CS 2302 Data Structures - Hash Tables and Sets

Author:
[Olac Fuentes]
Computer Science Department
University of Texas at El Paso
Last modified: 12/21/2022

Hash Tables

How long does it take to determine if an item belongs to a group of items?

- O(n) if we use a list.
- O(log n) if we use a balanced binary search tree.

We can perform this operation in O(1) time using a **hash table**.

How does a hash table work?

It uses a list of lists (called buckets) and a hash function *h*.

Given item *k*, *h(k)* is the index of the bucket where *k* should be stored in the list.

Thus, to determine if *k* is in table *H*, we go directly to bucket *h(k)*. If *k* is not in that bucket, it means that it is not in the table.

Since hash tables are designed to ensure buckets are short (usually 0 or 1 items), searching the appropriate bucket can be assumed to take O(1) time.

Sets

A set is an unordered collection data type that is iterable, mutable and has no duplicate elements. Python's set class represents the mathematical notion of a set.

The main advantage of sets over lists is that membership queries can be performed in O(1) time instead of O(n) using the **hash table** data structure described above.

Let's compare running times of membership queries for a list L and a set S, both containing the same elements.

First we'll build a list and a set with the integers from 0 to n in random order.

```
import numpy as np

n = 4000
L = list(np.random.permutation(n))
S = set(L)
```

```
L
```

```
S
```

Now we'll perform a membership query for each of the integers in the 0 to 2n range. Thus, half the queries will return True, and half of them will return False - the results of the queries are not relevant here; we are interested in comparing running times.

```
import time

start = time.time()
for i in range(2*n):
  t = i in L # Check membership and store in variable
elapsed_time1 = time.time() - start
print('elapsed time using list', elapsed_time1,'secs')

start = time.time()
for i in range(2*n):
  t = i in S  # Check membership and store in variable
elapsed_time2 = time.time() - start
print('elapsed time using set', elapsed_time2,'secs')

print('ratio',elapsed_time1/elapsed_time2)
```

```
elapsed time using list 3.487276792526245 secs
elapsed time using set 0.0015811920166015625 secs
ratio 2205.4733112183353
```

As with lists, sets can have elements of any type (string, integer, float, boolean).

```
S = set(['CS',2302,3.1416,True])
```

Set operations

Operations involving one set

```
S =  set(['red','green','blue'])
```

Length

```
len(S)
```

```
3
```

Creating an empty set

```
S = set()
len(S)
```

```
0
```

Adding an element to a set

```
S = set()
for i in range(4,-1,-1):
  S.add(i)
  print(S)
len(S)
```

```
{4}
{3, 4}
{2, 3, 4}
{1, 2, 3, 4}
{0, 1, 2, 3, 4}
5
```

If we try to add an element that is already in the set, the set doesn't change

```
print(S)
S.add(3)
print(S)
```

```
{0, 1, 2, 3, 4}
{0, 1, 2, 3, 4}
```

Notice that sets are not subscriptable, since they are unordered collections. See below:

```
print(S[0])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-ce3a7af0ac4e> in <module>
----> 1 print(S[0])

TypeError: 'set' object is not subscriptable
```

SEARCH STACK OVERFLOW

However, can iterate over the members of a set using a for loop without indexing

```
L = [1,2,3,4,5]
S = set(L)
for s in S:
  print(s)
```

```
1
2
3
4
5
```

Membership queries

```
S =  set(['red','green','blue'])
```

```
'red' not in S
```

```
False
```

```
'black' in S
```

```
False
```

```
'black' not in S
```

```
True
```

```
'red' in S
```

```
True
```

```
0 in S
```

```
False
```

Removing an element from a set

```
S = set(np.arange(5))
for i in range(5):
  S.remove(i)
  print(S)
len(S)
```

```
{1, 2, 3, 4}
{2, 3, 4}
{3, 4}
{4}
set()
0
```

If we try to remove an element that is not in the set, we get an error

```
S = set(np.arange(5))
S.remove(8)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-41-3f3735786c5d> in <module>
      1 S = set(np.arange(5))
----> 2 S.remove(8)

KeyError: 8
```

