自动增长散列表　算法说明

地球与空间科学学院　王冠力　1600012626

首先，这里的散列表应用的解决冲突的方法是线性探测；散列函数为对散列表大小求余。

在定义的 HashTable 类中增添 self.hold，表示当前已盛放数据的槽的数目；增添 self.limit，代表给定的负载因子阈值。

在放入新数据之前，检查加入数据后负载因子是否会溢出；如果会，当前散列表就倍增，同时对已有数据的位置做出调整，以提高查找的效率（减少冲突情况下线性寻找的时间开销）。具体的调整步骤如下：

散列表倍增后，逐个考察前一半的槽；对于非空的槽，先确定槽中数据在新的散列表中的哈希值。如果哈希值和当前槽的位置一致，说明不需要移动；如果不一致，则考察哈希值对应的槽是否为空槽。若为空，直接存入数据，同时将当前槽制成空槽即可；若不为空，则可再分为两种情况：哈希值对应槽中的数据不需要移动，此时需要对当前槽的数据进行再散列；哈希值对应槽中的数据需要移动，则该槽可以盛放当前数据，同时需要对该槽的原数据的位置进行调整。

完成上述倍增和调整过程后，再进行新数据的存入操作。这样，对于任意状态下的散列表，均能使散列表中数据盛放在满足散列函数的前提下的冲突达到最小。

例如，对于下面一段测试代码：

```
from Hash_Dynamic import HashTable
import random
h = HashTable(5, 0.9)
print(str(h))
for i in range(10):
    new = random.randrange(100)
    h.put(new)
    print("Inserting %d:      " % new, str(h))

print(h.search(7))
```

给定散列表的初始大小为 5，负载因子阈值为 0.9；存入 10 个在 0 - 100 间产生的随机数，分别输出每次存入后散列表的状态如下：

```
[None, None, None, None, None]
Inserting 41:     [None, 41, None, None, None]
Inserting 0:     [0, 41, None, None, None]
Inserting 4:     [0, 41, None, None, 4]
Inserting 10:     [0, 41, 10, None, 4]
Inserting 58:     [0, 41, 10, None, 4, None, None, None, 58, None]
Inserting 78:     [0, 41, 10, None, 4, None, None, None, 58, 78]
Inserting 3:     [0, 41, 10, 3, 4, None, None, None, 58, 78]
Inserting 7:     [0, 41, 10, 3, 4, None, None, 7, 58, 78]
Inserting 84:     [0, 41, None, 3, 4, 84, None, 7, None, None, 10, None, None, None, None, None, None, None, 58, 78]
Inserting 78:     [0, 41, 78, 3, 4, 84, None, 7, None, None, 10, None, None, None, None, None, None, None, 58, 78]
True
```

可以看出当散列表大小为 5 时，存入第 5 个数据时散列表倍增；散列表大小为 10 时，存入第 9 个数据时，散列表倍增。