

CS2040

Lab 5 – Hashing

Take-Home Assignment 2a – Join Strings

- Use custom linked list implementation (TailedLinkedList.java suffices)
- Use `ArrayList<TailedLinkedList>` (or a regular `TailedLinkedList[]` array)
- When concatenating string `b` to `a` (ie. string is `a + b`), set tail node of `a` to point to head node of `b`
- Update tail node pointer of `a` to tail node of `b`
- No need to set `b` to null in the array after concatenation, as `b` is never referenced again (but you can do so if you want)
- Use `StringBuffer` or `StringBuilder` when constructing the final answer

Take-Home Assignment 2b – Teque

- All operations need to run in $O(1)$ time
- For access to an element in $O(1)$ time (get() operation), this suggests the use of an array
- Pushing to the back can easily be done in $O(1)$ time too
- For adding to the front, instead of fixing index 0 as the front, allow for the front index to change
- This can be accomplished by using a circular array, with head and tail indices, as well as size attribute

Take-Home Assignment 2b – Teque

- How to insert to middle?
- Split into two circular arrays, one contains the front half (array 1), while the other contains the back half of the ADT (array 2)
- Balance out the number of elements between the two arrays after every push operation if necessary, so the middle is easily accessible
 - Should never need to move more than 1 element between the two arrays, so this is still $O(1)$
- For a `get(n)` operation using 2 arrays, access the n th element in array 1, or the $(n - (\text{size of array 1}))$ th element in array 2 if n exceeds the size of array 1

Lab 5 – Hashing

- Two different forms of hash tables are provided by Java
- HashMaps involve a key-value pair
 - The key is the object which is hashed, and then put into the HashMap depending on its resulting hash code
 - The value is a separate object that is associated with that key, and its contents are not considered when computing the hash code of the key
 - The entire key-value pair is called an entry
- HashSets simply involve one object, that acts as both the key and the value

Lab 5 – Hashing

- Note: there is also a Hashtable class which behaves in the same manner as a HashMap, but is generally slower than HashMap for the purposes of CS2040
 - The reason is similar to ArrayList vs Vector ie. synchronisation

Lab 5 – HashSet

Method name	Description	Time
<code>.add(YourClass element)</code>	Adds <i>element</i> to the HashSet	$O(1)$
<code>.clear()</code>	Clears the HashSet	$O(n)$
<code>.contains(Object o)</code>	Checks if <i>o</i> is in the HashSet, based off the object's <code>equals()</code> method	$O(1)$
<code>.isEmpty()</code>	Checks if the HashSet is empty	$O(1)$
<code>.remove(Object o)</code>	Removes <i>o</i> if it is in the HashSet, based off the object's <code>equals()</code> method	$O(1)$
<code>.size()</code>	Returns the number of elements in the HashSet	$O(1)$

Lab 5 – HashMap

- HashMaps are the first class encountered in this module that require the use of more than one generic
- To declare a HashMap with an Integer as the key, and a String as the value:
 - `HashMap<Integer, String> map = new HashMap<Integer, String>();`
- The two generics used need not be different eg. to use an integer as the key, and another integer as the value:
 - `HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();`

Lab 5 – HashMap

- HashMaps are generally useful in answering queries where the key is known, but the value is unknown
 - Eg. if we want to know which stall at a food court sells a certain food item, we could store a key-value pair of all foods, where the key is the food item's name, and the value is the stall name.
 - If we have a craving for a certain food item later, we use the food item's name to look up the HashMap, and check the value (stall name) associated with that key
- Effectively, using the key as a “reference” for the value

Lab 5 – HashMap

Method name	Description	Time
<code>.put(YourClass key, YourClass value)</code>	Adds <i>key</i> to the HashMap with the value <i>value</i>	$O(1)$
<code>.clear()</code>	Clears the HashMap	$O(n)$
<code>.containsKey(Object o)</code>	Checks if key <i>o</i> is in the HashMap, based off the object's <code>equals()</code> method	$O(1)$
<code>.containsValue(Object o)</code>	Checks if value <i>o</i> is in the HashMap, based off the object's <code>equals()</code> method	$O(n)$
<code>.get(Object o)</code>	Gets the value corresponding to the key <i>o</i>	$O(1)$
<code>.isEmpty()</code>	Checks if the HashMap is empty	$O(1)$
<code>.remove(Object o)</code>	Removes the entry with key <i>o</i> if it is in the HashMap, based off the object's <code>equals()</code> method	$O(1)$
<code>.size()</code>	Returns the number of elements in the HashMap	$O(1)$

Lab 5 – HashMap (cont.)

Method name	Description	Time
.entrySet()	Returns a set of all entries in the HashMap	O(1)
.keySet()	Returns a set of all keys in the HashMap	O(1)
.values()	Returns a collection of all values in the HashMap	O(1)

Unlike HashSet, it is not possible to iterate through a HashMap (eg. enhanced for-loop) directly. You will need to use the above methods to access an iterable form of the data stored within

One-Day Assignment 4 – Conformity

- Given the course combinations of students, determine how many students are taking one of the most popular combination of courses
 - There can be multiple combinations of courses which are tied for the most popular combination eg:
 - (100, 101, 102, 103, 104) – 4 students
 - (101, 102, 103, 104, 105) – 2 students
 - (102, 103, 104, 105, 106) – 4 students
 - Your answer should be $4 + 4 = 8$ (the students taking the first and third combinations)
- Note that course combinations can be given in different orders, but should be considered as the same course combination
 - Eg (100, 101, 102, 103, 488) is the same combination as (103, 102, 101, 488, 100) as given in Sample Input 1