# CSCI 3901 Assignment 2

Fall 2023

## Problem 1

### buoyData:

| Type of test case | Property of test case | Property of input data | Expected Result |
|---|---|---|---|
| Input Validation | Empty data stream | A valid buoyId but empty data stream | false |
| Input Validation | Null data stream | A valid buoyId with null data stream | false |
| Input Validation | Empty buouId | buouId as “” empty string | false |
| Input Validation | Null buouId | buoyId as null | false |
| Input Validation | Data stream is badly formatted | Data stream with no header data. **Assuming that there will be header data** | false |
| Input Validation | Data stream is badly formatted | Data stream with header data but no records | false |
| Input Validation | Data stream is badly formatted | Data stream with empty lines | false |
| Input Validation | Data stream is badly formatted | Data stream with missing height information | false |
| Input Validation | Data stream is badly formatted | Data stream with missing Timestamp information | false |
| Input Validation | Data stream is badly formatted | Timestamp data and height data are not tab separated | false |
| Input Validation | Timestamp data is badly formatted | Timestamp is not in a standard format. **Choice: Timestamp should be in YYYY-MM-DD HH:MM:SS format** | false |
| Input Validation | Height data is badly formatted | Height is not in standard format. **Choice: Height should be in meters. For instance, 140 m** | false |

| | | | |
|---|---|---|---|
| Boundary Validation | Max data points | Data stream with maximum allowed data points entry (timestamp + height) in the system | If adding beyond the limit, return false else add the data and return true |
| Boundary Validation | Timestamp data | Timestamp within the range of start of the day and end of the day format | If within the boundary, return true else false |
| Boundary Validation | Height data | Height within the max and min height boundaries | If within the range, return true else false |
| | | | |
| Control flow Validation | Data in chronological order | If the timestamp of upper data point is lower than the timestamp of lower data point throughout the data stream | If data is in chronological order, return true else false |
| Control flow Validation | Consistent Data | If each data points are separated by a constant time gap (for an instance 10 minutes) | If data is consistent, return true else false |
| Control flow Validation | Duplication | If same data points are repeated | Return false |
| Control flow Validation | Duplication | Trying to call buoyData with a buouId for which the data has been already stored in the system | Return false |
| | | | |
| Data flow Validation | Read and store measurement data for a buoy with valid input | A valid buouId with a valid data stream | true (indicating successful data reading) and store the buoyData in the system for future use |
| Data flow Validation | Invalid data stream | A valid buoyId but invalid data stream | false |
| Data flow Validation | Invalid buoyId | An invalid buouId with a valid data stream | false |

## selectData:

| Type of test case | Property of test case | Property of input data | Expected Result |
|---|---|---|---|
| Input Validation | Empty start timestamp | Start time stamp passed as “” empty string | Null |
| Input Validation | Null start timestamp | Start time stamp passed as “” null | Null |
| Input Validation | Empty end timestamp | End time stamp passed as “” empty string | Null |
| Input Validation | Null end timestamp | End time stamp passed as null | Null |
| Input Validation | Bad formatting | Start timestamp is not in the standard format | Null |
| Input Validation | Bad formatting | End timestamp is not in the standard format | Null |
| | | | |
| Boundary Validation | Minimum start timestamp | If start time stamp is less than the first data point’s timestamp in the data object | Return WaveHeight object with records starting from first data point |
| Boundary Validation | Maximum start timestamp | If start time stamp is more than the last data point’s timestamp in data object | Null (no data to extract) |
| Boundary Validation | Minimum end timestamp | If end timestamp is less than the first data point’s timestamp in data object | Return null |
| Boundary Validation | Maximum end timestamp | If end timestamp is more than the last data point’s timestamp in data object | Return WaveHeight object with records till the last data point |
| | | | |
| Control Flow Validation | Start timestamp > end timestamp | If start timestamp exceeds the end timestamp | Return null |
| Control Flow Validation | Equal check | If start timestamp = end timestamp | Return WaveHeight object with record at the specified start and end timestamp |
| Control Flow Validation | Start timestamp < end timestamp | End time stamp is greater than start time | Return WaveHeight object with specified |

