A ***graph*** *G is* a set *V* of ***vertices*** and a collection *E* of pairs of vertices from *V*, called ***edges***. The aim of this worksheet is to make **two implementations** of the Graph ADT one based on the ***adjacency matrix*** and other on the ***adjacency map*** representation. Figure 1 illustrates a graph with both representations.

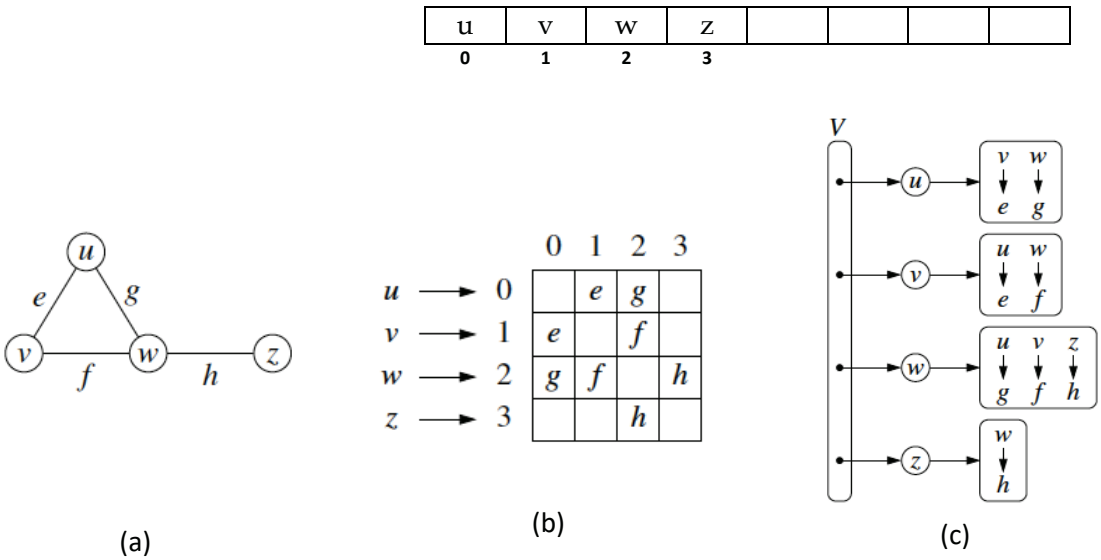
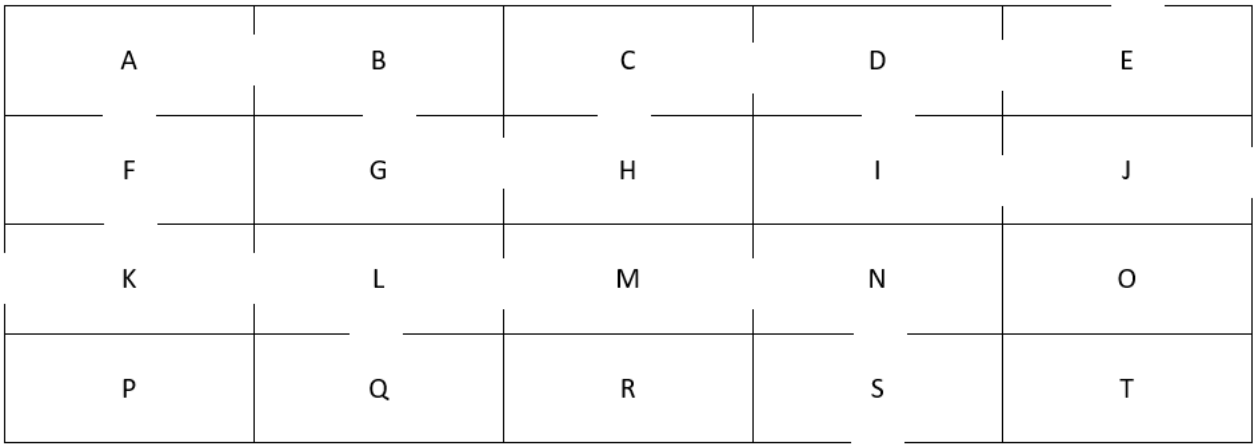


Figure1 - (a) An undirected graph G; (b) a schematic representation of the adjacency matrix
(c) a schematic representation of the adjacency map structure

The Edge class stores the weight of the edge and its both endpoints vertices. The Vertex class, in the adjacency map representation, stores the information associated with the vertex and a map with its outgoing edges.

