19BIO201 IBS -3

Assignment 3

Topic- Hamiltonian String Reconstruction

1. Write a program in any programming language (Java or Python) to carry out string reconstruction based on Hamiltonian path method for the sequences given below (Attempt for k=3 & k=5).

a) TAATGCT
b) GCGGTAATGCAGTCGAC
c) TTGAGTGCCAGCGAGCTAGGTCAGT

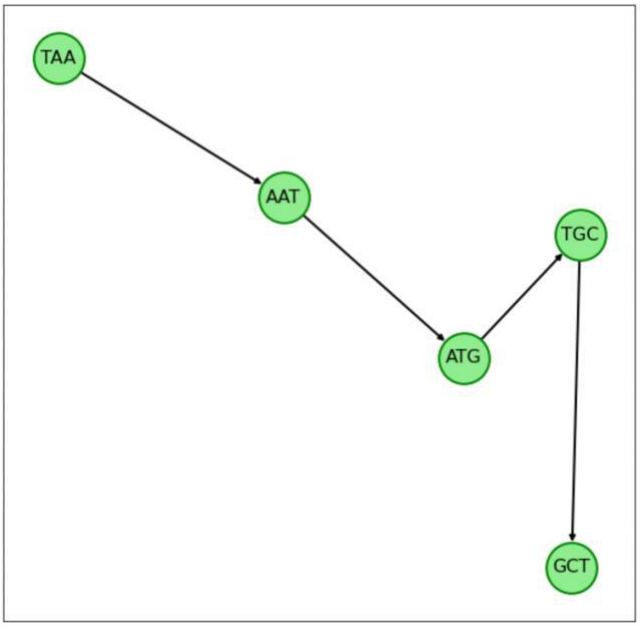
CODE: -

```
# Amruth
# BL.EN.U4AIE20002
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import pylab
import timeit
start = timeit.default_timer()
def kmers(read, k):
    KList=[]
    num_kmers = len(read) - k + 1
    for i in range(num_kmers):
        kmer = read[i:i+k]
        if i==0:
            KList.append(kmer)
        else:
            KList.append(kmer)
#   the below code sorts the kmer list
    for i in range(len(KList) - 1):
        for j in range(i + 1, len(KList)):
            if KList[i] > KList[j]:
                temp = KList[i]
                KList[i] = KList[j]
                KList[j] = temp
    return KList

input1="TAATGCT"
a=kmers(input1,3)

print(f"Kmers={a}")
print(" ")


def KmerListOfNodes(ls):
    ListOfNodes=[]
    for current in ls:
        for i in range(len(ls)):
            temp = ls[i]
            if current[1:len(current)]==temp[0:len(temp)-1]:
                ListOfNodes.append((current,temp))
    return ListOfNodes

print(KmerListOfNodes(a))
```

```
Kmers=['AAT', 'ATG', 'GCT', 'TAA', 'TGC']

edges = [('AAT', 'ATG'), ('ATG', 'TGC'), ('TAA', 'AAT'), ('TGC', 'GCT')]
```

```
# for diplaying the graph
G = nx.DiGraph()
G.add_edges_from(KmerListOfNodes(a),length=15)
totalNodes=G.nodes()
pos = nx.spring_layout(G)
options = {
    "font_size": 16,
    "node_size": 2000,
    "node_color": "lightgreen",
    "edgecolors": "green",
    "linewidths": 2,
    "width": 2,
    "edge_vmin":5
}
nodes=G.nodes()
edges=G.edges()
print(f"nodes = {nodes()}")
print(" ")
print(f"edges = {edges}")


plt.figure(2,figsize=(10,10))
nx.draw_networkx(G,pos,**options)
plt.show()
```

```
nodes = ['AAT', 'ATG', 'TGC', 'TAA', 'GCT']

edges = [('AAT', 'ATG'), ('ATG', 'TGC'), ('TGC', 'GCT'), ('TAA', 'AAT')]
```



```python
# assigning numerical values to the kmers
# so that adjacency matrix can be made
edge1=list(edges)
node1=list(nodes)
# print(node1)
# print(edge1)
DictNode = {}
count=0
for i in node1:
    DictNode.update({i:count})
    count+=1
print(DictNode)


def replaceKey(dict, key):
    if key in dict.keys():
        return dict[key]

edgeAlternate = [[0,0] for i in range(len(edge1))]

for i in range(len(edge1)):
    edgeAlternate[i][0]=replaceKey(DictNode,edge1[i][0])
    edgeAlternate[i][1]=replaceKey(DictNode,edge1[i][1])

print(edgeAlternate)
```

```
{'AAT': 0, 'ATG': 1, 'TGC': 2, 'TAA': 3, 'GCT': 4}
[[0, 1], [1, 2], [2, 4], [3, 0]]
```

```python
possibleOutcomes=[]
class Graph:
    # Constructor
    def __init__(self, edges, n):

        self.adjList = [[] for _ in range(n)]

        for (src, dest) in edges:
            self.adjList[src].append(dest)
            self.adjList[dest].append(src)

def hamiltonianPaths(graph, v, visited, path, n):

    if len(path) == n:
        # print the Hamiltonian path
        print(path)
        new = path
        # I have no idea why I put 2 lines below, it works DONT TOUCH!!-by Amruth
        # got it with trial and error
        for _ in range(n-1):
            new.append("hi")
        possibleOutcomes.append(new)
        return

    # Check if every edge starting from vertex `v` leads to a solution or not
    for w in graph.adjList[v]:

        # process only unvisited vertices as the Hamiltonian
        # path visit each vertex exactly once
        if not visited[w]:
            visited[w] = True
            path.append(w)

            # check if adding vertex `w` to the path leads to the solution or not
            hamiltonianPaths(graph, w, visited, path, n)

            # backtrack
            visited[w] = False
            path.pop()
```

```
def findHamiltonianPaths(graph, n):

    # start with every node
    for start in range(n):

        # add starting node to the path
        path = [start]

        # mark the start node as visited
        visited = [False] * n
        visited[start] = True

        hamiltonianPaths(graph, start, visited, path, n)


if __name__ == '__main__':
    edges = edgeAlternate
    print(edges)
    n = len(nodes)
    graph = Graph(edges, n)
    findHamiltonianPaths(graph, n)
```

```
[[0, 1], [1, 2], [2, 4], [3, 0]]
[3, 0, 1, 2, 4]
[4, 2, 1, 0, 3]
```

```
# cconvertring numerical values back to Kmers
new_dict = {value:key for (key,value) in DictNode.items()}
replaceKey(new_dict,0)

final = [[0 for _ in range(len(possibleOutcomes[0]))] for k in range(len(possibleOutcomes))]

for i in range(len(possibleOutcomes)):
    for j in range(len(possibleOutcomes[0])):
        final[i][j]=replaceKey(new_dict,possibleOutcomes[i][j])

print("All possible hamiltonian paths in the graph = ")
print(final)
```

