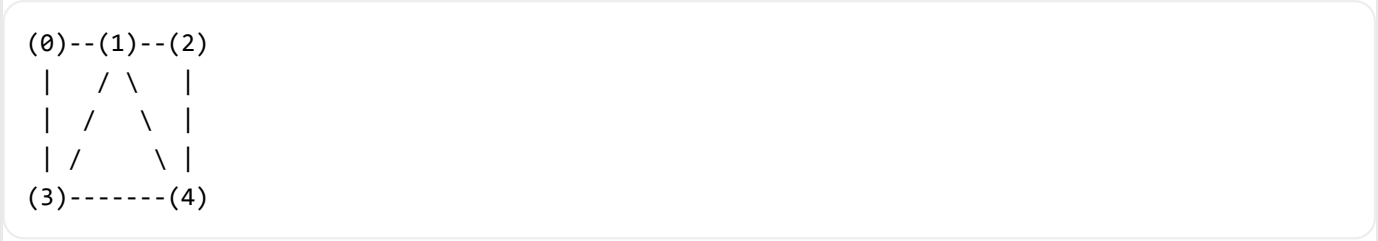
Backtracking | Set 6 (Hamiltonian Cycle)

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 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 print the path. Following are the input and output of the required function.

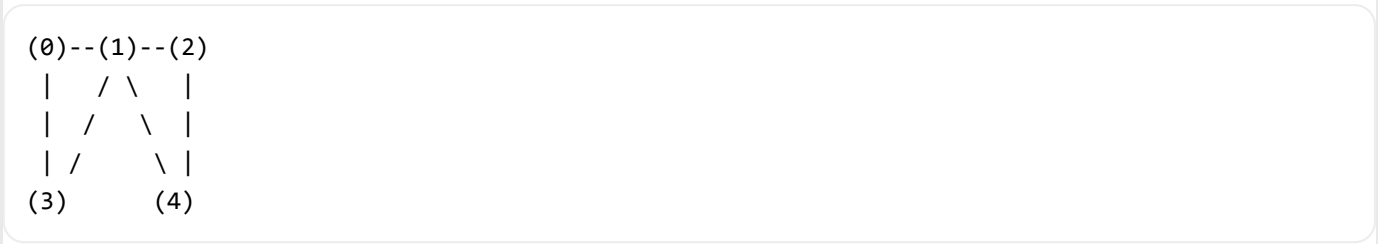
Input:
A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.

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}. There are more Hamiltonian Cycles in the graph like {0, 3, 4, 2, 1, 0}



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



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/C++

```
/* 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;
    }

    // Try different vertices as a next candidate in Hamiltonian Cycle.
    // We don't try for 0 as we included 0 as starting point in 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 )
    {
        printf("\nSolution does not exist");
        return false;
    }

    printSolution(path);
    return true;
}

/* A utility function to print solution */
void printSolution(int path[])
{
    printf ("Solution Exists:"
            " Following is one Hamiltonian Cycle \n");
    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;
}
```

Run on IDE

Java

```
/* Java program for solution of Hamiltonian Cycle problem
   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 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[][])
    {
        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[])
{
    System.out.println("Solution Exists: Following" +
                       " is one Hamiltonian Cycle");
    for (int i = 0; i < V; i++)
        System.out.print(" " + path[i] + " ");

    // 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
```

Run on IDE

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. Starting point should not matter as cycle can be started from any point. If you want to change the starting point, you should make two changes to 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.

52 Comments Category: Graph Tags: Backtracking, Graph

Related Posts:

- Iterative Depth First Traversal of Graph
- Print all Jumping Numbers smaller than or equal to a given value
- Hopcroft–Karp Algorithm for Maximum Matching | Set 2 (Implementation)
- Hopcroft–Karp Algorithm for Maximum Matching | Set 1 (Introduction)
- Length of shortest chain to reach a target word
- Find same contacts in a list of contacts

- Minimum time required to rot all oranges
- Karger's algorithm for Minimum Cut | Set 2 (Analysis and Applications)

(Login to Rate and Mark)

3 Average Difficulty : **3/5.0**
Based on **3** vote(s)

Add to TODO List
Mark as DONE

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

52 Comments GeeksforGeeks 1 Login

Recommend 5 Share Sort by Newest

Join the discussion…

Krishna Kumar • 2 months ago
I have tried to solve this in java in my post (solution is based on the concept presented here) :

http://krishnalearnings.blogsp...

