INTI International College Penang **School of Engineering and Technology**

3+0 Bachelor of Science (Hons) in Computer Science, in collaboration with Coventry University, UK

3+0 Bachelor of Science (Hons) in Computing, in collaboration with Coventry University, UK

Coursework Cover Sheet

Section A - To be completed by the student

Full Name: Chan Khai Shen	
CU Student ID Number: 11323943	
Semester: April 2022	
Lecturer: **Koo Lee Chun**	
Module Code and Title: **5003CEM Advanced Algorithms**	
Assignment No. / Title: **Courswork**	% of Module Mark **67%**
Hand out date: **28 April 2022**	Due date: **3 July 2022**
Penalties: No late work will be accepted. If you are unable to submit coursework on time due to extenuating circumstances you may be eligible for an extension. Please consult the lecturer.	
Declaration: I the undersigned confirm that I have read and agree to abide by the University regulations on plagiarism and cheating and Faculty coursework policies and procedures. I confirm that this piece of work is my own. I consent to appropriate storage of my work for plagiarism checking. Signature(s): -------------------------------------	

Section B - To be completed by the module leader

Intended learning outcomes assessed by this work:

LO2: Design and implement algorithms and data structures for novel problems.
LO3: Specify and implement methods to estimate solutions to intractable problems
LO5: Design and implement a basic concurrent application

Marking scheme	Max	Mark
Total	66	

Lecturer’s Feedback
Internal Moderator’s Feedback

Question 1

Program output

Add a new task

The user can add a task into the hash table by selecting option 1. Figure 1 shows the insertion of a task “Arrange table” with id 209.

```
1) Add a new task
2) Retrieve a task
3) Exit

Choose an option: 1
Task id: 209
Task description: Arrange table
Task is inserted.
```

Figure 1. Screenshot of add a new task (Question 1)

The user is not allowed to reuse the same id to ensure accuracy when retrieving the task. Figure 2 shows the error message when id 209 is reused.

```
Choose an option: 1
Task id: 209
Task description: Wipe window
Duplicate task id.
```

Figure 2. Screenshot of duplicate task id (Question 1)

Retrieve a task

The user can retrieve the task description based on task id by selecting option 2. Figure 3 shows the retrieval of task with id 209.

```
Choose an option: 2
Task id: 209
Task is: Arrange table
```

Figure 3. Screenshot of retrieve a task (Question 1)

When the task is not found, there is an error message. Figure 4 shows the error message when a non-existent id 214 is specified.

```
Choose an option: 2
Task id: 214
Task is not found.
```

Figure 4. Screenshot of task is not found (Question 1)

Exit

The user can display the hash table and exit the program by selecting option 3. Figure 5 shows the result when option 3 is selected.

```
Choose an option: 3

Summary:
Table[0]
Table[1]
Table[2]-->674(Clean whiteboard)
Table[3]
Table[4]
Table[5]
Table[6]-->209(Arrange table)-->349(Wipe window)
```

Figure 5. Screenshot of exit (Question 1)

Discussion on the solution

The solution used chaining as the collision handling technique. If collision occurs, the data will be appended to the end of the list at the slot. Using the example in Figure 5, key 209 hashes to slot 6. The task with key 209, “Arrange table”, is appended to the list at slot 6. After that, a task with key 349 comes in. Key 349 also hashes to slot 6. Collision occurs. Based on chaining collision handling technique, the task with key 349, “Wipe window”, is also appended to the list at slot 6, but at the back of “Arrange table”.

Weakness of the solution

The weakness of this solution is chaining takes up extra memory space for the list. At the same time, there may be more unused slots in the hash table, since the keys that have collision will not be stored at the other available slots.

Question 2

Program output

Figure 6 is the report of runtime results for the four trees.

```
Specify number of random numbers: 1000

Report
Time used to search number 2000 for 100000 times:
BST sorted: 6.883930399999372
AVL random order: 0.09412479999991774
AVL sorted: 0.08635909999975411
BST random order: 0.09045719999994617
```

Figure 6. Screenshot of runtime result (Question 2)

Discussion on the solution

The runtime for normal binary search tree with sorted input is 6.88. The runtimes for AVL tree with random input, and AVL tree with sorted input and normal binary search tree with random input, are 0.0941, 0.0864 and 0.0905 respectively. The runtimes for these three trees are nearly equal, whereas the runtime for normal binary search tree with sorted input is much longer than the three other trees.