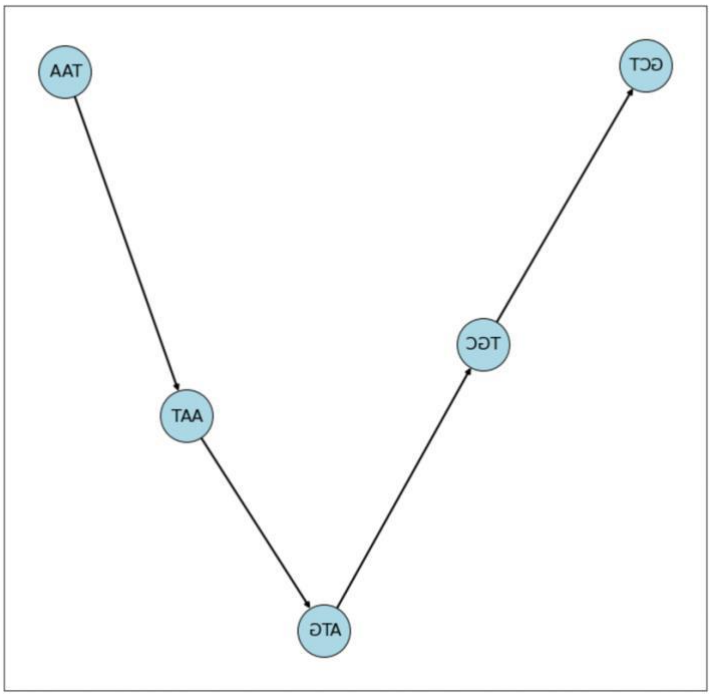
```
def FindHamiltonianString(ListofLists):
    Hamiltonian_String=[]
    for subList in ListofLists:
        m=subList[0]
        n=subList[1]
        i=0
        if m[1:]==n[:-1]:

            while i<len(subList)-2:
                m=subList[i+1]
                n=subList[i+2]
                if m[1:]==n[:-1]:
                    i+=1
                else:
                    break
            if i+2 == len(subList):
                Hamiltonian_String.append(subList)


    return Hamiltonian_String

required=FindHamiltonianString(final)
print(" ")
print("Succesful Hamiltonian Paths = ")
print(required)

# reconstructing the successful hamiltonian path
def reconstruct(List):
    j=0
    newString=""
    for i in range(len(List)):
        if len(List)-1==i:
            new1=List[j]
            newString=newString+new1
            j+=1
        else:
            new1=List[j]
            newString=newString+new1[0:1]
            j+=1
    return newString
```

```
print()
for i in required:
    print(f"Reconstructed String = {reconstruct(i)}")
```

```
All possible hamiltonian paths in the graph =
[['TAA', 'AAT', 'ATG', 'TGC', 'GCT'], ['GCT', 'TGC', 'ATG', 'AAT', 'TAA']]

Succesful Hamiltonian Paths =
[['TAA', 'AAT', 'ATG', 'TGC', 'GCT']]

Reconstructed String = TAATGCT
```

```
# diplaying the hamiltonian path

G1 = nx.DiGraph()
G1.add_edges_from(KmerListOfNodes(required[0]),length=15)

totalNodes=G1.nodes()
pos = nx.spring_layout(G1)
options = {
    "font_size": 16,
    "node_size": 2500,
    "node_color": "lightblue",
    "edgecolors": "black",
    "linewidths": 1,
    "width": 2,
    "edge_vmin":5
}

nodes=G1.nodes()
edges=G1.edges()
print(f"nodes = {nodes()}")
print(" ")
print(f"edges = {edges}")

plt.figure(1,figsize=(14,14))
nx.draw_networkx(G1,pos,**options)
plt.show()

print()
print(f"  Original String   = {input1}")
for i in required:
    print(f"Reconstructed String = {reconstruct(i)}")
```

```
nodes = ['TAA', 'AAT', 'ATG', 'TGC', 'GCT']

edges = [('TAA', 'AAT'), ('AAT', 'ATG'), ('ATG', 'TGC'), ('TGC', 'GCT')]
```




```
  Original String   = TAATGCT
Reconstructed String = TAATGCT
```

```
stop = timeit.default_timer()
print('Run Time: ', stop - start)
```

```
Run Time:  0.2807773750000706
```

