Finding Eulerian and Hamiltonian Cycles in Graphs

Marija Chaushevska
Institutuion
Jožef Stefan International Postgraduate School
Jamova 39, 1000 Ljubljana, Slovenia
e-mail: marija.chaushevska@ijs.si

ABSTRACT

The main objective of this paper is to find Eulerian and Hamiltonian Cycles in a graph, studying Eulerian and Hamiltonian Graphs, as well as finding their various aspects and applications in our real world. Both Hamiltonian and Euler paths are used in graph theory for finding a path between two vertices. Now a day's Eulerian and Hamiltonian graphs got height of achievement in many situations that occur in computer science, physical science, communication science, economics and many other areas can be analysed by using techniques found in a relatively new area of mathematics. The graphs concerns relationship with lines and points (nodes). The Euler graph can be used to represent almost any problem involving discrete arrangements of objects where concern is not with the internal properties of these objects but with relationship among them. On the other hand, the Hamiltonian Path Problem asks whether there is a route in a directed graph from a beginning node to an ending node, visiting each node exactly once. The Hamiltonian Path Problem is NP (nondeterministic polynomial) complete, achieving surprising computational complexity with modest increases in size. This challenge has inspired researchers to broaden the definition of a computer.
In this paper, we will be using Fleury's and Hierholzer's algorithms to find out if in a given graph exists an Eulerian circuit and print it or does not exist. As well as, we will use Backtracking algorithm to determine if a given graph is contains a Hamiltonian circuit or not. The algorithms are implemented in Python programming language.

Key words: Graph theory, Eulerian circuit, Eulerian path Hamiltonian circuit, Hamiltonian path, Fleury's algorithm, Hierholzer's algorithm, Backtracking algorithm…

1 INTRODUCTION

A graph *G* consists of a set *V* called the set of points (nodes, vertices) of the graph and a set of edges such that each edge $e \in E$ is associated with ordered or unordered pair of elements of *V* i.e., there is a mapping from set of edges *E* to set of ordered or unordered pairs of elements of *V*. The set *V(G)* is called the vertex set of *G* and *E(G*) is the edge set. The graph *G* with vertices *V* and edges E is written as *G = (V,E)* or *G (V,E)*. Types of graphs are simple graph, multi graph, pseudo graph. An undirected graph [1] *G* consists of sets of vertices *V* and a set of edges *E* such that each edges is associated with an unordered pair of vertices. A directed graph [1] *G* consists of a set of vertices *V* and a set of edges such that each edges *E* is associated with an ordered pair of vertices then the graph is called directed graph or digraph. The degree of a vertex of an graph is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex *v* in a graph *G* denoted by deg(*v*).
If *G= (V,E)* be an undirected graph with *e* edges. Then

$$\sum_{v \in V} \deg(v) = 2e$$

i.e., the sum of degrees of the vertices in an undirected graph is twice the number of edges (Handshaking theorem).

If *G= (V,E)* be a directed graph with *e* edges, then

$$\sum_{v \in V} \deg(v)^+ = \sum_{v \in V} \deg(v)^- = e$$

i.e., the sum of the out degrees of the vertices of a digraph *G* equals the sum of in degrees of the vertices which equals the number of edges in *G* .
In the second section of this paper we will present and explain Eulerian graph, it's origin, importance, algorithms (Fleury's and Hierholzer's algorithms) that find Eulerian circuits in a given graph and practical problems and applications for which this graph is used for. Whereas, in the third section we are focused on explaining Hamiltonian graphs, algorithm (Backtracking algorithm) that finds a Hamiltonian cycle in a given graph and real applications that this graph is used for. Finally, in the last section we are going to summarize all the theory and algorithms that we have been discussed in the previous sections.

2 EULERIAN GRAPH

2.1 Origin and Importance of Eulerian graph (Konigsberg problem)

The river called Pregel flows through the city Konigsberg (located in Russia) dividing the city into four land regions, two are river banks and two are islands or delta formation. The four land regions were connected by 7 bridges. There

was an entertaining or interesting exercise for the citizens of Konigsberg. Start from any land regions and come back to the starting point after crossing each of the seven bridges exactly once without repeating same path.

