UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Faculty of Engineering, Built Environment & IT

University of Pretoria

# COS212 - Data structures and algorithms

# Assignment 1 Specifications: Bi-variate Cartesian Plane Sparse Table

Release Date: 28-02-2022 at 06:00

Due Date: 25-03-2022 at 23:59

Total Marks: 70

# Contents

## 1 General instructions:

- This assignment should be completed individually, no group effort is allowed.
- Be ready to upload your assignment well before the deadline as no extension will be granted.
- You may not import any of Java's built-in data structures. Doing so will result in a mark of zero. You may only make use of 1-dimensional native arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- All submissions will be checked for plagiarism.
- Read the entire assignment before you start coding.
- You will be afforded three upload opportunities.

## 2 Plagiarism

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding.** Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

## 3 Outcomes

Upon successful completion of this assignment you would have implemented the following:

- Implemented a sparse table to model a compressed Cartesian plane for bi-variate functions.
- Make use of a strategy design pattern to simulate multiple functions

## 4 Background:

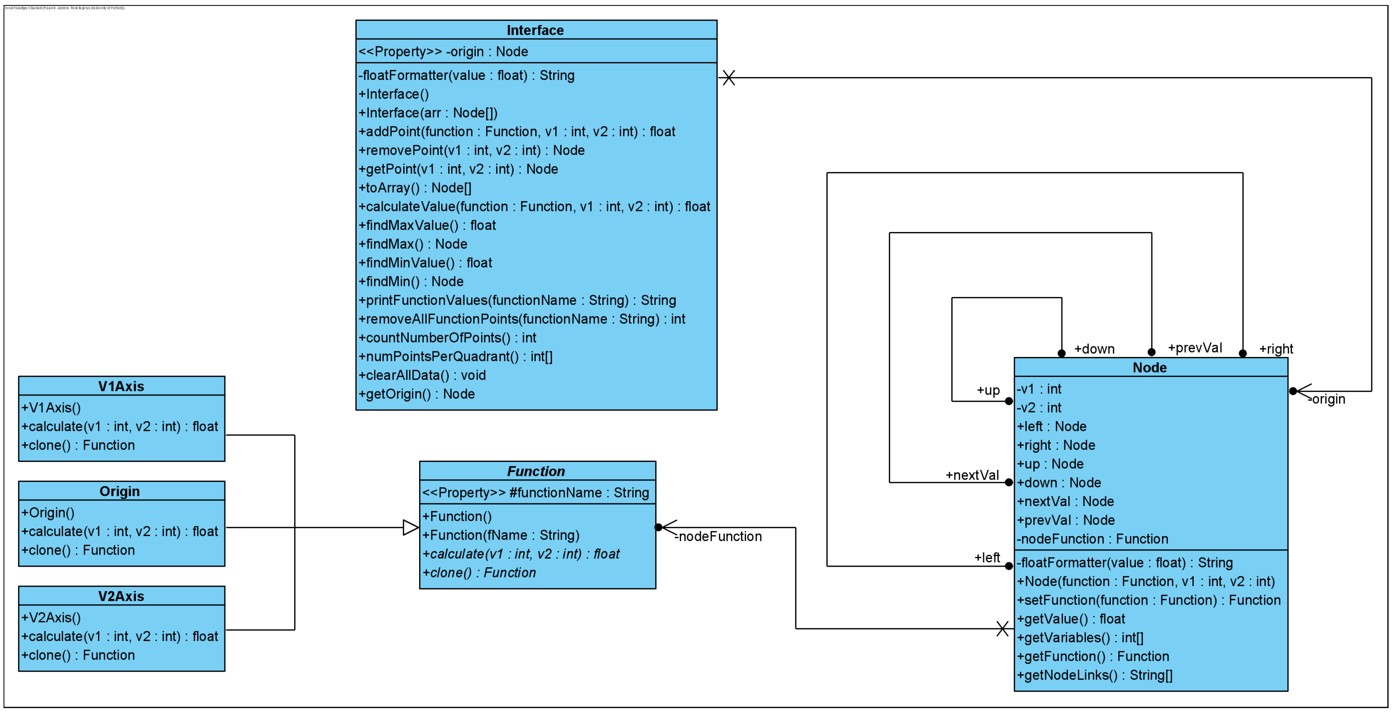
A bi-variate function is defined as a function that takes in two variables and produces a single result. The Ackermann function is a well known example of a bi-variate recursive function.
Normally to map out a bi-variate function a 2D or even 3D array would be needed (Depending on the implementation and desired outcome). This results in the following problems:

- Wasted memory when not all points needs to be mapped.
- Range and amount of data points needs to be known before hand.
- New points cannot be easily dynamically added as needed.
- Old unnecessary points cannot be easily removed.

