CGT Mini Project

PROGRAM TO FIND THE NO. OF DIFFERENT HAMILTONIAN CYCLE IN A GRAPH

Group members:

Charishma Priya	106121032
Harika	106121046
Sameeksha Jaiswal	106121112
Gopal	106121026

Problem Statement:

Find all Hamiltonian cycles in a graph

For a given graph a Hamiltonian cycle is a path that passes through every vertex exactly once and returns to the starting vertex. For a Hamiltonian cycle to be possible the graph must be connected. The given graph may be directed or undirected.

Program format

Input: Two-dimensional array containing only elements ‘0’ or ‘1’. The two dimensional array is an adjacency matrix is used to represent the graph.

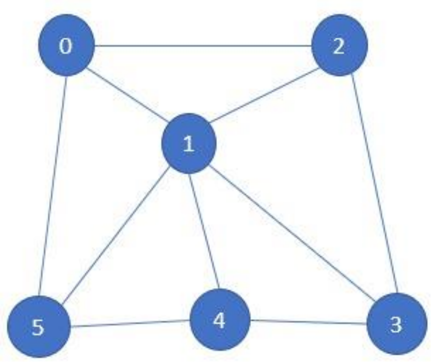
Output: If Hamiltonian cycle exists then all possible Hamiltonian cycles should be printed along with number of cycles. Else it should return message “No Cycle possible”.

Algorithm:

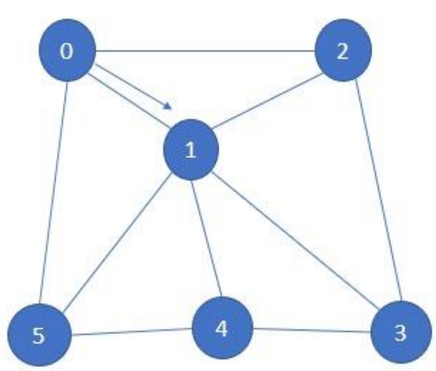
1. We create an integer array path which stores the vertices that have been traversed. And a boolean array **visited []** which checks if a given vertex has been visited or not.

2. We add starting vertex to the path.
3. Now, recursively add vertices to path one by one to find the cycle.
4. Before adding a vertex to path, check whether the vertex being considered is adjacent to the previously added vertex or not and is not already in path. If such a vertex is found, then add it to the path and mark its value as true in the **visited []** array.
5. If the length of path becomes equal to N, and there is an edge from the last vertex in path, then print the array.
6. After completing the above steps, if there exists no such path, then print **No Hamiltonian Cycle possible**.

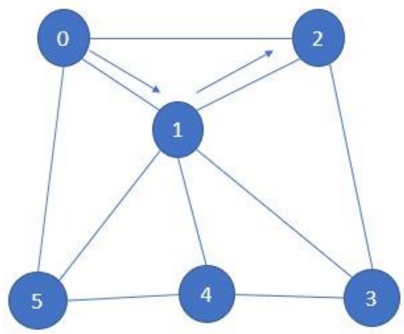
Approach:



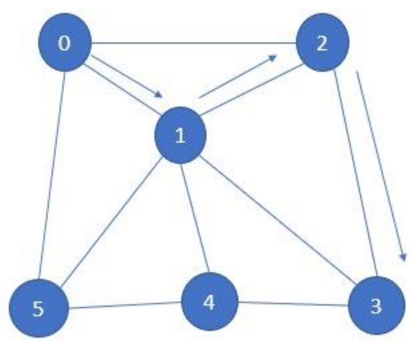
First mark our initial vertex which is 0 and add it to the path array. Now we iterate over vertices v from v=(0,N). Select the first vertex which shares an edge with 0 that has not been visited yet. In this case the vertex is 1. So, we add it to the path array**. Path: 0**