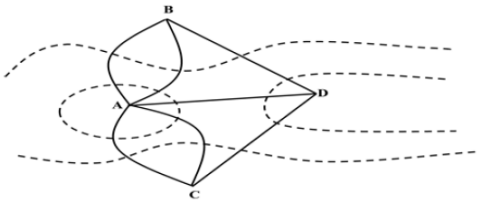


Figure 1. The bridges of Konigsberg problem

Euler suggested and further explained that it is impossible to do so by using the terminology of points (representing the land regions) and lines (representing the bridges). Hence, he titled his paper as "Solutions to a problem relating to the geometry of positions." Through this explanation, he laid the foundation for Graph Theory. Graph theory started from this problem.

He abstracted the case of Konigsberg by eliminating all unnecessary features. He drew a picture consisting of "dots" that represented the landmasses and the line-segments representing the bridges that connected those land masses. The resulting picture looked somewhat similar to the figure shown below in Figure 2.

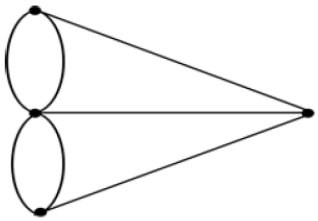


Figure 2.

This simplifies the problem to great extent. He came out with the then new concept of degree of nodes. The Degree of Node can be defined as the number of edges touching a given node. Euler proposed that any given graph can be traversed with each edge traversed exactly once if and only if it had, zero or exactly two nodes with odd degrees. The graph following this condition is called Eulerian circuit or path.

Using Euler's theorem we need to introduce a path to make the degree of two nodes even. And other two nodes can be of odd degree out of which one has to be starting and other at another the end point. Suppose we want to start our journey from node. So, the two nodes can have odd edges. But somehow we need to edit the actual graph by adding another edge to the graph such that the two other nodes have even degree.

Trail that visits every edge of the graph once and only once is called Eulerian trail. Starting and ending vertices are different from the one on which it began. A graph of this kind is said to be traversable. An Eulerian circuit is an Eulerian trail that is a circuit i.e., it begins and ends on the same vertex. A graph is called Eulerian when it contains an Eulerian circuit. A digraph in which the in-degree equals the out-degree at each vertex. A vertex is odd if its degree is odd and even if its degree is even.

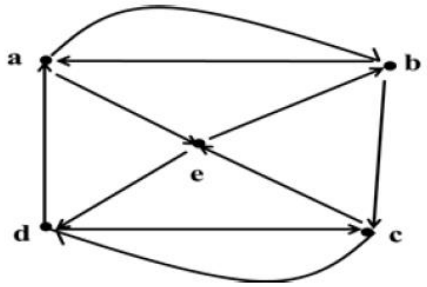


Figure 3. Example of an Eulerian graph

2.2 Existence of Euler path and circuit

Euler Path is a path that visits each edge of a graph exactly once. It may start and end at a different vertex. A graph contain Euler Path only if it has exactly 0 or 2 odd degree vertices.

There are several ways to find the existence of Euler path. Considering the existence of an Euler path in a graph is directly related to the degree of vertices in a graph. Euler formulated the theorems for which we have the sufficient and necessary condition for the existence of an Euler circuit or path in a graph respectively.

Theorem: An undirected graph has at least one Euler path if and only if it is connected and has two or zero vertices of odd degree.

Theorem: An undirected graph has an Euler circuit if and only if it is connected and has zero vertices of odd degree.

Let us take a graph having no odd vertices, the path can begin at any vertex and will end there; vice-versa in the case of two odd vertices, the path must begin at one odd vertex and end at the other. Any finite connected graph with two odd vertices is traversable. A traversable trail may begin at either odd vertex and will end at the other odd vertex. Finding an Euler path is a relatively simple problem it can be solve by keeping few guidelines in our mind:

- Always leave one edge available to get back to the starting vertex (for circuits) or to the other odd vertex (for paths) as the last step.
- Don't use an edge to go to a vertex unless there is another edge available to leave that vertex (except for the last step)

An Euler Circuit is a circuit that visits each edge of a graph exactly once. It starts and ends at the same vertex. A graph contain Euler Circuit only if it has exactly 0 odd degree vertices. A connected graph G is Eulerian if there exists a closed trail containing every edge of G. Such a trail is an Eulerian trail. The graph shown below in Figure 4 has several

possible Euler circuits, so in the Figure 5 is shown one of them.

