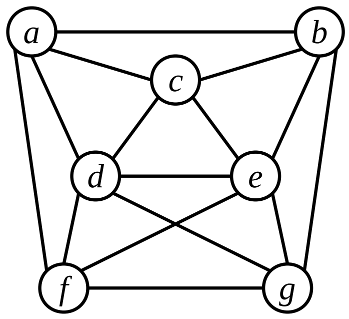
Homework #10

This homework assignment requests the submission of both written answers and C++ programs. Upload your written answers in a PDF file named `hw10.pdf`. Each individual question contains specific instructions on the submission and naming of code files.

Question 1 (8 pt.)

The complete code for this question should be uploaded in a file named `q1.zip` containing all source and header files needed to compile and execute the program. The program should compile without errors and run correctly on the CoE Linux machines.

Consider the following unweighted, undirected graph, equal to the graph presented in the support material for Unit 7:



a) (2 pt.) Write a main program that creates this graph using a representation based on an adjacency matrix, as done in the code given in the support material for the Floyd-Warshall algorithm. Insert the vertexes into the graph in the following order: *a*, *b*, *c*, *d*, *e*, *f*, *g*.

b) (2 pt.) Write an algorithm that computes whether a given set of vertexes forms a clique, that is, whether each vertex is connected to every other vertex of the set. The algorithm should be implemented as a member function of class `Graph`, with the following prototype:

```
bool Graph::isClique(std::vector<int> vertexes);
```

The argument of this function is a vector of integers representing the vertexes present in the set.

c) (2 pt.) Using your previous implementation, write an algorithm that solves the *clique* decision problem in the form of a member function with the following prototype:

```
bool Graph::CliqueExists(int k);
```

This function returns `true` if there is a clique of at least *k* vertexes. Here are some hints:

- The *clique* decision problem is NP-Complete, so you will need to explore all combinations of *k* vertexes in the graph. Obtaining <u>only</u> the combinations of *k* vertexes can be tricky, so it will be easier to just obtain all possible combinations of vertexes, and discard those that do not have exactly *k* vertexes.

- You can explore all combinations of $|V|$ elements by just having a looping variable *i* go from 0 to $2^{|V|}-1$. Each bit in variable *i* represents the presence of each vertex in the set. You have to ignore all values of *i* that do not have exactly *k* bits set to 1. For this purpose, implementing a helper function `CountOnes(int value)` can be useful, which returns the number of bits set to 1 in `value`.

You can take advantage of the trace presented in the support material to verify the correctness of your implementation.

d) (1 pt.) Using the previous implementation, write an algorithm that solves the *maximum clique* optimization problem in the form of a member function with the following prototype:

```
int Graph::getMaxClique();
```

This function returns the maximum number of vertexes forming a clique in the current graph. Extend your main program with a call to this function and print its result. Verify that the result matches your expectations.

e) (1 pt.) What is the temporal cost of functions `Graph::isClique()`, `Graph::CliqueExists()`, and `Graph::getMaxClique()`?

Question 2 (2 pt.)

Answer the following questions related with the NP-Complete class of problems:

a) (0.5 pt.) How is this class defined?

b) (0.5 pt.) Given the definition of a problem, what is the best strategy to proof that it is NP-Complete?

c) (0.5 pt.) Why is this class of problems especially intriguing to the scientific community?

d) (0.5 pt.) There are hundreds of problems that have been proven to be NP-Complete. Pick one and define it in one paragraph.