| | | | |
|---|---|---|---|
| | | stamp and data is available to extract between that range | start and end timestamps |
| | | | |
| Data flow validation | Invalid Data | Calling selectData without buoyData | Return null |
| Data flow validation | Valid input | Valid start timestamp and end timestamp | Return WaveHeight object with specified start and end timestamps. |
| Data flow validation | Empty object | Valid start and end timestamp but the data object is empty | Return null |
| Data flow validation | No data to extract | Valid start and end timestamps but there are no data available between the mentioned range | Return null |

## Cluster:

| Type of test case | Property of test case | Property of input data | Expected Result |
|---|---|---|---|
| Input Validation | Maximum clusters are null | Passing maximum clusters as null | Null |
| Input Validation | Maximum clusters are less than 0 | Passing maximum clusters as negative integers | Null |
| Input Validation | Buoy data is null | Valid maximum cluster with null buoy data | Null |
| Input Validation | Buoy data is empty | Valid maximum cluster with empty buoy data | Null |
| Input Validation | Non integer value | Passing maximum cluster as non-integer value | Null |
| Input Validation | Valid data | Valid max clusters and buoy data | Return set with 2 clusters |
| | | | |
| Boundary Validation | Minimum value for maximum clusters | Maximum clusters are equals to zero | Return Empty set |
| Boundary Validation | Maximum value for maximum clusters | Maximum clusters exceed the total clusters that can be formed from buoy data available | Return Set with the total number of clusters that can be formed from the available data |
| Boundary Validation | One less than the number of buoys | Maximum clusters parameter is set to one less than the number of buoys | Return Set with clustered data except of one buoy |
| | | | |
| Control Flow Validation | Buoys with unlike characteristics | If all buoys have different characteristics from one another | Return Set with max clusters defined in method parameter |
| Control Flow Validation | Similar Average wave height | If all buoys have similar average wave height and max clusters is more than 1 | Return a Set with 1 cluster |
| Control Flow Validation | Similar duration of big waves over time | If all buoys have similar duration of big waves over time and max clusters is more than 1 | Return a Set with 1 cluster |
| Control Flow Validation | Similar biggest waves | If all buoys have similar biggest waves and max clusters is | Return a Set with 1 cluster |

| | | | |
|---|---|---|---|
| | | more than 1 | |
| Control Flow Validation | Half clusters | If Half of the buoys are of one kind of similar characteristics and other half are with other kind of similar characteristics and max clusters are more than 2 | Return a Set with 2 clusters |
| Control Flow Validation | Odd one out | If one buoy is of other characteristics than all other buoys and max clusters is more than 2 | Return Set with 2 clusters (1 cluster of similar characteristics and other with that single buoy) |
| | | | |
| Data flow validation | Invalid Data | Calling cluster without calling buoyData | Return null |
| Data flow validation | Invalid Data | Calling cluster without calling selectData | Return null |
| Data flow Validation | Different buoy data | If no clusters can be formed from the given data | Return Set with max clusters passed (each cluster would have one buoy) |

# Problem 2

## kdTree

### Overview

The program is a collection of classes that we've put together to perform a process which includes following step:

1. Take the points or the coordinates as input and build a balanced kdTree or high dimension BST from those points.
2. The kdTree would have the functionality of searching for an element in the tree, adding an element to the tree, and finding the points between specified minimum and maximum range and it would also have a property to know the length or the size of the tree (basically total number of nodes in the tree).
3. The kdTree could also be printed for debugging purposes.

The goal is to build a data structure which would have the behavior and characteristics of a kdTree.