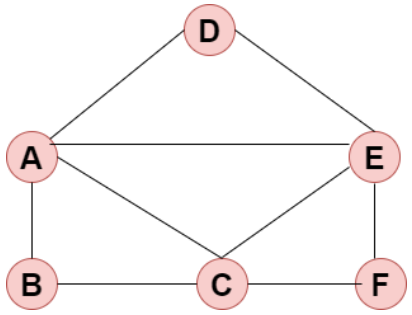


Figure 4. Example of graph that consists Eulerian circuit

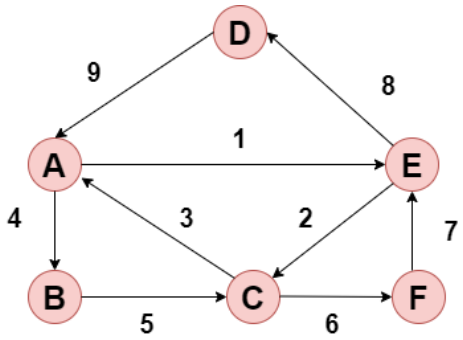


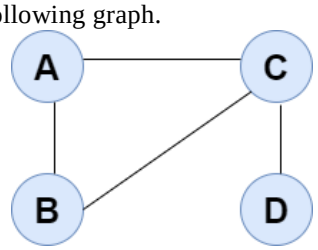
Figure 5. Eulerian circuit: AECABCFEDA

2.3 Finding Eulerian Circuit in a Graph using Fleury's Algorithm

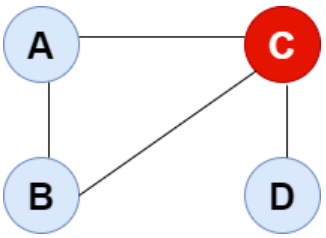
Fleury's algorithm is used to find a Euler Path [2] or a Euler Circuit [2] in a connected graph. The basic principle of Fleury's algorithm is very simple. In order to find the Euler Path or Euler Circuit, the bridge edge should be the last edge we want to cross. This is because the bridge is the only edge connecting the two components of a graph. If we crossed the bridge without visiting all edges in the first component then we cannot come back to the first component without re-visiting the bridge. As we already know, Euler Path or Euler Circuit visits each edge exactly once. Thus, the bridge edge is traversed only after visiting all the other edges in the component.
The steps that describe the Fleury's algorithm are following:

- Make sure the graph has either 0 or 2 odd vertices.
- If there are 0 odd vertices, start anywhere. If there are 2 odd vertices, start at one of them.
- Follow edges one at a time. If you have a choice between a bridge and a non-bridge, always choose the non-bridge.
- Continue until you're done, more precisely when you run out of edges.

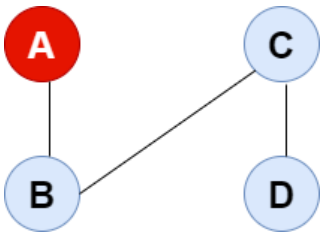
The idea is, "don't burn bridges" so that we can come back to a vertex and traverse remaining edges. For example let's consider the following graph.



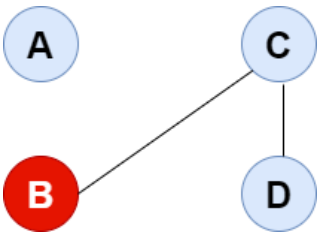
There are two vertices with odd degree, 'C' and 'D', we can start path from any of them. Let us start tour from vertex 'C'.



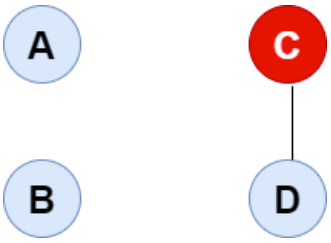
There are three edges going out from vertex 'C', which one to pick? We don't pick the edge 'C-D' because that is a bridge (we won't be able to come back to 'D'). We can pick any of the remaining two edge. Let us say we pick 'C-A'. We remove this edge and move to vertex 'A'.



