Specification for Assignment 3 of 4

Your submission for this assignment ***must include your full name*** *(as it appears on cuLearn) and you nine-digit* ***student number*** *at the top of* ***every file you submit****.* ***Source code submissions that crash*** *(i.e., terminate with an error) on execution will* ***receive a mark of 0****.*

Your submission for this assignment must be a ***single source file*** *with a* ***file name of 'comp3007_w19_#########_a3.hs'*** *(with the number signs replaced by your nine-digit student number). It* ***must be written using Haskell*** *and* ***must run in GHCi or WinGHCi****.*

Because this single source file will contain a large number of functions, you must ***be especially careful to ensure that the file you upload is free from errors****, as any such error would render the entire file unreadable and result in you receiving a mark of 0 for the assignment.*

You must also include a "README" text file, indicating the name of the function that corresponds to each of the different tasks that comprise this assignment. This file must have a file name of ***'comp3007_w19_#########_a3_README.txt'*** *(with the number signs replaced by your nine-digit student number).*

Compress your submissions *into a* ***zip*** *file named* ***'comp3007_w19_#########_3.zip'***

Late assignments will not be accepted *and will* ***receive a mark of 0****.*

The due date for this assignment is Saturday, March 16, 2019, by 11:00pm.

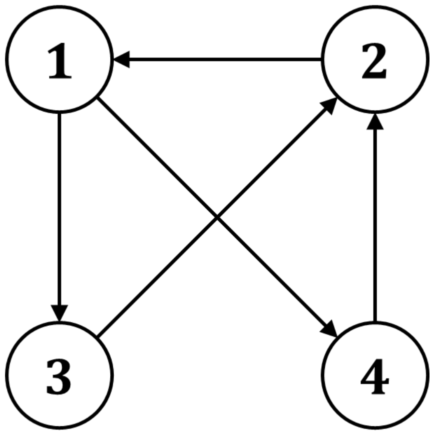
A directed graph G can be operationally defined as an ordered pair (i.e., a 2-tuple) of the form (V, E), where V is a set of nodes and E is a set of directed edges, and where each directed edge is, itself, an ordered pair of nodes (i.e., the node at which the directed edge originates, and the node at which it terminates, respectively). The union G ∪ G' of two graphs G = (V, E) and G' = (V', E') can also be operationally defined as the ordered pair (V ∪ V', E ∪ E' ∪ F), where F is the set of edges that connect a node in E with a node in E' (or vice versa).

Although it isn't an ideal representation, a directed graph could conceivably be recursively defined using this union operation as the graph constructor operator. Under the constraints that graphs can never be equal and that self loops (i.e., edges that originate and terminate at the same node) are not permitted, such a recursive definition would state that a graph is either a singleton node, or the union of two other graphs with some additional edges.

The following block of code (provided to you in a support file) will provide you with a custom data type for representing such a graph, and with this assignment you will write several functions for performing straightforward operations with such graphs.

```haskell
type Node = Int
type Edge = (Node, Node)
data Graph = Single Node | Union Graph Graph [Edge] deriving Show
```

As a clarifying example, consider the directed graph depicted right. In accordance with the types described on the previous page, this graph (which clearly is not an example of the singleton variant) could be represented as the union of G = ({1, 3}, {(1, 3)}) and G' = ({2, 4}, {(4, 2)}) with the additional edge set {(1, 4), (2, 1), (3, 2)}. Since G itself could be represented as the union of the singleton graph ({1}, {}) and the singleton graph ({3}, {}) with the additional edge set {(1, 3)}, and G' could be represented as the union of singleton graph ({2}, {}) and the singleton graph ({4}, {}) with the additional edge set {(4, 2)}.



This entails that the graph depicted above could be represented in Haskell as:

```
Union
      (Union
            (Single 1)
            (Single 3)
            [(1, 3)])
      (Union
            (Single 2)
            (Single 4)
            [(4, 2)])
      [(1, 4), (2, 1), (3, 2)]
```

In must, however, be explicitly noted that this representation is not unique - the same graph could also be represented as:

```
Union
      (Single 1)
      (Union
            (Single 2)
            (Union
                  (Single 3)
                  (Single 4)
                  [])
            [(3, 2), (4, 2)])
      [(1, 3), (1, 4), (2, 1)]
```

Additional representations of this same graph are possible, and it must be stressed that no representation should be considered superior or inferior to any other (as long as they are both correct).

Question 1: "Does this Node Appear in this Graph?"

For the first question of this assignment, you must design and implement a function that takes a Node as its first argument and a Graph as its second argument and has a Bool return value. This function must return True if and only if the first argument (i.e., the Node) is contained anywhere in the Graph that was passed as the second argument.

Question 2: "Does this Edge Appear in this Graph?"

For the second question of this assignment, you must design and implement a function that takes an Edge as its first argument and a Graph as its second argument and has a Bool return value. This function must return True if and only if the first argument (i.e., the Edge) is contained anywhere in the Graph that was passed as the second argument.

