Aviv Farag

Professor Boady W. Mark

CS502 - Data Structures & Algorithms

13 March 2022

Hierholzer's Algorithm

Graph theory is a branch in mathematics in which objects are represented as vertices and are connected using edges (either lines or arrows). This allow us to model and analyze various type of problems such as computer networks, and routes between several geographical locations. In this paper, we are going to review Hierholzer's algorithm which is used to find either an Euler path or an Euler circuit in a graph with linear time complexity. We will present the history of graph theory, Euler path, the problem that Hierholzer's algorithm solves, its application and complexity.

Background

Leonard Euler (1707 - 1783) was a Swiss influential figure who founded graph theory and topology. He is known for his contributions in mathematics, physics, and engineering [Dunham]. In the eighteenth century, there was a known problem among citizens of Konigsberg which is a city built on a river, so there were two islands connected to it with bridges. The citizens were curious to know if there is a way to walk across the city while crossing each of the seven bridges exactly once. Euler heard of this problem and published a paper describing his solution which was negative in that case. It was not possible to cross all bridges, each exactly once [Louridas].

In his work, Euler represented each landmass (islands and mainlands) by letter (A to D) as can be seen in figure 1. He drew lines to link between them removing the geometry of the city as presented in figure 2. According to Euler's solution if you enter a land that is neither the start nor the end, then you must exit this land. Therefore, those landmasses must have an even number of links (bridges). This was the beginning of graph theory founded by Euler who treated graph as an abstract data structure. A path is defined as a sequence of nodes and links, starting and ending at nodes. The problem described above is also known as Eulerian path, in which every link is used exactly once across the path. A further complication is to make

sure that the start and end node is exactly the same which is called an Eulerian circuit [Louridas].

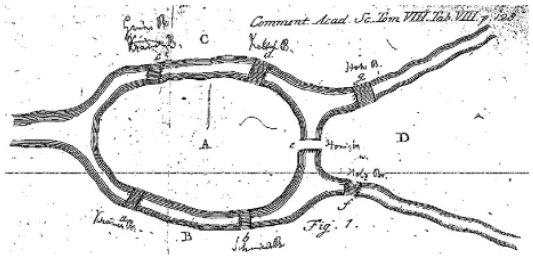

Figure 1: Map of Konigsberg [Louridas]

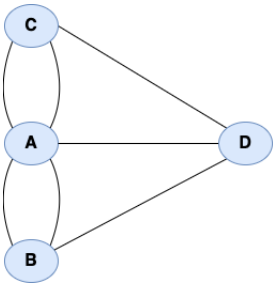

Figure 2: Graph of Konigsberg

There are two algorithms that find an Eulerian circuit in a graph: Hierholzer's algorithm and Fleury's algorithm. The former was published in 1873 [Hierholzer Carl] and is more efficient than the latter that was published in 1883.

The Problem

The goal is to find either an Euler circuit or an Euler path in a given graph. As mentioned in the background section, an Euler Path is such that every edge is used once, and an Euler Circuit is the same with a requirement that the starting node is also the ending node. Leonard Euler proved that Euler Paths exist in a graph if there are 2 odd degree vertices, while Euler Circuit exist if all nodes have an even degree (Pairs of incoming and outgoing edges cancelling one another).

Applications

Euler path and Euler circuit are being utilized in road transportation that requires traversing every edge exactly once such as The Chinese Postman Problem [Bondy], in computer networks to allow faster transmission of data [Fahad et al.], and to represent de Bruijn Sequences [Moreno] which reduce space required for storing information. For example, a set containing binary representation for numbers up to 15 ,{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111}, can be represented by the sequence 0000111100101101 as shown in the graph in figure 3. Using this method, one can encode 64 bits into only 16 bits.

