Hamiltonian Cycle | Backtracking-6

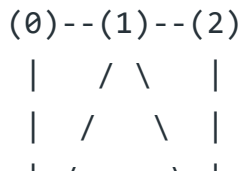
Difficulty Level : Hard • Last Updated : 06 Jan, 2020

Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian Path. 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.

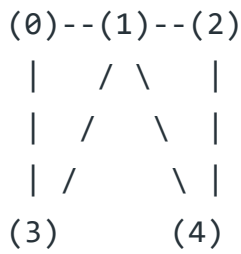
Input:
A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.

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.

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



Related Articles



Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

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.

Implementation of Backtracking solution

Following are implementations of the Backtracking solution.

```
/* C++ program for solution of Hamiltonian
Cycle problem using backtracking */
#include <bits/stdc++.h>
using namespace std;

// Number of vertices in the graph
#define V 5

void printSolution(int path[]);

/* A utility function to check if
the vertex v can be added at index 'pos'
in the Hamiltonian Cycle constructed
so far (stored in 'path[]') */
bool isSafe(int v, bool graph[V][V],
            int path[], int pos)
{
    /* Check if this vertex is an adjacent
    vertex of the previously added vertex. */
    if (graph [path[pos - 1]][ v ] == 0)
        return false;

    /* Check if the vertex has already been included.
    This step can be optimized by creating
    an array of size V */
    for (int i = 0; i < pos; i++)
        if (path[i] == v)
            return false;

    return true;
}

/* A recursive utility function
to solve hamiltonian cycle problem */
bool hamCycleUtil(bool graph[V][V],
                  int path[], int pos)
{
    /* base case: If all vertices are
    included in Hamiltonian Cycle */
    if (pos == V)
    {
        // And if there is an edge from the
        // last included vertex to the first vertex
        if (graph[path[pos - 1]][path[0]] == 1)
            return true;
        else
            return false;
    }
```

```
    // we included 0 as starting point in hamCycle()
    for (int v = 1; v < V; v++)
    {
        /* Check if this vertex can be added
        // to Hamiltonian Cycle */
        if (isSafe(v, graph, path, pos))
        {
            path[pos] = v;

            /* recur to construct rest of the path */
            if (hamCycleUtil (graph, path, pos + 1) == true)
                return true;

            /* If adding vertex v doesn't lead to a solution,
            then remove it */
            path[pos] = -1;
        }
    }

    /* If no vertex can be added to
    Hamiltonian Cycle constructed so far,
    then return false */
    return false;
}

/* This function solves the Hamiltonian Cycle problem
using Backtracking. It mainly uses hamCycleUtil() to
solve the problem. It returns false if there is no
Hamiltonian Cycle possible, otherwise return true
and prints the path. Please note that there may be
more than one solutions, this function prints one
of the feasible solutions. */
bool hamCycle(bool graph[V][V])
{
    int *path = new int[V];
    for (int i = 0; i < V; i++)
        path[i] = -1;

    /* Let us put vertex 0 as the first vertex in the path.
    If there is a Hamiltonian Cycle, then the path can be
    started from any point of the cycle as the graph is undirected */
    path[0] = 0;
    if (hamCycleUtil(graph, path, 1) == false )
    {
        cout << "\nSolution does not exist";
        return false;
    }
```

```
/* A utility function to print solution */
void printSolution(int path[])
{
    cout << "Solution Exists:"
            " Following is one Hamiltonian Cycle \n";
    for (int i = 0; i < V; i++)
        cout << path[i] << " ";

    // Let us print the first vertex again
    // to show the complete cycle
    cout << path[0] << " ";
    cout << endl;
}

// Driver Code
int main()
{
    /* Let us create the following graph
        (0)--(1)--(2)
        | / \ |
        | / \ |
        | / \ |
        (3)-------(4) */
    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}};

    // Print the solution
    hamCycle(graph1);

    /* Let us create the following graph
    (0)--(1)--(2)
    | / \ |
    | / \ |
    | / \ |
    (3) (4) */
    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}};

    // Print the solution
    hamCycle(graph2);
```

```
// This is code is contributed by rathbhupendra
```

