

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.

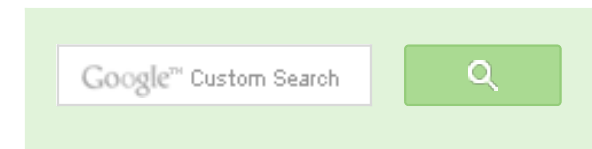
Output:

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}

```
(0) -- (1) -- (2)
|   /  \   |
|   /    \  |
|  /      \ |
(3) ----- (4)
```

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



GeeksforGeeks



53,525 people like [GeeksforGeeks](#).



[Interview Experiences](#)

[Advanced Data Structures](#)

[Dynamic Programming](#)

[Greedy Algorithms](#)

[Backtracking](#)

[Pattern Searching](#)

[Divide & Conquer](#)

[Mathematical Algorithms](#)

[Recursion](#)

[Geometric Algorithms](#)

(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 configurations
{
    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 is C/C++ implementation of the Backtracking solution.

```
// Program to print Hamiltonian cycle
#include<stdio.h>

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

void printSolution(int path[]);
```

Geometric Algorithms



Popular Posts

[All permutations of a given string](#)

[Memory Layout of C Programs](#)

[Understanding "extern" keyword in C](#)

[Median of two sorted arrays](#)

[Tree traversal without recursion and without stack!](#)

[Structure Member Alignment, Padding and Data Packing](#)

[Intersection point of two Linked Lists](#)

[Lowest Common Ancestor in a BST.](#)

[Check if a binary tree is BST or not](#)

[Sorted Linked List to Balanced BST](#)



This year,
make **progress**
on your money.

I'm ready



LV LEARNVE\$T™
PLANNING SERVICES

```
/* 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 ham
    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;
        }
    }
}
```



Recent Comments

Abhi You live US or India?

Google (Mountain View) interview · 8 minutes ago

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 47 minutes ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 51 minutes ago

Sanjay Agarwal bool

tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 1 hour ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 1 hour ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 1 hour ago

AdChoices

[► C++ Code](#)

[► Programming C++](#)

[► Java Source Code](#)

```

        /* 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 f
   then return false */
return false;
}

/* This function solves the Hamiltonian Cycle problem using Backtracki
   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 solutio
   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 i
       a Hamiltonian Cycle, then the path can be started from any poin
       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;
}


```

Output:

Solution Exists: Following is one Hamiltonian Cycle

0 1 2 4 3 0


Solution does not exist

AdChoices 

[► Graph C++](#)

[► C++ Example](#)

[► C++ Java](#)

AdChoices 

[► Graph Theory](#)

[► C++ Program](#)

[► Test C++](#)

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.

Free C# Code Generator



Create database & reporting apps
straight from your database! Try it



Related Tpoics:

- [Some interesting shortest path questions | Set 1](#)
- [Graph Coloring | Set 2 \(Greedy Algorithm\)](#)
- [Graph Coloring | Set 1 \(Introduction and Applications\)](#)
- [Johnson's algorithm for All-pairs shortest paths](#)
- [Travelling Salesman Problem | Set 2 \(Approximate using MST\)](#)
- [Travelling Salesman Problem | Set 1 \(Naive and Dynamic Programming\)](#)
- [Detect cycle in an undirected graph](#)
- [Find maximum number of edge disjoint paths between two vertices](#)



7



Tweet

0



0

Writing code in comment? Please use ideone.com and share the link here.

27 Comments**GeeksforGeeks**

Sort by Newest ▼



Join the discussion...

**arjomanD** · 2 days ago

i think my code is much simpler

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

^ | ▼ ·

**arjomanD** · 2 days ago

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

^ | ▼ ·

**hxgxs1** · 15 days agotime complexity should be $O(N!)$..there id a for loop in a recursive call so... $T(N) = N * (T(N-1) + O(1))$ or $T(N) = N * (N-1) * (N-2) ... = O(N!)$

^ | ▼ ·

**AlienOnEarth** · 20 days ago

What would be the time complexity for this problem? Wikipedia says it can not be solved in polynomial time. It is an NP-Complete problem

^ | ▼ ·



Guest • 5 months 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]:
if(hamcycle(graph,seen,i))="" return="" true;="" else="" top--;="" }="" }="" return
2 ^ | v •
```



Guest • 5 months 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]:
if(hamcycle(graph,seen,i))="" return="" true;="" else="" top--;="" }="" }="" return
to="" test="" above="" function="" int="" main()="" {="" *="" let="" us="" create=
-(2)="" |="" \="" |="" |="" \="" |="" |="" \="" |="" (3)------(4)="" *="" bool="" graph1
{1,="" 0,="" 1,="" 1,="" 1},="" {0,="" 1,="" 0,="" 0,="" 1},="" {1,="" 1,="" 0,="" 0,=
};="" print="" the="" solution="" bool="" seen[v]={false};="" seen[0]="true;" stack|
if(top=="V)" printf("there="" exists="" hamcycle\n");="" int="" i;="" for(i="0;i<V
return 0;
}
```


^ | v .



da3m0n · 6 months ago

Time Complexity please..

^ | v .



Guest · 7 months 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;

}

6 ^ | v .



Sriharsha g.r.v → Guest · 5 months ago

hi can u pls elaborate and explain with example..

^ | v .



Born Actor · 11 months 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](#)

^ | v .



AMIT · 11 months ago

I think a minor improvement is possible...using a matrix soln to indicate which can find isSafe in $O(1)$ time.

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

^ | v .



begfairouz → AMIT · 7 months 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 work (gives negative). Did you try it for big matrices?

^ | v .



neham · 11 months ago

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

^ | v .



GeeksforGeeks · a year ago

Ankit: The main task of program is to find out whether a given graph contains a cycle, then print the cycle. The above program always does that. The starting vertex 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.

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

^ | v .



Ankit Paharia · a year ago

The given input graph has a bug.... it won't work for all the input vertices. The input 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 c

^ | v .



NNavneet · a year ago

why in the input graph[i][i]=1 , if we start with vertex 1 or any other vertex than 1
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 to V-1
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++)
    {
        cout<<path[i]<<" ";
    }
    cout<<path[0]<<endl;
```

see more

^ | v .



GeeksforGeeks → NNavneet · a year ago

The main task of program is to find out whether a given graph contains is a cycle, then print the cycle. The above program always does that. 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 bel point :)

Also, we have removed self loops in the matrix representation of graph

^ | v .



Anshul Agrawal · a year ago

grt work buddy..really helped me alot..

^ | v .



AAZ · a year ago

What would be the runtime complexity of the program ?

^ | v .



spark9 · 2 years ago

IsSafe function complexity can be improved by using better data structures. W a vertex is visited or not.

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

^ | v .



atul · 2 years ago

code will not work for all inputs

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

^ | v .



kartik → atul · 2 years ago

Please provide more details of your comment. Why do think that the c
sample graph for which it didn't work?

^ | v .



atul · 2 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) */
```

^ | v .



atul → atul · 2 years ago

ignore above comment

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

^ | v .



kavish · 2 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)
```

^ | v ·



Venki · 2 years ago

How can we extend the solution to start at an arbitrary vertex? One way is to keep a count of zero when it crosses array bound, and include in the solution validity whether it is a valid vector.

Is there any better way?

^ | v ·



kartik → Venki · 2 years ago

One simple way is to generate the path array as it is being generated in all rotations of the path array. Every rotation of the path array will be a Hamiltonian path.

^ | v ·



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team