These problems can be solved by using a sparse table to create a dynamically sized compressed Cartesian plane for bi-variate functions. Using the sparse table multiple functions can be mapped on the same plane.
A sparse table is a table like data structure where all the "cells" in the table are linked together with link lists to the axes. Note that the link lists can have "gaps" in them if the "cells" where the "gaps" are, are empty to avoid wasting space.

## 5 Task:

You will be implementing a sparse table as a compressed Cartesian plane to map multiple bi-variate functions.



Interface
<<Property>> -origin : Node
-floatFormatter(value : float) : String
+Interface()
+Interface(arr : Node[])
+addPoint(function : Function, v1 : int, v2 : int) : float
+removePoint(v1 : int, v2 : int) : Node
+getPoint(v1 : int, v2 : int) : Node
+toArray() : Node[]
+calculateValue(function : Function, v1 : int, v2 : int) : float
+findMaxValue() : float
+findMax() : Node
+findMinValue() : float
+findMin() : Node
+printFunctionValues(functionName : String) : String
+removeAllFunctionPoints(functionName : String) : int
+countNumberOfPoints() : int
+numPointsPerQuadrant() : int[]
+clearAllData() : void
+getOrigin() : Node
V1Axis
+V1Axis()
+calculate(v1 : int, v2 : int) : float
+clone() : Function
Origin
+Origin()
+calculate(v1 : int, v2 : int) : float
+clone() : Function
V2Axis
+V2Axis()
+calculate(v1 : int, v2 : int) : float
+clone() : Function
Function
<<Property>> #functionName : String
+Function()
+Function(fName : String)
+calculate(v1 : int, v2 : int) : float
+clone() : Function
-nodeFunction
+down
+prevVal
+right
+up
+nextVal
+left
-origin
Node
-v1 : int
-v2 : int
+left : Node
+right : Node
+up : Node
+down : Node
+nextVal : Node
+prevVal : Node
-nodeFunction : Function
-floatFormatter(value : float) : String
+Node(function : Function, v1 : int, v2 : int)
+setFunction(function : Function) : Function
+getValue() : float
+getVariables() : int[]
+getFunction() : Function
+getNodeLinks() : String[]

Figure 1: Complete class diagram

### 5.1 Node

This is the node class that will be used for each "data" point in the sparse table. Please note that node and point can be used interchangeably in this specification.
Nodes to the left and right have the same v2 value where nodes above and below (up and down nodes) will have the same v1 value. Nodes with the same v1 and v2 will either be prevVal and nextVal of each other or form part of the same list.

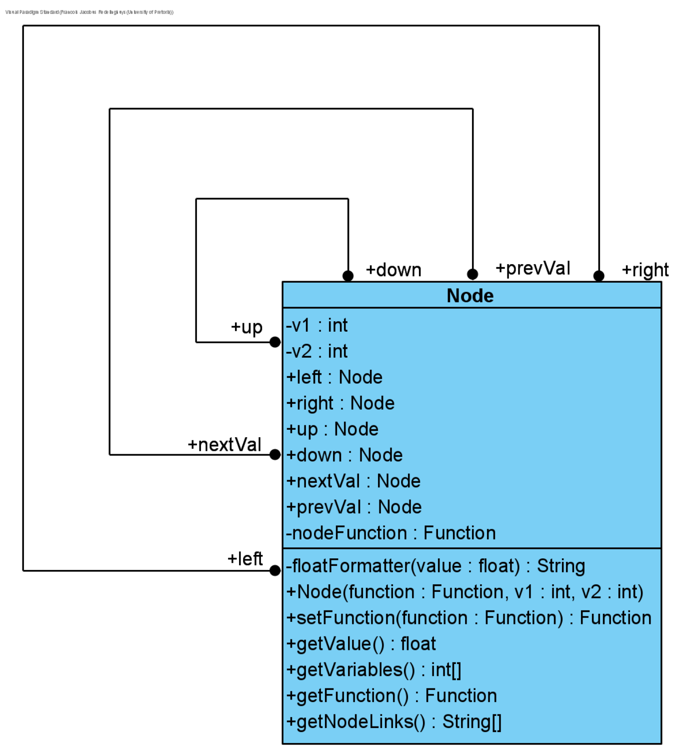


Figure 2: Node class

- Members:
  - v1: int
    - This is the first variable for the bi-variate function.
    - Nodes along the v2Axis will have a v1 value of 0.
  - v2: int
    - This is the second variable for the bi-variate function.
    - Nodes along the v1Axis will have a v2 value of 0.
  - left: Node
    - This is the node to the left of the current node.
  - right: Node
    - This is the node to the right of the current node.
  - up: Node
    - This is the node above the current node.
  - down: Node
    - This is the node below the current node.
  - nextVal: Node
    - This is the node with the same v1 and v2 as the current node.
    - See the interface class for the ordering of nodes with the same v1 and v2 values.