C

```
/* C program for solution of Hamiltonian Cycle problem
   using backtracking */
#include<stdio.h>

// Number of vertices in the graph
#define V 5

void printSolution(int path[]);

/* A utility function to check if the vertex v can be added at
   index 'pos' in the Hamiltonian Cycle constructed so far (stored
   in 'path[]') */
bool isSafe(int v, bool graph[V][V], int path[], int pos)
{
    /* Check if this vertex is an adjacent vertex of the previously
       added vertex. */
    if (graph [ path[pos-1] ][ v ] == 0)
        return false;

    /* Check if the vertex has already been included.
      This step can be optimized by creating an array of size V */
    for (int i = 0; i < pos; i++)
        if (path[i] == v)
            return false;

    return true;
}

/* A recursive utility function to solve hamiltonian cycle problem */
bool hamCycleUtil(bool graph[V][V], int path[], int pos)
{
    /* base case: If all vertices are included in Hamiltonian Cycle */
    if (pos == V)
    {
        // And if there is an edge from the last included vertex to the
        // first vertex
        if ( graph[ path[pos-1] ][ path[0] ] == 1 )
           return true;
        else
          return false;
    }
```

```
    {
        /* Check if this vertex can be added to Hamiltonian Cycle */
        if (isSafe(v, graph, path, pos))
        {
            path[pos] = v;

            /* recur to construct rest of the path */
            if (hamCycleUtil (graph, path, pos+1) == true)
                return true;

            /* If adding vertex v doesn't lead to a solution,
               then remove it */
            path[pos] = -1;
        }
    }

    /* If no vertex can be added to Hamiltonian Cycle constructed so far,
       then return false */
    return false;
}

/* This function solves the Hamiltonian Cycle problem using Backtracking.
  It mainly uses hamCycleUtil() to solve the problem. It returns false
  if there is no Hamiltonian Cycle possible, otherwise return true and
  prints the path. Please note that there may be more than one solutions,
  this function prints one of the feasible solutions. */
bool hamCycle(bool graph[V][V])
{
    int *path = new int[V];
    for (int i = 0; i < V; i++)
        path[i] = -1;

    /* Let us put vertex 0 as the first vertex in the path. If there is
       a Hamiltonian Cycle, then the path can be started from any point
       of the cycle as the graph is undirected */
    path[0] = 0;
    if ( hamCycleUtil(graph, path, 1) == false )
    {
        printf("\nSolution does not exist");
        return false;
    }

    printSolution(path);
    return true;
}

/* A utility function to print solution */
```

```
    for (int i = 0; i < V; i++)
        printf(" %d ", path[i]);

    // Let us print the first vertex again to show the complete cycle
    printf(" %d ", path[0]);
    printf("\n");
}

// driver program to test above function
int main()
{
   /* Let us create the following graph
      (0)--(1)--(2)
       |   / \   |
       |  /   \  |
       | /     \ |
      (3)-------(4)    */
   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},
                       };

    // Print the solution
    hamCycle(graph1);

   /* Let us create the following graph
      (0)--(1)--(2)
       |   / \   |
       |  /   \  |
       | /     \ |
      (3)       (4)    */
    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},
                        };

    // Print the solution
    hamCycle(graph2);

    return 0;
}
```

```
   using backtracking */
class HamiltonianCycle
{
    final int V = 5;
    int path[];

    /* A utility function to check if the vertex v can be
       added at index 'pos'in the Hamiltonian Cycle
       constructed so far (stored in 'path[]') */
    boolean isSafe(int v, int graph[][], int path[], int pos)
    {
        /* Check if this vertex is an adjacent vertex of
           the previously added vertex. */
        if (graph[path[pos - 1]][v] == 0)
            return false;

        /* Check if the vertex has already been included.
           This step can be optimized by creating an array
           of size V */
        for (int i = 0; i < pos; i++)
            if (path[i] == v)
                return false;

        return true;
    }

    /* A recursive utility function to solve hamiltonian
       cycle problem */
    boolean hamCycleUtil(int graph[][], int path[], int pos)
    {
        /* base case: If all vertices are included in
           Hamiltonian Cycle */
        if (pos == V)
        {
            // And if there is an edge from the last included
            // vertex to the first vertex
            if (graph[path[pos - 1]][path[0]] == 1)
                return true;
            else
                return false;
        }

        // Try different vertices as a next candidate in
        // Hamiltonian Cycle. We don't try for 0 as we
        // included 0 as starting point in hamCycle()
        for (int v = 1; v < V; v++)
        {
```

```
            path[pos] = v;

            /* recur to construct rest of the path */