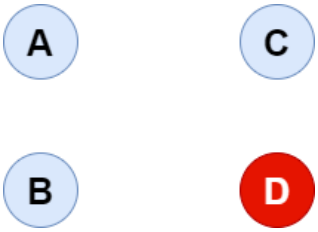
There is only one edge from vertex 'A', so we pick it, remove it and move to vertex 'B'. Euler tour becomes 'C-A-B'.



There is only one edge from vertex 'B', so we pick it, remove it and move to vertex 'C'. Euler tour becomes 'C-A-B-C'



Again there is only one edge from vertex C, so we pick it, remove it and move to vertex D. Euler tour becomes 'C-A -B-C-D'.



There are no more edges left, so we stop here. Final tour is 'C-A-B-C-D'.

Basically, we are traversing each vertex and each edge attached to that vertex. This takes O(V+E) comparisons. For each edge, we are checking if the edge is a bridge or not using a depth-first search.

Checking if an edge is a bridge or not requires O(V+E) comparisons.

Time Complexity = O(V+E) * O(V+E)

Complexity = $O((V+E)^2)$

Since the graph is connected, we can say that:

$$\frac{V*(V-1)}{2} \geq E \geq V-1$$

Therefore, time complexity of the implementation of Fleury's algorithm for a connected graph is $O(E^2)$. There are better algorithms to print Euler tour, Hierholzer's Algorithm finds in O(V+E) time. This algorithm will be clearly presented and explained in the next sub-section.

2.4 Finding Eulerian Circuit using Hierholzer's Algorithm

Remember that a directed graph has a Eulerian cycle if the following conditions are true:

- All vertices with nonzero degrees belong to a single strongly connected component.
- In degree and out-degree of every vertex is the same.

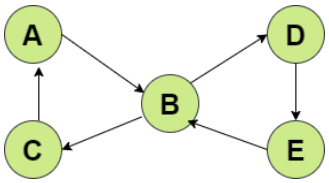
The algorithm assumes that the given graph has a Eulerian Circuit. The steps that describe the Hierholzer's algorithm are following:

- Choose any starting vertex v, and follow a trail of edges from that vertex until returning to v. It is not possible to get stuck at any vertex other than v, because indegree and outdegree of every vertex must be same, when the trail enters another vertex w there must be an unused edge leaving w.The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.
- As long as there exists a vertex u that belongs to the current tour, but that has adjacent edges not part of the tour, start another trail from u, following unused edges until returning to u, and join the tour formed in this way to the previous tour.

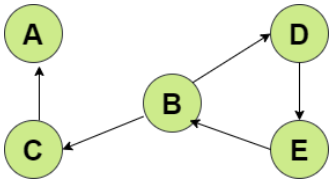
Thus the idea is to keep following unused edges and removing them until we get stuck. Once we get stuck, we backtrack to the nearest vertex in our current path that has unused edges, and we repeat the process until all the edges have been used. We can use another container to maintain the final path.

Let's take an example:

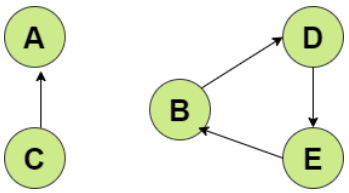
The initial directed graph is shown as below.



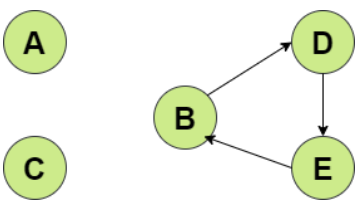
Let's start the path from vertex 'A' and use the edge 'A'->'B'. The current_path ={A} and circuit = {}.



Now, current_path= {A,B}, circuit = {}, similary we reach up to 'C' and then to 'A' again.



Now, current_path is {A,B,C} and circuit = {}. Then we go to A, now since A haven't got any unused edge we put A in circuit and back track till we find an edge.



Then, we have the current path = {A,B,C} and circuit = {A}. Similarly, when we backtrack to 'C', we don't find any

unused edge. Hence put ‘C’ in the circuit and backtrack again.

