的函数与这些代码基本一致，除了在调用散列函数 double() 的地方改为调用散列函数 linear()。

3. 为了读取 csv 文件、插入数据表、测试查找次数和运行时间以及保证双重散列的表长为质数，还实现了其他配套函数，因不属于核心算法，此处不列出。详情请见实际代码。

算法 4: search(key) 函数

```
1  def search(self, key):
2          k = ChineseHash(key)
3          for i in range(0, self.__size):
4                  j = double(k, i, self.__size)
5                  if(self.__table[j][0] == key):
6                          print("Successfully find")
7                          print(self.__table[j])
8                          return j
9                  if(self.__table[j] == EMPTY):
10                         break
11         print("Can't find ", end='')
12         print(key)
13         return -1
```

算法 5: insert(key) 函数

```
def insert(self, item):
        k = ChineseHash(item[0])
        for i in range(0, self.__size):
                j = double(k, i, self.__size)
                if(self.__table[j] == EMPTY or self.__table[j] == DELETED):
                        self.__table[j] = item
                        return j
        print("hash table overflow!")
        return -1
```

算法 6: delete(key) 函数

```
def delete(self, key):
        index = self.search(key)
        if(index == -1):
                print("and nothing to delete")
                return
        self.__table[index] = DELETED
        print("and successfully delete")
```

3 平均搜索时间的实验与分析

3.1 实验结果

1. 实验测试了依照所有各关键字查找所需次数的总和与所需时间的总和。为了研究散列表所用空间大小与查找所需次数与所需时间的关系，本实验从一倍于数据规模测到了 10 倍于数据规模。

2. 一个表中姓名的查找次数除以数据总数即为单个名字平均查找次数。一个表中姓名的查找时间除以数据总数即为单个名字平均查找时间时间的单位是纳秒。

散列表空间	总查找次数	单个姓名平均查找次数	总查找时间	单个姓名平均查找时间
1 倍	662	4.413333333	57846000	385640
2 倍	202	1.346666667	59843300	398955.3333
3 倍	181	1.206666667	43883300	292555.3333
4 倍	185	1.233333333	49866800	332445.3333

5 倍	167	1.113333333	46876500	312510
6 倍	158	1.053333333	46831900	312212.6667
7 倍	163	1.086666667	41890300	279268.6667
8 倍	158	1.053333333	43839900	292266
9 倍	168	1.12	48000900	320006
10 倍	157	1.046666667	46877300	312515.3333

表 1：双重散列法查找次数和查找时间实验

散列表空间	总查找次数	单个姓名平均查找次数	总查找时间	单个姓名平均查找时间
1 倍	896	5.973333333	81344600	542297.3333
2 倍	304	2.026666667	71963700	479758
3 倍	208	1.386666667	69644000	464293.3333
4 倍	242	1.613333333	73239200	488261.3333
5 倍	215	1.433333333	76416900	509446
6 倍	186	1.24	45436800	302912
7 倍	174	1.16	44159700	294398
8 倍	218	1.453333333	42797200	285314.6667
9 倍	166	1.106666667	42435000	282900
10 倍	212	1.413333333	43981700	293211.3333

表 2：线性探查法查找次数和查找时间实验

3.2 结果分析

1. 实验发现无论是双重散列法还是线性探查法，总体上查找次数和查找时间都随着散列表空间的增加而减少，但随着空间变得更大这一下降变得更慢。

2. 实验发现双重散列法所需时间更短。这是由于线性探查法存在一次群集的问题，随着槽数占用的增加而越发严重；而双重散列法产生的序列具有随机选择排列的许多特性，因此性能较好。