SEARCH STACK OVERFLOW

If the element we want to remove may not be in the set, we need to check membership first to prevent the error.

```
S = set(np.arange(5))
print(S)
for item in [1,4,8]:
  if item in S:
    S.remove(item)
  print(S)
```

```
{0, 1, 2, 3, 4}
{0, 2, 3, 4}
{0, 2, 3}
{0, 2, 3}
```

The same results can be obtained using the **discard** operation. *S.discard(item)* removes *item* from *S*, if *item* belongs to *S*, without generating an error if it does not.

```
S = set(np.arange(5))
print(S)
for item in [1,4,8]:
  S.discard(item)
  print(S)
```

```
{0, 1, 2, 3, 4}
{0, 2, 3, 4}
{0, 2, 3}
{0, 2, 3}
```

Operations involving two sets

```
S =  set(['red','green','blue'])
```

```
S1 = set(['red','orange','brown'])
S.isdisjoint(S1)
```

```
False
```

```
S1 = set(['pink','orange','brown'])
S.isdisjoint(S1)
```

```
True
```

```
print(S)
print(S1)
```

```
{'red', 'green', 'blue'}
{'pink', 'orange', 'brown'}
```

```
S1 = set(['red'])
S.issubset(S1)  # Is S a subset of S1?
```

```
S1 = set(['red'])
S1.issubset(S)  # Is S1 a subset of S?
```

```
True
```

```
S.issubset(S)
```

```
True
```

```
S1 = set(['red'])
S.issuperset(S)  # Is S a superset of S1?
```

```
True
```

```
S1 =  set(['red','green','blue'])
S2 = set(['red','orange','brown'])
print(S1.union(S2))
```

```
{'green', 'orange', 'blue', 'red', 'brown'}
```

```
print(S1.intersection(S2))
```

```
{'red'}
```

```
print(S1.difference(S2))  # Elements in S1 but not in S2
```

```
{'green', 'blue'}
```

```
print(S2.difference(S1))  # Elements in S2 but not in S1
```

```
{'orange', 'brown'}
```

Since membership queries, insertions, and deletions take O(1) time, issubset, issuperset, union, intersection, and difference take O(n) time.

Converting a list to a set and back to a list may alter the order of the elements, since sets are unordered collections.

```
L = ['Monday', 'Tuesday', 'Wednesday']
S = set(L)
L = list(S)
print(L)
```

```
['Monday', 'Tuesday', 'Wednesday']
```

Examples

Example 1. Write the function missing_item(L) that receives a list L that contains, in random order, all integers from 0 to len(L), except for one, and returns the missing integer.

```
def missing_item(L):
  S = set(np.arange(0,len(L)+1)).difference(set(L))
  return list(S)[0]
```

```
n = 10
L = list(np.random.permutation(n)[:-1])
print(L)
print('missing item:', missing_item(L))
```

```
[5, 2, 4, 0, 6, 9, 3, 1, 8]
missing item: 7
```

Example 2. Write the function has_duplicates(L) that receives a list L and determines if it contains duplicate elements.

Key observation: if we obtain a set from the elements of L, duplicates will be removed. We can thus just compare the length of L and the length of set(L) to determine if duplicates exist.

```
def has_duplicates(L):
  return len(L) > len(set(L))
```

```
has_duplicates([1,4,2,6,8,9])
```

```
False
```

```
has_duplicates([1,4,2,6,8,9,11,4,5])
```

```
True
```

Example 3. Write the function *missing_duplicate(L)* that receives a list *L* where every item appears twice except for one and returns the item that appears only once in *L*.

Idea:

Traverse *L*, the first time an item appears, we add it to an initially empty set; the second time it appears, we remove it. At the end, the set will contain only the item that appeared only once.

```
def missing_duplicate(L):
  seen_once = set()
  for item in L:
    if item in seen_once:
      seen_once.remove(item)
    else:
      seen_once.add(item)
  return list(seen_once)[0]
```

```
missing_duplicate([5,2,4,3,1,0,3,0,2,1,5])
```

```
4
```

```
import random

L = [chr(i) for i in range(ord('a'),ord('m')+1)]
L = L+L
random.shuffle(L)
L.pop()
print(L)
print('Missing duplicate:', missing_duplicate(L))
```

```
['f', 'i', 'e', 'j', 'h', 'a', 'b', 'b', 'c', 'i', 'm', 'a', 'h', 'c', 'f', 'k',
Missing duplicate: l
```