Current_path = {A,B} and circuit = {A,C}. After reaching ‘B’ we go to through unused edge ‘B’->’D’ and then ‘D’->’E’, ‘E’->’B’ until all edges have been traversed.

The contents of the two containers look as: current_path = {A,B,D,E,B} and circuit = {A,C}. Now as all edges have been used, the Current path is popped one by one into the circuit. Finally, we've circuit = {A,C,B,E,D,B,A}.

We print the circuit in reverse to obtain the path followed.

i.e., A->B->D->E->B->C->A

Using Heirholzer’s Algorithm, we can find the circuit/path in O(E), i.e., linear time.

2.5 Applications of Eulerian Graphs

Eulerian graphs can be used to solve many practical problems like Konisberg Bridge problem. They can also be used to by mail carriers who want to have a route where they don‟t retrace any of their previous steps. Euler graphs [8] are also useful to painters, garbage collectors, airplane pilots and all world navigators. In this section we will discussed some of the applications of Euler graphs.

2.5.1 Diagram tracing and Eulerian Graph

A popular old game that entertains children runs as follows: Can you trace with a pencil a diagram of points (with the small circles representing the points) and lines as shown in Figures 6a and 6b? The condition is that the diagram is to be traced beginning at a point and on completion end at the same point but the pencil should not be lifted till the diagram is completely traced and a line in the diagram should not be retraced (i.e. can be traced only once). A curious child will certainly try to find the answer by trial and error and will arrive at the conclusion after sometime that it is not possible in diagram in Figure 6b but it is possible in the diagram in Figure 6a. The question of whether it is possible to trace such a diagram, can be quickly answered if the concept of Eulerian graph is known. Indeed the diagram in Figure 6a is Eulerian whereas the diagram in Figure 6b is not. If e = xy is an edge in a graph, then x is called the start vertex and y, the end vertex of e. A path P (or u-v path P) in a graph is a sequence of edges so that the end vertex of an edge in the sequence is the start vertex of the next edge in the sequence and the path begins in the vertex u and ends in the vertex v. If vertices u and v are the same, then the path is called a circuit (some call it cycle). For example in the graph in Figure 6c, (a,b)(b,c)(c,b)(b,c)(c,d) is a path and (a,b)(b,c)(c,b)(b,c)(c,a) is a circuit. Vertices and/or edges can be repeated in a path or in a circuit. (A path is called a walk by some authors. Due to the diversity of people who use graphs for their own purpose, the naming of certain concepts has not been uniform in graph theory). For example in the graph in Figure 6c, (a,b)(b,c)(c,e)(e,d)(d,c)(c,a) is an Eulerian circuit and hence the graph in Figure 6c is an Eulerian graph.

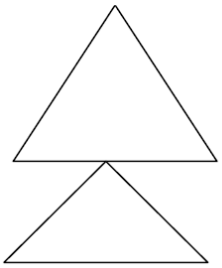


Figure 6a. Diagram that can be traced

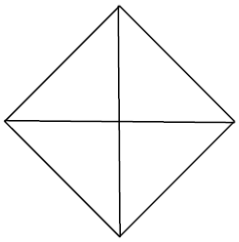


Figure 6b. Diagram that can not be traced

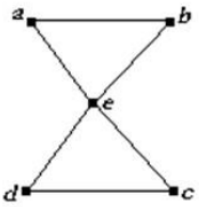


Figure 6c. An Eulerian Graph

2.5.2 Chinese Postman Problem

All vertices of a graph need not be of even degree and so a graph may not be Eulerian. But if a graph has all except two vertices of even degree then it has an Eulerian path which starts at one of the odd vertices and ends at the other odd vertex. A graph having an Eulerian path but not an Eulerian circuit is called semi-Eulerian. For example in the graph in Figure 7, (a,b)(b,c)(c,d)(d,b)(b,e)(e,d)(d,f) is an Eulerian path and hence the graph in Figure 7 is semi-Eulerian.

