Eulerian path approach for genome assembly

Department of Computer science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri Campus
Kollam 690525
Kerala

By Namitha S

Intelligence of Biological Systems 3 -19BIO201
S3 B.Tech CSE(AI)

***Abstract*—One of the most important scientific developments of the last decades has been the decoding of the human genome. A fragment assembly in DNA sequencing followed the "overlap–layout–consensus" paradigm that is used in all currently available assembly tools. Here we are trying to find a path in the graph that visits all edges exactly once. This path can be called a Eulerian path and it exists if and only if each vertex has the same in-degree and out-degree and all vertices with non-zero degree belong to a single strongly connected component. The birth of Graph theory can be attributed to the work of the famous Swiss mathematician Leonhard Euler (1707 – 1783) in solving the problem of "Seven Bridges of Konigsberg". The method used by Euler in solving this problem gave rise to the notion of 'graph', which essentially is a discrete structure useful for modelling relations among objects.**
In my approach, we abandon the classical "overlap–layout–consensus" approach in favour of a new Euler algorithm by using the de bruijn graph. Our main result is the reduction of the fragment assembly to a variation of the classical Eulerian path problem that allows one to generate accurate solutions of large-scale sequencing problems.

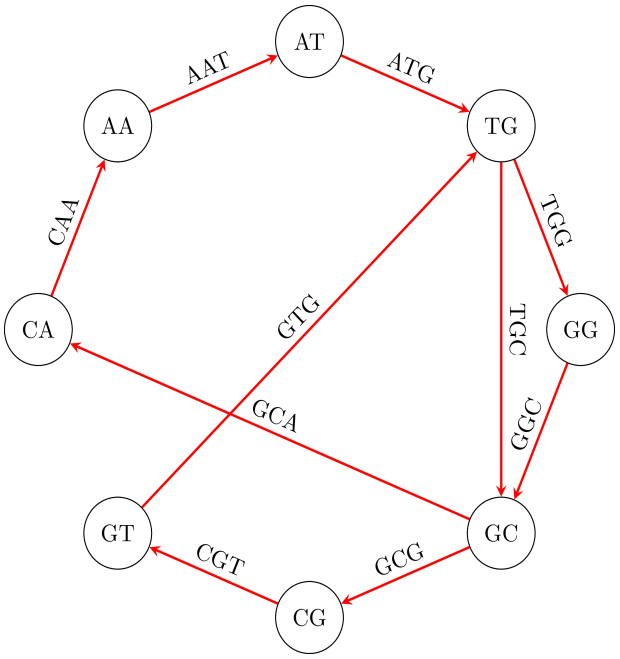
Keywords—Eulerian, DFS,De Bruijn

I. INTRODUCTION

Modern technology opens the opportunity for investigating genetic diseases, detecting mutations, studying genomes of extinct species, among other fascinating applications. 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.

Genomes are encoded in the DNA, a large organic molecule that is composed of a double helix. The double helix is made up of four bases: cytosine (C), guanine (G), adenine (A), and thymine (T). Each part of this double helix is constructed from a series of bases, like ACCGTATAG. The other part of the double helix is constructed from bases that are connected with their corresponding bases on the first part, according to the rules A-T, C-G.

In order to find the composition of an unknown DNA piece, we create many copies of the chain of the DNA sequence and then break them up into little fragments according to the user's wish. Hence this way we end up with a set of known fragments. We are then left with the problem of assembling the fragments to a DNA sequence, whose composition we will then know.
Let us consider an example :GTG, TGG, ATG, GGC, GCG, CGT, GCA, TGC, CAA, AAT. Each one of them has length 3. The size of k can be determined by the user according to the situation. To find the DNA sequence, we create a graph. In that graph, the vertices are polymers of length 2 (or, more generally, k−1) that are derived from the polymers of length 3 (or, k in the general case), taking for each polymer of length 3 (k) the first 2 (k−1) and the last 2 (k−1) polymers. So, from GTG we will get GT and TG, from TGG we will get TG and GG. In the graph, we add one edge for every one of the initial polymers or length 3 (k) that was used to derive the two vertices. We give the name of the polymer to that edge. So, from ATG we got vertices AT and TG and the edge ATG.



To find the initial DNA sequence we nee find a path in the graph that visits all edges exactly once which is known as Eulerian path and it exists if and only if each vertex has the same in-degree and out-degree and all vertices with non-zero degree belong to a single strongly connected component.