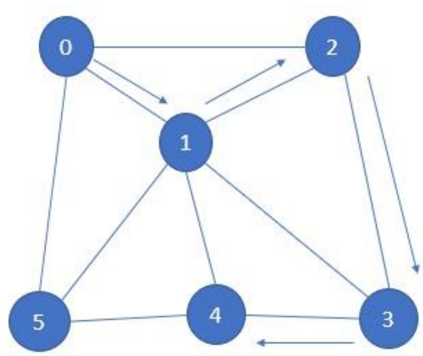
Now we repeat the same procedure for vertex 1. Vertices adjacent to 1 are 0, 2,5 and 4. We cannot go to vertex 0 as it has already been visited. Let we select 2 and add it to the path array. **Path: 0 1**



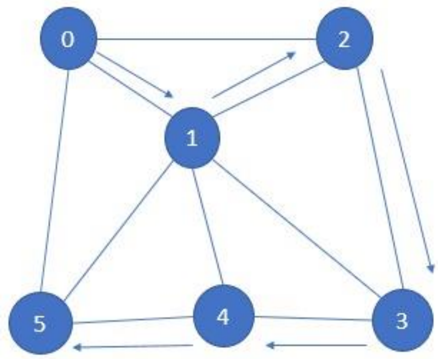
Perform same operation for vertex 2. Vertices adjacent to 2 are 0,1 and 3. We cannot go to vertex 0,1 as it has already been visited. So we select 3 and add it to the path array. **Path: 0 1 2**



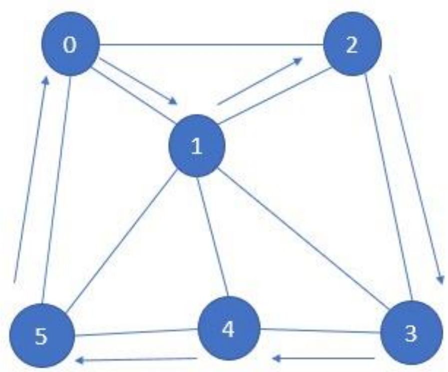
Perform same operation for vertex 3. Vertices adjacent to 3 are 2 and 4. We cannot go to vertex 2 as it has already been visited. So we select 4 and add it to the path array. **Path: 0 1 2 3**



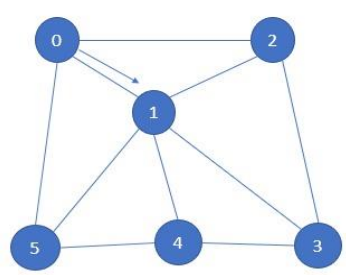
Repeat procedure for vertex 4. Vertices adjacent to 4 are 1,3 and 5. We cannot go to vertex 1 and 3 as they has already been visited. So we select 5 and add it to the path array. **Path: 0 1 2 3 4**



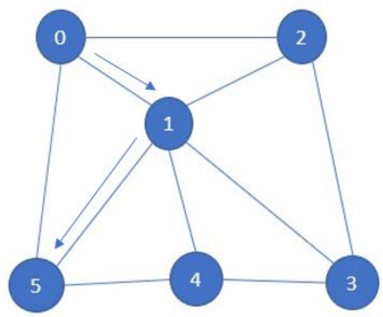
Now for vertex 5, size of the path array is equal to the number of vertices N. N = 6. We check if there is an edge from the last vertex 5 to the source vertex 0. If condition is satisfied add source vertex 0 to path array and print the result. Then pop the vertex 5 and go back to vertex 4**. Path:0 1 2 3 4 5. O/P: 0 1 2 3 4 5 0**.



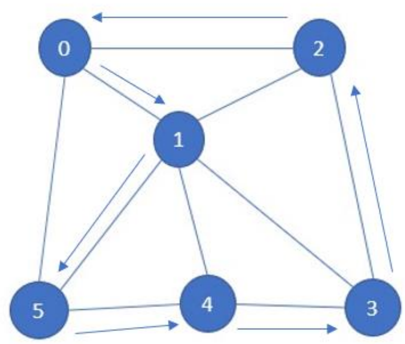
We check if there exists vertex greater than 4 and not equal to 5 that hasn't been traversed yet. Clearly there isn't so we pop value 4. Using the same logic, we pop elements 2 and 3. **Path: 0 1**