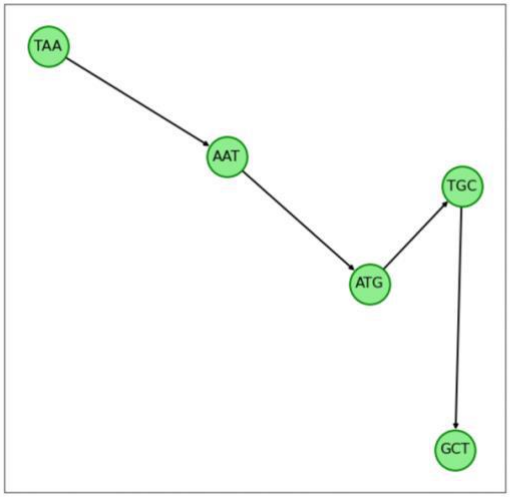
1. Case a

K=3

```
input1="TAATGCT"
a=kmers(input1,3)
```
(k=3)

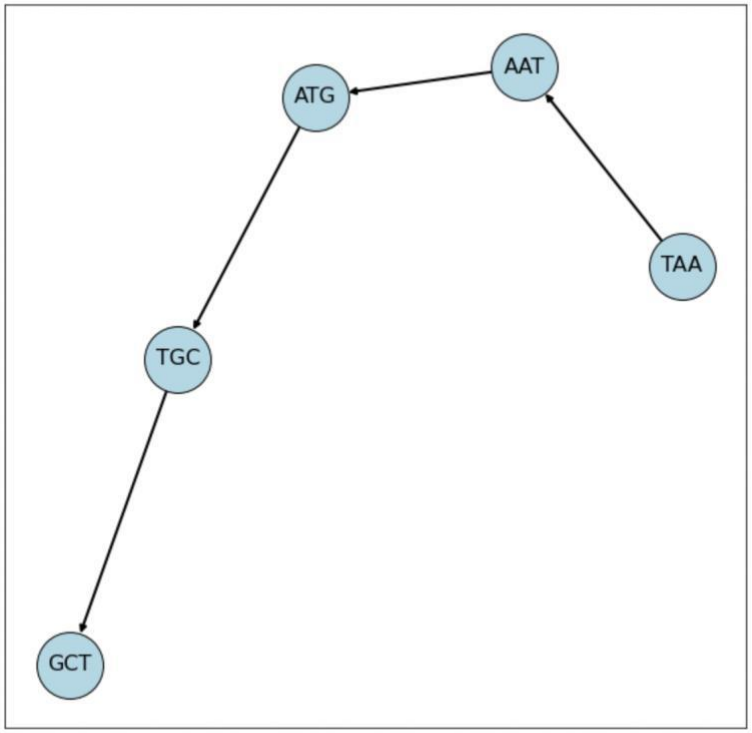
Nodes and Edges of the Constructed Graph

```
nodes = ['AAT', 'ATG', 'TGC', 'TAA', 'GCT']
edges = [('AAT', 'ATG'), ('ATG', 'TGC'), ('TGC', 'GCT'), ('TAA', 'AAT')]
```