            if (hamCycleUtil(graph, path, pos + 1) == true)
                return true;

            /* If adding vertex v doesn't lead to a solution,
               then remove it */
            path[pos] = -1;
        }
    }

    /* If no vertex can be added to Hamiltonian Cycle
       constructed so far, then return false */
    return false;
}

/* This function solves the Hamiltonian Cycle problem using
   Backtracking. It mainly uses hamCycleUtil() to solve the
   problem. It returns false if there is no Hamiltonian Cycle
   possible, otherwise return true and prints the path.
   Please note that there may be more than one solutions,
   this function prints one of the feasible solutions. */
int hamCycle(int graph[][])
{
    path = new int[V];
    for (int i = 0; i < V; i++)
        path[i] = -1;

    /* Let us put vertex 0 as the first vertex in the path.
       If there is a Hamiltonian Cycle, then the path can be
       started from any point of the cycle as the graph is
       undirected */
    path[0] = 0;
    if (hamCycleUtil(graph, path, 1) == false)
    {
        System.out.println("\nSolution does not exist");
        return 0;
    }

    printSolution(path);
    return 1;
}

/* A utility function to print solution */
void printSolution(int path[])
{
```

```
        // Let us print the first vertex again to show the
        // complete cycle
        System.out.println(" " + path[0] + " ");
    }

    // driver program to test above function
    public static void main(String args[])
    {
        HamiltonianCycle hamiltonian =
                                new HamiltonianCycle();
        /* Let us create the following graph
           (0)--(1)--(2)
            |   / \   |
            |  /   \  |
            | /     \ |
           (3)-------(4)    */
        int graph1[][] = {{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},
        };

        // Print the solution
        hamiltonian.hamCycle(graph1);

        /* Let us create the following graph
           (0)--(1)--(2)
            |   / \   |
            |  /   \  |
            | /     \ |
           (3)       (4)    */
        int graph2[][] = {{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},
        };

        // Print the solution
        hamiltonian.hamCycle(graph2);
    }
}
// This code is contributed by Abhishek Shankhadhar
```

```
# hamiltonian cycle problem

class Graph():
    def __init__(self, vertices):
        self.graph = [[0 for column in range(vertices)]
                            for row in range(vertices)]
        self.V = vertices

    ''' Check if this vertex is an adjacent vertex
        of the previously added vertex and is not
        included in the path earlier '''
    def isSafe(self, v, pos, path):
        # Check if current vertex and last vertex
        # in path are adjacent
        if self.graph[ path[pos-1] ][v] == 0:
            return False

        # Check if current vertex not already in path
        for vertex in path:
            if vertex == v:
                return False

        return True

    # A recursive utility function to solve
    # hamiltonian cycle problem
    def hamCycleUtil(self, path, pos):

        # base case: if all vertices are
        # included in the path
        if pos == self.V:
            # Last vertex must be adjacent to the
            # first vertex in path to make a cyle
            if self.graph[ path[pos-1] ][ path[0] ] == 1:
                return True
            else:
                return False

        # Try different vertices as a next candidate
        # in Hamiltonian Cycle. We don't try for 0 as
        # we included 0 as starting point in hamCycle()
        for v in range(1,self.V):

            if self.isSafe(v, pos, path) == True:

                path[pos] = v
```

```
                # lead to a solution
                path[pos] = -1

        return False

    def hamCycle(self):
        path = [-1] * self.V

        ''' Let us put vertex 0 as the first vertex
            in the path. If there is a Hamiltonian Cycle,
            then the path can be started from any point
            of the cycle as the graph is undirected '''
        path[0] = 0

        if self.hamCycleUtil(path,1) == False:
            print ("Solution does not exist\n")
            return False

        self.printSolution(path)
        return True

    def printSolution(self, path):
        print ("Solution Exists: Following",
                "is one Hamiltonian Cycle")
        for vertex in path:
            print (vertex, end = " ")
        print (path[0], "\n")

# Driver Code

''' Let us create the following graph
    (0)--(1)--(2)
    | / \ |
    | / \ |
    | /     \ |
    (3)-------(4) '''
g1 = Graph(5)
g1.graph = [ [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], ]

# Print the solution
g1.hamCycle();

''' Let us create the following graph
    (0)--(1)--(2)
    | / \ |
```

```
g2.graph = [ [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], ]

# Print the solution
g2.hamCycle();

# This code is contributed by Divyanshu Mehta
```

