

The purpose of this assignment is to practice your knowledge of hash tables. The assignment must be completed in pairs.

1. The Problem

We have been asked to construct a hash table of strings. We know that hash tables are extremely valuable in performing insertions and searches in constant time, on average.

The way in which we resolve collisions in a hash table is critical to attaining a running time of $O(1)$ in the average case. Such collisions are unavoidable because we have many more possible items than positions in the table. However, the number of collisions strongly depends on the effectiveness of the hash function at distributing items evenly throughout the table. Thus, we are interested in evaluating the performance of two strategies for resolving collisions (quadratic probing and separate chaining), as well as, the performance of several hashing functions to determine which strategy best guarantees the $O(1)$ average-case running time. (Recall that a good hash function not only attempts even distribution, but is also very simple and fast, as it must be invoked many times.)

2. Requirements

1. Setinterface

The Set interface has been provided for you. It contains all of the operations that your hash table classes must implement.

2. QuadProbeHashTable class

To the `assignment10` package add a `QuadProbeHashTable` class which implements the `Set` interface. The class must represent a hash table of `String` objects and must use quadratic probing to resolve collisions.

To get you started, the header for your `QuadProbeHashTable` class is given here.

```
public class QuadProbeHashTable implements Set<String>
```

Your `QuadProbeHashTable` class must contain the following constructor.

NAVIGATION

Home

■ My home

Site pages

My profile

My courses

Computer Science

Previous Semester

CS 1410-1-S13

CS_2100_S_13

CS2420-S13

Participants

General

Getting started;
Java review

Generic
programming;
Object Oriented
Programming

Algorithm
analysis; Data
Structures

Basic Sorting
Algorithms

Recursive Sorting
Algorithms

Linked Lists

Stacks and
Queues

```

/** Constructs a hash table of the given capacity that uses the l
 * specified by the given functor.
 */
public QuadProbeHashTable(int capacity, HashFunctor functor)

```

To allow your `QuadProbeHashTable` class to be used with any hash function, notice that the constructor accepts a function object containing the `int hash(String item)` method, and that method should be used whenever an item's hash code is needed. The `HashFunctor` interface is given below.

Finally, recall that the table size should be a prime number and that the table should be resized and rehashed when λ exceeds 0.5. (Hint: If the capacity given in the constructor is not prime, use the next largest prime number as the table size.)

3. ChainingHashTable class

To the `assignment10` package add a `ChainingHashTable` class which implements the `Set` interface. The class must represent a hash table of `String` objects and must use separate chaining to resolve collisions. You do not need to implement rehashing for this hash table, since it will never be full, however, if you don't implement rehashing, performance may suffer, depending on the initial size of the table.

The header for `ChainingHashTable` is similar to that given above.

`ChainingHashTable` must also contain a constructor `public ChainingHashTable(int capacity, HashFunctor functor)`.

Since `ChainingHashTable` will require an array of `LinkedList<String>`, which is a generic type, Java will give a warning when initializing it. We must specifically cast a non-generic array to a generic one (just like in assignment 3). Use the following as a guideline for declaring your generic array:

Declare the array in your class:

```
private LinkedList<String>[] storage;
```

Initialize the array in your constructor:

```
storage = (LinkedList<String>[]) new LinkedList[capacity];
```

The above line will give a warning, but it is safe to add

Trees


Graphs

Spring Break!

Hash Tables

 [Lab 7](#)


 [Slides](#)

 [Linear probing applet](#)

 [Review slides](#)

 [clicker solutions](#)

 [Assignment 10](#)

 [Analysis Document](#)

 [Assignment 10 files](#)

 [Exam 2](#)

Binary Heaps

File Compression

Comprehensive

Project;

Multithreading

Wrap Up

Final Exam and Review

29 April - 5 May

SETTINGS

Assignment administration
Submission

Course administration

4. HashFunctorinterface

The HashFunctor interface has been provided for you and serves as a guide for how to define a functor that contains a hashing function for `String` items (i.e., the hash method).

For example, to define a really bad hash function, one might write the following class.

```
public class ReallyBadHashFunctor implements HashFunctor {  
    public int hash(String item) {  
        return 0;  
    }  
}
```

You must provide three functors, each containing different hashing functions for `String` items.

- Create one hashing function that you think is bad (although not as bad as the one above). You should expect that it will result in many collisions. Put the hashing function in the class `BadHashFunctor`.
- Create one hashing function that you think is mediocre. You should expect that it will result in fewer collisions than the bad one. Put the hashing function in the class `MediocreHashFunctor`.
- Create one hashing function that you think is good. You should expect that it will result in few or no collisions. Put the hashing function in the class `GoodHashFunctor`. Do not use `String`'s built-in `hashCode` method, you must implement your own - although you can implement the same algorithm that it uses (you just have to write it yourself). Consult the web for `String` hash function ideas.

Each of your functor classes `BadHashFunctor`, `MediocreHashFunctor`, and `GoodHashFunctor` must implement the `HashFunctor` interface and must be in the `assignment10` package.

Create your own tests and submit them with your program.

3. When preliminary coding is complete and your program compiles without error or warning, test the program thoroughly and systematically.

Your code should be well-commented (Javadoc comments are recommended) and

formatted such that it is clear and easy to read. Be sure to put the names of both programming partners in the header comment of each file.

Zip your source code files (.java only) and **upload the zip file here by 5p on March 29**. Please submit just one solution per pair.

4. Analysis Document (must be written and submitted by each programming partner) **due March 29 at 5p**

Due date:	Friday, 29 March 2013, 5:05 PM
------------------	--------------------------------

Submission feedback



Paymon Saebi

Monday, 29 April 2013, 9:16 AM

Grade: 90.00 / 100.00

- Please see the attached file for grading detail and report feedback.
- Unfortunately -5 standard for not uploading a PDF



Barsketis.pdf



Barsketis.txt

Submission



Anaylisis10.docx



Assignment10Final.zip

No further submissions are allowed.

--	--