The theoretical time complexity for searching a number in a binary search tree with height h is equal to O(h). For a normal binary search tree with n sorted inputs, the height of the tree is almost equal to n. Thus, the time complexity is O(n). On the other hand, the height of a normal binary search tree with n random inputs, is approximately log n. Thus, the time complexity is O(log n).

The height of an AVL tree is always approximately log n because AVL tree is maintained balanced during insertion. So, the time complexity for searching a number in an AVL tree, for both with n sorted inputs and n random inputs, is O(log n).

In conclusion, the runtime results are tally with the theoretical time complexity, whereby the normal binary search tree with sorted input, which time complexity is O(n), has much longer runtime result than the normal binary tree with random input, the AVL tree with sorted input and the AVL tree with random input, which time complexity is O(log n).

Question 3

<u>Program output</u>

<u>Adjacent places</u>

When two adjacent places are inserted, the distance between them is displayed. Figure 7 shows the result when the inputs are A and E.

```
Places: A, B, C, D, E, F
Select any two places.
Place 1: A
Place 2: E

Report:
The distance between A and E is 48m.
```

Figure 7: Screenshot of result of execution (Question 3, Adjacent places)

<u>Non-adjacent places</u>

When the places inserted are not adjacent to each other, the route between them is displayed, together with the total distance and distances between each point on the journey. Figure 8 shows the output when the inputs are F and B.

```
Places: A, B, C, D, E, F
Select any two places.
Place 1: F
Place 2: B

Report:
Route from F to B:
F -18m-> D -22m-> A -32m-> B
Total distance: 72m
```

Figure 8. Screenshot of result of execution (Question 3, Non-adjacent places)

<u>Discussion on the solution</u>

The solution used adjacent list to represent the edges of the graph. Each vertex stores a list of neighbouring vertices with the associated distance.

The solution used breadth first search to find the path from one vertex to another. When the user inserts two places, say place1 and place2, the solution uses breadth first search to traverse from place1 to all places in the graph and stops when it reaches place2.

<u>Weakness</u>

One weakness of the solution is the distance found by the solution is not necessarily the shortest distance. For example, when the inputs are B and C, the program finds the direct route B->C first, then it reports the distance is 70m. However, the shortest distance is 40m via route B->A->C.

Figure 9 shows the program output when the input is B and C.

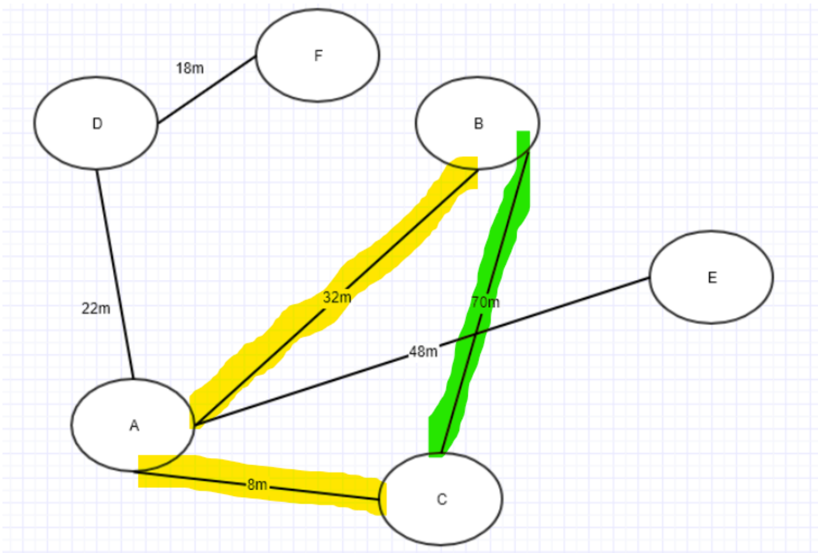
```
Places: A, B, C, D, E, F
Select any two places.
Place 1: B
Place 2: C

Report:
The distance between B and C is 70m.
```

Figure 9. Screenshot of result of execution (Question 3, Input B and C)

Graph 1 illustrates the routes from B to C. The route highlighted in green is the direct route B->C, which distance is 70m, whereas the route highlighted in yellow is the shortest route B->A->C, which distance is 40m.