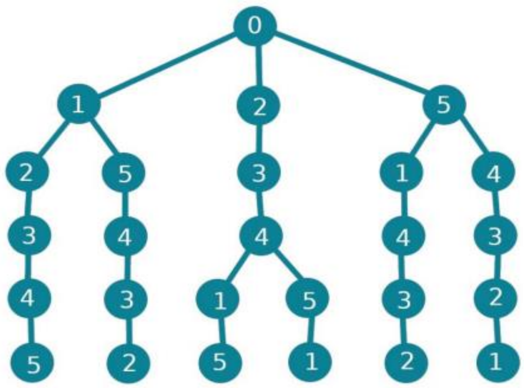
For vertex 1, vertex 4 is greater and hasn't been traversed yet so we push it to the path array. But no Hamiltonian cycle can be formed after traversing vertex 4. So, we backtrack to 5 and repeat the procedure. **Path: 0 1 5**



Hamiltonian cycle is possible for path from vertex 5. This is an example of backtracking which is used to store useful solutions and discard wrong ones. We repeat this process for all permutations. This gives a recursive call tree. **Path: 0 1 5 4 3 2 0**.



The diagram shows paths of the recursive call tree that successfully form a Hamiltonian cycle. Any sequence of vertices starting from the root to any leaf node form a Hamiltonian cycle. (We add the root node at the end). Note that we haven't shown those recursive calls that do not satisfy the above condition.



Code Implementation:

```cpp
main.cpp

// C++ program for the above approach
#include<bits/stdc++.h>
using namespace std;

// To check if there exists at least 1 hamiltonian cycle
bool hasCycle;
int cycleCount = 0;

// Function to check if a vertex v can be added at index pos in the Hamiltonian Cycle
bool isSafe(int v, int graph[][6], vector<int> path, int pos)
{

    // If the vertex is adjacent to the vertex of the previously added vertex
    if (graph[path[pos - 1]][v] == 0)
        return false;

    // If the vertex has already been included in the path
    for (int i = 0; i < pos; i++)
        if (path[i] == v)
            return false;

    // Both the above conditions are not true, return true
    return true;
}

// Recursive function to find all hamiltonian cycles
void FindHamCycle(int graph[][6], int pos, vector<int> path, bool visited[], int N)
{
    // If all vertices are included in Hamiltonian Cycle
    if (pos == N) {

        // If there is an edge from the last vertex to the source vertex
        if (graph[path[path.size() - 1]][path[0]] != 0) {

            // Include source vertex into the path and print the path
            path.push_back(0);
```

main.cpp

```cpp
            // Include source vertex into the path and print the path
            path.push_back(0);

            cycleCount++;
            for (int i = 0; i < path.size(); i++) {
                cout << path[i] << " ";
            }
            cout << endl;

            // Remove the source vertex added
            path.pop_back();

            // Update the hasCycle as true
            hasCycle = true;
        }
        return;
    }

    // Try different vertices as the next vertex
    for (int v = 0; v < N; v++) {

        // Check if this vertex can be added to Cycle
        if (isSafe(v, graph, path, pos) && !visited[v]) {

            path.push_back(v);
            visited[v] = true;

            // Recur to construct rest of the path
            FindHamCycle(graph, pos + 1, path, visited, N);

            // Remove current vertex from path and process other vertices
            visited[v] = false;
            path.pop_back();
        }
    }
}

```

main.cpp

```cpp
    }
}

// Function to find all possible hamiltonian cycles
void hamCycle(int graph[][6], int N)
{
    // Initially value of boolean flag is false
    hasCycle = false;

    // Store the resultant path
    vector<int> path;
    path.push_back(0);

    // Keeps the track of the visited vertices
    bool visited[N];

    for (int i = 0; i < N; i++)
        visited[i] = false;

    visited[0] = true;

    // Function call to find all hamiltonian cycles
    FindHamCycle(graph, 1, path, visited, N);

    if (!hasCycle) {

        // If no Hamiltonian Cycle is possible for the given graph
        cout << "No Hamiltonian Cycle" << "possible " << endl;
        return;
    }
    else {
        cout<<endl;
        cout<< "Number of Hamiltonian cycles possible:"<<cycleCount<<endl<<endl;
    }

}

int main()
```

