Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 44 No. 3 (2023)

A Comprehensive Study On Some Fundamental Problems In Graph Theory Using PYTHON

[1]Dr. A.Venkatesan, [2]Dr. S. Muthukumaran,
[3]Dr.A. Victoria Anand Mary,[4]Dr. C. Christy

[1]Assistant Professor, PG and Research Department of Mathematics,
[2]Assistant Professor, PG and Research Department of Computer Science
St. Joseph's College of Arts & Science (Autonomous), Cuddalore-607001.

E-mail: [1]suresh11venkat@gmail.com[2]muthu.svk06@gmail.com
[3]victoria.mary1106@gmail.com [4]vincentchristy4@gmail.com

Abstract: In this paper some fundamental problems in graph theory such as finding the shortest path, Hamiltonian path, minimum weight of the spanning tree and determining whether a given graph is Eulerian are obtained by Python.

Keywords: Eulerian Circuit - Hamiltonian path - Dijkstra's Algorithm - Prim's Algorithm - Minimum Spanning Tree.

1. Introduction:

In this paper we assume a graph G is finite, undirected and connected. Graph theory is a branch of mathematics that deals with the study of graphs and networks. A graph G consists of a pair (V(G), X(G)) where V(G) is a non-empty finite set whose elements are called points or vertices and X(G) is a set of unordered pair of distinct elements of V(G). The elements of X(G) are called lines or edges of the graph G[6]. Python is a high-level programming language that is used for a wide range of applications. It was first released in 1991 by Guido van Rossum and has since become one of the most popular programming languagesin the world. The Shortest Path, Hamiltonian Path, Minimum weight of the Spanning Tree and determining whether a given graph is Eulerian are some of the most well-studied topics in graph theory and computer science.Here we discuss these fundamental problems in graph theory using Python.

2. Preliminaries:

2.1 Definitions: ([7])

A *walkW*, connecting vertices u and v in a graph G is a finite alternating sequence of verticesand edges in the form $u = u_0, e_1, u_1, e_2, u_2, e_3, u_3, e_4, u_4, e_5, u_5, \ldots, u_{n-1}, e_n, u_n = v$. The walk W is called *closed walk* is u = v. In a walk, an edge or a vertex may repeat.

The number of edges in a walk is called its *length.* A walk is called a *trial* if all edges in it are distinct and a trial is called a *circuit* if it is closed.

A walk W is called a *path* if all vertices in it are distinct. A path is called a *cycle* if it is closed. Obviously, all edges in a path are also distinct and only first and last vertices are repeated.

2.2 Definition: ([7])

A graph G is called *weightedgraph* if each edge e in G is assigned a definite non-negative real number denoted by (e), called weight of e. The non-negative real number may be length, cost, time, distance, fuel etc.

2.3 Definition: ([3])

A circuit in a connected graph G is called *Eulerian circuit* if it contains all the edges of graph and the graph is called *Eulerian graph* if such a Eulerian circuit exists in G.

2.4 Definition: ([3])

A path in a connected graph G is called *Hamiltonian path* if it contains all vertices in G (not necessarily all edges) and the graph is called *Hamiltonian graph* if such Hamiltonian path exists in G.

2.5 Definition: ([4])

A *tree* is a simple graph G such that there is a unique simple nondirected path between each pair of vertices of G.

2.6 Definition: ([4])

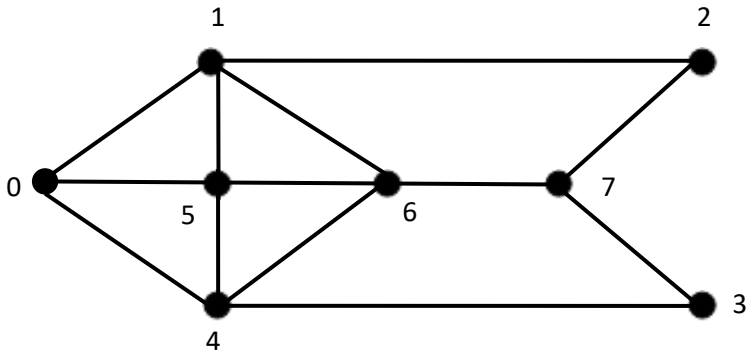
A subgraph H of a graph G is called a *spanning tree* of G if H is a tree and contains all the vertices of G.