  - prevVal: Node
    - This is the node with the same v1 and v2 as the current node.
    - See the interface class for the ordering of nodes with the same v1 and v2 values.
  - nodeFunction: Function
    - This is the function object associated with the current node.
    - The origin node and nodes belonging to the v1Axis and v2Axis have specialized functions.

- Functions:
  - floatFormatter(value: float): String
    - This function is given. DO NOT CHANGE THIS FUNCTION.
    - This function will return a string containing the passed in value rounded to two decimal points without additional zeros.
    - Examples:
      - floatFormatter(2.4568) = 2.46
      - floatFormatter(0.100) = 0.1
      - floatFormatter(1) = 1
  - Node(function: Function, v1: int, v2: int)
    - This is the parametrized constructor for the node class.
    - A deep copy of the passed in function should be made.
    - The v1 and v2 members should be initialized accordingly.
  - setFunction(function: Function): Function
    - This function should replace the current function with the passed in function.
    - The function should return the original function.
  - getValue(): float
    - This function should return the value calculated by the function member using v1 and v2.
    - If the function member is null the function should return Float.NEGATIVE_INFINITY
    - *Hint: Call the appropriate function belonging to the function member*
  - getVariables(): int[]
    - This function should return v1 and v2 in the form of a 1D array of size 2.
    - Index 0 of the result will contain v1.
    - Index 1 of the result will contain v2.
  - getFunction(): Function
    - This function should return the function member property.

- getNodeLinks(): String[]
    - This function should return a string array of size 6 which contain information of the adjacent nodes to the current node.
    - The indexes and format are given below (Note that [ ] are square brackets):

| Index | Contains | If not null | If null |
|---|---|---|---|
| 0 | Up member's information | U[*up's v1*][*up's v2*]{*up's value*} | U[][]{} |
| 1 | Down member's information | D[*down's v1*][*down's v2*]{*down's value*} | D[][]{} |
| 2 | Right member's information | R[*right's v1*][*right's v2*]{*right's value*} | R[][]{} |
| 3 | Left member's information | L[*left's v1*][*left's v2*]{*left's value*} | L[][]{} |
| 4 | PrevVal member's information | P[*prevVal's v1*][*prevVal's*]{*prevVal's value*} | P[][]{} |
| 5 | NextVal member's information | N[*nextVal's v1*][*nextVal's*]{*nextVal's value*} | N[][]{} |

    - Note that if the node is null there should be **NO** spaces between the brackets (square or curly)
    - The value between the { } should be printed using the floatFormatter()
    - Example: Using slide 11 of the provided example slideshow the node at v1=2, v2=2 and a function of Efunc, the result will look as follows:
        - {U[][]{}; D[1][2]{3}; R[][]{}; L[1][2]{2}; P[2][2]{4}; N[][]{}}

## 5.2 Function

This will be the class responsible for the calculations of the functions.
During development/testing custom function classes will need to be written for testing purposes. Note the custom classes should inherit from this Function class.. **DO NOT SUBMIT THESE CUSTOM CLASSES**. An example of such a custom class will be provided.
**THIS CLASS IS GIVEN. DO NOT CHANGE OR ADD MEMBERS/FUNCTIONS TO THIS CLASS**

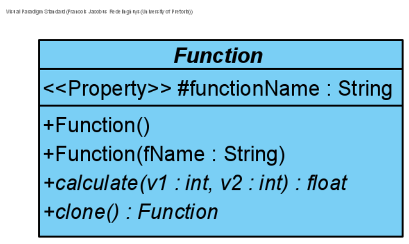

Visual Paradigm Standard(Francois Jacobus Redelinghuys(University of Pretoria))
Function
<<Property>> #functionName : String
+Function()
+Function(fName : String)
+calculate(v1 : int, v2 : int) : float
+clone() : Function

Figure 3: Interface class

- Members:
    - functionName: String
        - This is the name of the function

- Functions:
  - Function()
    - This is the default constructor. It sets the function name to a predefined name.
  - Function(fName: String)
    - This is the parameterized constructor. It sets the function name to the passed in variable.
  - calculate(v1: int, v2: int): float
    - This is an abstracted function used to calculate a value based on the passed in v1 and v2.
  - clone(): Function
    - This is an abstracted function used to create a copy of a function object.

