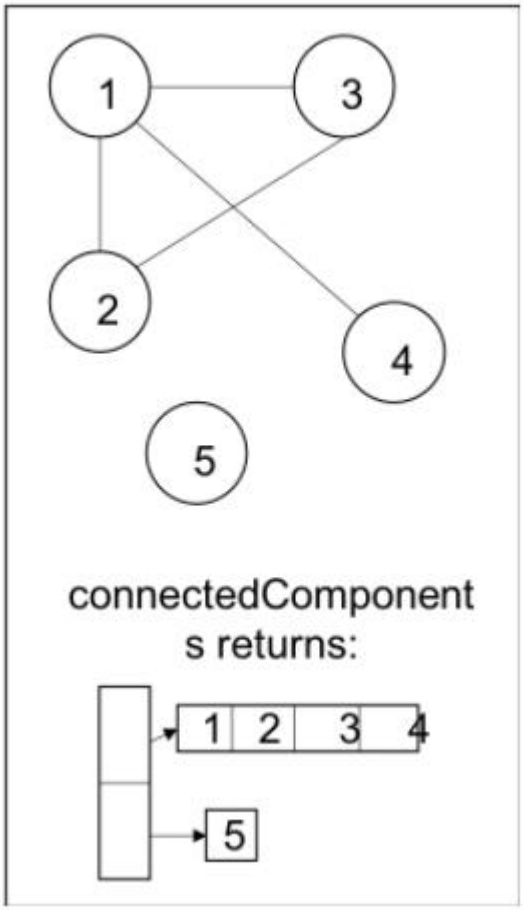
Assignment 3

Graphs and Algorithm Design

Exercise 1

Implement a class of graphs MyUndirectedUnweightedGraphImpl with operations addVertex(AnyType vertex), addEdge(AnyType sourceVertex, AnyType targetVertex).

- Implement a method "public boolean isConnected()" that checks if the graph is connected. The method should execute in worst-case O(|E|+|V|).
- Implement a method connectedComponents() that returns a myList of myLists of vertices (MyList<MyList<AnyType>> connectedComponents()) that returns the vertices in each connected component of the graph. Next figure shows an example of what connectedComponents() should return. The MyList is the same as in assignment 1; therefore, you can reuse the implementation that you did some weeks ago. The method should execute in worst-case O(|E|+|V|).

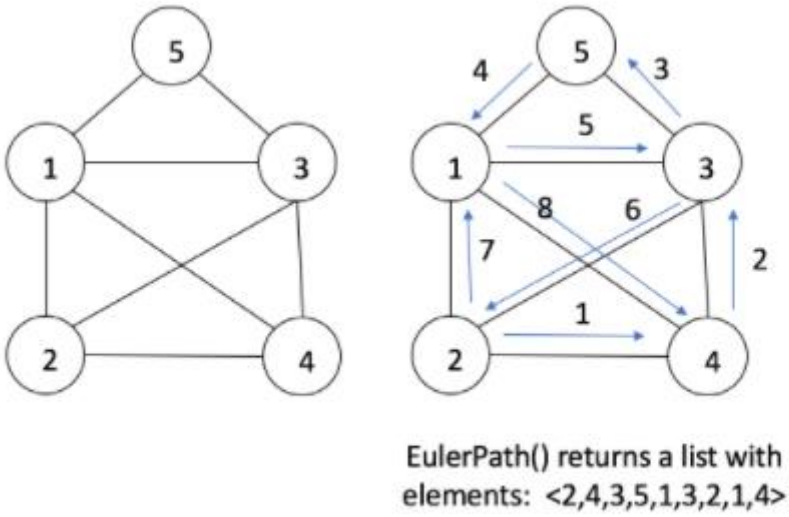


Exercise 2

A common game is to find how to traverse all lines in a drawing passing only one time through each line and without lifting the pen. In graphs, an Euler path is a path that traverses every edge exactly once (see Chapter 9.6.3 in the course reference book, which explains how to implement it).

Implement in MyUndirectedUnweightedGraphImpl methods:

- public boolean hasEulerPath(), which returns true if the graph has an Euler path. This method must execute in O(|E|+|V|).
- public myList<AnyType> eulerPath(), which returns a list of vertices of length |E|+1. The vertices in the list represent how to traverse the graph to complete an Euler path. This method should execute in $O(|E|^2)$ (we will only check that the output is correct; we will not create a graph with millions of elements to measure execution times; therefore, just be sure that the method is not embarrassingly slow). Next figure shows an example.



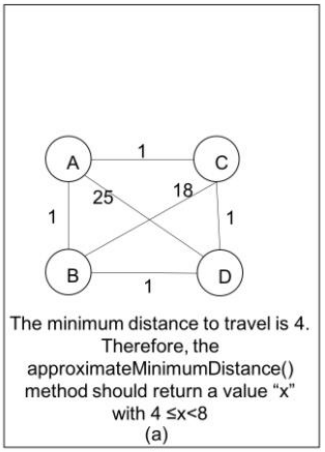
Exercise 3

This exercise includes dynamic programming and graphs. You must implement a class MyNeighborhood which solves a slightly modified version of Exercise 3 in the exam 31st October 2019 (published with the course material). The modifications are the following:

- Part (a) only has to return the computed distance to walk. The name of the method is “approximateMinimumDistance”. The time complexity of the algorithm has to be polynomial in the number of vertices (exponential time solutions are not allowed) and must not depend on the values of the weights in edges.

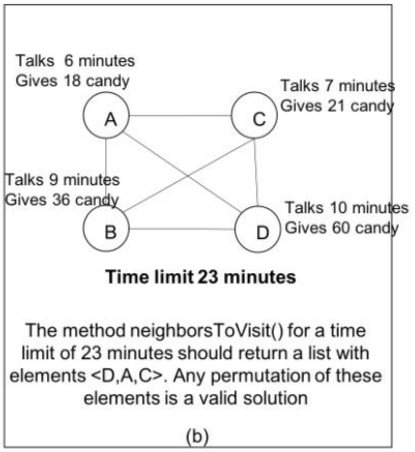
- Part (b) needs to return the set of neighbours that are visited (in other words : IGNORE THE “NOTE 1” in the problem statement. It applied to the exam because the available time to solve the exercise was much more limited). The method name is "neighborsToVisit". The set of visited neighbours is returned in a list “myList”. The algorithm should execute in O(N*T).

See the next figure for clarification.



The minimum distance to travel is 4. Therefore, the approximateMinimumDistance() method should return a value "x" with 4 ≤x<8

(a)



The method neighborsToVisit() for a time limit of 23 minutes should return a list with elements <D,A,C>. Any permutation of these elements is a valid solution

(b)

Important: You must implement the ADTs without reusing Graph data structures from libraries. You can use libraries for arrays and arrayLists in this assignment.

Hint: You may feel that you need to associate an integer index with each vertex of the generic class AnyType. That feeling is OK. To do it, you **must NOT use anything in the java libraries** like HashSets, HashTables, HashMaps, etc.; **but you can reuse your own myHashTableImpl** that you implemented in Assignment 2.

Important2: Respect the interfaces provided in the .zip. These are the interfaces that we use to execute your code.