A picture containing diagram

Description automatically generated

Data Structures and Algorithms II

## Assignment Guidelines

Read the following instructions carefully before you start the assignment. If you do not understand any of them, ask your lecturer.

* The assignment coversheet should be the first sheet in your assignment. Moreover, the coversheet should be fully completed with all the necessary details.
* All text\code must be properly referenced. In the absence of proper referencing, the assignment will be regarded as plagiarised.
* Copying is strictly prohibited and will be penalized in line with the College’s disciplinary procedures.
* When the deadline specified by your lecturer is due, you shall hand all the required deliverables as explained in class.
* You are also required to submit your assignment to the relevant plagiarism detection service by the same deadline. If necessary, your lecturer will forward you details in order to submit your assignment to this service.
* The lecturer may hold a post-submission interview. Attendance to such interview is mandatory. Moreover, marks assigned to the criteria may be affected by the interview performance.
* **All work that has been carried out, must be written down and included within the assignment as evidence. No marks will be awarded for work that is not presented.**
* The deadline for this assignment is Monday 17th January 2022.

# Section 1

Hash Tables and Perfect Hashing (AA3.2, 7 marks)

### Scenario:

A new efficient implementation of a hash table is required. Research and Analyse Perfect Hashing.

### Task:

1. Research and prepare a write up (about 1 page) on the following topics: (3 marks)

Hash Tables,

Collision Resolution,

Chaining and Open Addressing.

Include at least 1 academic reference for this subtask.

1. Research about Perfect hashing, and prepare a write up (about 2 pages) including the following information: (3 marks)

What is Perfect hashing and how it avoids collision resolution,

Dynamic Perfect hashing,

Load factor and Dynamic Perfect hashing.

Include at least 2 academic references for this subtask.

1. Research and write about Universal families of Hash Functions. (1 mark)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grading guideline information | | | | |
| AA3.2 | Inadequate Work 0 marks |  | Superior Work 7 marks | Score Achievement |
| Answer the questions above. | Answer is poor and poorly written or partly plagiarised. |  | All topics discussed in detail, with proper explanation of all the topics required. Properly referenced and paraphrased sources. |  |

# Section 2

Extend existing Data Structures to implement a Custom Hashtable (AA1.4, 7 marks)

### Task:

Create a Hash table interface (3 marks) with Key and Value generic types and the following method signatures:

bool Insert(Key key, Value value);

bool Update(Key key, Value newValue);

Value Search(Key key);

bool Delete(Key key);

int GetCollisions( );

int Count( ); and

double GetLoadFactor( )

Create a struct of type Bucket that stores a Key and Value generic types (2 marks). You can include additional custom information in the struct, if you need.

Create a Custom hash table that implements the Hash table interface (0.5 marks).

Implement the following in your hashtable class:

Add an array of type Bucket to store the data (0.5 marks);

Implement the Count method that returns the number of key value pairs currently stored in the hash table (starting with 0) (0.5 marks); and

Implement the GetLoadFactor method that returns the load factor of the hashtable (0.5 marks).

Note: Since in this assignment your Custom hashtable will rehash whenever a collision occurs instead of other more typical collision resolution strategies.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grading guideline information | | | | |
| AA1.4 | Inadequate Work 0 marks |  | Superior Work 7 marks | Score Achievement |
| Answer the questions above. | No implementation or implementation does not work. |  | Well explained, working implementation. |  |

# Section 3:

Research how to extend and apply algorithms to your solution (AA4.2, 7 marks)

### Task:

Implement the remaining operations in the hash table described within this section.

1. The Insert operation takes a key and a value as parameters and returns a Boolean that indicates whether the Insert operation was successful. The return type will be used in Section 4 of the assignment;
2. The Update operation takes a key and a value as parameters and returns a Boolean that indicates whether the Update operation was successful. The return type will be used in Section 4 of the assignment;
3. The Search operation takes a key as a parameter and return the value associated with that parameter; and
4. The Delete operation takes a key as a parameter and returns a Boolean that indicates whether the Delete operation was successful. The return type will be used in Section 4 of the assignment.