#### 5.2.1 Origin

This will be the function used by the origin node in the Interface class described below. This class is a derived class of the Function class
**THIS CLASS IS GIVEN. DO NOT CHANGE OR ADD MEMBERS/FUNCTIONS TO THIS CLASS**

#### 5.2.2 V1Axis

This will be the function used by the nodes which will form the v1Axis (nodes with a v2 = 0) in the Interface class described below. This class is a derived class of the Function class
**THIS CLASS IS GIVEN. DO NOT CHANGE OR ADD MEMBERS/FUNCTIONS TO THIS CLASS**

#### 5.2.3 V2Axis

This will be the function used by the nodes which will form the v2Axis (nodes with a v1 = 0) in the Interface class described below. This class is a derived class of the Function class
**THIS CLASS IS GIVEN. DO NOT CHANGE OR ADD MEMBERS/FUNCTIONS TO THIS CLASS**

### 5.3 Interface class

This is the interface class for the sparse table which will be responsible for all sparse table functionality. The origin node will form the head of effectively 4 link lists. Up and down will form two link lists forming the v2 axis. Left and right will also form two link lists forming the v1 axis. Nodes should then be linked to the appropriate v1Axis and v2Axis to form a sparse table.

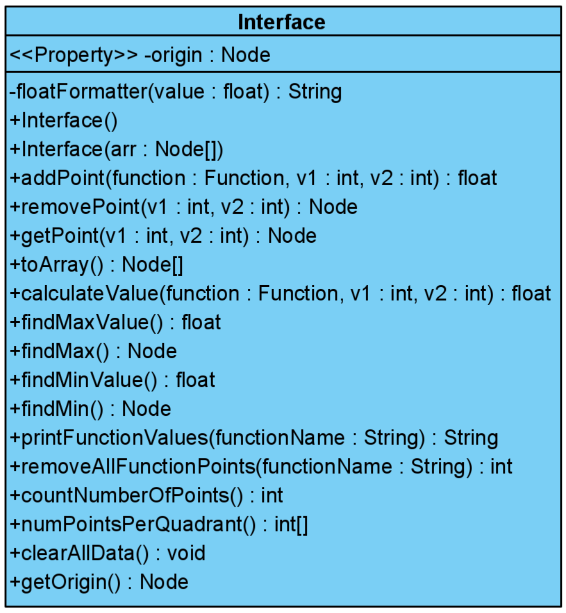

Interface
<<Property>> -origin : Node
-floatFormatter(value : float) : String
+Interface()
+Interface(arr : Node[])
+addPoint(function : Function, v1 : int, v2 : int) : float
+removePoint(v1 : int, v2 : int) : Node
+getPoint(v1 : int, v2 : int) : Node
+toArray() : Node[]
+calculateValue(function : Function, v1 : int, v2 : int) : float
+findMaxValue() : float
+findMax() : Node
+findMinValue() : float
+findMin() : Node
+printFunctionValues(functionName : String) : String
+removeAllFunctionPoints(functionName : String) : int
+countNumberOfPoints() : int
+numPointsPerQuadrant() : int[]
+clearAllData() : void
+getOrigin() : Node

Figure 4: Interface class

- Members:
  - origin: Node
    - This will be the origin of the cartesian plane.
    - The origin's v1 and v2 should be initialized to 0.
    - The origin's function should be an object of type Origin.
    - The origin's left property will consist of negative v1 values with a v2 value of 0.
    - The origin's right property will consist of positive v1 values with a v2 value of 0.
    - The origin's up property will consist of positive v2 values with a v1 value of 0.
    - The origin's down property will consist of negative v2 values with a v1 value of 0.
    - The origin's prevVal and nextVal will always be null.
- Functions:
  - floatFormatter(value: float): String
    - This function is given.
    - DO NOT CHANGE THIS FUNCTION.
    - This function will return a string containing the passed in value rounded to two decimal points without additional zeros.
    - For examples see Node's floatFormatter function
  - Interface()
    - This constructor should only initialize the origin Node.