Hamiltonian path after string reconstruction:

```
nodes = ['TAA', 'AAT', 'ATG', 'TGC', 'GCT']
edges = [('TAA', 'AAT'), ('AAT', 'ATG'), ('ATG', 'TGC'), ('TGC', 'GCT')]
```




```
  Original String   = TAATGCT
Reconstructed String = TAATGCT
```

```
Run Time:  0.2664676250001321
```

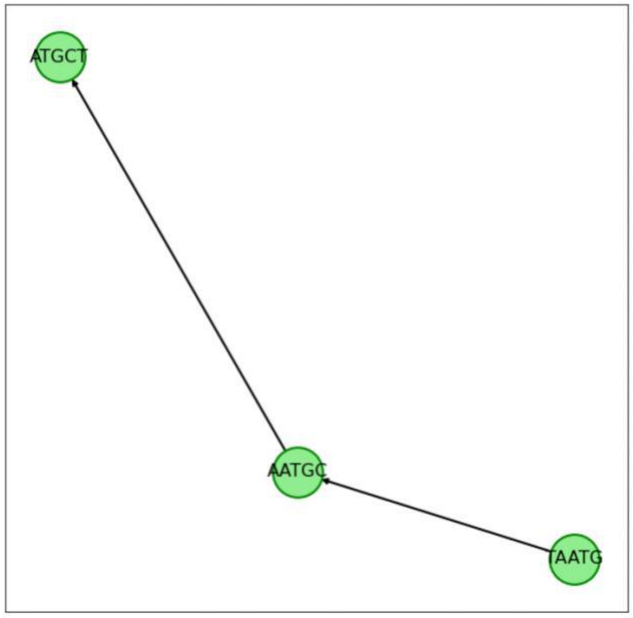
Note: the previous graph and this looks similar, but when the string length is higher, there is a considerable difference

K= 5

```
input1="TAATGCT"
a=kmers(input1,5)
```

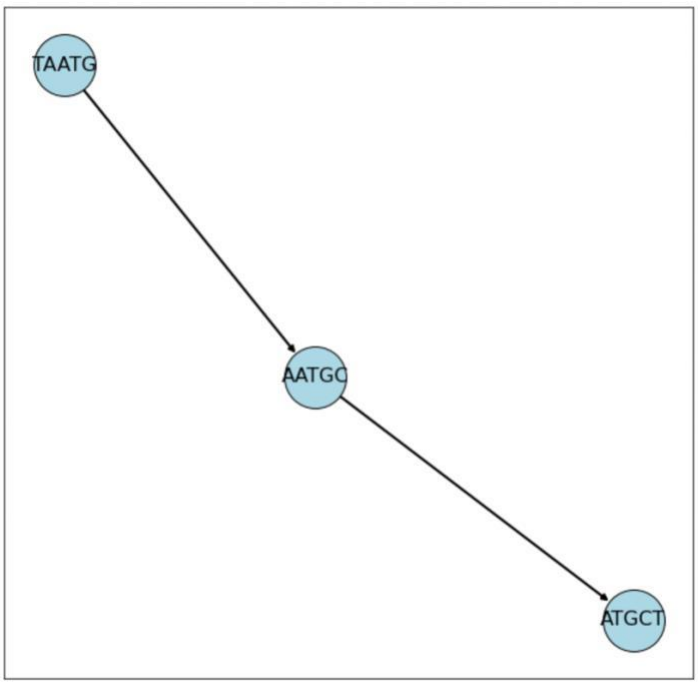
Nodes and Edges of the Constructed Graph

```
nodes = ['AATGC', 'ATGCT', 'TAATG']

edges = [('AATGC', 'ATGCT'), ('TAATG', 'AATGC')]
```



Hamiltonian path after string reconstruction:

```
nodes = ['TAATG', 'AATGC', 'ATGCT']

edges = [('TAATG', 'AATGC'), ('AATGC', 'ATGCT')]
```




```
  Original String   = TAATGCT
Reconstructed String = TAATGCT
```

```
Run Time:  0.27530504199967254
```