In a graph Gp that models streets and street corners in a town, with the street corners as the vertices and the streets as the edges, someone starting in a street corner and walking along the streets one after another can end up in a street corner, giving rise to a path in the graph. Note that walking along a street more than once from one corner to another, corresponds to repeating an edge or a vertex in the graph. Suppose that a postman has to deliver letters to the residents in all the streets of a village. Assume that the village is small enough for the postman to be assigned this task every day. If the graph Gp that represents the streets and street corners as mentioned above, is semi-Eulerian so that there is an Euler path in the graph Gp, then this path gives rise to a route following which the postman can start in a street corner, deliver letters going

through every street exactly once. This is a desirable 297 | P a g e situation for the postman. But if no such Euler path exists in Gp, then the postman may have to repeat some of the streets. This problem is called the Chinese postman problem in honour of a Chinese mathematician Meigu Guan who proposed this problem.

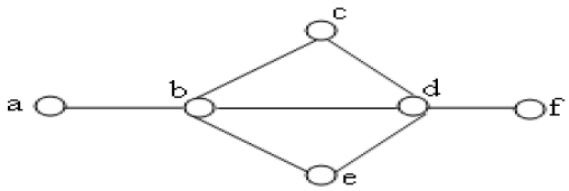


Figure 7. A Semi-Eulerian graph

2.5.3 DNA fragment assembly

DNA (deoxyribonucleic acid) is found in every living organism and is a storage medium for genetic information. A DNA strand is composed of bases which are denoted by A (adenine), C (cytosine), G (Guanine) and T (thymine). The familiar DNA double helix arises by the bondage of two separate strands with the Watson-Crick complementarity (A and T are complementary; C and G are complementary) leading to the formation of such double strands. DNA sequencing and fragment assembly is the problem of reconstructing full strands of DNA based on the pieces of data recorded. It is of interest to note that ideas from graph theory, especially Eulerian circuits have been used in a recently proposed approach to the problem of DNA fragment assembly. We do not enter into the details but only mention that this brings out the application of graph theory in the field of bioinformatics [4].

3 HAMILTONIAN GRAPH

In the mathematical field of graph theory [5], a Hamiltonian path (or traceable path) is a path in an undirected or directed graph [1] 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 [6]. Being a circuit, it must start and end at the same vertex. The Hamilton cycle problem is closely related to a series of famous problems and puzzles (traveling salesman problem, Icosian game) and, due to the fact that it is NP-complete, it was extensively studied with different algorithms to solve it. Therefore, resolving the Hamiltonian cycle [6] is an important problem in graph theory and computer science as well.
A Hamiltonian path also visits every vertex once with no repeats, but does not have to start and end at the same vertex. If a (connected) graph *G(V,E)* has a Hamiltonian circuit, then the graph is called a Hamiltonian graph [7]. Hamiltonian paths and cycles are named after William Rowan Hamilton (1805-1865) who invented the icosian game, now also known as Hamilton's puzzle, which involves finding a Hamiltonian cycle in the edge graph of the dodecahedron. Hamilton solved this problem using the icosian calculus, an algebraic structure based on roots of unity with many similarities to the quaternions (also invented by Hamilton). This solution does not generalize to arbitrary graphs.
Due to their similarities, the problem of an Hamiltonian Cycle [7] is usually compared with Euler's problem, but solving them is very different. There exists a very elegant, necessary and sufficient condition for a graph to have Euler Cycles. Also, literature presents many solutions that generate efficient algorithms for finding Euler Cycles, such as Fleury's and Hierholzer's algorithms. Unfortunately, there is a lack of solutions for the Hamilton problem.

3.1 Hamiltonian Path Properties

A simple graph with N vertices contains a Hamiltonian path if:

- The graph contains a pair of distinct non-adjacent vertices
- For each such pair, (degrees of vertex pair) (length of the shortest path for that pair) ≥ N +1

For a connected graph *G(V,E)* , a random pair of a distinct vertex is *(M,P)∈ V*. Now according to the property : $d(M)+d(P)+\delta(M,P) \geq N+1$.

Here, *d(M), d(P),* denotes the degree of the vertex *M*, *P* and *N* denotes the number of the vertex in *G*. The other notation $\delta(M,P)$ is used to denote the shortest path between *M* and *P* in *G*.

The graph shown in Figure 8 does not contain any Hamiltonian Path, because there is not any path that visits each vertex in the graph. On the other hand, the graph shown in Figure 9 contains two Hamiltonian Paths: A->B->C->D and A->B->D->C.