3. Python Programs For Some Fundamental Problems Of Graph Theory:

3.1 Python Program To Check Whether The Given Graph Is Eulerian:

```
from collections import defaultdict
def add_edge(adj_list, node1, node2):
   adj_list[node1].append(node2)
   adj_list[node2].append(node1)
def DFS(adj_list, visited_nodes, start_node):
   visited_nodes[start_node] = True
   for neighbor in adj_list[start_node]:
      if not visited_nodes[neighbor] and len(adj_list[neighbor])>0:
DFS(adj_list, visited_nodes, neighbor)
def check_eulerian_circuit(adj_list):
   start_vertex = next(iter(adj_list))
visited = defaultdict(lambda: False)
DFS(adj_list, visited, start_vertex)
for i in visited.keys():
      if not visited[i] and len(adj_list[i])>0:
         return False
   odd_nodes = [v for v in adj_list.keys() if len(adj_list[v])%2!=0]
   return len(odd_nodes) == 0 or len(odd_nodes) == 2
adj_list = defaultdict(list)
add_edge(adj_list, 0, 1)
add_edge(adj_list, 0, 4)
add_edge(adj_list, 1, 2)
add_edge(adj_list, 2, 3)
add_edge(adj_list, 3, 4)
add_edge(adj_list, 0, 5)
add_edge(adj_list, 1, 6)
add_edge(adj_list, 2, 7)
add_edge(adj_list, 3, 7)
add_edge(adj_list, 4, 6)
add_edge(adj_list, 5, 6)
add_edge(adj_list, 6, 7)
if check_eulerian_circuit(adj_list):
print("The graph is Eulerian.")
else:
print("The graph is not Eulerian.")
```

Output:
The graph is not Eulerian.
Manual Output:



G

The graph G is not Eulerian because, the vertices 0 and 7 are of odd degree.

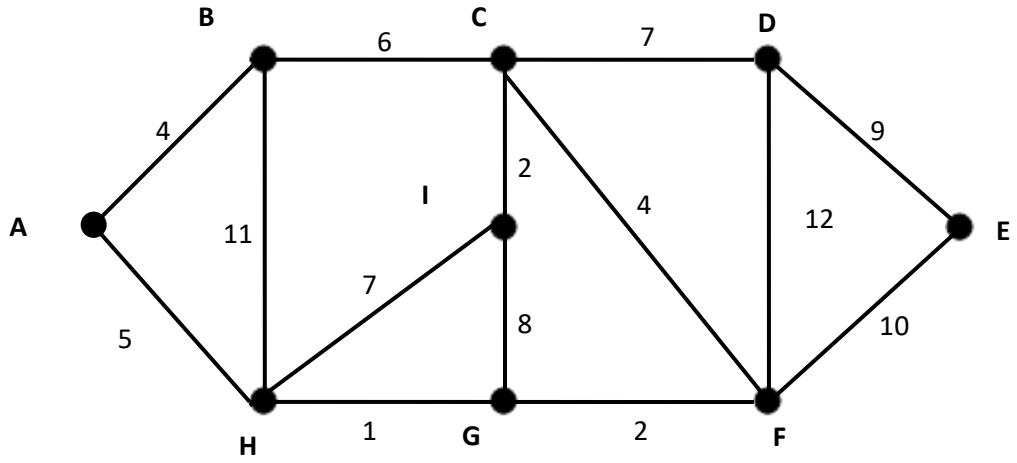
3.2 Python Program To Find The Shortest Path

```
import heapq
def shortest_path(graph, start_node):
    distance = {vertex: float('inf') for vertex in graph}
    distance[start_node] = 0
    heap = [(0, start_node)]
    while heap:
        (dist, cur) = heapq.heappop(heap)
        if dist > distance[cur]:
            continue
for nbr, weight in graph[cur].items():
            tentative_distance = dist + weight
            if tentative_distance < distance[nbr]:
                distance[nbr] = tentative_distance
heapq.heappush(heap, (tentative_distance, nbr))
    return distance
graph = {
    'A': {'B': 4, 'H': 5},
    'B': {'A': 4, 'C': 6, 'H': 11},
    'C': {'B': 6, 'D': 7, 'F': 4, 'I': 2},
    'D': {'C': 7, 'E': 9, 'F': 12},
    'E': {'D': 9, 'F': 10},
    'F': {'C': 4, 'D': 12, 'E': 10, 'G': 2},
    'G': {'F': 2, 'H': 1, 'I': 8},
    'H': {'A': 5, 'B': 11, 'G': 1, 'I': 7},
    'I': {'C': 2, 'G': 8, 'H': 7}
}
start_vertex = 'A'
distances = shortest_path(graph, start_vertex)
for vertex in distances:
print(" The Shortest Distance from the vertex", start_vertex, "to the vertex", vertex, "is", distances[vertex])
```

Output:

The Shortest Distance from the vertex A to the vertex A is 0
The Shortest Distance from the vertex A to the vertex B is 4
The Shortest Distance from the vertex A to the vertex C is 10
The Shortest Distance from the vertex A to the vertex D is 17
The Shortest Distance from the vertex A to the vertex E is 18
The Shortest Distance from the vertex A to the vertex F is 8
The Shortest Distance from the vertex A to the vertex G is 6
The Shortest Distance from the vertex A to the vertex H is 5
The Shortest Distance from the vertex A to the vertex I is 12

Manual Output:



The shortest distance between A to A is 0.
The shortest distance between A to B is 4.
The shortest distance between A to C is 10.
The shortest distance between A to D is 17.
The shortest distance between A to E is 18.
The shortest distance between A to F is 8.
The shortest distance between A to G is 6.
The shortest distance between A to H is 5.
The shortest distance between A to I is 12.