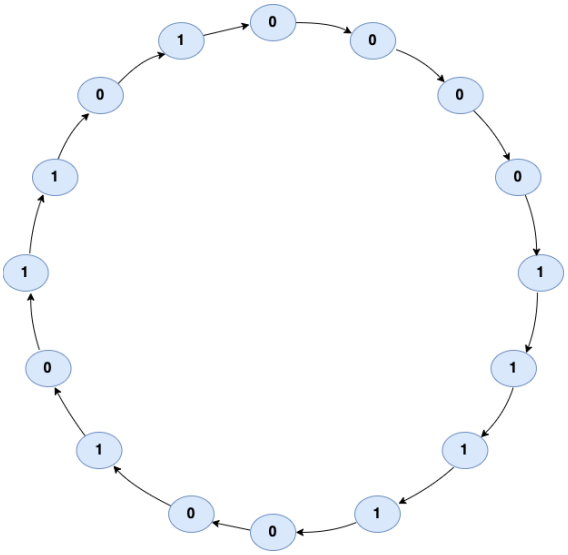

Figure 3: de-Bruijn sequence

The example in figure 3 is being utilized in sequencing DNA [O'Regan] [Pevzner et al.]. For instance, DNA sequence is composed of the letters A, T, G, and C. So, one can store the information by breaking the sequence into smaller pieces as can be seen in the figure below which represent the DNA sequence $ATGCGTGGCA$ [Louridas].

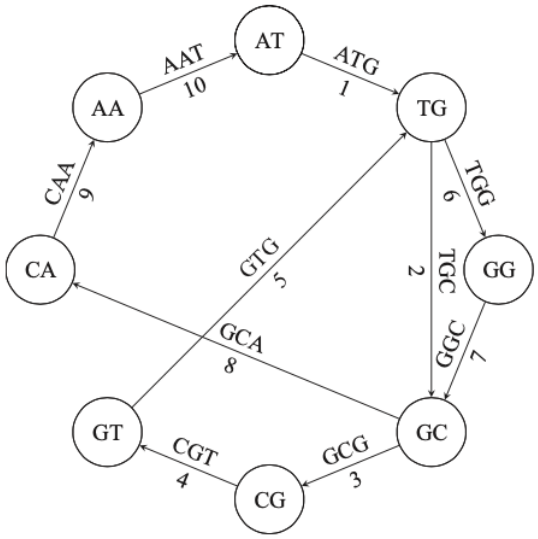

Figure 4: DNA fragments [Louridas]

Implementation

The implementation reviewed in this report relates to a directed graph, but can be easily modified in order to be utilized in an undirected graph as well. The difference is that in a directed graph one must follow the direction of the edges, as opposed to undirected in which any direction is applicable.

The first step is to ensure that there are either 0 or 2 odd degree vertices, the former is the case of an Euler Circuit, and the latter finds the Euler Path. In the first case (fig. 5) any node can be chosen as the starting point, while in the other case (fig. 6) the node with out-degree greater than the in-degree must be the starting point.

The algorithm, which appears in the next page, follows the edges and finds closed tours which are paths that start at node v, end at node v, but do not cover all of the nodes [Louridas]["Hierholzer's Algorithm"]. Since they do not traverse every edge, such tours are neither an Euler Circuit nor an Euler Path. While following edges, we remove them from the graph in order to prevent traversing them once again. The nodes that are visited, are being pushed into a data structure (e.g. stack) to store them according to their visiting order.

Back trailing occurs after a closed tour was found, there are no outgoing edges from the current node, and there exist edges in the graph that were no yet traversed. In this case, the algorithm goes back to the previous node and pushes the current node into another data structure (e.g. a stack). The algorithm back trails as long as the current node has no edges. Once all edges were removed from the graph, the algorithm merge the data structures and print the results.

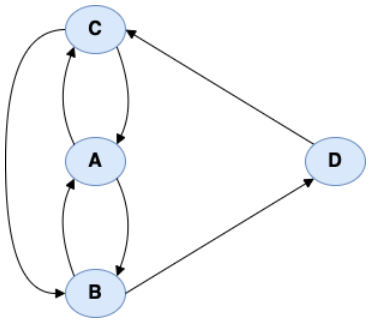

Figure 5: All nodes have even degree

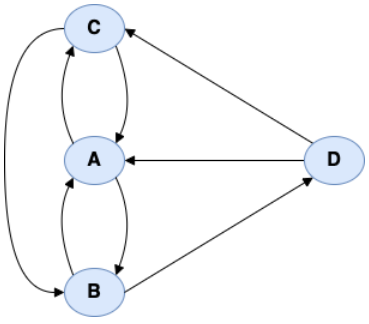

Figure 6: A & D have odd degree. D is the starting node

Algorithm 1: Hierholzer's Algorithm [Louridas]["Hierholzer's Algorithm"]