NOTE: We got The desired outputs with both k=3 & 5

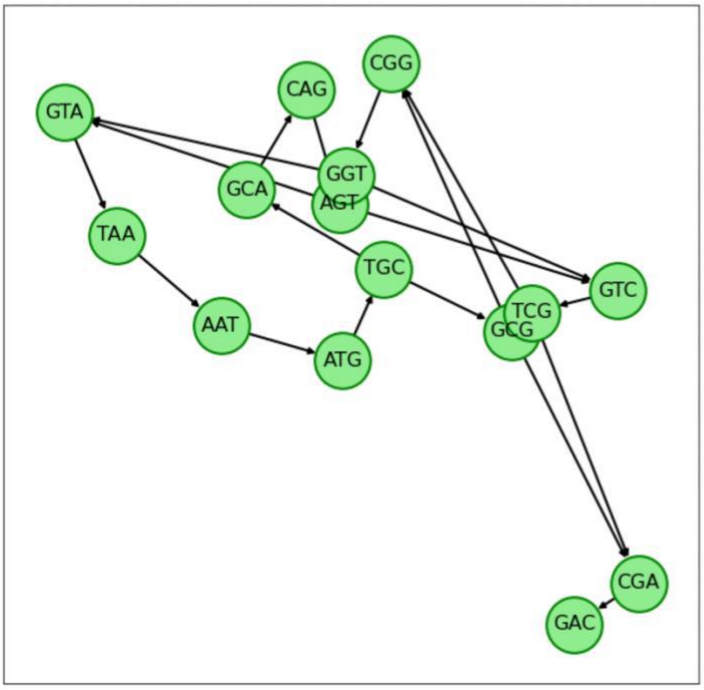
2. Case b

K=3

```
input1="GCGGTAATGCAGTCGAC"
a=kmers(input1,3)
```

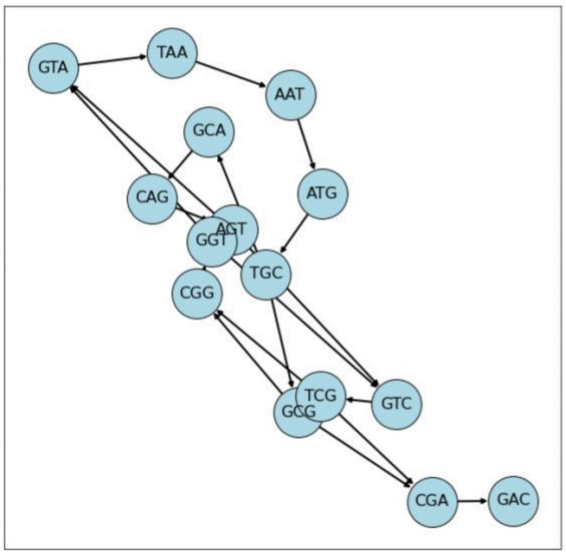
Nodes and Edges of the Constructed Graph :

```
nodes = ['AAT', 'ATG', 'AGT', 'GTA', 'GTC', 'TGC', 'CAG', 'CGA', 'GAC', 'CGG', 'GGT', 'GCA', 'GCG', 'TAA', 'TCG']

edges = [('AAT', 'ATG'), ('ATG', 'TGC'), ('AGT', 'GTA'), ('AGT', 'GTC'), ('GTA', 'TAA'), ('GTC', 'TCG'), ('TGC', 'G
CA'), ('TGC', 'GCG'), ('CAG', 'AGT'), ('CGA', 'GAC'), ('CGG', 'GGT'), ('GGT', 'GTA'), ('GGT', 'GTC'), ('GCA', 'CAG'
), ('GCG', 'CGA'), ('GCG', 'CGG'), ('TAA', 'AAT'), ('TCG', 'CGA'), ('TCG', 'CGG')]
```



Hamiltonian path after string reconstruction:

```
nodes = ['GCA', 'CAG', 'AGT', 'GTA', 'GTC', 'TAA', 'AAT', 'ATG', 'TGC', 'GCG', 'CGG', 'CGA', 'GGT', 'TCG', 'GAC']

edges = [('GCA', 'CAG'), ('CAG', 'AGT'), ('AGT', 'GTA'), ('AGT', 'GTC'), ('GTA', 'TAA'), ('GTC', 'TCG'), ('TAA', 'A
AT'), ('AAT', 'ATG'), ('ATG', 'TGC'), ('TGC', 'GCA'), ('TGC', 'GCG'), ('GCG', 'CGG'), ('GCG', 'CGA'), ('CGG', 'GGT'
), ('CGA', 'GAC'), ('GGT', 'GTA'), ('GGT', 'GTC'), ('TCG', 'CGG'), ('TCG', 'CGA')]
```




```
  Original String   = GCGGTAATGCAGTCGAC
Reconstructed String = GCAGTAATGCGGTCGAC
Reconstructed String = GCGGTAATGCAGTCGAC
```

```
Run Time:  0.3830885830002444
```

