**6.11 Undirected Graph**

**Aim:** The aim is to determine if an undirected graph contains a cycle that visits every single vertex exactly once.

**Algorithm:**

1. Let the undirected graph be represented as G(V, E) with n = |V| vertices.

2. The task is to check whether there exists a cycle (v₁, v₂, …, vn, v₁) such that:  
  - Every vertex in V appears exactly once in the sequence.  
  - Consecutive vertices (vi, vi+1) are connected by an edge in E.  
  - There is also an edge (vn, v₁) to close the cycle.

3. We define a recursive backtracking procedure: HamiltonianCycle (path, visited)  
  - path → sequence of chosen vertices so far  
  - visited → Boolean array marking whether a vertex has already been included

4. Base condition:  
  - If path contains n vertices and there is an edge between the last vertex path[n−1] and the first vertex path[0], then a Hamiltonian cycle exists.

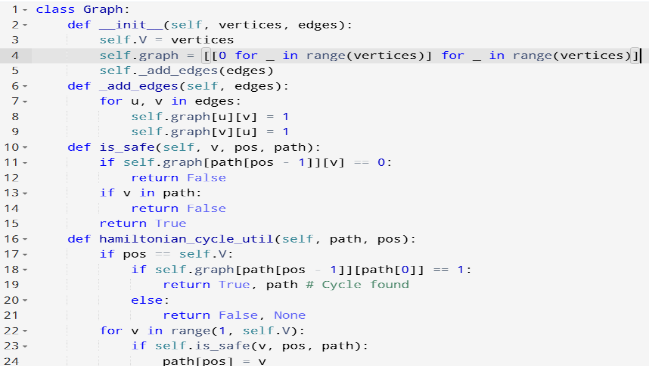
5. Recursive case:  
  - Let u = path[last] be the last vertex in the current path.  
  - For each vertex v ∈ V:  
    a. If (u, v) ∈ E and visited[v] = false:  
      - Mark visited[v] = true.  
      - Add v to path.  
      - Recursively call HamiltonianCycle(path, visited).  
      - If successful, return true.  
      - Otherwise, backtrack: remove v from path and set visited[v] = false.

6. If no valid extension of path is possible, return false.

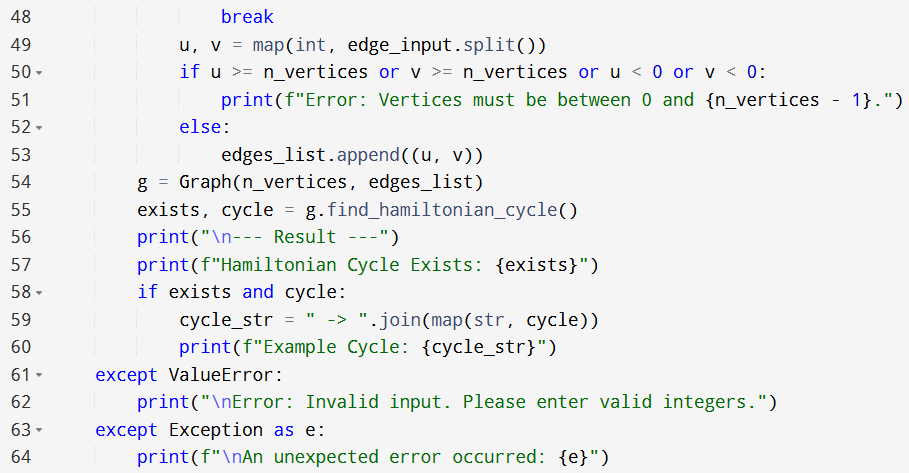
7. Initially, choose an arbitrary starting vertex v₀, set visited[v₀] = true, and call: HamiltonianCycle([v₀], visited)

8. If the procedure returns true, the graph contains a Hamiltonian cycle. Otherwise, no such cycle exists.

**Program:**

****

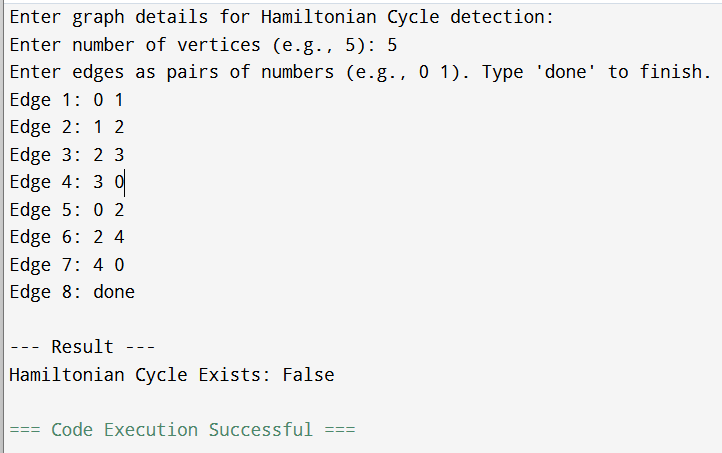




**Input:**

* Enter graph details for Hamiltonian Cycle detection:
* Enter number of vertices (e.g., 5):
* Enter edges as pairs of numbers (e.g., 0 1). Type 'done' to finish.

**Output:**

****

**Result:** Thus, the program is executed successfully and output is verified.

**Performance analysis:**

* Time Complexity: O(n\*n)
* Space Complexity: O(n^2).