For the purposes of this section, the method should always be successful and return value for the Insert, Update and Delete operations should always be true. These return values will be used in Section 5.

Your hashtable implementation must obtain a random hash function from a universal family of hash functions.

Your hashtable should have an integer field, called collisionCount. The collisionCount should count the number of collisions found since the last resizing event. When a new hashtable is instantiated or the hashtable is resized, the collisionCount should be set to 0.

P.T.O.

When inserting a new element, if a collision occurs, then:

If the load factor is larger than 0.7, double the hashtable size and select a new hash function at random from the universal family of hash functions. Set the collisionCount to 0;

Else if the collisionCount > 2, double the hashtable size and select a new hash function at random from the universal family of hash functions. Set the collisionCount to 0;

Else, rehash the elements of the hashtable, keeping the same size but selecting a new hash function at random from the universal family of hash functions. Increment the collisionCount.

Note that rehashing may cause further collisions, in which case, the same procedure as above applies.

Based on the research carried out in section 1, implement the custom hash table as described above using an appropriate universal family of hashing functions. Include screenshots of your code.

Note: It is important that you implement a hash table that can accept key/value pairs. The key can be of type integer, while the value can either be of type object or a generic type. Failure to implement a hash table can cause issues in other tasks.

Hint: Use a good design to allow your implementation to be extendable for the following sections.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grading guideline information | | | | |
| AA4.2 | Inadequate Work 0 marks | Inferior Work 2 marks | Superior Work 7 marks | Score Achievement |
| Answer the questions above. | No implementation or implementation does not work. | A hash set is implemented instead of a hash table. | Well explained, working implementation. |  |

# Section 4

Apply hashing algorithms to your Hashtable (AA2.3, 7 marks)

### Task:

Research and implement a universal family of hashing algorithms that can be used by integers in your hashtable. Use this universal family of hashing functions in your hashtable (4 marks).

Research and implement a hashing algorithm that can be used by strings in your hashtable. Use this universal family of hashing functions in your hashtable (3 marks).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grading guideline information | | | | |
| AA2.3 | Inadequate Work 0 marks | Inferior Work 4 marks | Superior Work 7 marks | Score Achievement |
| Answer the questions above. | No implementation or implementation does not work. | Only the integer hashing function is implemented. | Well explained, working implementation. |  |

# Section 5:

Design and implement algorithms (SE3.3, 10 marks)

### Scenario:

The implementation will be used in a highly parallel environment, with multiple threads but low probability of collision. In the task below, you will update your implementation of the hashtable that implements optimistic concurrency.

### Task:

Create either a wrapper class around your implementation of the hashtable or a subclass of your hashtable.

* Add a field of type ReaderWriterLockSlim to your hashtable. Use LockRecursionPolicy.SupportsRecursion in the constructor (1 mark).
* Whenever a read operation is carried out, obtain a ReadLock. Make sure to release the ReadLock after you are ready (1 mark)
* Whenever a rehash is made in your hashtable (whether or not a resize is required), you need to enter a WriteLock. Make sure that you always (by using appropriate code) release the WriteLock after the rehash and that you adjust the collisionCount while holding the WriteLock (1 mark).
* Whenever a call is made to an Insert, Update or Delete operation, before applying the Insert, Update or Delete (3 marks):
  + Obtain a ReadLock (or an Upgradeable Read Lock for the Insert method); and
  + Lookup and store the value in bucket where you would have to Insert, Update or Delete.

P.T.O.

* After the previous step, for the Update and Delete operations and for the Insert operation if the Insert operation does NOT require a rehash due to a collision, use the Interlocked.CompareExchange method to update value the bucket, as required by the operation, using the value looked up earlier as the comparand (1 mark).
* If the return value of the Interlocked.CompareExchange is equal to the comparand, then the operation has worked and you can return true (1 mark).
* If the return value of the Interlocked.CompareExchange is different, the required operation and not been carried out and a false is returned to inform the user (1 mark).
* In the case that a rehash is required, the Upgradeable Read Lock should be updated to a Write Lock. In this case, the Insert operation should always work and return true (1 mark).

