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 Sunday 15th January 2023.

# Section 1

Open-Addressing and Chaining in Hash Tables (AA3.2, 7 marks)

### Scenario:

A hash table implementation that uses chaining is required. Research and Analyse collision resolution strategies using open addressing and chaining.

### 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 how the load factor affects hash table performance. Produce a write up (about 2 pages) including the following information:

The impact of the Load Factor for *at least* 2 different open-addressing collision-resolution strategies (1 mark);

The impact of the Load Factor for the chaining collision resolution strategy (0.5 marks); and

The difference in speed and time performance of open-addressing and chaining collision resolution strategies. A good level of detail is required to compare the number of operations required and the extra space required in both strategies. (2.5 marks)

Include at least 2 academic references for this subtask.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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:

// The Insert operation adds a new key, value pair to the hash table

// If the key already exists, the Insert operation fails

// The operation returns true if the key, value pair is successfully added

// The operation returns false if the operation fails

// If the operation fails, no changes are made to the hash table

bool Insert(Key key, Value value);

// The Update operation updates the value of an existing key, value pair

// If the key does not already exist in the hash table, the operation fails

// If the key exists in the hash table, the value for the key, value pair that is

// identified by the key is replaced by the new value.

// The operation returns true if the operation succeeds and false if it fails

// If the operation fails, no changes are made to the hash table

bool Update(Key key, Value newValue);

// The Search operation will return the value associated with the key

// If the key is not found in the hash table, an Exception should be thrown

// The Search operation should never make changes in the hash table

Value Search(Key key);

// The Delete operation will remove the key, value pair associated with the key

// If the key is not found, the operation fails and nothing is removed

// The Delete will return true if the key, value pair is successfully removed and

// it will return a false otherwise

// If the operation fails, no changes are made to the hash table

bool Delete(Key key);

// Return the number of key, value pairs stored within the hash table

int Count( ); and

// Returns the load factor of the hash table

double GetLoadFactor( )

Create a class, as required to store Key and Value generic types (2 marks). You can include additional custom information in the class to meet the requirements of your implementation.

Create a Custom hash table that uses the chaining collision resolution strategy and implements the Hash table interface. The implementation should:

Contain an array that stores a linked list of key, value pairs stored in the implemented class. (1 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).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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. |  | Working implementation. Good programming structure and well commented. |  |

# 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 as described within section 2.

1. The Insert operation (2.5 marks);
2. The Update operation (1 mark);
3. The Search operation (1 mark); and
4. The Delete operation (2.5 marks).

Your hashtable should start with an underlying array of size 16. The hashtable should rehash and double in size whenever the load factor reaches 0.9.

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 a hashing algorithm that can be used in a Universal Family of hash functions and is able to use the GetHashCode() method from the class Object.

Implement your own Universal Family of hash functions that can be used to obtain a random hashing algorithm to be used in your hashtable. (3 marks)

Research and implement a universal family of hashing algorithms that can be used to obtain a random hashing function to be used for integer keys in your hashtable. (2 marks)

Update your hash table to allow for dependency injection of the Universal Family of hash functions to be used. Use in-line comments to explain how. (2 marks)

Hint: You can use the Strategy pattern of the Factory pattern to satisfy the requirements.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grading guideline information | | | | |
| AA2.3 | Inadequate Work 0 marks | Inferior Work 3 marks | Superior Work 7 marks | Score Achievement |
| Answer the questions above. | No implementation or implementation does not work. | Implementations carried out, but not used by the hashtable. | Well researched and well explained, working implementation, applied to the hashtable. |  |

# 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 hash table that implements optimistic concurrency.

### Task:

Create a copy of your implementations for this section so that you do not modify the work carried out for the previous tasks.

Note: For the methods below to work, you need to make sure that the key and values are reference types or primitive types.

You will need to update the counters used to store the number of elements as follows (1 mark):

* Add the volatile keyword to each of the counters;
* Use Interlocked.Increment and Interlocked.Decrement to increment or decrement the counters, instead of the usual code; and
* If necessary, use the Volatile.Read to read from the counters (this is useful if there are multiple read/write operations within the same method.

