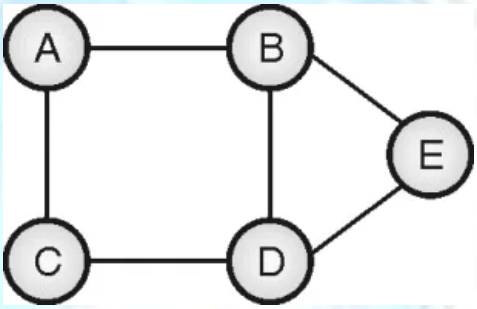
Hamiltonian Cycle

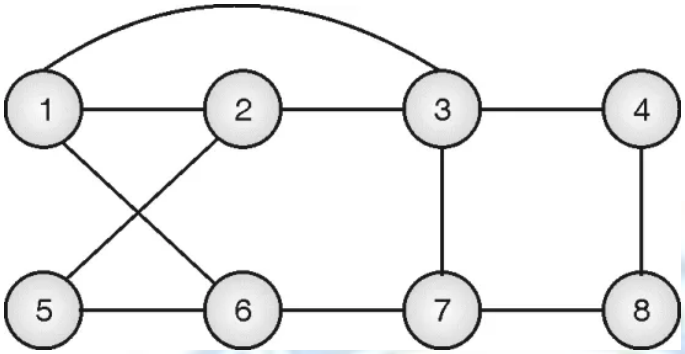
The Hamiltonian cycle is the cycle in the graph which visits all the vertices in graph exactly once and terminates at the starting node. It may not include all the edges

- The Hamiltonian cycle problem is the problem of finding a Hamiltonian cycle in a graph if there exists any such cycle.
- The input to the problem is an undirected, connected graph. For the graph shown in Figure (a), a path A – B – E – D – C – A forms a Hamiltonian cycle. It visits all the vertices exactly once, but does not visit the edges <B, D>.



- The Hamiltonian cycle problem is also both, decision problem and an optimization problem. A decision problem is stated as, “Given a path, is it a Hamiltonian cycle of the graph?”.
- The optimization problem is stated as, “Given graph G, find the Hamiltonian cycle for the graph.”
- We can define the constraint for the Hamiltonian cycle problem as follows:
- In any path, vertex i and (i + 1) must be adjacent.
- 1st and (n – 1)th vertex must be adjacent (nth of cycle is the initial vertex itself).
- Vertex i must not appear in the first (i – 1) vertices of any path.
- With the adjacency matrix representation of the graph, the adjacency of two vertices can be verified in constant time.

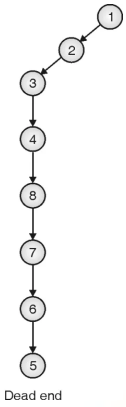
Example:



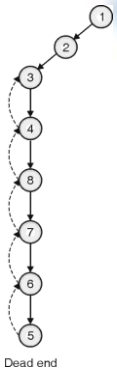
Solution:

The backtracking approach uses a state-space tree to check if there exists a Hamiltonian cycle in the graph. Figure (f) shows the simulation of the Hamiltonian cycle algorithm. For simplicity, we have not explored all possible paths, the concept is self-explanatory.

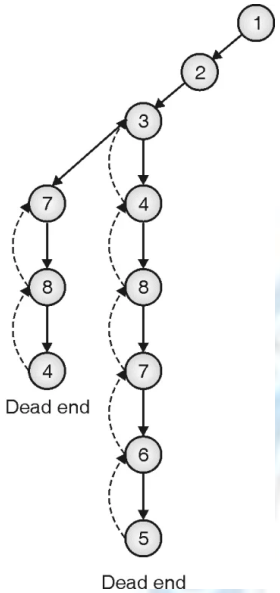
Step 1: Tour is started from vertex 1. There is no path from 5 to 1. So it's the dead-end state.



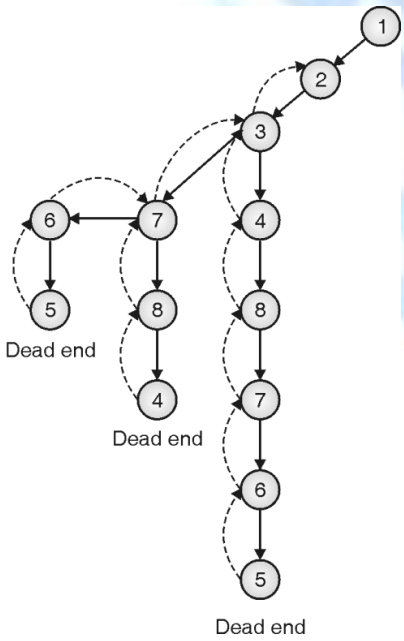
Step 2: Backtrack to the node from where the new path can be explored, that is 3 here



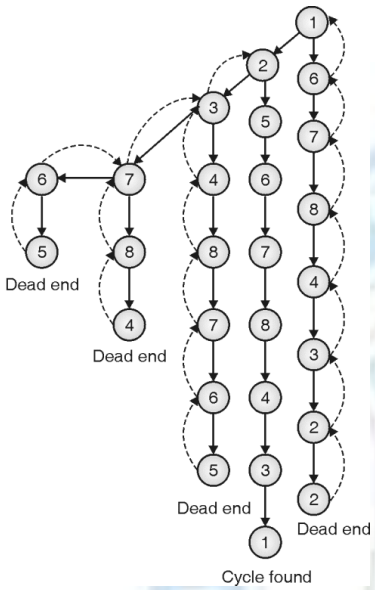
Step 3: New path also leads to a dead end so backtrack and explore all possible paths