Input: G: Graph
Output: Print Euler Path or Circuit

```
1
2  if G has 0 odd degree nodes then
3  |  print("Euler Circuit: ")
4  |  v ← randomly chosen node from G
5  else if G has 2 odd degree nodes then
6  |  print("Euler Path: ")
7  |  v ← Node with out − degree > in − degree
8  else
9  |  print("No Euler Path/Circuit Exist")
   |  /* Abort function                                          */
10 |  return;
11 end
12
13 Initialize Stack path_st
14 Initialize Stack circuit_st
15 path_st ← v
16
17 while path_st is not empty do
18 |
19 |  u ← path_st.top
20 |
21 |  if u has no edges then
   |  |  /* Closed tour                                         */
22 |  |  push(path_st.pop, circuit_st)
23 |  else
   |  |  /* There are edges to go through                       */
24 |  |  x ← randomly chosen node connected to u
25 |  |  push(x, path_st)
26 |  |  Remove edge (u, x) from G
27 |  end
28 end
29
   /* Print Euler Circuit                                       */
30 while circuit_st is not empty do
31 |  print circuit_st.pop
32 end
```

Example

The figure below illustrate iterations of Hierholzer's Algorithm. Here all nodes have an even degree, so D was chosen randomly as the starting point. At each iteration, the current node is colored in orange, traversed edges were removed, and each stack status is shown with nodes on top of each stack appear on the right-most side.