C#

```
// C# program for solution of Hamiltonian
// Cycle problem using backtracking
using System;

public class HamiltonianCycle
{
    readonly int V = 5;
    int []path;

    /* A utility function to check
    if the vertex v can be added at
    index 'pos'in the Hamiltonian Cycle
    constructed so far (stored in 'path[]') */
    bool isSafe(int v, int [,]graph,
                int []path, int pos)
    {
        /* Check if this vertex is
        an adjacent vertex of the
        previously added vertex. */
        if (graph[path[pos - 1], v] == 0)
            return false;

        /* Check if the vertex has already
        been included. This step can be
        optimized by creating an array
        of size V */
        for (int i = 0; i < pos; i++)
            if (path[i] == v)
                return false;

        return true;
    }

    /* A recursive utility function
```

```
        are included in Hamiltonian Cycle */
        if (pos == V)
        {
            // And if there is an edge from the last included
            // vertex to the first vertex
            if (graph[path[pos - 1],path[0]] == 1)
                return true;
            else
                return false;
        }

        // Try different vertices as a next candidate in
        // Hamiltonian Cycle. We don't try for 0 as we
        // included 0 as starting point in hamCycle()
        for (int v = 1; v < V; v++)
        {
            /* Check if this vertex can be
            added to Hamiltonian Cycle */
            if (isSafe(v, graph, path, pos))
            {
                path[pos] = v;

                /* recur to construct rest of the path */
                if (hamCycleUtil(graph, path, pos + 1) == true)
                    return true;

                /* If adding vertex v doesn't
                lead to a solution, then remove it */
                path[pos] = -1;
            }
        }

        /* If no vertex can be added to Hamiltonian Cycle
        constructed so far, then return false */
        return false;
    }

    /* This function solves the Hamiltonian
    Cycle problem using Backtracking. It
    mainly uses hamCycleUtil() to solve the
    problem. It returns false if there
    is no Hamiltonian Cycle possible,
    otherwise return true and prints the path.
    Please note that there may be more than
    one solutions, this function prints one
    of the feasible solutions. */
    int hamCycle(int [,]graph)
```

```
    /* Let us put vertex 0 as the first
    vertex in the path. If there is a
    Hamiltonian Cycle, then the path can be
    started from any point of the cycle
    as the graph is undirected */
    path[0] = 0;
    if (hamCycleUtil(graph, path, 1) == false)
    {
        Console.WriteLine("\nSolution does not exist");
        return 0;
    }

    printSolution(path);
    return 1;
}

/* A utility function to print solution */
void printSolution(int []path)
{
    Console.WriteLine("Solution Exists: Following" +
                    " is one Hamiltonian Cycle");
    for (int i = 0; i < V; i++)
        Console.Write(" " + path[i] + " ");

    // Let us print the first vertex again
    //  to show the complete cycle
    Console.WriteLine(" " + path[0] + " ");
}

// Driver code
public static void Main(String []args)
{
    HamiltonianCycle hamiltonian =
                            new HamiltonianCycle();
    /* Let us create the following graph
    (0)--(1)--(2)
        | / \ |
        | / \ |
        | /     \ |
    (3)-------(4) */
    int [,]graph1= {{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},
    };
```

```
        /* Let us create the following graph
        (0)--(1)--(2)
            | / \ |
            | / \ |
            | /     \ |
        (3)     (4) */
        int [,]graph2 = {{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},
        };

        // Print the solution
        hamiltonian.hamCycle(graph2);
    }
}

// This code contributed by Rajput-Ji
```