Graph 1. Graph to illustrate different routes from B to C



Question 4

Program output

The solution for Question 4 is run twice to observe the difference. Figure 10 and Figure 11 shows the output of each execution respectively.

```
Enter number 1: 10
Enter number 2: 2

10 - 2 = 8
10 * 2 = 20
10 / 2 = 5
10 + 2 = 12
```

Figure 10. Screenshot of result of first execution (Question 4)

```
Enter number 1: 10
Enter number 2: 2

10 - 2 = 8
10 / 2 = 5
10 + 2 = 12
10 * 2 = 20
```

Figure 11. Screenshot of second execution (Question 4)

Discussion on the solution

Comparing the two results, the sequences of add, subtract, multiply and divide operations are different. This is because each operation runs as a separate thread and the threads are created almost at the same time. The order of execution of threads is out of developer’s control so the sequence of the operation can be different each time the program runs.

Reflection

From this coursework, I have learned the implementation of hash table and collision handling, implementation of binary search tree and AVL tree, measuring execution time of code, implementation of graph and breadth first search, and concurrent programming.

References

Geeksforgeeks. (No date). & [princiraj1992]., [rathbhupendra]., [Akanksha_Rai]., [sohamshinde04]., [nocturnalstoryteller]., [rdtank]., [kaiwenzheng644]., & [hardikkoriintern]. (Eds.) (2022, June 17). *AVL tree* | *Set 1 (insertion)* [Online]. Geeksforgeeks. Available at: https://www.geeksforgeeks.org/avl-tree-set-1-insertion/ (Accessed 24 June 2022)

Geeksforgeeks. (No date). & [ishankhandelwals]. (Eds.) (2022, June 22). Python programme for breadth first search or BFS for a graph [Online]. Geeksforgeeks. Available at: https://www.geeksforgeeks.org/python-program-for-breadth-first-search-or-bfs-for-a-graph/ (Accessed 28 June 2022)

[ss6437p]. (No date). & [simmytarika5]., [surinderdawra388]., [adnanirshad158]., &[abhishek0719kadiyan]. (Eds.) (2021, July 16). *How to check the execution time of Python script* [Online]. Geeksforgeeks. Available at: https://www.geeksforgeeks.org/how-to-check-the-execution-time-of-python-script/ (Accessed 24 June 2022)

Techie Delight. (No date). *Graph implementation in Python* [Online]. Techie Delight. Available at: https://www.techiedelight.com/graph-implementation-python/ (Accessed 28 June 2022)

Tutorialspoint. (No date). *Concurrency in Python – Quick guide* [Online]. Tutorialspoint. Available at: https://www.tutorialspoint.com/concurrency_in_python/concurrency_in_python_quick_guide.htm (Accessed 1 July 2022)

Appendix:

Solution for Question 1:

```python
class HashNode:
    def __init__(self, id, description):
        self.id = id
        self.description = description

class HashTable:
    def __init__(self, maxsize):
        self.__MAXSIZE = maxsize
        self.__hashTable = []
        self.__hashTable = [[] for _ in range(maxsize)]

    def __hashing(self, id):
        """ Method to convert the id to a hash value.
            Returns integer.
        """
        return id % self.__MAXSIZE  # 7 11

    def insertTask(self, id, description):
        """ Method to insert task into index given and display appropriate 
message.
            No return value.
        """
        index = self.__hashing(id)
        for hashNode in self.__hashTable[index]:
            if hashNode.id == id:
                print("Duplicate task id.")
                return
        self.__hashTable[index].append(HashNode(id, description))
        print("Task is inserted.")

    def retrieveTask(self, id):
        """ Method to display task description based on id.
            No return value.
        """
        index = self.__hashing(id)
        for hashNode in self.__hashTable[index]:
            if hashNode.id == id:
                print(f'Task is: {hashNode.description}', end='\n')
                return
        print("Task is not found.")

    def displayHashTable(self):
        """ Method to display the hash table.
            No return value. """
        for index in range(len(self.__hashTable)):
            print(f'Table[{index}]', end='')
            for hashNode in self.__hashTable[index]:
                print(f'-->{hashNode.id}({hashNode.description})', end='')
            print()  # Next line


if __name__ == "__main__":
    hashTable = HashTable(7)

    print("1) Add a new task")
    print("2) Retrieve a task")
```

```python
print("3) Exit")

while True:

    option = input("\nChoose an option: ")

    # Add task
    if option == "1":
        id = int(input("Task id: "))
        description = input("Task description: ")
        hashTable.insertTask(id, description)

    # Retrieve task
    elif option == "2":
        id = int(input("Task id: "))
        hashTable.retrieveTask(id)

    # Exit
    elif option == "3":
        print("\nSummary:")
        hashTable.displayHashTable()
        break
```