main.cpp

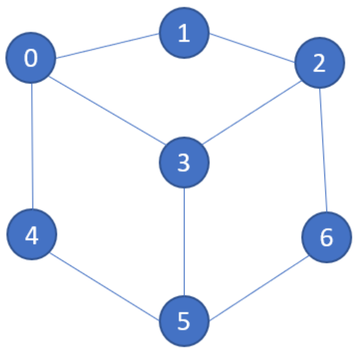
```cpp
    for (int i = 0; i < N; i++)
        visited[i] = false;

    visited[0] = true;

    // Function call to find all hamiltonian cycles
    FindHamCycle(graph, 1, path, visited, N);

    if (!hasCycle) {

        // If no Hamiltonian Cycle is possible for the given graph
        cout << "No Hamiltonian Cycle" << "possible " << endl;
        return;
    }
    else {
        cout<<endl;
        cout<< "Number of Hamiltonian cycles possible:"<<cycleCount<<endl<<endl;
    }

}

int main()
{
    int graph[][6] = {
            { 0, 1, 1, 0, 0, 1 },
            { 1, 0, 1, 0, 1, 1 },
            { 1, 1, 0, 1, 0, 0 },
            { 0, 0, 1, 0, 1, 0 },
            { 0, 1, 0, 1, 0, 1 },
            { 1, 1, 0, 0, 1, 0 },
        };
    hamCycle(graph, 6);

    return 0;
}
```

Output:

```
0 1 2 3 4 5 0
0 1 5 4 3 2 0
0 2 3 4 1 5 0
0 2 3 4 5 1 0
0 5 1 4 3 2 0
0 5 4 3 2 1 0

Number of Hamiltonian cycles possible:6



...Program finished with exit code 0
Press ENTER to exit console.
```

Examples:

1)

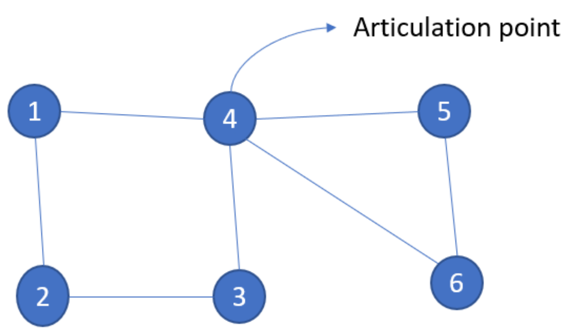


Input: graph[][]={{0,1,0,1,1,0,0}, {1,0,1,0,0,0,0} , {0,1,0,1,0,0,1}, {1,0,1,0,0,1,0},{1,0,0,0,0,1,0},{0,0,0,1,1,0},{0,0,1,0,0,1,0}}

Output:

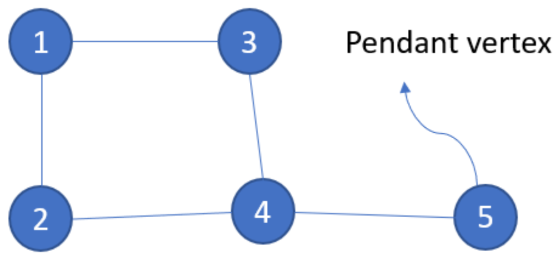
For the given graph, no Hamiltonian cycle is possible.

2)



For the given graph, no Hamiltonian cycle is possible.

3)



For the given graph, no Hamiltonian cycle is possible.

Application:

The Hamiltonian cycle (HC) problem has many applications such as time scheduling, the choice of travel routes. It is used in computer graphics, electronic circuit design, and many more. A real-life application of the Hamiltonian cycle includes genome mapping. Another example includes framing a bus route for school students. Here nodes = students, edges = paths to be traveled and all paths are to be traversed only once.