• Reply • Share ›

Harsh Jain • 3 months ago
Correct me if I am wrong but this problem looks like just another version of DFS. Keep recursing until you run out of unvisited adjacent vertices, then check if last node has a connection to the first node. If so, check if all vertices have been visited in the current path.

• Reply • Share ›

Krishna Kumar Harsh Jain • 2 months ago
hmm.actually in usual DFS we traverse all the vertices but order does not matter. Here we need to traverse all the nodes but in a special order.

• Reply • Share ›

Jeff • 6 months ago
Prints Hamcycle And if HamCycle is not present then prints the HamPath if present.
http://ideone.com/JcKaXJ

• Reply • Share ›

soumyajit bhattacharyay • 6 months ago

c++ easy implementation

http://ideone.com/GjLC2W

• Reply • Share ›

phantom • 7 months ago

This doesn't print all the hamiltonian cycles; it only prints the path if it exists starting from 0. What if I want to print any Hamiltonian Cycle?

My suggestion would be to traverse from v = 0 through N in the function hamilCycleUtil

• Reply • Share ›

dev21 • 7 months ago

Easy implementation using STL. Prints Hamilton path as well is hamilton cycle doesnot exist. https://ideone.com/OLJwNP

2 • Reply • Share ›

Ayush Srivastava • 7 months ago

http://ideone.com/ysUUcQ

• Reply • Share ›

vikas mangal • 8 months ago

Do we really need to write **path[pos] = -1** in hamCycleUtil(). What difference does it make if we don't write it.

• Reply • Share ›

Anshu Singh ➚ vikas mangal • 7 months ago

Backtracking is not possible without this step. It's an indication that the previously considered vertex is wrong, hence needs to be backtracked.

• Reply • Share ›

Mr. Lazy • 9 months ago

http://ideone.com/uCETqY

A better and bit efficient version of the above implementation!. For those who didn't understand the above code check this one. Its commented well. :)

3 • Reply • Share ›

Billionaire ➚ Mr. Lazy • 4 months ago

Vote up for simpler code and use visited[] array

• Reply • Share ›

Santhosh • a year ago

Same thing in java

```
package com.santhosh.test;

import java.util.ArrayList;

public class HamiltonCycle {

public static ArrayList<graphnode> visitedNodes = new ArrayList<graphnode>();

public static boolean isHamiltonCycle(GraphPath path, GraphNode A) {

System.out.println("Checking for Path From:" + A);

if (visitedNodes.size() > path.node.size()) {

if (visitedNodes.get(0).equals(A)) {

System.out.println("yes Hamilton Graph");

return true;
```

see more

• Reply • Share ›

Dheeraj Sachan • a year ago

https://github.com/dheeraj9198...

• Reply • Share ›

Meena Chaudhary • a year ago

I solved it using DFS and time complexity is O(V^2) Below is the code. Hope it helps.

```
public boolean check() {

Stack<node> nodeStack = new Stack<>();

nodeStack.add(root);

root.setIsOnStack();

while (!nodeStack.isEmpty()) { //O(V), where V is total number of vertices

Node currentNode = nodeStack.peek(); //1

System.out.println(currentNode.getLabel());

for (Entry<node, boolean=""> entry : currentNode.getNeighbourList().entrySet()) { // V
```

```
Node currentNeighbourer = entry.getKey();

if (!currentNeighbourer.isOnStack()) { // 1
```

see more

• Reply • Share ›

Billionaire ➚ Meena Chaudhary • 4 months ago

This is impossible, since it time complexity is O(N!)
http://stackoverflow.com/a/200...

• Reply • Share ›

Harsh Kara • a year ago

@geeksforgeeks
IS THIS TRUE ?
if the maximun length of cycle is equal to the number of vertices then we can say that hamilton cycle is present otherwise it's NOT
so we can easily inplement this by dfs
please correct me if wrong :)

• Reply • Share ›

Arpit ➚ Harsh Kara • a year ago

also the cycle should contain all the vertices

EXACTLY ONCE except the starting vertex considering it to be a cycle.