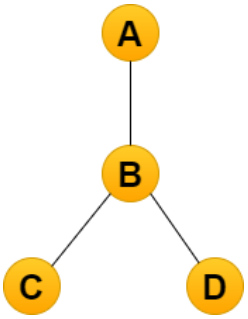


Figure 8. Graph without Hamiltonian Path

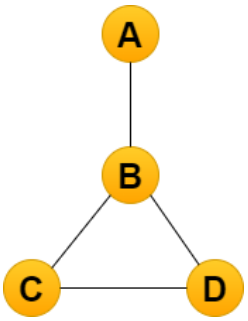


Figure 9. Graph with two Hamiltonian Paths

3.2 Hamiltonian Cycle Properties

According to the definition for Hamiltonion Cycle that we explained earlier in this section, we can notice that the following graph, shown in Figure 10 has a Hamiltonian Cycle and the path is following: A->G->F->E->C->D->B->A.

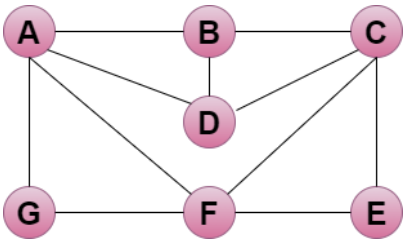


Figure 10. Graph that has a Hamiltonian Cycle

If we remove the edge BD we get the following graph (Figure 11), in which Hamiltonian Cycle does not exist:

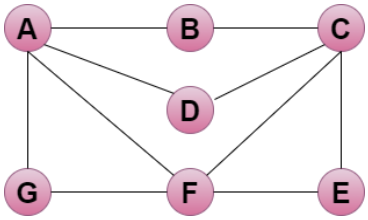


Figure 11. Graph that does not have a Hamiltonian Cycle

A sort of "proof" is the following consideration:
Any circuit can be written so that it starts at any vertex you like. So suppose you try to follow a circuit starting at vertex B. No matter whether you travel left or right, there is no way for you to go back to B without retracing an edge unless you skip vertex D. But if you skip vertex D, then you don't have a Hamilton circuit.
Also, there are graphs in which Hamiltonian Cycles are not possible, such as the graph that is shown in Figure 12. In this graph it does not matter from which vertex we are starting the cycle, because however it is not possible to return on the same starting vertex without re-visiting some vertex. In the graph below the vertex 'C' is called connecting point of the graph or articulation point. So, if there is an articulation point in a graph, then Hamiltonion Cycle is not possible in that graph.

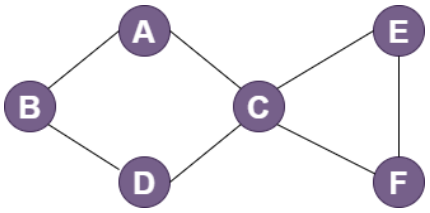


Figure 12. A Graph in which a Hamiltonion Cycle is not possible

An encouraging note is that we can list though a number of conditions that must be held in a graph that has a Hamilton circuit:
Assume G=(V, E) has more than 2 nodes

- No vertex of degree 1. If node a has degree 1, then the other endpoint of the edge incident to a must be visited at least twice in any circuit of G.
- If a node has degree 2, then both edges incident to it must be in any Hamilton circuit.
- No smaller circuits contained in any Hamilton circuit (the start/endpoint of any smaller circuit would have to be visited twice).
- There must exist a subgraph of G with the following properties:
    - H contains every vertex of G
    - H is connected
    - H has the same number of edges as vertices
    - H has every node with degree 2

Then , the subgraph H is the Hamilton circuit in the graph G

3.3 Finding Hamiltonian Circuit using Backtracking Algorithm (for determing a Hamiltonian Path)

Backtracking Algorithm is commonly used solution to determine if a given graph contains a Hamilatonian Cycle or not.
An input of the Backtracking algorithm is 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. As an example we consider the graph shown in Figure 13 and it's corresponding adjacency matrix is shown in Figure 14.

