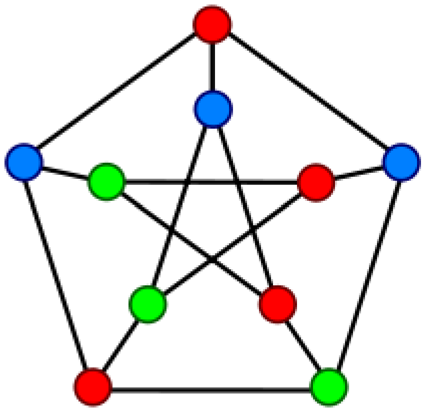
# Graph Colouring Problem

Given an undirected graph and a number m, determine if the graph can be coloured with at most m colours such that no two adjacent vertices of the graph are coloured with same colour. Here colouring of a graph means assignment of colours to all vertices.

*Input:*
1) A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.
2) An integer m which is maximum number of colours that can be used.

*Output:*
An array colour[V] that should have numbers from 1 to m. colour[i] should represent the colour assigned to the ith vertex. The code should also return false if the graph cannot be coloured with m colours.

Following is an example graph (from Wiki page ) that can be coloured with 3 colours.



**Naive Algorithm**
Generate all possible configurations of colours and print a configuration that satisfies the given constraints.

```
while there are untried configuration
{
   generate the next configuration
   if no adjacent vertices are coloured with same colour
   {
      print this configuration;
   }
}
```

There will be V^m configurations of colours.

**Backtracking Algorithm**

The idea is to assign colours one by one to different vertices, starting from the vertex 0. Before assigning a colour, we check for safety by considering already assigned colours to the adjacent vertices. If we find a colour assignment which is safe, we mark the colour assignment as part of solution. If we do not a find colour due to clashes then we backtrack and return false.

The most obvious solution to this problem is arrived at through a design referred to as *backtracking*.

Recall that the essence of backtracking is:

1. Number the solution variables [$v_0$ $v_1$, …, $v_{n-1}$].
2. Number the possible values for each variable [$c_0$ $c_1$, …, $c_{k-1}$].
3. Start by assigning $c_0$ to each $v_i$.
4. If we have an acceptable solution, stop.
5. If the current solution is not acceptable, let $i = n-1$.
6. If $i < 0$, stop and signal that no solution is possible.
7. Let j be the index such that $v_i = c_j$. If $j < k-1$, assign $c_{j+1}$ to $v_i$ and go back to step 4.
8. But if $j \geq k-1$, assign $c_0$ to $v_i$, decrement i, and go back to step 6.

Although this approach will find a solution eventually (if one exists), it isn't speedy. Backtracking over n variables, each of which can take on k possible values, is $O(k^n)$.

For graph colouring, we will have one variable for each node in the graph. Each variable will take on any of the available colours.