Solution for Question 2:

```
import timeit
import random


class Node:
    def __init__(self, number):
        self.number = number
        self.leftChild = None
        self.rightChild = None


class BST:
    def __init__(self):
        self.__root = None

    def insertNumber(self, number):
        """ Method to insert a new node.
            No return value.
        """
        # Root is None
        if not self.__root:
            self.__root = Node(number)
            return

        # Root is not None
        current = self.__root
        parent = current
        while current:
            parent = current
            if current.number > number:
                current = current.leftChild
            elif current.number < number:
                current = current.rightChild
            else:
                return  # Duplicate entry
        if parent.number > number:
            parent.leftChild = Node(number)
        else:
            parent.rightChild = Node(number)

    def searchNumber(self, number):
        """ Method to search a number.
            Returns true if found, otherwise false.
        """
        current = self.__root
        while current:
            if current.number > number:
                current = current.leftChild
            elif current.number < number:
                current = current.rightChild
            else:
                return True  # Found
        return False  # Not found


# references: https://www.geeksforgeeks.org/avl-tree-set-1-insertion/
class AVL_Node:
    def __init__(self, number):
        self.number = number
        self.leftChild = None
```

```
        self.rightChild = None
        self.height = 1


class AVL:
    def __init__(self):
        self.__root = None

    def __getHeight(self, AVL_node):
        """ Method to get height of a node.
            Returns integer.
        """
        if not AVL_node:
            return 0
        return AVL_node.height

    def __leftRotate(self, AVL_node):
        """ Method to perform left rotation on a subtree.
            Returns the subtree new root.
        """
        rightChild = AVL_node.rightChild
        leftChildOfRightChild = rightChild.leftChild

        # Perform rotation
        rightChild.leftChild = AVL_node
        AVL_node.rightChild = leftChildOfRightChild

        # Update heights
        AVL_node.height = 1 + max(self.__getHeight(AVL_node.leftChild),
self.__getHeight(AVL_node.rightChild))
        rightChild.height = 1 + max(self.__getHeight(rightChild.leftChild),
self.__getHeight(rightChild.rightChild))

        return rightChild  # New root

    def __rightRotate(self, AVL_node):
        """ Method to perform right rotation on a subtree.
            Returns the subtree new root.
        """
        leftChild = AVL_node.leftChild
        rightChildOfLeftChild = leftChild.rightChild

        # Perform rotation
        leftChild.rightChild = AVL_node
        AVL_node.leftChild = rightChildOfLeftChild

        # Update heights
        AVL_node.height = 1 + max(self.__getHeight(AVL_node.leftChild),
self.__getHeight(AVL_node.rightChild))
        leftChild.height = 1 + max(self.__getHeight(leftChild.leftChild),
self.__getHeight(leftChild.rightChild))

        return leftChild  # New root

    def __insertRecursive(self, AVL_node, number):
        """ Method to insert a new node recursively.
            Returns the current node to the calling function.
        """
        # Normal BST insertion
        if not AVL_node:
            return AVL_Node(number)
```

```
        elif AVL_node.number > number:
            AVL_node.leftChild = self.__insertRecursive(AVL_node.leftChild,
