**To find all Hamiltonian Circuits assuming that the given undirected simple graph is not complete.**

Shubham Vashishtha, Yash Sharma, Aditya Kumawat

IT Department,Indian Institute of Information Technology,Allahabad  
Deoghat,Jhalwa-211015 India

ihm2014007@iiita.ac.in

ihm2014006@iiita.ac.in

icm2014001@iiita.ac.in

**Abstract**—*This report proposes algorithms to find all the Hamiltonian Circuits present in an undirected simple graph assuming the given graph is not complete. First algorithm is a naive brute force approach. Second one uses backtracking / dfs concept. Detailed discussion on memory consumed and time complexity analysis of algorithms is also presented.*

**Keywords** — Undirected *Graph , Hamiltonian Circuits*

1. **Motivation**

A Hamiltonian Circuit[2] is a closed walk in a connected graph such that all vertices in the graph are visited exactly once except the starting vertex at which the walk ends. The problem of finding all such circuits in a graph is NP-Complete. There are no nice necessary-and-sufficient conditions known for a graph to have a Hamiltonian circuit. The aim of our work is to compare the possible methods and their efficiency for graphs having large number of vertices.

1. **Problem Description**

We have a Graph G(V,E) in the form of Adjacency Matrix A of size [VxV], Our aim is:

* To print all Hamiltonian Circuits if they exist.
* Memory Analysis of the algorithm proposed.
* Time Complexity Analysis of the algorithm proposed and comparison with other algorithms.

1. **Approach**

We consider only undirected simple graphs to simplify the discussion. We propose the following algorithms:

**Naive Approach :** A Hamiltonian Circuit[2] in a graph having N vertices is nothing but a permutation of the vertices of the graph [v0, v1 ..... vN-2, vN-1] , such that there is an edge between vi and v(i+1)%N where 0 ≤ i < N. So it can be checked for all permutations of the vertices whether any of them represents a Hamiltonian Circuit or not. For example, for the graph given in Fig. 1 there are 3 vertices, which means total 6 possible permutations, out of which only following represents a Hamiltonian Circuit :

1-2-3-1

3-2-1-3

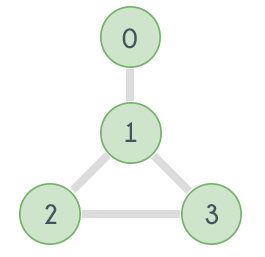


Fig. 1

**Algorithm :** To Print All Hamiltonian Circuits Given Adjacency Matrix

**Input :** Adjacency Matrix

**Output :** All Hamiltonian Circuits

n: no. of vertices in Graph

**Pseudo Code :**

find\_circuit\_using\_permutation(adj\_mat[n][n], n) {

for i = 0 to n-1 {

p[i]=i

}

hamiltonian\_graph = false

while next\_permutation is possible {

valid = true

for i = 0 to n-1 {

if adj[p[i]][p[(i+1) % n]] == false {

valid = false

break

}

}

if valid == true {

hamiltonian\_graph = true

print(p)

}

p = get\_next\_permutation(p)

if hamiltonian\_graph == false {

print(“no hamiltonian circuit exists”)

}

}

**Time Complexity Analysis:** O(N\*N!)

For a graph having N vertices it visits all the permutations of the vertices, i.e. N! iterations and in each of those iterations it traverses the permutation to see if adjacent vertices are connected or not i.e N iterations, so the complexity is O(N\*N!).

**Memory Usage Analysis:** O(N^2)

For storing adjacency matrix.

**Using Depth First Search:** Depth first search[1] and backtracking can also help to check whether a Hamiltonian Circuit exists in a graph or not. Simply apply depth first search starting from every vertex v and do labeling of all the vertices. All the vertices are labelled as either "IN STACK" or "NOT IN STACK". A vertex is labelled "IN STACK" if it is visited but some of its adjacent vertices are not yet visited and is labelled "NOT IN STACK" if it is not visited. If at any instant the number of vertices with label "IN STACK" is equal to the total number of vertices in the graph then a Hamiltonian Circuit exists in the graph.

**Algorithm :** To Print All Hamiltonian Circuits Given Adjacency MAtrix

**Input :** Adjacency Matrix

**Output :** All Hamiltonian Circuits

n: no. of vertices in Graph

**Pseudo Code :**

find\_circuit\_using\_dfs(adj\_mat[n][n], n) {

for i = 1 to n {

label[i] = NOT\_IN\_STACK

}

for i = 1 to n {

label[i] = IN\_STACK

stack.push(i)

if dfs(adj\_mat, label, n, stack) is true

then print(stack)

label[i] = NOT\_IN\_STACK

stack.pop()

}

}

bool dfs(adj\_mat[][], label[], n, stack) {

if stack.size == n && adj[stack.top][stack.bottom]

then return true

v = stack.top

for i = 1 to n {

if (adj[v][i] && label[i]==NOT\_IN\_STACK)

then label[i] = IN\_STACK

stack.push(i)

if (dfs(adj\_mat, label, n, stack)

then return true

label[i] = NOT\_IN\_STACK

stack.pop()

}

return false

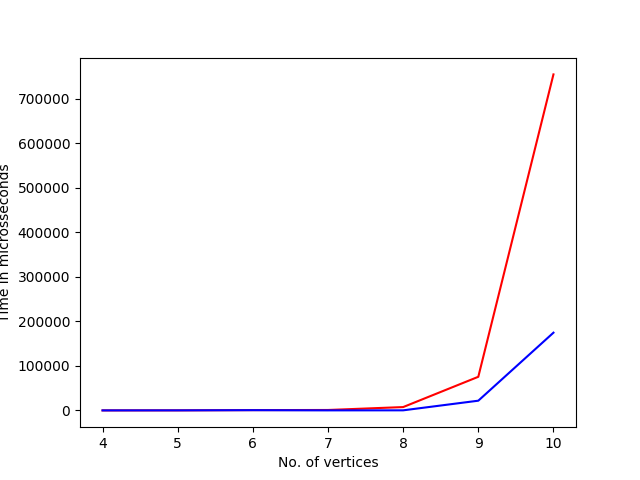
}

**Time Complexity** : Worst case complexity of using DFS and backtracking is O(N!).

**Memory Usage Analysis:** O(N)

1. **Comparison Of Time Taken By Algorithms**

The following graph shows the comparison between time taken by the two methods. Y - Axis represents the number of vertices in the graph and X - Axis represents the time taken (in microseconds) to find all the Hamiltonian circuits in the graph.

****

The Red curve shows the time taken by the naive approach and the Blue curve shows the time taken by the approach with DFS & Backtracking.

1. **Conclusion**

There is no good algorithm known to find a Hamilton circuit/cycle. The most obvious way is to check every one of the n! possible permutations of the vertices to see if things are joined up that way. Some of the known algorithms are of exponential order which check for existence of Hamiltonian Cycle. They don’t scale very well for very large graphs. Also, proof of non-existence of non-exponential algorithms doesn’t exist either.

6. **References**

1. *https://www.hackerearth.com/practice/algorithms/graphs/hamiltonian-path/tutorial/*
2. *https://www.cs.sfu.ca/~ggbaker/zju/math/euler-ham.html#ham*