**Hamiltonian Cycle USING BACKTRACKING**

* **INTRODUCTION**
* **EXAMPLE**
* **ALGORITHM**
* **CODE**

**INTRODUCTION-** 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 edgesHamiltonian path-in an undirected graph is a path that visits each vertex exactly once. Determine whether a given graph contains Hamiltonian Cycle or not. If it contains, then prints the path. Following are the input and output of the required function.*Input:*   
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.  
*Output:*   
An array path[V] that should contain the Hamiltonian Path. path[i] should represent the ith vertex in the Hamiltonian Path. The code should also return false if there is no Hamiltonian Cycle in the graph.

**ALGORITHM**

* If a graph is given then we have to start from a vertex then visit all the vertex exactly once and come back to the starting vertex. It is Hamiltonian cycle.
* Now problem is: Is there any Hamiltonian cycle possible? If possible then find out all those cycles.
* Problem looks similar to travelling salesman problem. In travelling salesman problem we have start from a vertex then visit all vertex and come back to the starting vertex.  but it is optimization problem where we have to find out the minimum cost tour.
* Graph may be directed or undirected but it should be connected. If graph is not connected then Hamiltonian cycle is not possible.
* Finding Hamiltonian cycle is not easy problem. Its NP-Hard Problem

For example, a Hamiltonian Cycle in the following graph is {0, 1, 2, 4, 3, 0}.

(0)--(1)--(2)

 |   / \   |

 |  /   \  |

 | /     \ |

(3)-------(4)

And the following graph doesn’t contain any Hamiltonian Cycle.

(0)--(1)--(2)

 |   / \   |

 |  /   \  |

 | /     \ |

(3)      (4)

**Naive Algorithm**   
Generate all possible configurations of vertices and print a configuration that satisfies the given constraints. There will be n! (n factorial) configurations.

while there are untried conflagrations

{

   generate the next configuration

   if ( there are edges between two consecutive vertices of this

      configuration and there is an edge from the last vertex to

      the first ).

   {

      print this configuration;

      break;

   }

}

**Backtracking Algorithm**   
Create an empty path array and add vertex 0 to it. Add other vertices, starting from the vertex 1. Before adding a vertex, check for whether it is adjacent to the previously added vertex and not already added. If we find such a vertex, we add the vertex as part of the solution. If we do not find a vertex then we return false.

* If a graph is given then we have to start from a vertex then visit all the vertex exactly once and come back to the starting vertex. It is Hamiltonian cycle.

|  |
| --- |
| #include <bits/stdc++.h>  **using** **namespace** std;    #define V 5    **void** printSolution(**int** path[]);    **bool** isSafe(**int** v, **bool** graph[V][V],  **int** path[], **int** pos)  {  **if** (graph [path[pos - 1]][ v ] == 0)  **return** **false**;    **for** (**int** i = 0; i < pos; i++)  **if** (path[i] == v)  **return** **false**;    **return** **true**;  }    **bool** hamCycleUtil(**bool** graph[V][V],  **int** path[], **int** pos)  {    **if** (pos == V)      {  **if** (graph[path[pos - 1]][path[0]] == 1)  **return** **true**;  **else**  **return** **false**;      }  **for** (**int** v = 1; v < V; v++)      {    **if** (isSafe(v, graph, path, pos))          {              path[pos] = v;      **if** (hamCycleUtil (graph, path, pos + 1) == **true**)  **return** **true**;                path[pos] = -1;          }      }  **return** **false**;  }    **bool** hamCycle(**bool** graph[V][V])  {  **int** \*path = **new** **int**[V];  **for** (**int** i = 0; i < V; i++)          path[i] = -1;        path[0] = 0;  **if** (hamCycleUtil(graph, path, 1) == **false** )      {          cout << "\nSolution does not exist";  **return** **false**;      }        printSolution(path);  **return** **true**;  }    **void** printSolution(**int** path[])  {      cout << "Solution Exists:"              " Following is one Hamiltonian Cycle \n";  **for** (**int** i = 0; i < V; i++)          cout << path[i] << " ";          cout << path[0] << " ";      cout << endl;  }    **int** main()  {  **bool** graph1[V][V] = {{0, 1, 0, 1, 0},                          {1, 0, 1, 1, 1},                          {0, 1, 0, 0, 1},                          {1, 1, 0, 0, 1},                          {0, 1, 1, 1, 0}};        hamCycle(graph1);    **bool** graph2[V][V] = {{0, 1, 0, 1, 0},                           {1, 0, 1, 1, 1},                           {0, 1, 0, 0, 1},                           {1, 1, 0, 0, 0},                           {0, 1, 1, 0, 0}};        hamCycle(graph2);    **return** 0;  } |

**Output:**

Solution Exists: Following is one Hamiltonian Cycle

 0  1  2  4  3  0

Solution does not exist

Note that the above code always prints a cycle starting from 0. The starting point should not matter as the cycle can be started from any point. If you want to change the starting point, you should make two changes to the above code.   
Change “path[0] = 0;” to “path[0] = s;” where s is your new starting point. Also change loop “for (int v = 1; v < V; v++)” in hamCycleUtil() to “for (int v = 0; v < V; v++)”.