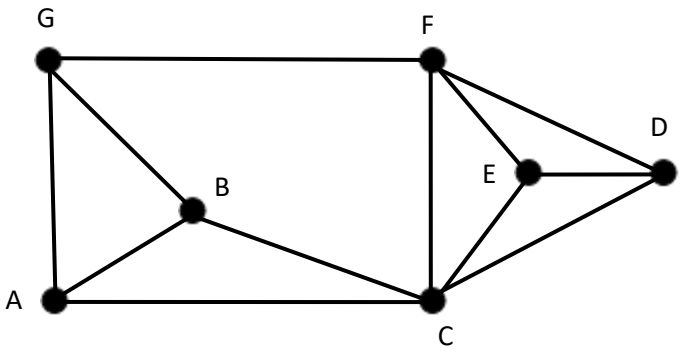
3.3 Python Program To Find The Hamiltonian Path:

```
def hamilton_path(graph):
    path = []
for start_vertex in graph.keys():
        visited = set([start_vertex])
path.append(start_vertex)
if explore(graph, visited, path):
            return path
path.pop()
    return []
def explore(graph, visited, path):
    if len(path) == len(graph):
        return True
    current_vertex = path[-1]
```

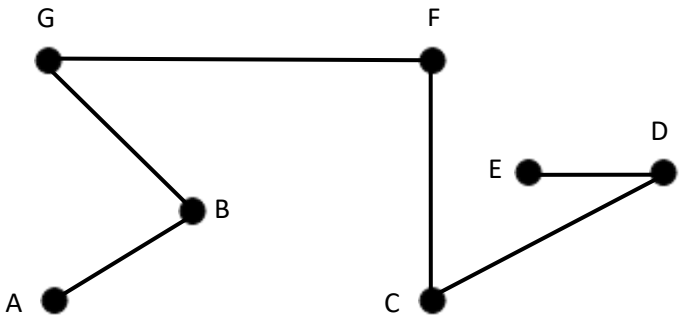
```
    for nbr in graph[current_vertex]:
        if nbr not in visited:
            visited.add(nbr)
path.append(nbr)
if explore(graph, visited, path):
                return True
visited.remove(nbr)
path.pop()
    return False
graph = {
    'A': {'B': 1, 'C': 1, 'G':1},
    'B': {'A': 1, 'C': 1, 'G':1},
    'C': {'A': 1, 'B': 1, 'D': 1, 'E':1, 'F':1},
    'D': {'C': 1, 'E': 1, 'F':1},
    'E': {'C': 1, 'D': 1, 'F': 1},
    'F': {'C': 1, 'D': 1, 'E':1},
    'G': {'A': 1, 'B': 1, 'F': 1}
}
path = hamilton_path(graph)
if path:
print("Hamiltonian path:", " -> ".join(path))
else:
print("No Hamiltonian path exists for the graph.")
```

Output:
Hamiltonian path: A -> B -> G -> F -> C -> D -> E

Manual Output:



The Hamiltonian path of the above graph is A -> B -> G -> F -> C -> D -> E



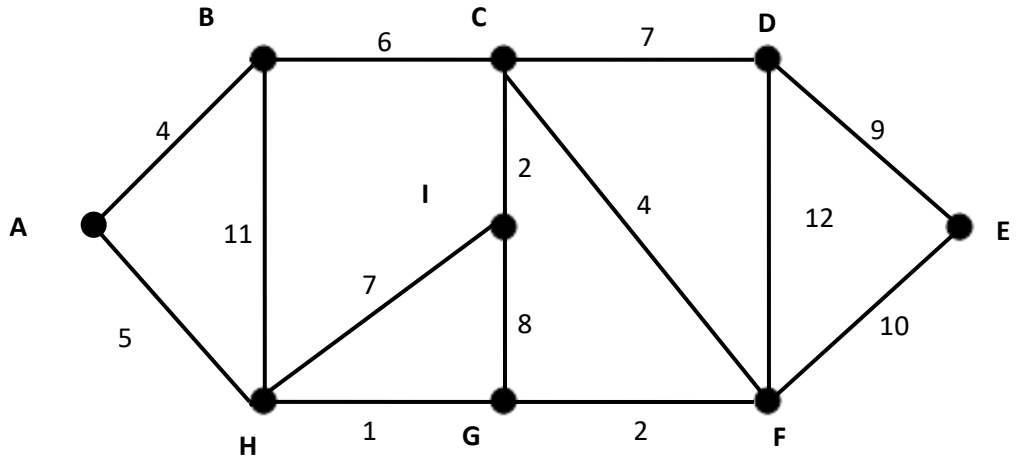
3.4 Python Program To Find The Minimum Weight Of The Spanning Tree:

