GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computing & Mathematics

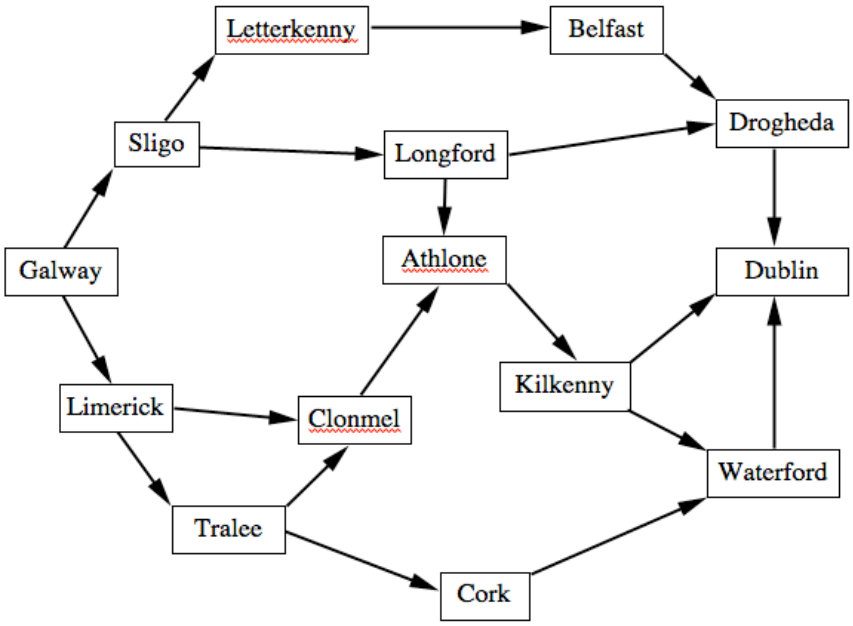
CP2SD – Data Structures & Algorithms

Lab 9: Graphs

Graph theory is a branch of discrete mathematics that has its origins in the work of Leonhard Euler (1707-1783). Despite the contributions of many great mathematicians over a period of two centuries (including the Irishman William Rowan Hamilton), it was not until the advent of digital computing in the second half of the 20th century that graph theory could be applied to practical applications.

In the context of computer science, graphs are the most abstract, and therefore the most reusable, data structures. All the data structures we have studied so far, including lists, trees and maps, are all specialized types of graph. The high degree of specialization inherent in these data structures limits their application and simplifies the algorithms required to operate on them. Consequently, while we typically can manipulate such specialized data structures in linear time, the abstract nature of graphs requires generalized algorithms whose time and memory complexity depends on the graph's topography.

In this practical, we will examine how an adjacency list can be used to represent the following directed acyclic graph. We will also examine how both a depth-first and breadth-first search can be applied to the graph to exhaustively search all nodes.



- Download the Zip archive ***lab9-graphs.zip from*** Moodle and extract the files into a folder on your hard disk. As the classes involved are all packaged, ensure that you create the correct directory structure and study the source code before you compile.

The following Java source files are included in the Zip archive:

- ***Vertex.java:*** An abstraction of a graph vertex. *Vertex* maintains a string representing the node state and an instance variable called visited that is used to determine if a vertex has already been encountered during a search. Painting vertices like this is vital if the graph may contain cycles.
- ***Graph.java:*** Uses a *HashMap* to map vertices to a list of edges. *Graph* exposes the methods required to build and query the graph.
- ***SearchType.java:*** An enum that is used to parameterises the search algorithm.
- ***Search.java:*** An interface that defines the method void *search(Vertex vertex)* to be implemented by search implementations.
- ***GraphSearch.java:*** An implementation of the interface *Search*. *GraphSearch* can perform either a DFS or BFS, depending on the *SearchType* passed to the ***search()*** method.
- ***GraphRunner.java:*** Constructs the graph by creating vertices and edges using the *Graph* class.

Note that there is no direct correlation between the mathematical depiction of a graph (logical) and the programming implementation (physical).

- Execute the class ***GraphRunner.java***. A DFS should be performed using the class *GraphSearch*. Sketch the progress of the search over the image of the graph on the preceding page.

- Alter line 24 of *GraphSearch.java* and change the ***for*** loop as follows:

  for (int i = destinations.length - 1; i >= 0; i--)

  What effect does this have on the order of transversal?

- Change line 15 of ***GraphRunner.java*** to perform a breadth-first search as follows and execute the programme again:

  Search s = new GraphSearch(graph, SearchType.BREADTH_FIRST);

  What effect does using a BFS have on the order of transversal?

- In ***GraphRunner.java***, add a new edge between Dublin and Athlone as follows:

  graph.addEdge(dublin, athlone);

  Comment out line 20 of ***GraphSearch.java*** as follows: *next.setVisited(true);*
  Execute the programme again and explain the output (or possible lack of it…)