Hashing, Collisions, Sets, and Maps

January 7, 2019

---

1 Hashing

- A hashing function is a function which generates an integer value from an item
  - An array, called *table*, is create and the index of the item will be:

    ```
    abs(hashcode) % len{table}
    ```

  - Now the table can be immediately examined to see whether the item is present or not

---

2 Collisions

- A problem with hashing is that two different items may have the same index, this is called a *Collision*
- A solution to *Collisions* is:
  - Open addressing
    - find another address
  - Chaining
    - Linked list at each table entry
    - Simply append items to the linked list

---

3 Code for implementing HashSets

```
class HashSet:
    def __init__(self):
        # Create a list to use as the hash table
        self.table = [None]*10

    def add(self, item):
        # Find the hash value
        h = hash(item)
        # Get the index
        index = h % len(self.table)
        print(index)

        if self.table[index] == None:
            # Generates a LinkedList
            self.tabl[index] = LinkedList()
        if item not in self.table[index]:
            # Don't add the item if already present
            self.table[index].add(item)
        print(self.table[index])
```

---

4 Complexity

- Average time complexity of *contains*, *add*, and *remove* is $O(1)$
- Each operation operates on a linked list of about 1 element
- If the table size is smaller than the number of items
  - Then the basic operations become 4 times slower but is still $O(1)$
- With some operations, you need to examine every entry in the linked list
  - $O(k+n)$ where k is the table size and n is the size of the linked list
- When the hash table gets crowded, performance deteriorates
  - The load factor is the size of the set over the size of the table: n/k and usually, we increase the hash table size if the load factor goes above say 0.75
  - Increasing the size of the table is expensive as each item's hash has to be recalculated

---

5 Hash Function

- An ideal hash function gives a good range of values
  - If we were hashing phone numbers, it would be bad to base the hash on the first three digits. If we did, then any phone number with the same area code would have the same hash
    - There would be many collisions
- *hash(item)*
  - Returns the hash code of the item
    - It will be an integer
  - However, you may overwrtie the hash function
    - `__hash__()`
- Ideally, the hash would use the contents of the item to generate the hash code
- The

  `__hash__`

  method would look like:

```
def __hash__():
    hashVal = 0
    for c in self:
        hashVal = hashVal + ord(c)

    return hashVal
```