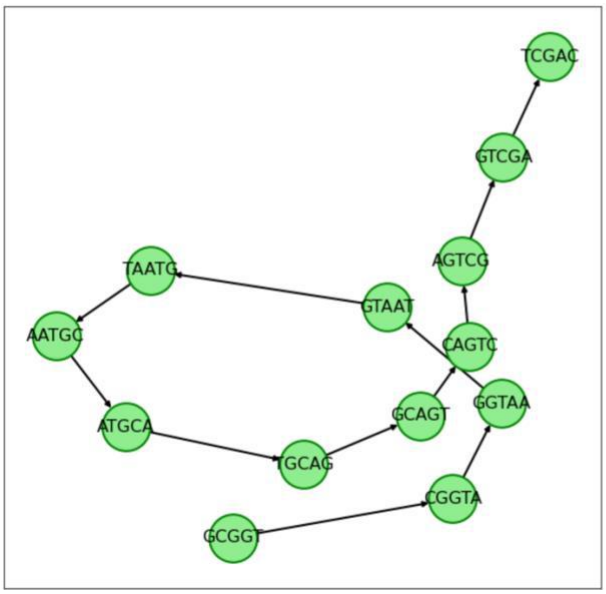
NOTE: In this case we are getting 2 possible Reconstructed Strings,

K= 5

```
input1="GCGGTAATGCAGTCGAC"
a=kmers(input1,5)
```

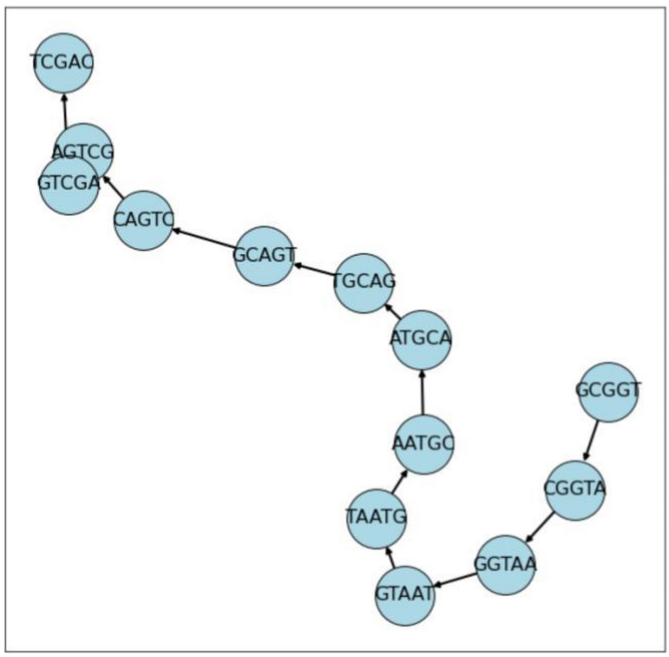
Nodes and Edges of the Constructed Graph :

```
nodes = ['AATGC', 'ATGCA', 'AGTCG', 'GTCGA', 'TGCAG', 'CAGTC', 'CGGTA', 'GGTAA', 'GCAGT', 'GCGGT', 'GTAAT', 'TAATG'
, 'TCGAC']

edges = [('AATGC', 'ATGCA'), ('ATGCA', 'TGCAG'), ('AGTCG', 'GTCGA'), ('GTCGA', 'TCGAC'), ('TGCAG', 'GCAGT'), ('CAGT
C', 'AGTCG'), ('CGGTA', 'GGTAA'), ('GGTAA', 'GTAAT'), ('GCAGT', 'CAGTC'), ('GCGGT', 'CGGTA'), ('GTAAT', 'TAATG'), (
'TAATG', 'AATGC')]
```



Hamiltonian path after string reconstruction:

```
nodes = ['GCGGT', 'CGGTA', 'GGTAA', 'GTAAT', 'TAATG', 'AATGC', 'ATGCA', 'TGCAG', 'GCAGT', 'CAGTC', 'AGTCG', 'GTCGA'
, 'TCGAC']

edges = [('GCGGT', 'CGGTA'), ('CGGTA', 'GGTAA'), ('GGTAA', 'GTAAT'), ('GTAAT', 'TAATG'), ('TAATG', 'AATGC'), ('AATG
C', 'ATGCA'), ('ATGCA', 'TGCAG'), ('TGCAG', 'GCAGT'), ('GCAGT', 'CAGTC'), ('CAGTC', 'AGTCG'), ('AGTCG', 'GTCGA'), (
'GTCGA', 'TCGAC')]
```




```
  Original String    = GCGGTAATGCAGTCGAC
Reconstructed String = GCGGTAATGCAGTCGAC  Run Time:  0.32912070900010804
```