A graph instance maintains the number of vertices and edges of the graph, a boolean variable that designates whether the graph is directed and a list with all its vertices.

1. Implement a **LabyrinthCheater** class, which is intended to represent a labyrinth through a map of rooms and doors. Each room has a name and potentially provides an exit. Doors represent connections between rooms.



   a) Declare the class, and its attributes, using an **adjacency matrix** to represent the labyrinth (suggestion: create private inner classes to represent rooms and doors)

   b) Implement a method to list all rooms which are reachable from a particular room (identified by name)

   c) Implement a method to return the name of the nearest room with an exit (from a particular room)

   d) Implement a method to return the path with the sequence of rooms to reach the nearest room with an exit (from a particular room)

   e) Create unit tests for methods developed in b. to d.

2. Represent a **network of airports** and the flight routes between them using a **directed graph**. For simplicity, consider that each airport is represented by its name, the distance between airports is in miles and the flight routes only include the traffic or the number of passengers carried.

   a) Make the declaration of the class AirportNet using the ***adjacency map*** representation and considering the private class Route.

   Add the following methods to the class AirportNet:

   b) Return the numeric key of a given airport

   c) Return the airport with a specific numeric key

d) Return the traffic between two directly linked airports, note that the traffic (number of passangers carried) between two airports may be different in the two connections.

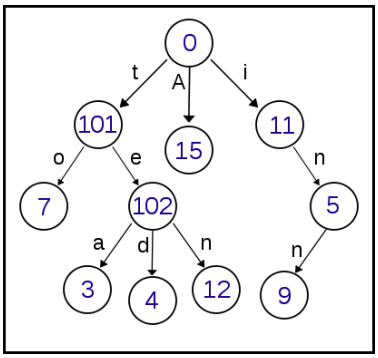
e) Return the miles between two directly linked airports

f) Return the number of routes origin and destination for all airports

g) Return the airports with the greatest connection (more miles)

h) Check whether two airports are reachable

3. A trie is a data structure widely used in Information Retrieval for character string recognition. Generally a trie can be seen as an n-ary tree with symbols in the branches and numerical values in the nodes, where in this case the **terminal nodes** have numerical values **smaller than 100**, the **non terminal nodes** have numerical values **greater than 100** and the initial node has the value zero.



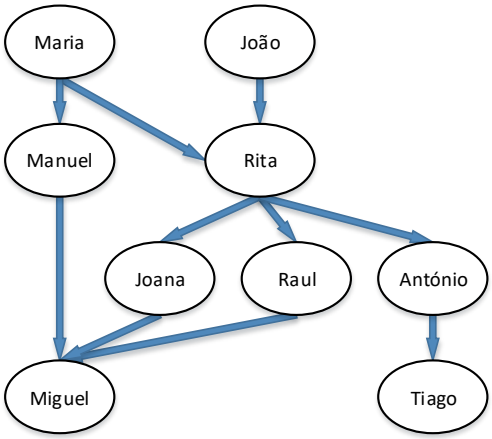
https://pt.wikipedia.org/wiki/Trie

To recognize a sequence of characters, the trie begins at the start node and uses each of the sequence characters to traverse through the branch with that character, if the branch exists. If, at the end of the sequence is at a terminal node the sequence is recognized. The above trie recognizes the sequences: {to, tea, ted, ten, A, i, in, inn}, but the sequences {t, te, x, tent, Al} are not recognized.

Write a method that validates whether a sequence of characters is recognized by the trie. It must return the number corresponding to the terminal node if the sequence is recognized or -1 if it is not.

```
public Integer checkSequence(Graph<Integer,Character> trie, String sequence)
```

4. Consider an oriented and acyclic graph where each vertex represents an employee and each branch x -> y represents semantically that the employee y can only be promoted if the employee x is also promoted. For example, in the graph of the figure if we want to find 2 promotions, there are 2 possible lists: {Maria, João} or {Maria, Manuel}. In case you want 3 promotions, the options are {Maria, João, Manuel} or {Maria, João, Rita}.



Implement a method that, given a graph and a n number of promotions, return a list with the employees to promote. The goal is only to find one of the solutions. The method to be developed must obey the interface:

```
List<String> getPromotions(Graph<String,Integer> g, Integer n)
```