- Interface(arr: Node[])
  - This constructor takes in an array containing Node objects.
  - Using the passed in array the sparse table should be populated making deep copies of the each non null node.
  - If a index of the array is null it should just be skipped and not inserted into the sparse table.
  - The order of the nodes in the array cannot be assumed. I.e. the order of v1, v2 or functions cannot be assumed. The order of nodes may be randomized.
  - *Hint use the function described below*
- addPoint(function: Function, v1: int, v2: int): float
  - This function will be used to add new nodes into the sparse table.
  - The function will receive a Function object, and two int parameters v1 and v2 respectively. These should be passed to the new node that will be inserted into the sparse table.
  - It may be assumed that the passed in function will never be null.
  - The function should return the value of the newly inserted node.
  - If the v1Axis does not contain the passed in v1 value the v1Axis should be extended such that it contains the new v1 value. Ensure that the new node in the v1Axis is inserted in the correct position.
  - If the v2Axis does not contain the passed in v2 value the v2Axis should be extended such that it contains the new v2 value. Ensure that the new node in the v2Axis is inserted in the correct position.
  - If there are already nodes with the same v1 and v2 value as the new node in the sparse table, the new node should be added to the list of nodes at position v1 and v2. This list should then be sorted in alphabetical order based on each node's function's name. The node with the lowest alphabetical order should be the head of the list of nodes at the given v1 and v2.
  - *Hint: for nodes that are linked using the prevVal and nextVal link list, the up left, down and right members should be null.*
  - If addPoint is called with v1 and/or v2 set to 0 the function should only return Float.NaN. No new node should be inserted into the sparse table.
  - Examples: See provided slideshow for additional examples

- removePoint(v1: int, v2: int): Node
  - This function should remove the node at the given v1 and v2 and return the node that was removed.
  - If the node at the given v1 and v2 has a prevVal then the prevVal node should become the new head of the list of nodes with the given v1 and v2. *Remember to set the new head's nextVal to null*
  - If the removal of the node at v1 and v2 results in the v1Axis node with a value of v1, having no up or down nodes that v1Axis node should be removed from the v1Axis.
  - If the removal of the node at v1 and v2 results in the v2Axis node with a value of v2, having no left or right nodes that v2Axis node should be removed from the v2Axis.
  - If there is no node at given v1 and v2 then the function should just return null.
  - If removePoint is called with v1 and/or v2 set to 0 then the function should return null without removing anything from the sparse table.
  - Remember to set up,down,left,right,prevVal,nextVal of the removed node to null before returning.
  - Examples: See provided slideshow for additional examples.
- getPoint(v1: int, v2: int): Node
  - This function should return the "head" node at the given v1 and v2.
  - If no node is at the given v1 and v2 then the function should return null.
  - If v1 and/or v2 is 0 then the function should also return null.
  - Examples: The following examples use the sparse table on slide 11 of the provide.
    - If getPoint(-2,-2) is called node with value 0 and function Efunc() should be returned.
    - If getPoint(0,0) is called then null should be returned.
- toArray(): Node[]
  - This function will return the sparse table as a 1D array.
  - The array size should be the exact number of nodes that is present in the sparse table. Nodes with a v1 and/or v2 of 0 should not be used to determining the size of the array.
  - The function should add nodes to the array starting from the right most v1 value and upper most v2 value for that v1 value. The function should then add them from highest v2 value to lowest v2 value for the v1 value before moving the v1's left node. This process should be repeated until all the nodes are placed in the array
  - Remember to also add all the nodes with the same v1 and v2.
  - No node with a v1 and/or v2 value of 0 should be added to the array.
  - Example: Using slide 11 of the provided slideshow the result will be as follows:
    - 4;4;3;2;0