A. Eulerian

It was first discussed by Leonhard Euler while solving the famous Seven Bridges of Königsberg problem in 1736. Euler proved that there exists a condition for the existence of Eulerian circuits. The condition was that all vertices in the graph have an even degree, and stated without proof that connected graphs with all vertices of even degree have an Eulerian circuit. The first complete proof of this latter claim was published posthumously in 1873 by Carl Hierholzer.

Euler Graph - A connected graph G is called an Euler graph, if there is a closed trail which includes every edge of the graph G.A graph G=(V(G),E(G)) is considered Eulerian if the graph is both connected and has a closed trail (a walk with no repeated edges) containing all edges of the graph.

Euler Path - An Euler path is a path that uses every edge of a graph exactly once. An Euler path starts and ends at different vertices. For an Eulerian path, one can relax the above requirements. Exactly two nodes then have to have an odd degree. For both of these nodes, one edge remains at the end; therefore one has to leave one of them one more time and arrive at the other one more time. These two nodes will then be the start and end node of the Eulerian path. Such a graph is called semi-Eulerian.

To detect the path and circuit, we have to follow these conditions −

- The graph must be connected.
- When exactly two vertices have odd degree, it is an Euler Path.
- Now when no vertices of an undirected graph have odd degree, then it is an Euler Circuit.

Euler Circuit - A Euler circuit is a circuit that uses every edge of a graph exactly once. An Euler circuit always starts and ends at the same vertex. A connected graph G is an Euler graph if and only if all vertices of G are of even degree, and a connected graph G is Eulerian if and only if its edge set can be decomposed into cycles.It starts and ends on the same vertex.

B. De Bruijn Graph

The de Bruijn graph is a data structure first brought to bioinformatics as a method to assemble genomes from the experimental data generated by sequencing by hybridization. It later became the key algorithmic technique in genome assembly that resulted in dozens of software tools. In addition, the de Bruijn graphs have been used for repeat classification, de novo protein sequencing, synteny block construction, multiple sequence alignment, and other applications in genomics and proteomics.

C. Depth First Search traversal

DFS of a tree can begin from a node, then traverse its adjacent (or children) without caring about cycles. And if we begin from a single node (root), and traverse this way, it is guaranteed that we traverse the whole tree as there is no dis-connectivity. Depth-first search is a method for exploring a tree or graph. In a DFS, you go as deep as possible down one path before backing up and trying a different one. DFS is like walking through a corn maze. You explore one path, hit a dead end, and go back and try a different one. We are implementing DFS function in such a manner that it will return a number of nodes in the subtree whose root is a node on which DFS is performed.

Applications of DFS are:

- ➢ *Solving puzzles with only one solution.*
- ➢ *To test if a graph is bipartite.*
- ➢ *Topological Sorting for scheduling the job and many others.*
- ➢ *Detecting a cycle in a graph*

- ➢ *Finding a path in a graph*
- ➢ *Finding Strongly Connected Components of a graph*

For the problem at hand, graph traversal techniques need to be employed and so Depth First Search is used to traverse the graph.

II. Literature Review

The project had used An Eulerian path approach to local multiple alignments for DNA sequences by Yu Zhang and Michael S. Waterman present an Eulerian path approach to local multiple alignments for DNA sequences. The computational time and memory usage of this approach are approximately linear to the total size of sequences analyzed; hence, it can handle thousands of sequences or millions of letters simultaneously. By constructing a De Bruijn graph, most of the conserved segments are amplified as heavy Eulerian paths in the graph, and the original patterns distributed in sequences are recovered even if they do not exist in any single sequence. This approach can accurately detect unknown conserved regions, for both short and long, conserved and degenerate patterns. In graph theory, an n-dimensional De Bruijn graph of m symbols is a directed graph representing overlaps between sequences of symbols. It has m^n vertices, consisting of all possible length-n sequences of the given symbols. There can also be situations where the same symbol may appear multiple times in a sequence. A de Bruijn graph can be constructed for any length of sequences, may it be too short or too large. To construct a de Bruijn Graph, the first step is to choose a k-mer size and split the original sequence into its k-mer components. Then a directed graph is constructed by connecting pairs of k-mers with overlaps between the first k-1 nucleotides and the last k-1 nucleotides. The direction of the directed graph goes from the k-mer, whose last k-1 nucleotides are overlapping, to the k-mer, whose first k-1 nucleotides are overlapping. Our project involves the construction of de Bruijn Graph to form the graph to be given in the eulerian pathfinding algorithm