Note: You may wish to either update the existing code or create a class that turns the integer into a type that has atomic operations.

Replace the original Linked List implementation with a custom Singly Linked List that:

* Has a ContainsKey operation which returns true whenever the key being searched appears in the Singly Linked List (0.5 marks);
* Has an InsertFirst operation (used for the hash table Insert operation). The Insert operation will work as follows (4 marks):
  + Store a reference to the head of the linked list;
  + Create a linked list node with the key, value pair to be inserted. The next node of the new node should point towards the head of the linked list;
  + Check whether the key is contained within the linked list;
  + If the key is already stored within the linked list, the InsertFirst and the Insert operation will fail;
  + If the key is not already stored within the linked list, use the Interlocked.CompareExchange to update the head of the linked list with the newly created node;
  + If the CompareExchange fails, the InsertFirst and the Insert operation will fail, otherwise the Insert First operation has succeeded; and
  + Increment the counters as appropriate. (Note that there is a moment where the element is added but the counters are not yet incremented).
* Has an UpdateValue operation (used for the hash table Update operation). The Update operation will work as follows (1.5 marks):
  + Check whether the key is contained within the linked list. Obtain and store the value that belongs to the key using the Search operation;
  + If the key is not contained within the linked list, the UpdateValue and the Update operation will fail;
  + Otherwise, find the node in the linked list with the matching key;
  + Use the Interlocked.CompareExchange to replace the old value, while comparing it with the value obtained through the Search operation, with the new value.
  + If the CompareExchange fails, the UpdateValue and the Update operations will fail, otherwise they will succeed.
* Has a DeleteKey operation (used for the hash table Delete operation). The Delete operation will work as follows (3 marks):
  + Use a lock on the DeleteKey operation. Only a single thread may execute the DeleteKey operation to avoid interactions between threads that can cause issues. Make sure to release the lock after the method has terminated.
  + Check whether the key is contained within the linked list. Find and store the reference to the node with the key being searched. The reference could be the link to the head of the list (if the key is the first item in the list) or the using the next reference of the previous node;
  + If the key is not contained within the linked list, the DeleteKey and the Delete operation will fail;
  + Otherwise, find the reference to the node after the one to be deleted and use the Interlocked.CompareExchange to replace the reference to the node with the key being deleted, while comparing with the original reference found, with the reference to the node after the node being deleted.
  + If the CompareExchange fails, the DeleteKey and the Delete operations will fail, otherwise they will succeed.
  + Decrement the Count counter and increment the Collision counter as appropriate. (Note that there is a moment where the element is added but the counters are not yet incremented/decremented).

Note: Since there are moments when the Count and Collision counters are out of synchronisation, these values cannot be used to reliably make important decisions (such as whether the hash table is empty).

Note: The above methods are assuming that values cannot be added to the middle of the linked list. Adding this functionality can cause issues to the operations.

Note: The CompareExchange and the implementation described above can be vulnerable to the ABA problem. More information and related work can be found in Section 7.

IMPORTANT NOTE: The CompareExchange does not use the overloaded Equals operation for objects. Instead, it only checks for reference equality!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 structured, well commented, working implementation. |  |

# Section 6:

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

Implement unit tests to show that the hash-table implemented in Section 3 works properly:

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)

Describe the ABA problem, how it can affect the implementation given in Section 5 and describe solutions that can be applied to solve this problem (2 marks).

Implement a *data-driven* unit tests to test the ABA problem and attempt to prove that issues related to the ABA problem exist when working in a multi-threaded environment. The data can be generated randomly and the tests carried out must attempt to uncover issues related to the ABA problem in a multi-thread environment.

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

Create a method that uses the selected data to insert and delete the data in the hash table using a parallel method to try and force an ABA issue (3 marks).

Verify whether the hash table contains all the expected data (1 mark).

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

Describe your observations and draw conclusions from this observation (1 mark).

Note: Unit tests must be correctly implemented 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 implemented properly and can detect issues in the implementation 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. |  |  |  |  |