# COMP 2002 – Assignment 2 (5%)

**Due: 11:59pm, Oct. 17, 2022**

## Learning Objectives

The goal of this assignment is to gain experience with trees.

## Instructions

Download the Python source file `bsptree.py`. The `BSPTree` class implements a one-dimensional Binary Space Partitioning tree.

Binary Space Partitioning trees recursively subdivide a region of space, separating the objects contained in that region into two sets. A node's two children (if any) represent the left and right sides of that area along the division point.

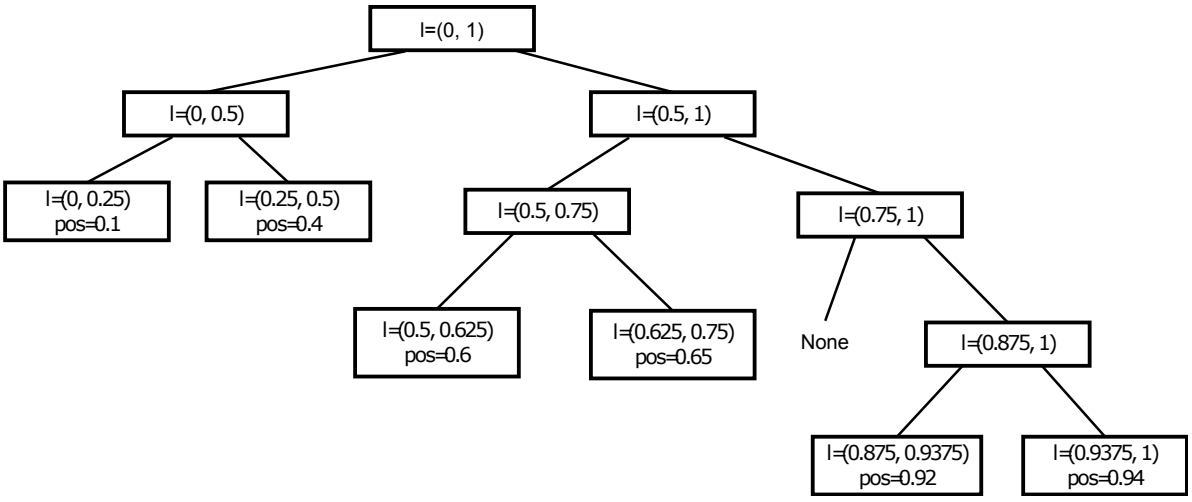
For this assignment, the `BSPTree` assumes data points have a position and a value, and that positions are within the range (0, 1). Each node of the `BSPTree` has a length, given as a tuple, which stores the minimum and maximum spatial range of the node.

The `BSPTree` class subdivides nodes at their midpoint. Each spatial region is subdivided until it encloses only a single data point. That is, external or leaf nodes contain only one point. New data points are inserted into the tree using the `insert_point()` method. Nodes store their size, that is, the total number of data points they enclose.

For example, take the data set with points located at `[0.1, 0.4, 0.6, 0.65, 0.92, 0.94]`.

A `BSPTree` constructed for this data set will subdivide the region recursively in two equal halves until a node contains only a single data point. (Note how one region contains no points. Match this to the corresponding part of the tree diagram.)


The tree diagram for this `BSPTree` is given below. The root node encloses the full space between (0, 1). Its left child encloses the space (0, 0.5) and its right child (0.5, 1.0). These are further subdivided until a node encloses only a single data point. Note that for one of the nodes, it only has a right child. Its left child is None. This occurs when one child node encloses two or more points and the other child node encloses none.




**Importantly, do not modify the constructor or any of the other methods** in the `_Node` or `BSPTree` classes except those explicitly instructed in the questions. The only modifications allowed to these classes are in those methods requested in the questions below.

**Question 1: (20 pts)**

Implement the `height()` methods in the `BSPTree` and `_Node` classes. For `BSPTree`, this method returns the height of the tree. For `_Node`, it returns the height of the node.

**Question 2: (20 pts)**

Implement the method `node_count()` in the `BSPTree` class. This method returns the total number of nodes contained in the tree.

**Question 3: (20 pts)**

Implement the method `contains()` in the `BSPTree` class. This method accepts a position and returns `True` if the tree contains a data point at that position, otherwise `False`.

**Question 4: (20 pts)**

Nodes in the BSPTree store a position and value. However, for internal nodes, this position and value is not meaningful. The position and value only has meaning for external nodes, as they represent the position and value of the data point that node contains.

Modify the `insert_point()` method so that the position and value of interior nodes store the average position and value of all data points that they enclose.

For example, in the tree diagram above, the root node would have the average position ~0.60167, representing the average of all data points. The left child of the root node would have average

position 0.25 (average of 0.1 and 0.4) and its right child average position 0.7775 (average of 0.6, 0.65, 0.92, and 0.94).

The average position and values for internal nodes can be updated incrementally as new data points are inserted in to the tree. If the average value of $n - 1$ data points is $a_{\text{n-1}}$, and one additional data point $k$ is added, the new average for $n$ data points, $a_{\text{n}}$, is given by

$$a_{\text{n}} = a_{\text{n-1}} + \frac{k - a_{\text{n-1}}}{n}. \tag{1}$$

## Submission

Submit a single zip file containing the Python source file with your modifications to the `BSPTree` and `_Node` classes through the Assignment submission folder in Brightspace.

Python source code should be `*.py` plain text. The only file types allowed aside from Python source code (`*.py`) are pdfs and plain text (`*.txt`). Do not submit Word documents or rich text format documents. They will not be marked. Only submit a single zip archive. Do not submit files archived in rar format. That may result in your assignment not being graded.

Name all files with the format "firstname_lastname_studentid_...". Make sure to include your name and student ID as comments at the top of all Python source files.

Late submissions will be subject to a 10% penalty for each hour past the deadline.

## Attribution

Submissions must represent your independent work.

Submissions should include an attribution section indicating any sources of material, ideas or contribution of others to the submission.

You are encouraged to use any resources to help with your solution, but your solution must represent independent work. If your submitted work includes unacknowledged collaboration, code materials, ideas or other elements that are not your original work, it may be considered plagiarism or some other form of cheating under MUN general regulations 6.12.4.2 and academic penalties will be applied accordingly.

Avoid academic penalties by properly attributing any contribution to your submission by others, including internet sources and classmates. This will also help distinguish what elements of the submission are original. You may not receive full credit if your original elements are insufficient, but you can avoid penalties for plagiarism or copying if you acknowledge your sources.

## Github

I encourage you to store and version your work on Github. It is good practice to do so as everyone uses git in the real world.

However, **it is a requirement that git repositories containing assignment material be private.** University regulations section 6.12.4.2 consider it cheating if you allow your work to be copied. There will be zero tolerance for this.