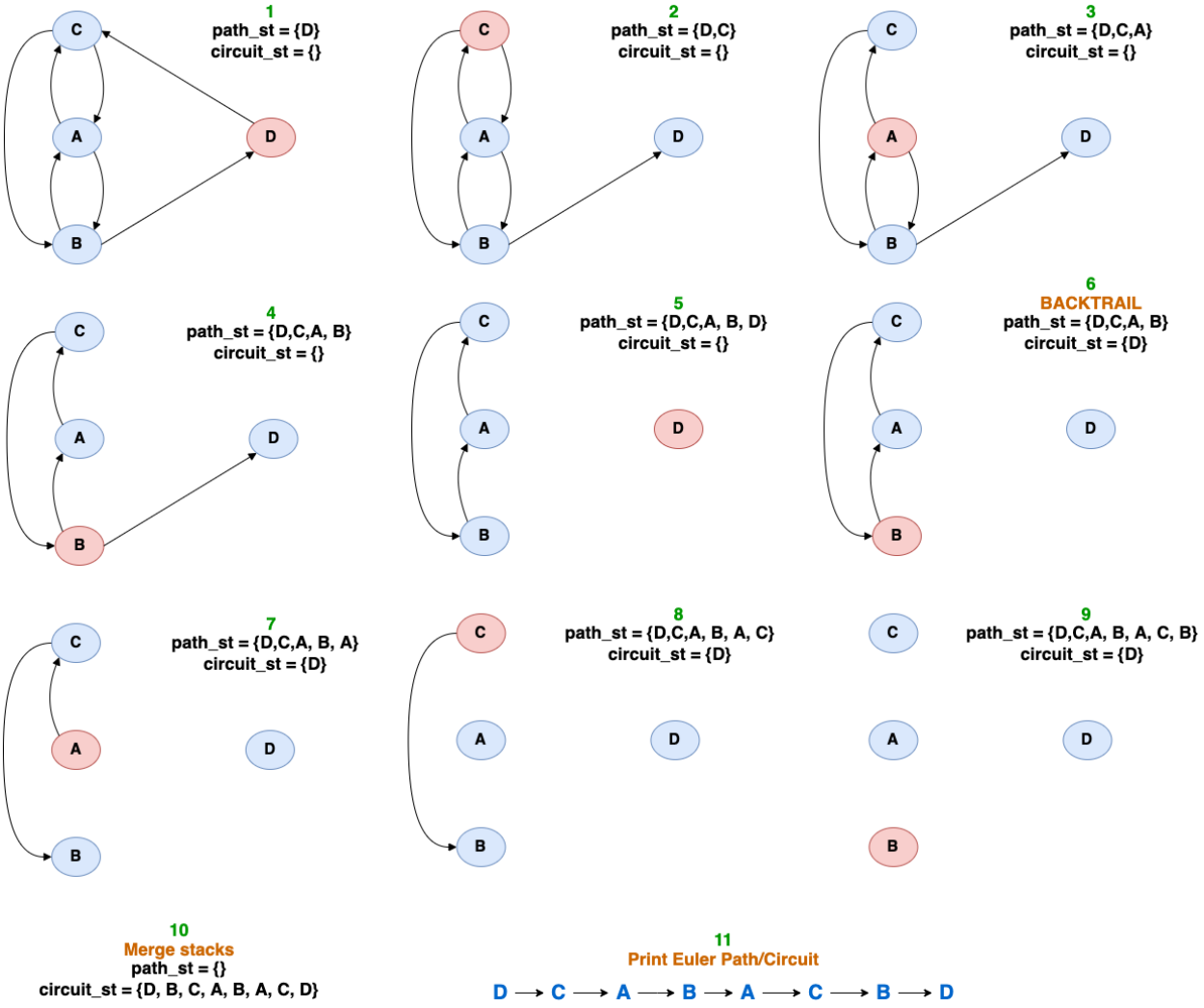

Figure 7: An example of Hierholzer's Algorithm

In iterations 1 to 5 the algorithm pushes nodes to *path_st* stack and delete all edges from the graph. Iteration 5 ends at node D and there is no out-coming edge, so in iteration 6 the algorithm back trail to the previous node (B). At this point, node D is being pop from *path_st* and pushed into *circuit_st*. Then, the algorithm follows other edges from node B to A, C and back to B. In the last stage (10), no edges are left, so

the nodes are being pop from *path_st* and pushed into *circuit_st* one after the other. Once *path_st* is empty, the algorithm prints the nodes by removing them one after the other from *circuit_st*. The printed circuit is either an Euler Path or an Euler Circuit depends on the number of odd degree vertices. In our example, the Euler Circuit is:

$$D \to C \to A \to B \to A \to C \to B \to D$$

Complexity

Hierholzer's algorithm is an efficient algorithm to find an Euler circuit in a given graph. Time complexity of this algorithm is $O(E)$ where E is the number of edges [Louridas]. This algorithm is more efficient than Fluery's algorithm whose time complexity is $O(E^2)$.

Conclusions

In this paper, we reviewed an algorithm for finding Euler Paths and Circuits within a graph. The algorithm runs through all edges, removing them from the graph, and storing the nodes in order to reconstruct the final path. It does so in linear time $O(E)$, where E is the number of edges, which is the best algorithm for that purpose. The other algorithm developed for that purpose runs in $O(E^2)$. Both algorithms have various type of applications such as DNA sequencing, and an efficient road transportation.

Works Cited

Bondy, J. A. (John Adrian). *Graph theory with applications*. Elsevier Science Publishing Co., 1976.

Dunham, William. *Euler : the master of us all*. The Dolciani mathematical expositions ; no. 22, Mathematical Association of America, 1999.

Fahad, Muhammad, et al. "Asymptotically Effective Method to Explore Euler Path in a Graph." *Mathematical problems in engineering*, vol. 2021, 2021, pp. 1–7. doi:10.1155/2021/8018373.

Hierholzer Carl, Wiener Chr. "Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren." *Mathematische Annalen*, vol. 6, 1873, pp. 30–32. doi:10.1007/BF01442866.

"Hierholzer's Algorithm." slaystudy.com/hierholzers-algorithm/.

Louridas, Panos. *Algorithms*. The MIT Press essential knowledge series, MIT P, 2020.

Moreno, Eduardo. "De Bruijn sequences and De Bruijn graphs for a general language." *Information processing letters*, vol. 96, no. 6, 2005, pp. 214–219.

O'Regan, Gerard. "Graph Theory." *Mathematics in Computing: An Accessible Guide to Historical, Foundational and Application Contexts*, Springer London, 2013, pp. 267–275, doi:10.1007/978-1-4471-4534-9_16.

Pevzner, P A, et al. "An Eulerian Path Approach to DNA Fragment Assembly." *Proceedings of the National Academy of Sciences - PNAS*, vol. 98, no. 17, 2001, pp. 9748–9753.