III. METHODS

The process of finding the eulerian path had many algorithms. Two of the algorithms include;

A. Fleury's algorithm

Fleury's algorithm is an elegant but inefficient algorithm that dates to 1883. Consider a graph known to have all edges in the same component and at most two vertices of odd degree. The algorithm starts at a vertex of odd degree, or, if the graph has none, it starts with an arbitrarily chosen vertex. At each step, it chooses the next edge in the path to be one whose deletion would not disconnect the graph, unless there is no such edge, in which case it picks the remaining edge left at the current vertex. It then moves to the other endpoint of that edge and deletes the edge. At the end of the algorithm there are no edges left, and the sequence from which the edges were chosen forms an Eulerian cycle if the graph has no vertices of odd degree, or an Eulerian trail if there are exactly two vertices of odd degree.

B. Hierholzer's algorithm

The basic idea of Hierholzer's algorithm is the stepwise construction of the Eulerian cycle by connecting disjunctive circles. Hierholzer's 1873 paper provides a different method for finding Euler cycles that are more efficient than Fleury's algorithm.

Algorithm:

1) *We pick a starting node, u.*
2) *We go from node to node until we return to u. The path that we have traced to this point does not necessarily cover all edges. It is not possible to get stuck at any vertex other than u, because the even degree of all vertices ensures that, 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.*
3) *As long as there exists a vertex v that belongs to the path we have traced, but is part of an edge that does not belong to that path:*
   a) *We start another path from v, using edges that we have not used yet until we return to v. Then we splice this path to the path we have already traced.*
4) *Since we assume the original graph is connected, repeating the previous step will exhaust all edges of the graph.*

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.

The algorithm this paper has chosen is Hierholzer's algorithm as it is more efficient than Fleury's algorithm.

C. Dataset

We will be taking a genome sequence to be used as an input to our project, to find the eulerian path of it.I have tried two datasets on the program.

[1] In our implementation, I have tried on: Human coronavirus OC43 (*to isolate KEN/CHA001/2016 spike glycoprotein (S) gene, partial cds (Abidha,C.A., et al.).Coronaviruses are enveloped, positive-stranded RNA viruses with a genome of approximately 30 kb.)* Based on genetic similarities, coronaviruses are classified into three groups. Two group 2 coronaviruses, human coronavirus OC43 (HCoV-OC43) and bovine coronavirus (BCoV), show remarkable antigenic and genetic similarities. Human coronavirus (HCoV)OC43 belongs to the Betacoronavirus genus of the Coronaviridae and its genome is formed by a positive-sense, single-stranded RNA of ca. 31.5 kb . HCoV OC43 and HCoV 229E are responsible for one-third of all common colds, infecting all age groups, but there have been reports of a more severe lower respiratory tract involvement.

ATTRIBUTES
Nuc Completeness: partial
Length: 2835
Mol Type: RNA
Host: Homo
Geo Location: Kenya: Kilifi
Collection Date: 2016-01-22
Available from: https://www.ncbi.nlm.nih.gov/nuccore/MN630522

[2] I have also tried the dataset :Human coronavirus 229E (*isolate HCoV 229E/human/ITA/TE5146/2020 replicase polyprotein 1ab and replicase polyprotein 1a genes, partial cds; and spike protein, ORF4a, ORF4b, envelope protein, membrane protein, and nucleocapsid protein genes, complete cds*.) The Human coronavirus strain 229E (HCoV-229E), is one of the causative agents of the common cold, is increasingly associated with more severe respiratory infections in children, elders, and individuals with underlying medical conditions . The virus also has neuroinvasive and neurotropic properties. The presence of HCoV-229E within clinical samples (e.g., nasal or throat swabs, nasopharyngeal aspirate, bronchoalveolar lavage, or saliva) is typically confirmed by molecular or serological methods .