Show screenshots of your code.

A sample pseudo code for the Insert Operation is described below.

Note that depending on your design, your implementation may be different. This is permissible if the goal of optimistic concurrency is achieved properly.

IMPORTANT NOTE: The CompareExchange does not use the overloaded Equals operation for objects. Instead, it only checks for reference equality! Bear in mind, that if you update the fields only, the object reference does not change.

// updated insert method

bool Insert(Key key, Value value) {

// Obtain an upgradeable read lock

rwls.EnterUpgradeableReadLock();

try {

// A new search operation that returns the

// Bucket for the given key instead of the Value

// b can be null

Bucket b = SearchBucket(key);

// Check whether a collision has happened

If ( b != null && ! b.IsEmpty ) {   
 // a collision has happened

// obtain a Write lock and rehash as normal

// Design and implement your own code here

} else { // no collision has occurred

// create a new instance of the bucket to be

// replace the old bucket

Bucket newBucket = new Bucket(key, value);

Bucket oldBucket = Interlocked.CompareExchange(ref b, newBucket, b);

If (Object.ReferenceEquals(oldBucket, b) {

Return true; // Insert is successful!

} else {

Return false; // Insert did not occur

}

}

} // end try

finally { // release locks being held

rwls.ExitUpgradeableReadLock( );

}

} // end method

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grading guideline information | | | | |
| SE3.3 | Inadequate Work 0 marks | Inferior Work 3 marks | Superior Work 10 marks | Score Achievement |
| Answer the questions above. | No implementation or implementation does not work. |  | Well explained, working implementation. |  |

# Section 6:

Illustrate how an algorithm works for a given problem (KU4.1, 5 marks)

Implement unit tests to show that the implemented hash-table works properly, as follows:

Implement a unit test for the Select method (1 mark).

Implement a unit test for the Insert method (1 mark).

Implement a unit test for the Update method (1 mark).

Implement a unit test for the Delete method (1 mark).

Give screenshots of your code and unit test results (1 mark).

Note: Unit tests must be correctly implemented and pass to obtain the relevant mark.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grading guideline information | | | | |
| KU4.1 | Inadequate Work 0 marks |  | Superior Work 5 marks | Score Achievement |
| Answer the questions above. | No implementation or implementation does not work. |  | All unit tests are sound and work properly. |  |

# Section 7:

Evaluate the correctness of an implemented solution to a given problem (SE4.3, 10 marks)

Implement a *data-driven* unit tests to show that the implemented hash-table works properly in a multi-threaded environment. The data can be generated randomly and the tests carried out must check that the implementation works well in the multi-thread environment).

Create a method that generates data to be used for the hashtable (2 marks).

Create a method that uses the selected data to insert the data into the hashtable using in parallel (4 marks)

Verify that the hashtable contains all the data that was inserted (1 mark).

Show screenshots of your testing code and test results (1 mark).

The unit test passes (2 marks).

Note: Unit tests must be correctly implemented and pass to obtain the relevant mark.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grading guideline information | | | | |
| SE4.3 | Inadequate Work 0 marks |  | Superior Work 10 marks | Score Achievement |
| Answer the questions above. | No implementation or implementation does not work. |  | Unit tests is and works properly as described above. |  |

Online multiple choice

KU1.1 is assessed through online multiple choice on Moodle.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grading guideline information | | | | |
| KU1.1 | Inadequate Work 0 marks |  | Superior Work 5 marks | Score Achievement |
| Answer questions on the Moodle multiple choice section. |  |  |  |  |

KU1.2 is assessed through online multiple choice on Moodle.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grading guideline information | | | | |
| KU1.2 | Inadequate Work 0 marks |  | Superior Work 5 marks | Score Achievement |
| Answer questions on the Moodle multiple choice section. |  |  |  |  |

KU2.1 is assessed through online multiple choice on Moodle.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grading guideline information | | | | |
| KU2.1 | Inadequate Work 0 marks |  | Superior Work 5 marks | Score Achievement |
| Answer questions on the Moodle multiple choice section. |  |  |  |  |