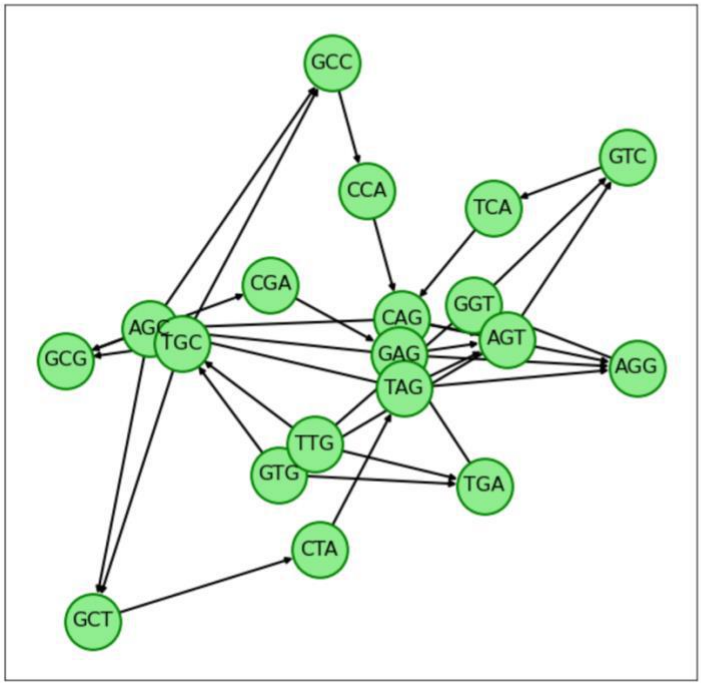
Note: when k=5, we are getting only 1 possible reconstructed string

3. Case 3

```
input1="TTGAGTGCCAGCGAGCTAGGTCAGT"
a=kmers(input1,3)
```

Nodes and Edges of the Constructed Graph :

```
nodes = ['AGC', 'GCC', 'GCG', 'GCT', 'AGG', 'GGT', 'AGT', 'GTC', 'GTG', 'CAG', 'CCA', 'CGA', 'GAG', 'CTA', 'TAG', '
TCA', 'TGA', 'TGC', 'TTG']

edges = [('AGC', 'GCC'), ('AGC', 'GCG'), ('AGC', 'GCT'), ('GCC', 'CCA'), ('GCG', 'CGA'), ('GCT', 'CTA'), ('AGG', 'G
GT'), ('GGT', 'GTC'), ('GGT', 'GTG'), ('AGT', 'GTC'), ('AGT', 'GTG'), ('GTC', 'TCA'), ('GTG', 'TGA'), ('GTG', 'TGC'
), ('CAG', 'AGC'), ('CAG', 'AGG'), ('CAG', 'AGT'), ('CCA', 'CAG'), ('CGA', 'GAG'), ('GAG', 'AGC'), ('GAG', 'AGG'),
('GAG', 'AGT'), ('CTA', 'TAG'), ('TAG', 'AGC'), ('TAG', 'AGG'), ('TAG', 'AGT'), ('TCA', 'CAG'), ('TGA', 'GAG'), ('T
GC', 'GCC'), ('TGC', 'GCG'), ('TGC', 'GCT'), ('TTG', 'TGA'), ('TTG', 'TGC')]
```



