CSCI 102 assignment 4 – Hash maps

Oct 30, 2024

1 Using maps

In this part of the assignment, you will be using maps. You'll see that using unsorted maps

- Build a `main` method in `UnsortedMap` where you create a `Map` variable that points to an `UnsortedMap` that has values of type `String` with keys of type `DoublyLinkedList<Double>`. Add entries with keys [3.43,5.432], [7], [812.4, 12.76, 123.4] and values "One", "Two", "Three". Print out the result of `get([7])`. Try to add "four" with key [7] and print the result of `get([7])` again.
- Override `hashCode` in `DoublyLinkedList` to get a hash code that turns the elements in the list into a string and then hashes the concatenated string with commas in between and brackets on the ends– i.e. a list that has the doubles `3.43` and `5.432` should have the hash of the string "[3.43,5.432]". To get the string of an element, use the built in Object method `toString()`.
- Write another method `hashCodeAlternative` for `DoublyLinkedList` that implements a polynomial hash code.
- Build a `main` method in `HashMapSC` where you create a `Map` variable that points to an `UnsortedMap` that has values of type `String` with keys of type `DoublyLinkedList<Double>`. Add entries with keys [3.43,5.432], [7], [812.4, 12.76, 123.4] and values "One", "Two", "Three". Print out the result of `get([7])`. Try to add "four" with key [7] and print the result of `get([7])` again. You should reuse the code from the first part of this problem!

2 Hashing trees

- Give an example of two different trees that have the same preorder traversal, two binary trees with the same inorder traversal, two trees with the same postorder traversal, and two trees with the same breadth-first-search traversal.
- Say we implemented a method `preOrderString` that prints out a string **with brackets** as such:

  ```
  root.toString()(preOrderString(first_child))(preOrderString(second_child))...
  ```

  where `preOrderString(root.left)` outputs `preOrderString` for the subtree rooted at `root.left`. Can two different trees have the same output `preOrderString` (assume `E.toString()` cannot contain brackets)? If so, give an example; if not, explain why not.
- Implement `preOrderString` for `LinkedTree<E>`.
- Override `hashCode` and `equals` in `LinkedTree<E>` with a more sensible implementation.
- (Bonus) Why is this not a good hash code for binary trees? Give an example. How could we modify it so it works fine for binary trees?
- (Bonus) Give an example of where this is a bad hash code if `E.toString()` contains brackets.

3 Open addressing

Implement a hash map `HashMapOA` that resolves collisions with open addressing with linear probing. Don't worry about implementing `values`, `keySet`, or `entrySet`; just build a constructor and implement `get`, `put`, and, `remove`. For a placeholder, use an entry which has both key and value set to null. Don't worry about the case where every index in the list has an entry.
Please submit your code and answers to the questions in a zipped folder on Brightspace by Nov 6.