number)
        elif AVL_node.number < number:
            AVL_node.rightChild =
self.__insertRecursive(AVL_node.rightChild, number)

        # Update the height
        AVL_node.height = 1 + max(self.__getHeight(AVL_node.leftChild),
self.__getHeight(AVL_node.rightChild))

        # Get the balance factor
        balanceFactor = self.__getHeight(AVL_node.leftChild) -
self.__getHeight(AVL_node.rightChild)

        # Node is balanced
        if balanceFactor in (-1, 0, 1):
            return AVL_node

        # Node is imbalance

        # Case 1: Left left imbalance
        if balanceFactor > 1 and AVL_node.leftChild.number > number:
            return self.__rightRotate(AVL_node)

        # Case 2: Right right imbalance
        if balanceFactor < -1 and AVL_node.rightChild.number < number:
            return self.__leftRotate(AVL_node)

        # Case 3: Left right imbalance
        if balanceFactor > 1 and AVL_node.leftChild.number < number:
            AVL_node.leftChild = self.__leftRotate(AVL_node.leftChild)
            return self.__rightRotate(AVL_node)

        # Case 4: Right left imbalance
        if balanceFactor < -1 and AVL_node.rightChild.number > number:
            AVL_node.rightChild = self.__rightRotate(AVL_node.rightChild)
            return self.__leftRotate(AVL_node)

    def insertNumber(self, number):
        """ Method to insert a new node.
            No return value.
        """
        self.__root = self.__insertRecursive(self.__root, number)

    def searchNumber(self, number):
        """ Method to search a number.
            Returns true if found, otherwise false.
        """
        current = self.__root
        while current:
            if current.number > number:
                current = current.leftChild
            elif current.number < number:
                current = current.rightChild
            else:
                return True   # Found
        return False   # Not found


if __name__ == "__main__":
```

```python
    n = int(input("Specify number of random numbers: "))

    # Create BST/AVL
    BST_randomOrder = BST()
    BST_sorted = BST()
    AVL_randomOrder = AVL()
    AVL_sorted = AVL()

    # Prepare list of numbers
    count = 1
    numberList = []
    while count <= n:
        randomNumber = random.randint(1, 1000)
        if randomNumber not in numberList:
            numberList.append(randomNumber)
            count += 1

    # Insert numbers
    # Random order
    for num in numberList:
        BST_randomOrder.insertNumber(num)
        AVL_randomOrder.insertNumber(num)
    # Sorted
    numberList.sort()
    for num in numberList:
        BST_sorted.insertNumber(num)
        AVL_sorted.insertNumber(num)

    print("\nReport")
    print("Time used to search number 2000 for 100000 times: ")

    # references: https://www.geeksforgeeks.org/how-to-check-the-execution-
time-of-python-script/
    # Search for 2000
    # BST sorted
    def execute_BST_sorted():
        BST_sorted.searchNumber(2000)
    print(f'BST sorted:
{timeit.timeit(stmt=execute_BST_sorted,number=100000)}', end='\n')

    # AVL random order
    def execute_AVL_random():
        AVL_randomOrder.searchNumber(2000)
    print(f'AVL random order:
{timeit.timeit(stmt=execute_AVL_random,number=100000)}', end='\n')

    # AVL sorted
    def execute_AVL_sorted():
        AVL_sorted.searchNumber(2000)
    print(f'AVL sorted:
{timeit.timeit(stmt=execute_AVL_sorted,number=100000)}', end='\n')

    # BST random order
    def execute_BST_random():
        BST_randomOrder.searchNumber(2000)
    print(f'BST random order:
{timeit.timeit(stmt=execute_BST_random,number=100000)}', end='\n')
```