ATTRIBUTES
Nuc Completeness: partial
Length: 26811
Mol Type: RNA
Host: Homo sapiens
Isolate: oronasopharynx
Geo Location: Italy: Atri-Teramo
Collection Date: 2020-03-18
Available from :https://www.ncbi.nlm.nih.gov/nuccore/MW039392

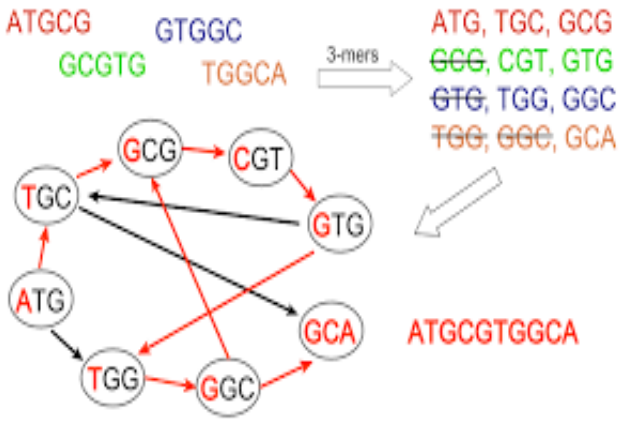
IV. IMPLEMENTATION

The project aims in building the eulerian path of the dataset using Hierholzer's algorithm. To begin, the input genome sequence is given in the form of a txt file, with the genome dataset in it. This dataset is being read by the program. In order to find the composition of the unknown DNA piece, we create many copies of the chain of the DNA sequence and then break them up into little fragments according to the user's wish. Hence this way we end up with a set of known fragments. We are then left with the problem of assembling the fragments to a DNA sequence, whose composition we will then know. The value of k to build the k-mers is being given by the user and based on that the genome sequence is formed. We then move forward in making a de Bruijn graph out of the k-mers .To make a de Bruijn graph from the set of DNA sequences, we break each sequence into overlapping k-tuples. Two adjacent k-tuples overlap at (k-1) letters. Each k-tuple is represented by a directed edge in the graph, and identical k-tuples are represented by the same edge. Two edges are connected by a node if they represent at least one pair of adjacent k-tuples in a sequence, where the node represents the (k-1)-tuple overlap. Finally, we record the position and sequence index of each k-tuple in the corresponding edge. During the formation of the de Bruijn graph, we can notice that duplicate elements and the last element are removed by a function to make sure to get a unique output of the graph.

This helps us in forming vertices and edges. These vertices and edges are given as the input to our Hierholzer's algorithm.

In the algorithm, we traverse from node to node using Depth-first search traversal and it continues until we reach the starting node. The argument vertices is a list and contain the vertices traversed. If all adjacent edges of the starting vertex are already traversed, 'vertices' are empty after the call. This yields the first circle in the graph. If this circle covers all nodes it is an Eulerian cycle and the algorithm is finished. Otherwise, one chooses another node among the cycles' nodes with unvisited edges and constructs another circle, called sub tour. By choice of edges

in the construction, the new circle does not contain any edge of the first circle. However, both circles must intersect in at least one node by choice of the starting node of the second circle. Hence both circles can be represented as one new circle. To do so, we iterate the nodes of the first circle and replace the sub tours starting node by the complete node sequence of the sub tour. Thus, one integrates additional circles into the first circle. If the extended cycle does include all edges the algorithm is finished. Otherwise, we can find another cycle to include. An example of this process is shown below. After returning the list of the vertices of the eulerian path, the , list is joined together to form an assembled dna sequence of the eulerian path.



V. RESULT

The program was successful in finding the eulerian path of the genome sequence with Hierholzer's algorithm. The program's output will be the DNA sequence that it assembled. Few results are shown below:-

The result from the Human coronavirus OC43 dataset of genome sequence are:-

Value of k	Eulerian path
2	TCGTCAATGGT
3	TCAGCAATGTGGACATTTTGAAAGCTGTA TACTCATCCTCTCCCCGCACGGGTAGC
4	TCAATAATTGCGTATGTAGGGCCCGAGTAT GGCGATTTCGTTTGTCAATCACCCACTCT CAGAATCAGTTGGGATCTAAGGACGCTCA CAAGTGAGATATAAATTGTCCCTTATACAT CTTTTTCTCTAAGTCCTCCCAAAAGCAAG AACAATGTGGATAGCACATAGAAACAGGG CCGGAGGGTTTTCCCACAATAGTCGGCG
	CTCTATA
5	TCAAAATTTTATGGGTTCTGTAGCATTTTT ATCAACTTGTAAATATTTTTGAGTTTGTCT GATTTGCGTCTATATATAATCGCAACAACC CAGATAAACCCAGTCACAAGAGTGATGTA AGAACTACATTCACTATTTGAGGTGAGGT GCTATACTAAATCGAGAATACAGATAGAAT AGATTTCCTCGAGTCCTCCCTGTCAAAAG CTTTCATAGAGCACGGAGATTGTATGGTG GCGCGGTGTACACCACTCCCCACAAAGC CTCTCATGTGTTGTTATTATTGTTGTCTCG TTGCTCGTTGAAATCGGGATTAGGGAATA CTATATAAAAAAAACCCCGTTGGGAAATTT AACAGACCTTTCCCACAACTTATGGAGA