• Reply • Share ›

arjomanD • 2 years ago

i think my code is much simpler

http://paste.ubuntu.com/743219...

• Reply • Share ›

arjomanD • 2 years ago

I had a question . (or maybe im wrong cuz i havn't still read the code) does this program find
Hamiltonian path ?

1 • Reply • Share ›

hxgxs1 • 2 years ago

time complexity should be O(N!)..there id a for loop in a recursive call so...
T(N)=N*(T(N-1) + O(1)) or
T(N)=N*(N-1)*(N-2)...=O(N!)

1 • Reply • Share ›

Billionaire ➚ hxgxs1 • 4 months ago

Exactly!

http://stackoverflow.com/a/200...

• Reply • Share ›

AlienOnEarth • 2 years ago

What would be the time complexity for this problem? Wikipedia says it can not be solved in Polynomial time as its a NP-Complete problem

• Reply • Share ›

Billionaire ➚ AlienOnEarth • 4 months ago

O(N!)

http://stackoverflow.com/a/200...

• Reply • Share ›

iyer ➚ AlienOnEarth • a year ago

i have the same question. did you get the answer?

• Reply • Share ›

Herman ➚ iyer • 4 months ago

NP-Complete does not mean Non-polynomial. It means not-deterministic polynomial. So the time is indeed polynomial.

• Reply • Share ›

Guest • 2 years ago

bool hamCycle(bool graph[V][V],bool*seen,int vertex)

{
int i;
for(i=0;i<v;i++) {="" if(graph[vertex][i]="=1" &&="" seen[i]="=false)" {="" seen[i]="true;" stack[++top]="i;" if(hamcycle(graph,seen,i))="" return="" true;="" else="" top--;="" }="" }="" return="" false;="" }="">

2 • Reply • Share ›

Guest • 2 years ago

include <stdio.h>
#include <stdbool.h>
// Number of vertices in the graph
#define V 5

void printSolution(int path[]);

```
int stack[V];
int top;
bool hamCycle(bool graph[V][V],bool*seen,int vertex)

{
int i;
for(i=0;i<v;i++) {="" if(graph[vertex][i]="=1" &&="" seen[i]="=false)" {="" seen[i]="true;"
stack[++top]="i;" if(hamcycle(graph,seen,i))="" return="" true;="" else="" top--;="" }="" }="
return="" false;="" }="" 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="" bool="" seen[v]="
{false};" seen[0]="true;" stack[0]="0;" hamcycle(graph1,seen,0)="" if(top="V)"
printf("there="" exists="" hamcycle\n");="" int="" i;="" for(i="0;i&amp;lt;V;i++)" printf("%d="" --
="">  ",stack[i] );
return 0;
}
```

• Reply • Share ›

da3m0n • 2 years ago

Time Complexity please..

• Reply • Share ›

Billionaire ➚ da3m0n • 4 months ago

It is O(N!)

http://stackoverflow.com/a/200...

• Reply • Share ›

Guest • 2 years ago

Algorithm for finding whether a graph is Hamiltonian or not:
Do DFS &maintain count of the adjacent unvisited vertices
If (there is no adjacent unvisited vertex){
if the count is equal to the num of vertices
return true;
else
return false;
}

8 • Reply • Share ›

Siya ➚ Guest • 8 months ago

There is a check whether all the vertices's are present but here is no condition to check whether every vertex is visited exactly once. I am not convinced with your

check whether every vertex is visited exactly once. I am not convinced with your solution.

• Reply • Share ›

Sriharsha g.r.v Guest • 2 years ago

hi can u pls elaborate and explain with example..