Question 3: "List All Nodes that Appear in this Graph"

For the third question of this assignment, you must design and implement a function that takes a Graph as its only argument and has a list of Nodes (i.e., a [Node]) as its return value. The return value must be the list containing each of the Nodes that appears in the Graph.

Question 4: "List All Edges that Appear in this Graph"

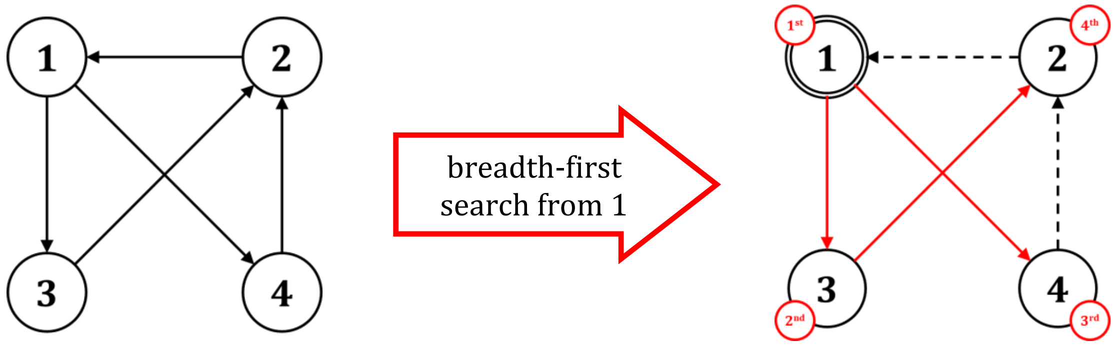
For the fourth question of this assignment, you must design and implement a function that takes a Graph as its only argument and has a list of Edges (i.e., an [Edge]) as its return value. The return value must be the list containing each of the Edges that appears in the Graph.

Question 5: "Is this Graph a Singleton?"

For the fifth question of this assignment, you must design and implement a function that takes a Graph as its only argument and has a Bool return value. This function must return True if and only if the Graph is the singleton variant (i.e., a single Node with no Edges).

Question 6: "Compute a Breadth-First Search Traversal"

For the sixth question of this assignment, you must design and implement a function that takes a Graph as its first argument and a Node as its second argument and has a list of Nodes (i.e., a [Node] as its return value). The return value must be the list of Nodes traversed by a breadth-first search traversal of the Graph argument, starting from the Node argument, in the order in which they appear during the traversal. When your breadth-first search algorithm has a "choice" of which Node to explore, it must process the nodes in numerical order. As a clarifying example, please note that the breadth first search traversal of the Graph included as the previous example starting from Node 1 should be [1, 3, 4, 2] and would correspond to the traversal depicted on the following page:



Question 7: "Create a Graph from an Adjacency List"

For the seventh question of this assignment, you must design and implement a function that takes an 'adjacency list' as its first argument has a Graph return value. For the purposes of this question, an adjacency list will take the form [(Node, [Node])]. To clarify, an adjacency list is a list of ordered pairs, where each element of that list specifies a Node (as its first component) and all of the Nodes that are adjacent from that node (as its second component). The graph associated with the example depicted previously would be represented using the adjacency list:

```
[(1, [3, 4]), (2, [1]), (3, [2]), (4, [2])]
```

Please also note that, in accordance with the simplifying constraints mentioned on the first page, the function you write to address this question should produce an error if supplied with an empty adjacency list argument.

Question 8: "Create a Graph from an Adjacency Matrix"

For the eighth question of this assignment, you must design and implement a function that takes an 'adjacency matrix as its first argument has a Graph return value. For the purposes of this question, an adjacency matrix will take the form [[Bool]]. To clarify, an adjacency matrix should be a square matrix of Boolean values, where a value of True is present in the matrix at row X and column Y if and only if there is a directed edge from X to Y. You must label each of the Nodes in the resulting graph with consecutive integer values starting from the value of 1. The graph associated with the example depicted previously would be represented using the adjacency matrix:

```
[[False, False, True, True], [True, False, False, False],
       [False, True, False, False], [False, True, False, False]]
```

Please also note that, in accordance with the simplifying constraints mentioned on the first page, the function you write to address this question should produce an error if supplied with an empty adjacency matrix argument or an argument that is not a square matrix.

You may use only the list constructor operator (`:`), the list concatenation operator (`++`), and the arithmetic operators to complete this assignment, and you may not use any built-in functions. This means that you will need to write your own implementations of basic functions like `length`, `take`, `drop`, and `reverse`, should you required them.

The programs you write must be a completely original works, authored by you and you alone, prepared for this offering (i.e., Winter 2019) of COMP3007. Do not discuss this (or any other) question with anyone except the instructor or the teaching assistants, and do not copy materials from the internet or any other source.