The results from the Human coronavirus 229E dataset is :-

Value of k	Eulerian Path
2	ATTGCAGACCA
3	ATCTGTCCACGTTTGCATATATATTTGCCC CAGAAAGTCTTATCGCAGGGTAGCGGCA AA
4	ATCGCTTTCTGACCCTCACATGGGTTCCT CTCTTGGAAAGACGTGTATTCGTAATAATG TTTATAAACTCCGCACGTGGGTTGAGTAT GTGCTATCGTTAAAGAACACAGTGCCAGA CGGAGATAGTTTCTGAACTGAACAACTTA TGCCCACGTGCTTTCATCCATACGCAATT TAGCGATCGTGACATCGGGCGAGCATGT CAGCTCGGGCCATTAGAGCGGGGAGCG GACGCTATCAGAACCCCACCCCA
5	ATCGTGCTGTTGTACGGGTTTTTGGCTTC GCTTATATTGTGGAGTGGGCGGTGAGAG AATATAGCTGTCTCGACACACCCAGATAG TGATAGAAAGGGAAGTGAAACTCAGTCAA AAACAGCAACAACGGGAAACAAATACAAT AACAAATTTTGAAAGCTTTTTCAAACATGT ATGTAACTTCTCCCTTTCTTGGACTATCAT GACGTGCAGTACGAGGGATTATTCCATAC CGAGTGCACCGTCACTGAGTCTGCGTGT TGTTTTGCGTACTTTCCACTTTATTTCTTTT TAGTTTGATTGTCCCACTTACGTTTTGGGT TCACTTCTCTGCTAAATCTGGGTGGTAAC AGTGATTTGTAACCTCTCCTGTAAATGTAC CTAAATTGCCTGTTCTTGGTTTGCTGCGG AGGTCTGCTCACTATATAAATCACTAGCGA TTTCGCAGATCTGAGGCGTATGTGCTACT CGCATGTATCTGTCCCCCAGTGCCAGAG CTCGAGATAAAACCTTCTGTGGGTGATGG GAGCGTTTCTATATACGCCCGAGGGACAG TGATTAAAAAGACGCAGCGGGCGTATCC GTTCGCTTACGATACCACCGTCAGACGTG GAAACATAGACATGGGAACATTATATCTCG CTGAAAGTATAAAAGACAATGTAGATTGAC TAGATCGCGCATTACAAACGCGACCCCCC GCGAGTACGCTCGTTATGTTCAAGAACAC CCACCACAAACTGTCCATCCTACGCTCCG CTTATAGATACGTAACATCGGGCACGAAG

	AGGCCGAAAGGAGTGGGACCCGACCCC TAGCGTAGATGTCTGAGTCTCCACCCAGT CTCGACTCGCCAGGGAACATGTGTCGGA AGGCACGTCTCCACTCTACCGGCCAGCC GAGATAGTGGA

VI. DISCUSSIONS

I have developed a eulerian path finding approach of genome assembly using the hierholzer's algorithm and have come out with satisfying outputs. Both the results showed real success even though there are chances of the existence of eulerian circuits in the graph.We can try out different types of the dataset to experiment with the project to get the eulerian paths.The approach we used will be able to predict the eulerian path of many more datasets with larger sizes .

VII. CONCLUSION

Eulerian paths can be used in bioinformatics to reconstruct the DNA sequence from its fragments. They can also be used in a CMOS circuit design to find an optimal logic gate order. There are some algorithms for processing trees that rely on an Euler tour of the tree. The de Bruijn sequences can be constructed as Eulerian trails of de Bruijn graphs. Eulerian graphs can be used to solve many practical problems like the Konigsberg Bridge problem. They can also be used by mail carriers who want to have a route where they don"t retrace any of their previous steps. Euler graphs are also useful to painters, garbage collectors, aeroplane pilots and all world navigators.

References

[1] Eulerian path https://en.wikipedia.org/wiki/Eulerian_path

[2] National Center for Biotechnology Information https://www.ncbi.nlm.nih.gov/

[3] What is the difference between the breakpoint graph and the de Bruijn graph? PA. Pevzner-MS. RM. Idury-H. PA. Pevzner-E. DR. Zerbino-PA.

