COMP2521 24T1
Applications of Hash Tables

Kevin Luxa
.cs2521@cse.unsw.edu.au

set adt
counter adt
assorted problems
A hash table is a data structure that stores key-value pairs, where keys are unique

**Operations**

**Insert:** Insert or replace key-value pair

**Lookup:** Given a key, get its associated value

**Delete:** Given a key, delete its key-value pair

**Performance**

**Average-case:** $O(1)$

Assuming good hash function and appropriate resizing

**Worst-case:** $O(n)$

If all keys hash to the same value (extremely unlikely with good hash)
Applications of Hash Tables

Hash tables are used everywhere due to their efficiency
A set is an unordered collection of distinct elements

**Operations**

**Insert:** Insert an item into the set  
**Membership:** Check if an item is in the set  
**Delete:** Delete an item from the set
/** Creates a new empty set */
Set SetNew(void);

/** Free memory used by set */
void SetFree(Set set);

/** Inserts an item into the set */
void SetInsert(Set set, int item);

/** Checks if an item is in the set */
bool SetContains(Set set, int item);

/** Deletes an item from the set */
void SetDelete(Set set, int item);

/** Returns the size of the set */
int SetSize(Set set);

/** Displays the set */
void SetShow(Set set);
<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Insert</th>
<th>Membership</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unordered array</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Ordered array</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Ordered linked list</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>AVL tree</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Hash table</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
How to implement the Set ADT using a hash table?

**Insert**
Insert item into the hash table as a key
Can use anything as the value

**Contains**
Check if the item exists in the hash table

**Delete**
Delete the item from the hash table
## Set ADT

### Implementations

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Insert</th>
<th>Membership</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unordered array</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Ordered array</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>AVL tree</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Hash table*</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

*average costs*
A counter is a collection of items where each distinct item has a count

Operations

**Add:** Add one to the count of an item  
**Get:** Get the count of an item
/** Creates a new empty counter */
Counter CounterNew(void);

/** Free memory used by counter */
void CounterFree(Counter c);

/** Add one to the count of an item */
void CounterAdd(Counter c, int item);

/** Get the count of an item */
int CounterGet(Counter c, int item);
## Counter ADT

### Implementations

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Add</th>
<th>Get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unordered array</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Ordered array</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>AVL tree</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Hash table</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
How to implement the Counter ADT using a hash table?

Use hash table to map items to their counts

Add
Look up item’s count in the hash table
Then re-insert the item into the hash table
with count increased by 1

Get
Look up item’s count in the hash table
## Counter ADT (Implementation)

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Add</th>
<th>Get</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unordered array</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Ordered array</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>AVL tree</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Hash table*</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

* average costs
Hash tables are often used as sets or counters to solve problems efficiently

Examples
Two sum
Odd occurring elements
Anagram
Problem

Given an array of integers and a target sum $S$, determine whether the array contains two integers that sum to $S$.

Examples

Consider the array $A = [12, 6, 3, 3, 7, 8]$

- $\text{twoSum}(A, 13) \Rightarrow \text{true}$
- $\text{twoSum}(A, 16) \Rightarrow \text{false}$
- $\text{twoSum}(A, 3) \Rightarrow \text{false}$
- $\text{twoSum}(A, 6) \Rightarrow \text{true}$
Problem

Given an array of integers, return the number of distinct integers that occur an odd number of times.

Examples

oddOccurring([4, 3, 4, 8, 8, 4]) ⇒ 2
oddOccurring([7, 2, 1, 5, 6, 9]) ⇒ 6
oddOccurring([1, 1, 3, 3, 7, 7]) ⇒ 0
Problem

Given two strings $s$ and $t$, determine whether they are anagrams.

Two strings are anagrams if they contain the same amount of each character.

Examples

anagram("abcde", "edcba") ⇒ true
anagram("abcde", "fdcba") ⇒ false
anagram("abcde", "abcdef") ⇒ false
anagram("aaabb", "ababa") ⇒ true
anagram("aaabb", "babab") ⇒ false
Bonus content!

Python has built-in syntax for hash tables, which are called **dictionaries**.
Demo: Python Dictionaries

**Operations**

Create a dictionary

```python
my_dictionary = {}
```

Insert a key-value pair

```python
my_dictionary[key] = value
```

Check if a key exists

```python
key in my_dictionary
```

Get the value associated with a key

```python
my_dictionary[key]
```

Delete a key-value pair

```python
del my_dictionary[key]
```
https://forms.office.com/r/5c0fb4tvMb