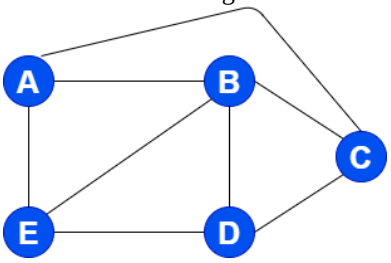


Figure 13. Graph for explaining the Backtracking Algorithm for finding Hamiltonian Cycle

	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	1	1
C	1	1	0	1	0
D	0	1	1	0	1
E	1	1	0	1	0

Figure 14. Adjacency matrix for the graph shown in Figure 12

First of all, we have to create an empty path array and add vertex A, as a starting point to it. It does not matter which vertex we choose as a starting point. The cycle can be started from any point. Then we add other vertices, starting from the vertex B. Before adding a vertex, we need to check in the adjacency matrix whether it is adjacent to the previously added vertex and to make sure that that vertex is not already added in the path array. 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.
The output of the Backtracking algorithm is an array path[V] that should contain the Hamiltonian Path. Path[i] should represent the i^{th} vertex in the Hamiltonian Path. The code return false if there is no Hamiltonian Cycle in the graph.
For the graph represented in the Figure 13, one of the Hamiltonian cycles that could be found using the Backtracking Algorithm is: A->B->E->D->C->A.

3.4 Applications of Hamiltonian Graphs

A search for these cycles isn't just a fun game for the afternoon off. It has real applications in such diverse fields as computer graphics, electronic circuit design, mapping genomes, and operations research. A very simple application is planning bus route to pick up students, where nodes are students, roads are edges and bus path is the Hamiltonian path.

For instance, when mapping genomes scientists must combine many tiny fragments of genetic code ("reads", they are called), into one single genomic sequence (a 'superstring'). This can be done by finding a Hamiltonian path or cycle, where each of the reads are considered nodes in a graph and each overlap (place where the end of one read matches the beginning of another) is considered to be an edge.

In a much less complex application of exactly the same math, school districts use Hamiltonians to plan the best route to pick up students from across the district. Here students may be considered nodes, the paths between them edges, and the bus wishes to travel a route that will pass each students house exactly once.

4 CONCLUSION

In this section, we're going to summarise all the theory that we discussed in the previous sections regarding Eulerian and Hamiltonian graphs, paths, cycles, algorithms and applications that these graphs are used for.
At this point it must be obvious that the similarity of the two problems discussed is just illusionary. There are many differences both in their practical appliances and their theoretical analysis. First of all the Eulerian problem takes edges in consideration while the Hamiltonian one has to do with vertices. The first one tries to pass through every edge of the graph exactly once, while the second one tries to visit all the nodes exactly once without retracing any edge. The nature of the problems shows that in the first case we are not interested whether the graph is weighted or not, while the second one has much bigger practical use if the graph is weighted.
The analysis showed that despite their "similarity" the Eulerian problem can be solved in linear time while the question whether a Hamilton cycle exists or not can only be answered in NP (nondeterministic polynomial) -time in the general case, O(N!) in fact!

References

[1] R. Williams, On the Application of Graph Theory to Computer Data Structures, Advanced Computer Graphics pp 775-801.

[2] David M. Clark, SUNY New Paltz, Graphs and Optimization, Journal of inquiry-based learning in Mathematics, No. 2, (Mar. 2007)

[3] Aigner, Martin; Ziegler, Günter (2009). Proofs from THE BOOK (4th ed.). Berlin, New York: Springer-Verlag.

[4] Abdul Samad Ismail , Roslan Hasni and K. G. Subramanian, "SOME APPLICATIONS OF EULERIAN GRAPHS", International Journal of Mathematical Science Education, © Technomathematics Research Foundation, Vol. 2, No. 2, 1 – 10, 2009.

[5] J.A. Bondy, V. Chvátal, A method in graph theory, Discrete Math. 15 (1976) 111–136.

[6] M. Sohel Rahman, M. Kaykobad, On Hamiltonian cycles and Hamiltonian paths, Information Processing Letters 94 (2005) 37–41.

[7] Hertel A, Hamiltonian cycles in sparse graphs. Master of Science Thesis, University of Toronto, (2004).

[8] Ashish Kumar, A study on Euler Graph and it"s applications, International Journal of Mathematics Trends

and Technology (IJMTT) – Volume 43 Number 1- March 2017.