PHP

```
<?php
// PHP program for solution of
// Hamiltonian Cycle problem
// using backtracking
$V = 5;

/* A utility function to check if
the vertex v can be added at index 'pos'
in the Hamiltonian Cycle constructed so far
(stored in 'path[]') */
function isSafe($v, $graph, &$path, $pos)
{
    /* Check if this vertex is
    an adjacent vertex of the
    previously added vertex. */
    if ($graph[$path[$pos - 1]][$v] == 0)
        return false;

    /* Check if the vertex has already been included.
    This step can be optimized by creating an array
    of size V */
    for ($i = 0; $i < $pos; $i++)
        if ($path[$i] == $v)
```

```php
/* A recursive utility function
to solve hamiltonian cycle problem */
function hamCycleUtil($graph, &$path, $pos)
{
    global $V;

    /* base case: If all vertices are included in
    Hamiltonian Cycle */
    if ($pos == $V)
    {
        // And if there is an edge from the
        // last included vertex to the first vertex
        if ($graph[$path[$pos - 1]][$path[0]] == 1)
            return true;
        else
            return false;
    }

    // Try different vertices as a next candidate in
    // Hamiltonian Cycle. We don't try for 0 as we
    // included 0 as starting point hamCycle()
    for ($v = 1; $v < $V; $v++)
    {
        /* Check if this vertex can be added
        to Hamiltonian Cycle */
        if (isSafe($v, $graph, $path, $pos))
        {
            $path[$pos] = $v;

            /* recur to construct rest of the path */
            if (hamCycleUtil($graph, $path,
                                    $pos + 1) == true)
                return true;

            /* If adding vertex v doesn't lead to a solution,
            then remove it */
            $path[$pos] = -1;
        }
    }

    /* If no vertex can be added to Hamiltonian Cycle
    constructed so far, then return false */
    return false;
}

/* This function solves the Hamiltonian Cycle problem using
```

```
this function prints one of the feasible solutions. */
function hamCycle($graph)
{
    global $V;
    $path = array_fill(0, $V, 0);
    for ($i = 0; $i < $V; $i++)
        $path[$i] = -1;

    /* Let us put vertex 0 as the first vertex in the path.
    If there is a Hamiltonian Cycle, then the path can be
    started from any point of the cycle as the graph is
    undirected */
    $path[0] = 0;
    if (hamCycleUtil($graph, $path, 1) == false)
    {
        echo("\nSolution does not exist");
        return 0;
    }

    printSolution($path);
    return 1;
}

/* A utility function to print solution */
function printSolution($path)
{
    global $V;
    echo("Solution Exists: Following is ".
         "one Hamiltonian Cycle\n");
    for ($i = 0; $i < $V; $i++)
        echo(" ".$path[$i]." ");

    // Let us print the first vertex again to show the
    // complete cycle
    echo(" ".$path[0]." \n");
}


// Driver Code

/* Let us create the following graph
(0)--(1)--(2)
    | / \ |
    | / \ |
    | / \ |
(3)-------(4) */
$graph1 = array(array(0, 1, 0, 1, 0),
```

```
);

// Print the solution
hamCycle($graph1);

/* Let us create the following graph
(0)--(1)--(2)
    | / \ |
    | / \ |
    | / \ |
(3) (4) */
$graph2 = array(array(0, 1, 0, 1, 0),
                array(1, 0, 1, 1, 1),
                array(0, 1, 0, 0, 1),
                array(1, 1, 0, 0, 0),
                array(0, 1, 1, 0, 0));

// Print the solution
hamCycle($graph2);

// This code is contributed by mits
?>
```

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 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++)".

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Next

Sudoku | Backtracking-7

RECOMMENDED ARTICLES

01 **Number of Hamiltonian cycle**
14, Feb 19

02 **Proof that Hamiltonian Cycle is NP-Complete**
17, Jun 20

03 **Proof that Hamiltonian Path is NP-Complete**
04, Jun 18

04 **Hamiltonian Path (Using Dynamic Programming)**
09, Apr 21

05 **Detect Cycle in a Directed Graph**
03, Apr 12

06 **Disjoint Set (Or Union-Find) | Set 1 (Detect Cycle in an Undirected Graph)**
28, Oct 12

07 **Detect cycle in an undirected graph**
11, Oct 13

08 **Detect Cycle in a directed graph using colors**
12, Mar 16

Article Contributed By :

GeeksforGeeks

Easy | Normal | Medium | Hard | Expert

Improved By : Vidhayak_Chacha, Rajput-Ji, rathbhupendra, md1844, Mithun Kumar, Akanksha_Rai

Article Tags : Amazon, DFS, Backtracking, Graph

Practice Tags : Amazon, DFS, Graph, Backtracking

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Practice

Courses

Company-wise

Topic-wise

How to begin?

Contribute

Write an Article

Write Interview Experience

Internships

Videos