[4] https://www.tutorialspoint.com/Eulerian-Path-and-Circuit

[5] Idury, R. & Waterman, M. S. (1995) J. Comp. Biol. 2, 291–306.

[6] Pevzner, P. A., Tang, H. & Waterman, M. S. (2001) Proc. Natl. Acad. Sci. USA

[7] Zhang, Y. & Waterman, M. S. (2003) J. Comp. Biol. 10, 803–819.

[8] Smith, R. F. & Smith, T. F. (1990) Proc. Natl. Acad. Sci. USA 87, 118–122.

[9] Smith, R. F. & Smith, T. F. (1992) Protein Eng. 5, 35–41.

[10] Waterman, M. S. & Jones, R. (1990) Methods Enzymol. 183, 221–237.

[11] Schuler, G. D., Altschul, S. F. & Lipman, D. J. (1991) Proteins 9, 180–190

[12] Carl Hierholzer. Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. Mathematische Annalen, 6(1):30–32, 1873.

[13] Complete Genome Sequence of Human Coronavirus Strain 229E Isolated from Plasma Collected from a Haitian Child in 2016
Tania Bonny-Kuttichantran Subramaniam-Thomas Waltzek-Maha Elbadry-Valery Beau De Rochars-Taina Telisma-Mohammed Rashid-J Morris-John Lednicky - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5701476/

APPENDIX

```
import itertools
from collections import OrderedDict
str=''
list_final=[]
stri=[]
strings=[]
fir=[]
V=[]
E=[]
listf=[]
out=[]
class Graph:
    def __init__(self, nodes=None, edges=None):
        self.nodes, self.adj = [],{}
        if nodes != None:
            self.add_nodes_from(nodes)
        if edges != None:
            self.add_edges_from(edges)
    def __iter__(self):
        return iter(self.nodes)
    def __getitem__(self, x):
        return iter(self.adj[x])
    def add_node(self, n):
        if n not in self.nodes:
            self.nodes.append(n)
            self.adj[n] = []
    def add_nodes_from(self, i):
        for n in i:
            self.add_node(n)
    def add_edge(self, u, v):   # undirected
unweighted graph
        self.adj[u] = self.adj.get(u, []) + [v]
        self.adj[v] = self.adj.get(v, []) + [u]
    def add_edges_from(self, i):
        for n in i:
            self.add_edge(*n)
```

```
    def number_of_edges(self):
            return sum(len(l) for _, l in
self.adj.items()) // 2
#removes the duplicates and its last element
def rem(strings):
    if(len(strings[-1])!=len(strings[0])):
            strings.pop()
    reads = list(OrderedDict.fromkeys(strings))
    return reads
def hierholzer(g):
    start = next(g.__iter__())
    circuit = [start]
    traversed = {}
    ptr = 0
    while len(traversed) // 2 <
g.number_of_edges() and ptr < len(circuit):
            subpath = []
            dfs(g, circuit[ptr], circuit[ptr],
subpath, traversed)
            if len(subpath) != 0:
    circuit = list(itertools.chain(circuit[:ptr+1],
subpath, circuit[ptr+1:]))
            ptr += 1
    return circuit
def dfs(g, u, root, subpath, traversed):
    for v in g[u]:
            if (u,v) not in traversed or (v,u) not in
traversed:
                    traversed[(u,v)] =
traversed[(v,u)] = True
                    subpath.append(v)
                    if v == root:
                            return
                    else:
                                    dfs(g, v, root,
subpath, traversed)
######
for i in open("dna1.txt","r"):
    strl=i.split()
    stri=stri+strl
for line in stri:
    str=str+line
k=int(input("Enter value of k: "))
for m in range(0,len(str),k):
    strings.append(str[m:m+k])
reads=rem(strings)
#make vertices and edges
for st in reads:
    fir.append(st[:-1])
    fir.append(st[1:])
    t=(st[:-1],)+(st[1:],)
    E.append(t)
V=rem(fir)
g = Graph(nodes=V, edges=E)
out=hierholzer(g)
first=out[0]
for i in out:
    list_final.append(i.rstrip())
finalstring=""
finalstring=finalstring+first
del list_final[0]
for i in list_final:
    finalstring=finalstring+i[len(first)-1]
listf.append(finalstring)
print("Eulerian Path :")
print(listf)
```