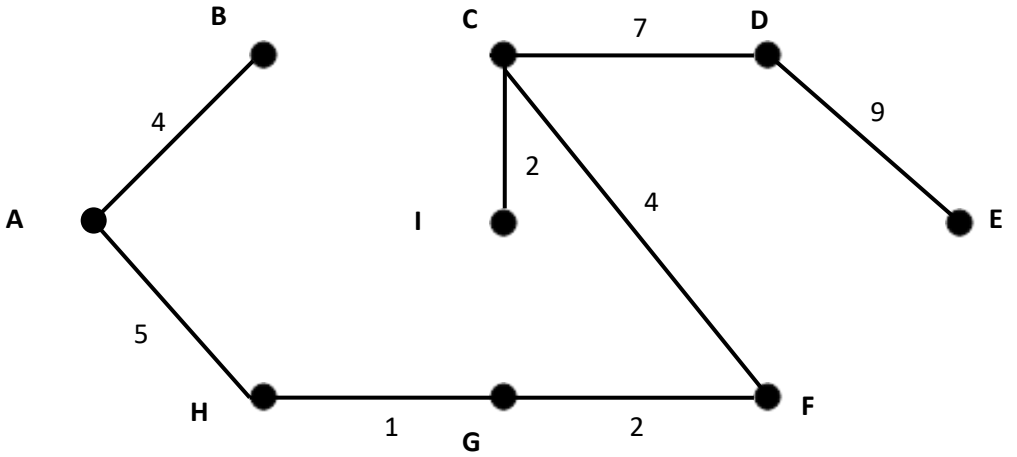
```
import heapq
def prim(graph):
visited = set()
   edges = [(0, None, 'A')]
   min_spanning_tree_weight = 0
   while edges:
weight, node1, node2 = heapq.heappop(edges)
      if node2 not in visited:
         visited.add(node2)
         min_spanning_tree_weight += weight
for nbr, weight in graph[node2].items():
            if nbr not in visited:
heapq.heappush(edges, (weight, node2, nbr))
   return min_spanning_tree_weight
graph = {
   'A': {'B': 4, 'H': 5},
   'B': {'A': 4, 'C': 6, 'H': 11},
   'C': {'B': 6, 'D': 7, 'F': 4, 'I': 2},
   'D': {'C': 7, 'E': 9, 'F': 12},
   'E': {'D': 9, 'F': 10},
   'F': {'C': 4, 'D': 12, 'E': 10, 'G': 2},
   'G': {'F': 2, 'H': 1, 'I': 8},
   'H': {'A': 5, 'B': 11, 'G': 1, 'I': 7},
   'I': {'C': 2, 'G': 8, 'H': 7}
}
min_spanning_tree_weight = prim(graph)
print("Minimum weight of the spanning tree:", min_spanning_tree_weight)
```

Output:

Minimum weight of the spanning tree: 34

Manual Output:





The Minimum weight of spanning tree of the above graph is 34.

4. Conclusion:

In this paper we have used Python program codes to determine some fundamental problems in Graph Theory such as finding the shortest path, Hamiltonian path, minimum weight of the spanning tree and determining whether the given graph is Eulerian.

References:

[1] Afsana Khan, AfridaAnzumAesha, JuthiSarker, "*A New Algorithmic approach to Finding Minimum Spanning Tree*", https://www.researchgate.net/publication/328389207

[2] Ashish Kumar, "*A study on Euler Graph and its Applications*", International Journal of Mathematics Trends and Technology, Vol.43(1), 2017.

[3] H.B.Maharjan, and L.N. Sharma, "*An Introduction to Graph Theory*", PaluwaPrakashan, Kathmandu Nepal, 2008.

[4] Joe L. Matt, Abraham Kandel, Theodore P. Baker, "*Discrete Mathematics for Computer Scientists and Mathematicians",* Prentice-Hall of India Pvt. Ltd, 2008.

[5] Ram Lakhan Sah, "*Dijkstra's Algorithm for determining shortest path"*, Journal of Emerging Technologies and Innovative Research, Vol.7(4), 2020.

[6] S. Arumugam, S. Ramachandran, "*Invitation to Graph Theory*", Scitech Publications (India) Pvt. Ltd., 2001.

[7] S.M. Maskey, "*First Course in Graph Theory*", RatnaPustakBhandar, Kathmandu, Nepal, 2008.