### Files and external data

All in all, there are 9 classes and 2 interfaces in the project:

1. Main.java:
    a. Contains the main method and basically holds the driver code.
    b. Creates the kdTree object and initializes the dimensions.
2. kdTree.java:
    a. This class implements two interfaces – Searchable (Searchable.java) and TreeDebug (TreeDebug.java).
    b. Holds the underlying logic for the entire implementation of the kdTree.
    c. Implements the following methods:
        i. Build – build the tree from given points.
        ii. Add – add a node to the tree.
        iii. Find – find a node in the tree.
        iv. Size – length or total number of nodes in the tree.
        v. FindInRange – find elements between specific ranges.
        vi. printTree – print the entire tree.
    d. Also has some helper functions including:
        i. breadthFirstOrder – print tree in breadth first fashion.
        ii. currentLevel – get current level.
        iii. depthFirstOrder – print tree in depth first fashion.
        iv. length – get the length of the tree.
3. Node.java - Holds the logic for an individual node in the tree.
4. Point.java – Holds the logic for an individual point in the node.
5. TreeBuilding.java - Holds the logic for building the tree from the given set of points.
6. FindAdd.java - Holds the logic for finding and adding an element to the tree.
7. FindInRange.java – Hold the logic for finding all elements between the specified range.

8. PrintTree.java: Hold the logic for printing the tree once it has all the nodes.
9. PointComparator.java: Used as a custom Comparator when creating a TreeSet. This comparator will essentially define how TreeSet should order the Point object.

## Data structures and their relations to each other

Mostly, the entire project consists of primitive data structures.

1. Node –
    a. Basic structure of the node in the kdTree.
    b. Consists of points which act as values of the node.
    c. Stores the node on the left and right.
    d. And have depth variable which keeps track of the current depth of the node.
2. Point –
    a. Consists of an ArrayList of the coordinates.
    b. Consists of the following methods:
        i. Dimension – get the dimension of the point.
        ii. getCoordinates – get the value of the point.
        iii. compareTo – compares the value between two points based on the current dimension.

## Assumptions

1. Assumed that the first level or the depth would start from 0. Although, this is just for simplicity purposes to access elements through index. This won't have any impact while we are printing the tree. The tree would be printed from level 1 and so on.

## Choices

1. Used recursive approach to execute almost every functionality of the kdTree. That's because I found a way to use recursion on one functionality and so later it was very easy to repeat the same thing to implement other functionalities.
2. Find in range would also include all the points which fall on the edge lines of the rectangle. Maximum, there can be 4 such points.

## Key algorithms and design elements

1. Building the tree:
    a. Using recursive approach to build the tree from the input points.
    b. Basically, I've used two pointers (lower index and upper index) which would be used to partition the array and find median.
    c. There is also a depth parameter which would be used to sort array at each level with respect to the current depth.
    d. And finally, the array of points for which the tree is to be built.
2. Adding the node to the tree:
    a. Check if node to be added is less or greater than the root node (with respect to the depth)
    b. If smaller, check if root has any nodes on left. If not, add element to the left of node

and if there's already a node on left, repeat from step one by incrementing the depth by 1.
    c. If greater, check if root has any nodes on right. If not, add element to the right of node and if there's already a node on right, repeat from step one by incrementing the depth by 1.
3. Finding an element in the tree:
    a. This is sort of like adding element.
    b. Here, we would check one more condition, which is to see if the current element is equal to the node which is to be searched.
    c. If not, check if the element to be searched is smaller or greater than the current node. (Similar to add)
    d. If smaller, go to the left node of current node and if greater, go to the right node of current node. Increment depth by 1.
    e. Repeat the process.
4. Getting the length/size of kdTree - The approach is to traverse through each node in the tree and keep adding one to the count.
5. Find in range:
    a. Used recursive approach.
    b. Traversing through each node in the tree and checking if the node is less than or equal to maximum corner and greater than or equal to minimum corner.
    c. If the above condition is true, add element to the set and return the set after each node in the tree is traversed.

## Limitations

1. After adding the element, tree becomes unbalanced. This can be implemented in future versions of code to make a complete pure balanced kdTree.