1 • Reply • Share ›

Born Actor • 3 years ago

[sourcecode language="C++"]
/* #include <iostream>
#include<string>
#include<sstream>
#include<iomanip>
include <stdio.h>
include <math.h>
#include <vector>
#include <stdlib.h>
using namespace std;
int a[50][50];
int n;

std::vector < pair <int, int > >edges;
int visited[50];
int cycle(int node);
void print();
int main()

see more

• Reply • Share ›

AMIT • 3 years ago

I think a minor improvement is possible...using a matrix soln to indicate which indices has already been visited,we can find isSafe in O(1)time.

```
 /* Paste your code here (You may delete these lines if not writing code) */
// Program to print Hamiltonian cycle
#include<stdio.h>

// Number of vertices in the graph
#define V 5
void print(int soln[V])
{
    int i,j;
    printf("solution exists\n");
```

```
    for(i=0;i<V;i++,printf("\n"))
    //for(j=0;j<V;j++)
    printf("%d  ",soln[i]);
    printf("\n\n");
}
```

see more

• Reply • Share ›

begfairouz AMIT • 2 years ago

Hi,

In the beggining i have to thank you for your program, it works well.
But i tried to use it for a graph[V][V] with V=120 but the program doesn't return result (neither positive or negative). Did you try it for big matrices?

• Reply • Share ›

neham • 3 years ago

just a quick question - Hamiltonian Cycle belongs to NP-Complete problem how can we solve it using backtracking technique?

• Reply • Share ›

GeeksforGeeks • 3 years ago

Ankit: The main task of program is to find out whether a given graph contains Hamiltonian Cycle or not. And if there is a cycle, then print the cycle. The above program always does that. The starting point of cycle doesn&#039t matter, a cycle is a cycle and can be printed in any way.

If you don&#039t like 0 as a starting point of cycle, we have added a note below to change the starting point :)

Also, we have removed self loops in the matrix representation of graph. Keep it up!

• Reply • Share ›

Ankit Paharia • 3 years ago

The given input graph[][] has a bug.... it won&#039t work for all the input vertices such as 1,2,3,4... Why input graph is considered having self loops like graph[0][0]=1, graph[1][1]=1 etc.
It should be -
bool graph[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},

};.

If the loop in bool hamCycleUtil() is changed as -
for (int v = 0; v < V; v++){}.
it will work for all the vertices having hamiltonian cycle.... please check if I am correct or not....

• Reply • Share ›

NNavneet • 3 years ago

why in the input graph[i][i]=1 , if we start with vertex 1 or any other vertex than 0 as starting vertex then code does not return correct path , it actually return that there is no path existing.

you correct this by making graph[i][i]=0 for all i's, and stating the for from v=0 in the hamCycleUtil function
Here is the code :

```
 #include<stdio.h>
#include<iostream>
// Number of vertices in the graph
#define V 5
using namespace std;

void printSolution(int path[])
{
        for(int i=0;i<V;i++)
        {
```

see more

• Reply • Share ›

GeeksforGeeks ➚ NNavneet • 3 years ago

The main task of program is to find out whether a given graph contains Hamiltonian Cycle or not. And if there is a cycle, then print the cycle. The above program always does that. The starting point of cycle doesn't matter, a cycle is a cycle and can be printed in any way.

If you don't like 0 as a starting point of cycle, we have added a note below the code to change the starting point :)

Also, we have removed self loops in the matrix representation of graph. Keep it up!

• Reply • Share ›

Anshul Agrawal • 3 years ago

grt work buddy..really helped me alot..

• Reply • Share ›

AAZ • 3 years ago

What would be the runtime complexity of the program ?

• Reply • Share ›

Billionaire ➚ AAZ • 4 months ago

O(N!)

http://stackoverflow.com/a/200...

• Reply • Share ›

spark9 • 4 years ago

IsSafe function complexity can be improved by using better data structures. We can use an array[V] to mark whether a vertex is visited or not.

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

atul • 4 years ago

code will not work for all inputs

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

kartik ➚ atul • 4 years ago

Please provide more details of your comment. Why do think that the code will not work? Do you have some sample graph for which it didn't work?

• Reply • Share ›

atul • 4 years ago

why assumption has been made that node 0 is connected to node 1.
if given graph is the one given below..then code will fail right??

```
(0)--(4)--(3)
 |   / \   |
 |  /   \  |
 | /     \ |
```

```
(2)-------(1)
```

[sourcecode language=""]
/* Paste your code here (You may delete these lines if not writing code) */

• Reply • Share ›

atul atul • 4 years ago

ignore above comment

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

kavish • 4 years ago

good post....a more generalised version of knight's tour problem...

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

Load more comments