CMPT225, Spring 2023
Homework Assignment 4
Due date: Sunday, March 19, 2023, 23:59

You need to implement the following classes:

- *avltree.AVLNode*
- *avltree.AVLTree*
- *graph.Graph*

Note that all files must be under the correct package (folder)**.**
You may add more classes to your solution if necessary.

Submit all your files in ***assignment4.zip*** to CourSys.
Make sure your zip file can be unzipped using the command "*unzip assignmen4.zip*" in CSIL.
The zip file needs to contain exactly one folder: ***src***.
In the ***src*** folder there must be at least two folders/packages: ***avltree*** and ***graph***.
You need to submit the following files (and any additional files and folders if needed):

- src/*avltree/AVLNode.java*
- src/*avltree/AVLTree.java*
- src/*graph/Graph.java*

Discussion with others: You may discuss the assignment with your classmates/tutors (or anyone else), but coding must be entirely your own.

References: You may use textbooks, wiki, stack overflow, geeksforgeeks, etc. If you do, specify the references in comments. Posting questions online asking for solutions (e.g. using chegg.com) is prohibited.

Readability: Your code should be readable using the standard Java conventions. Add comments wherever is necessary. If needed, write helper functions or add classes to improve readability.

Compilation: Your code MUST compile in CSIL using javac.
Make sure that your code compiles without warnings/errors.
If the code does not compile in CSIL the grade on that part will be 0 (zero).
Even if you can't solve a problem completely, make sure it compiles.

The assignment will be graded mostly **automatically**, with some exceptions.

Do not add main() to your solutions. The main() method will be in the test files.

Warnings: Warnings during compilation will reduce points.
More importantly, they indicate that something is probably wrong with the code.

Testing: Test your code. Examples of tests are included. Your code will be tested using the provided tests as well as additional tests. You should create more tests to check your solution.

Good luck!

AVL Tree [50 points]

The class represents an AVL tree with elements of generic type. The insertion and deletion algorithms need to be exactly as we saw in class.
See also https://www.cs.usfca.edu/~galles/visualization/AVLtree.html for reference.

You need to implement the following two classes:

```
public class AVLNode<T extends Comparable<T>>
public class AVLTree<T extends Comparable<T>>
```

The class AVLTree needs to contain the following methods:

public AVLTree()
  creates an empty AVL tree

public AVLNode<T> find(T item)
  returns a node containing the item
  if item is not in the tree, the method throws NoSuchElementException

public AVLNode<T> insert(T item)
  inserts a new node with the give item into the AVL tree
  returns the new node

public void remove(T item)
  removes a node with the give item from the tree
  if item is not in the tree, the method throws NoSuchElementException

public int height()
  Returns the height of the tree in O(1) time

public int size()
  Returns the size of the tree in O(1) time

public T getMin()
  returns the minimal element of the tree in O(1) time
  if the tree is empty, the method throws NoSuchElementException

public Collection<T> lessThanK(T k)
  returns a collection of all elements in the tree for which element.compareTo(k) < 0
  if the list is empty, the method returns an empty list

Graph [50 points]

The class represents a graph. That is a collection of vertices and edges. It has one constructor that gets the number of vertices, and the number of vertices does not change.

You need to implement the following methods:

public Graph(**int** n)
    creates an empty graph on n nodes.
    the "names" of the vertices are 0,1,..,n-1

public void addEdge(**int** i, **int** j)
    adds the edge (i,j) to the graph

public void removeEdge(**int** i, **int** j)
    removes the edge (i,j) from the graph

public boolean areAdjacent(**int** i, **int** j)
    checks if (i,j) is an edge in the graph

public boolean distanceAtMost2(**int** i, **int** j)
    checks if the distance between i and j in the graph is at most 2

public int degree(**int** i)
    returns the degree of i

public Iterator<Integer> neighboursIterator(**int** i)
    Returns an iterator that outputs the neighbors of i in the increasing order
    Assumption: the graph is not modified during the use of the iterator

public int numberOfVertices()
    Returns the number of vertices in the graph

public int numberOfEdges()
    Returns the number of edges in the graph

public static Graph generateRandomGraph(**int** n, **double** p)
    Generates a random graph on n vertices such that each edge appears in the graph
    with probability p independently of all others
    You may use Math.random() to generate a random number between 0 and 1