- calculateValue(function: Function, v1: int, v2: int): float
  - This function should simply call the passed in function's calculate function using v1 and v2 and return the result.
  - If the function is null then Float.NaN should be returned.
- findMaxValue(): float
  - This function should return the value of the node with the highest value.
  - Any node with a v1 and/or v2 with a value of 0 should not be considered
  - If the table is empty the function should return Float.NaN
  - Example: Using slide 11 of the provided slideshow the following will be the result
    - 4
- findMax(): Node
  - This function should return the node with the highest value.
  - Any node with a v1 and/or v2 with a value of 0 should not be considered
  - If multiple nodes have the same highest value the node with the lowest (left most) v1 value and the highest (upper most) v2 value. *Hint: First check the v1 values. If multiple highest values have the same v1 then check the v2 values*
  - If the same v1 and v2 values have multiple nodes that will produce the same highest value, the node with the lowest alphabetical order should be returned.
  - If the table is empty the function should return null.
  - Example: Using slide 11 of the provided slideshow the result will be as follows:
    - Node with a value of 4 with a function of Efunc and a v1=2 and v2=2.
- findMinValue(): float
  - This function should return the value of the node with the lowest value.
  - Any node with a v1 and/or v2 with a value of 0 should not be considered
  - If the table is empty the function should return Float.NaN
  - Example: Using slide 11 of the provided slideshow the result will be as follows:
    - 0
- findMin(): Node
  - This function should return the node with the lowest value.
  - Any node with a v1 and/or v2 with a value of 0 should not be considered
  - If multiple nodes have the same highest value the node with the lowest (left most) v1 value and the highest (upper most) v2 value. *Hint: First check the v1 values. If multiple lowest values have the same v1 then check the v2 values*
  - If the same v1 and v2 values have multiple nodes that will produce the same lowest value, the node with the lowest alphabetical order should be returned.
  - If the table is empty the function should return null.
  - Example: Using slide 11 of the provided slideshow the result will be as follows:
    - Node with a value of 0 with a function of Efunc and v1=-2 and v2=-2.

- printFunctionValues(functionName: String): String
  - This function should print the values of all of the nodes with a function member that has a function name that matches the passed in functionName.
  - The nodes' values should be added starting from the v1Axis's left most node's most downward node. The function should then proceed to add all the nodes till the upper most node is reached before moving the to v1's right node.
  - If the function does not have any nodes in the sparse table the function should just return an empty string.
  - The nodes should be separated by ";".
  - **Use the floatFormatter function to format the float before adding it to the string.**
  - Nodes with a v1 and/or v2 of 0 should be ignored.
  - *Hint: remember some v1 v2 combinations have multiple nodes with different functions.*
  - Example: Using slide 11 of the provided slideshow the result will be as follows:
    - Using function Efunc: 0;3;4
    - Using function unknownFunc: empty string
- removeAllFunctionPoints(functionName: String): int
  - This function should remove all nodes whose function member's name is equal to the passed in functionName and return the number of nodes that were removed.
  - The removeAllFunctionPoints function should also adhere to all applicable constraints that are stipulated by the removePoints function.
- countNumberOfPoints(): int
  - This function will return the number of nodes in the sparse table.
  - Nodes with a v1 and or v2 with a value of 0 should not be counted.
  - If no nodes are present in the sparse table the function should return 0.
  - Example: Using slide 11 of the provided slideshow the result will be as follows:
    - 5
- numPointsPerQuadrant(): int[]
  - This function should count the number of nodes in each quadrant separately and store them in their respective index in an array. The array that will be returned should be of size 4.
  - The array is divided as follows:
    - Index 0 consists of quadrant 1 ($v1 > 0$ && $v2 > 0$)
    - Index 1 consists of quadrant 2 ($v1 < 0$ && $v2 > 0$)
    - Index 2 consists of quadrant 3 ($v1 < 0$ && $v2 < 0$)
    - Index 3 consists of quadrant 4 ($v1 > 0$ && $v2 < 0$)
  - If a quadrant does not contain any nodes the respective count for that quadrant should be 0.

    * Example: Using slide 11 of the provided slideshow the result will be as follows:
        · {4;0;1;0}
    * Any node with a v1 and/or v2 with a value of 0 should not be counted.
  – clearAllData(): void
    * This function should remove all the nodes from the sparse table.
    * The only node that should remain is the origin node.
  – getOrigin(): Node
    * This function should return the origin node.

## 6 Submission instructions

You need to create your own makefile for local development/testing. Note the makefile will be over-written.
Do not submit any custom function classes.
Your code should be able to be compiled with the following command:
**javac *.java**
Once you are satisfied that everything is working, you must tar all of your Java code, including any additional files which you've created, into one archive called uXXXXXXXX.tar.gz where XXX is your student number. Submit your code for marking under the appropriate link (Assignment 1) before the deadline.

Please ensure that you thoroughly test your code before submitting. Just because your code runs with local testing does not mean that your code works for every situation. **DO NOT USE FITCH FORK AS A DEBUGGER**

## 7 Change log:

- 02/03/2022:
  – Added getOrigin(): Node to the diagrams
  – Added getOrigin() function description
  – Added clarification to getMinValue()
  – Added clarification to getMaxValue()
  – Added clarification to getMin()
  – Added clarification to getMax()
- 03/03/2022:
  – Updated example for getNodeLinks()
  – Added clarification to getMin()
  – Added clarification to getMax()