**6.11 Undirected Graph**

**Aim:** The aim is to determine if a given 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 goal is to check if there exists a cycle (v₁, v₂, …, vn, v₁) such that:  
  - Every vertex vi ∈ V is included exactly once.  
  - For every consecutive pair (vi, vi+1), the edge belongs to E.  
  - An edge also exists between (vn, v₁) to close the cycle.

3. Define a recursive backtracking procedure:  
  HamiltonianCycle (path, visited)  
  - path → ordered sequence of vertices chosen so far  
  - visited → Boolean array to mark which vertices are already in the path

4. Base condition:  
  - If path contains all n vertices and the last vertex is adjacent to the first vertex, then a Hamiltonian cycle is found → return true.

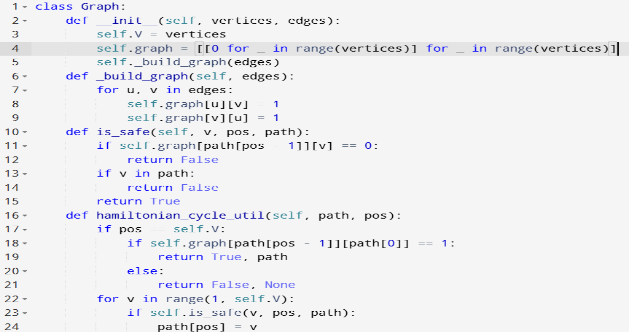
5. Recursive case:  
  - Let u = last(path).  
  - For each vertex v ∈ V: a. If (u, v) ∈ E and visited[v] = false:  
      - Mark visited[v] = true.  
      - Append v to path.  
      - Recursively call HamiltonianCycle(path, visited).  
      - If the recursive call succeeds, return true.  
      -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

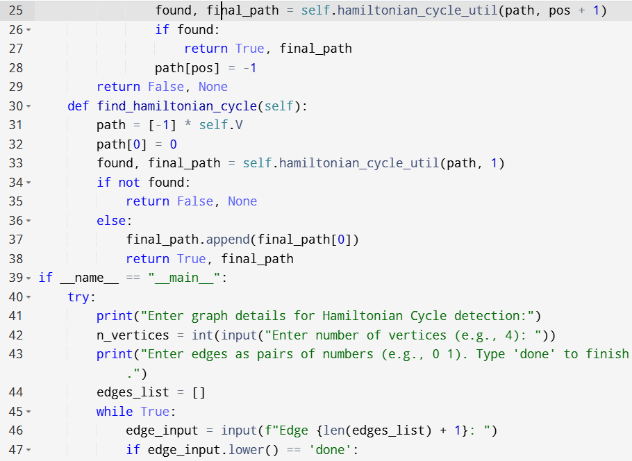
6. If no vertex can be added to path, return false.

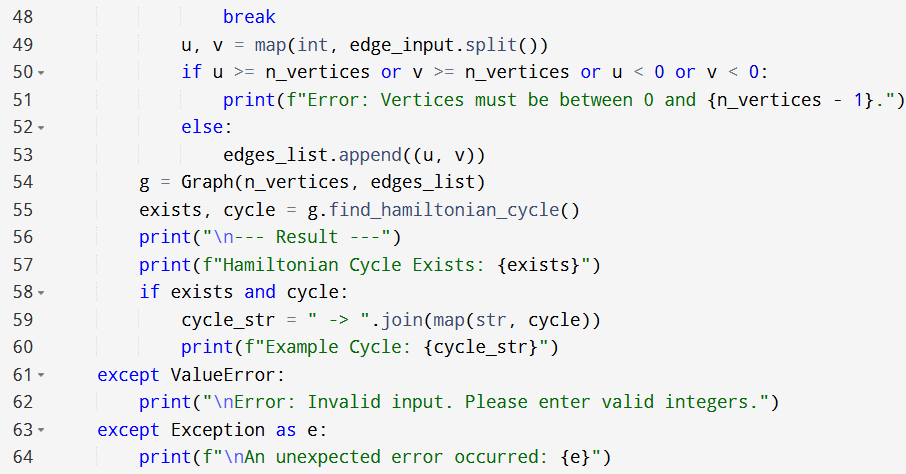
7. Initialization:  
  - Choose an arbitrary starting vertex v₀.  
  - Mark visited[v₀] = true.  
  - Call HamiltonianCycle([v₀], visited).

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

**Program:**

****

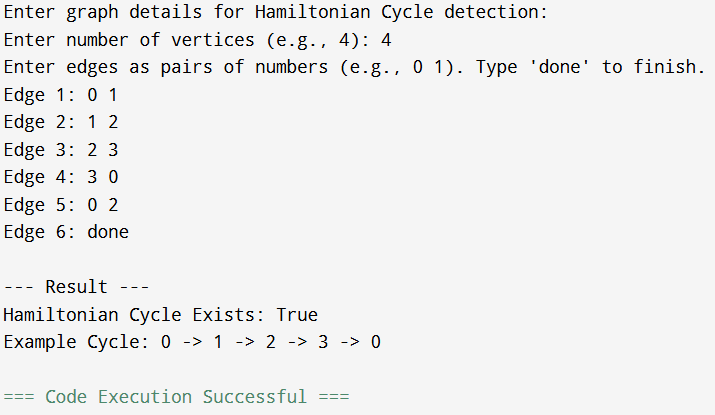




**Input:**

* Enter graph details for Hamiltonian Cycle detection:
* Enter number of vertices (e.g., 4):
* 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).