Code for Question 3:

```
#references : https://www.techiedelight.com/graph-implementation-python/
class Graph:
    # Constructor for Graph
    def __init__(self, edgeList, placeList):

        # Initialize neighbourDictionary
        self.__neighbourDictionary = {}
        for place in placeList:
            self.__neighbourDictionary[place] = []

        # Set up a list of neighbouring places and distance for each place
        for (origin, destination, distance) in edgeList:
            self.__neighbourDictionary[origin].append((destination,
distance))
            self.__neighbourDictionary[destination].append((origin,
distance))

    def isNextToEachOther(self, place1, place2):
        """ Method to display the distance between two given places.
            Returns True if the two places are neighbours, otherwise
returns False.
        """
        for (neighbour, distance) in self.__neighbourDictionary[place1]:
            if neighbour == place2:
                print(f'The distance between {place1} and {place2} is
{distance}m.', end='')
                return True
        return False

    # references: https://www.geeksforgeeks.org/python-program-for-breadth-
first-search-or-bfs-for-a-graph/
    def isReachableViaOtherPlace(self, place1, place2):
        """ Method to determine whether two given places are reachable via
other
            places and display message to indicate the result.
            No return value.
        """

        # Performing breadth first search
        # Start from place1
        queue = [place1]
        visitedDictionary = {place1: {"Distance": 0, "Previous": None}}

        reachedPlace2 = False

        # Traverse through points in the graph
        while queue and not reachedPlace2:
            current = queue.pop(0)

            for (neighbour, distance) in
self.__neighbourDictionary[current]:
                if neighbour not in visitedDictionary:
                    queue.append(neighbour)
                    distanceFromPlace1 =
visitedDictionary[current]["Distance"] + distance
                    visitedDictionary[neighbour] = {"Distance":
distanceFromPlace1, "Previous": current}

                    # Stop when reached place2
                    if neighbour == place2:
```

```
                        reachedPlace2 = True
                        break

        # Display message
        if reachedPlace2:
            print(f'Route from {place1} to {place2}:', end='\n')

            # Make the list of passed-by places
            passedByList = [place2]
            current = place2
            while current != place1:
                previous = visitedDictionary[current]["Previous"]
                passedByList.append(previous)
                current = previous
            passedByList.reverse()

            # Calculate distances between each place
            accumulatedDistance = 0
            for passedByPlace in passedByList:
                if passedByPlace == place1:
                    print(f'{place1}', end='')
                else:
                    nowAccumulatedDistance = 
visitedDictionary[passedByPlace]["Distance"]
                    distance = nowAccumulatedDistance - accumulatedDistance
                    print(f' -{distance}m-> {passedByPlace}', end='')
                    accumulatedDistance = nowAccumulatedDistance

            print(f'\nTotal distance: {accumulatedDistance}m', end='')

        else:
            print(f'Cannot find route between {place1} and {place2}.', 
end='')


if __name__ == "__main__":
    # Create a Graph object
    edges = [
        ["A","D",22], ["A","B",32], ["A","E",48], ["A","C",8], 
["D","F",18], ["C","B",70]
    ]
    placeList = ["A","B","C","D","E","F"]
    graph = Graph(edges, placeList)

    # Ask user to input two places
    print("Places: A, B, C, D, E, F")
    print("Select any two places.")
    place1 = input("Place 1: ")
    place2 = input("Place 2: ")
    print("\nReport:")

    # Check whether the two places are next to each other
    if not graph.isNextToEachOther(place1, place2):
        # Check whether the two places is reachable via other places
        graph.isReachableViaOtherPlace(place1, place2)
```

Code for Question 4:

```
import _thread
import time


# Reference:
https://www.tutorialspoint.com/concurrency_in_python/concurrency_in_python_
quick_guide.htm
def calculate(number1, number2, operation):
    """ Function to perform the specified operation on the two numbers
given.
        Returns the result.
    """
    if operation == "+":
        return number1 + number2
    elif operation == "-":
        return number1 - number2
    elif operation == "*":
        return number1 * number2
    elif operation == "/":
        return number1 / number2


def print_result(number1, number2, operation):
    """ Function to call calculate() and print the result in proper
statement.
    """
    result = calculate(number1, number2, operation)
    print(f'\n{number1} {operation} {number2} = {int(result)}', end='')


# Get 2 numbers from the user
number_1 = int(input("Enter number 1: "))
number_2 = int(input("Enter number 2: "))

# Create one thread for each operation
_thread.start_new_thread(print_result, (number_1, number_2, "+",))
_thread.start_new_thread(print_result, (number_1, number_2, "-",))
_thread.start_new_thread(print_result, (number_1, number_2, "*",))
_thread.start_new_thread(print_result, (number_1, number_2, "/",))

# Delay the main process so that it finishes after all threads finished
time.sleep(10)
```