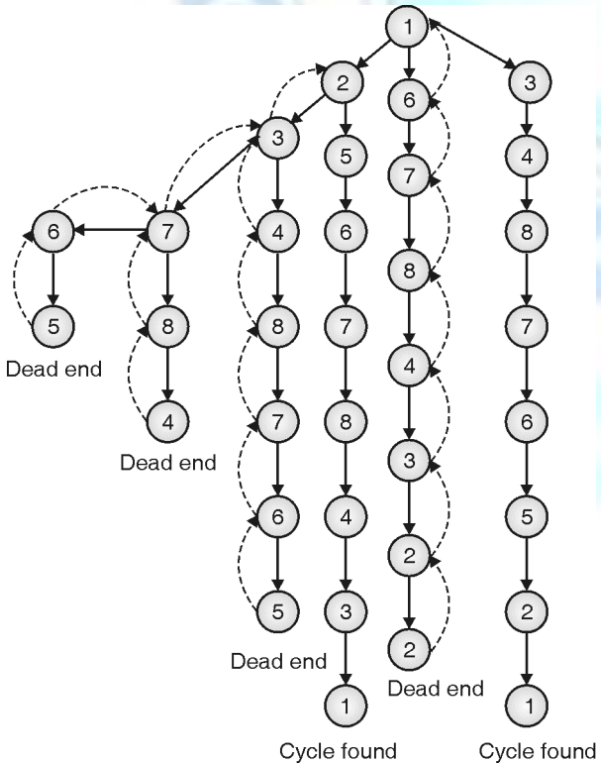
Step 4: Next path is also leading to a dead-end so keep backtracking until we get some node that can generate a new path, i.e .vertex 2 here



Step 5: One path leads to Hamiltonian cycle, next leads to a dead end so backtrack and explore all possible paths at each vertex



Step 6: Total two Hamiltonian cycles are detected in a given graph



Graph Coloring

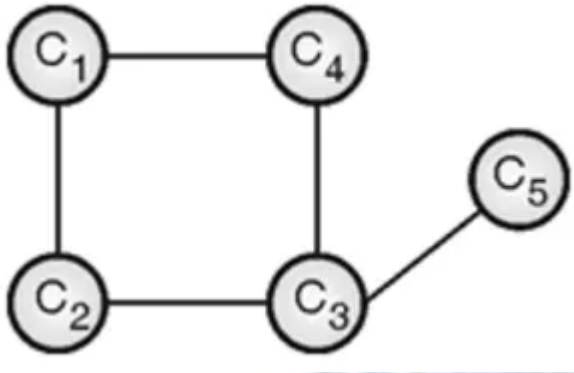
Graph coloring refers to the problem of coloring vertices of a graph in such a way that no two adjacent vertices have the same color. This is also called the vertex coloring problem.

- If coloring is done using at most k colors, it is called k-coloring.
- The smallest number of colors required for coloring graph is called its chromatic number.
- The chromatic number is denoted by X(G). Finding the chromatic number for the graph is NP-complete problem.
- Graph coloring problem is both, decision problem as well as an optimization problem. A decision problem is stated as, "With given M colors and graph G, whether such color scheme is possible or not?".
- The optimization problem is stated as, "Given M colors and graph G, find the minimum number of colors required for graph coloring."
- Graph coloring problem is a very interesting problem of graph theory and it has many diverse applications.

Applications of Graph Coloring Problem

- Design a timetable.
- Sudoku
- Register allocation in the compiler
- Map coloring
- Mobile radio frequency assignment:

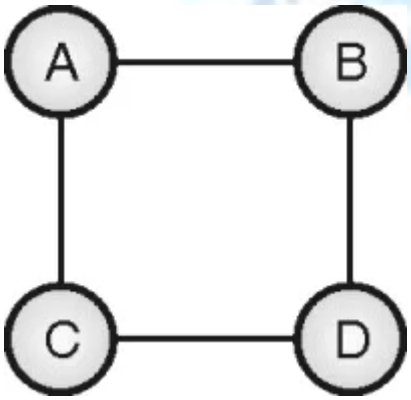
The input to the graph is an adjacency matrix representation of the graph. Value M(i, j) = 1 in the matrix represents there exists an edge between vertex i and j. A graph and its adjacency matrix representation.



	C_1	C_2	C_3	C_4	C_5
C_1	0	1	0	1	0
C_2	1	0	1	0	0
C_3	0	1	0	1	1
C_4	1	0	1	0	1
C_5	0	0	1	0	0

Adjacency matrix for graph G

Algorithm backtracks whenever color i is not possible to assign to any vertex k and it selects next color i + 1 and test is repeated. Consider the graph shown in Figure



If we assign color 1 to vertex A, the same color cannot be assigned to vertex B or C. In the next step, B is assigned some different colors 2. Vertex A is already colored and

vertex D is a neighbor of B, so D cannot be assigned color 2. The process goes on. State-space tree is shown in Figure