Hamiltonian path after string reconstruction:

```
All possible hamiltonian paths in the graph =
[['AGC', 'GCC', 'CCA', 'CAG', 'TCA', 'GTC', 'GGT', 'AGG', 'GAG', 'CGA', 'GCG', 'TGC', 'GCT', 'CTA', 'TAG', 'AGT', '
GTG', 'TGA', 'TTG'], ['GCC', 'AGC', 'GCG', 'CGA', 'GAG', 'AGG', 'GGT', 'GTG', 'TGA', 'TTG', 'TGC', 'GCT', 'CTA', 'T
AG', 'AGT', 'GTC', 'TCA', 'CAG', 'CCA'], ['GCG', 'AGC', 'GCC', 'CCA', 'CAG', 'TCA', 'GTC', 'GGT', 'AGG', 'TAG', 'CT
A', 'GCT', 'TGC', 'TTG', 'TGA', 'GTG', 'AGT', 'GAG', 'CGA'], ['GCT', 'AGC', 'GCC', 'CCA', 'CAG', 'TCA', 'GTC', 'GGT
', 'AGG', 'GAG', 'CGA', 'GCG', 'TGC', 'TTG', 'TGA', 'GTG', 'AGT', 'TAG', 'CTA'], ['AGG', 'GGT', 'GTC', 'TCA', 'CAG'
, 'CCA', 'GCC', 'AGC', 'GCG', 'CGA', 'GAG', 'AGT', 'GTG', 'TGA', 'TTG', 'TGC', 'GCT', 'CTA', 'TAG'], ['GGT', 'AGG',
'GAG', 'CGA', 'GCG', 'AGC', 'GCC', 'CCA', 'CAG', 'TCA', 'GTC', 'AGT', 'GTG', 'TGA', 'TTG', 'TGC', 'GCT', 'CTA', 'TA
G'], ['AGT', 'GTC', 'GGT', 'GTG', 'TGA', 'TTG', 'TGC', 'GCG', 'CGA', 'GAG', 'AGG', 'TAG', 'CTA', 'GCT', 'AGC', 'GCC
', 'CCA', 'CAG', 'TCA'], ['GTC', 'GGT', 'AGG', 'GAG', 'CGA', 'GCG', 'AGC', 'GCT', 'CTA', 'TAG', 'AGT', 'GTG', 'TGA'
, 'TTG', 'TGC', 'GCC', 'CCA', 'CAG', 'TCA'], ['GTG', 'GGT', 'AGG', 'CAG', 'CCA', 'GCC', 'AGC', 'GCG', 'CGA', 'GAG',
'TGA', 'TTG', 'TGC', 'GCT', 'CTA', 'TAG', 'AGT', 'GTC', 'TCA'], ['CAG', 'CCA', 'GCC', 'AGC', 'GCG', 'CGA', 'GAG', '
AGG', 'GGT', 'GTG', 'TGA', 'TTG', 'TGC', 'GCT', 'CTA', 'TAG', 'AGT', 'GTC', 'TCA'], ['CCA', 'GCC', 'AGC', 'GCG', 'C
GA', 'GAG', 'AGG', 'GGT', 'GTC', 'TCA', 'CAG', 'AGT', 'GTG', 'TGA', 'TTG', 'TGC', 'GCT', 'CTA', 'TAG'], ['CGA', 'GC
G', 'AGC', 'GCC', 'CCA', 'CAG', 'AGG', 'GAG', 'AGT', 'TAG', 'CTA', 'GCT', 'TGC', 'TTG', 'TGA', 'GTG', 'GGT', 'GTC',
'TCA'], ['GAG', 'CGA', 'GCG', 'AGC', 'GCC', 'CCA', 'CAG', 'AGG', 'GGT', 'GTG', 'TGA', 'TTG', 'TGC', 'GCT', 'CTA', '
TAG', 'AGT', 'GTC', 'TCA'], ['CTA', 'GCT', 'AGC', 'GCC', 'CCA', 'CAG', 'AGG', 'TAG', 'AGT', 'GAG', 'CGA', 'GCG', 'T
GC', 'TTG', 'TGA', 'GTG', 'GGT', 'GTC', 'TCA'], ['TAG', 'CTA', 'GCT', 'AGC', 'GCC', 'CCA', 'CAG', 'AGG', 'GGT', 'GT
G', 'TGA', 'TTG', 'TGC', 'GCG', 'CGA', 'GAG', 'AGT', 'GTC', 'TCA'], ['TCA', 'GTC', 'GGT', 'AGG', 'CAG', 'CCA', 'GCC
', 'AGC', 'GCG', 'CGA', 'GAG', 'AGT', 'GTG', 'TGA', 'TTG', 'TGC', 'GCT', 'CTA', 'TAG'], ['TGA', 'GTG', 'GGT', 'AGG'
, 'GAG', 'CGA', 'GCG', 'AGC', 'GCC', 'CCA', 'CAG', 'TCA', 'GTC', 'AGT', 'TAG', 'CTA', 'GCT', 'TGC', 'TTG'], ['TGC',
'GCC', 'CCA', 'CAG', 'TCA', 'GTC', 'GGT', 'AGG', 'GAG', 'CGA', 'GCG', 'AGC', 'GCT', 'CTA', 'TAG', 'AGT', 'GTG', 'TG
A', 'TTG'], ['TTG', 'TGA', 'GTG', 'GGT', 'AGG', 'CAG', 'CCA', 'GCC', 'AGC', 'GAG', 'CGA', 'GCG', 'TGC', 'GCT', 'CTA
', 'TAG', 'AGT', 'GTC', 'TCA']]

Succesful Hamiltonian Paths =
[]
```

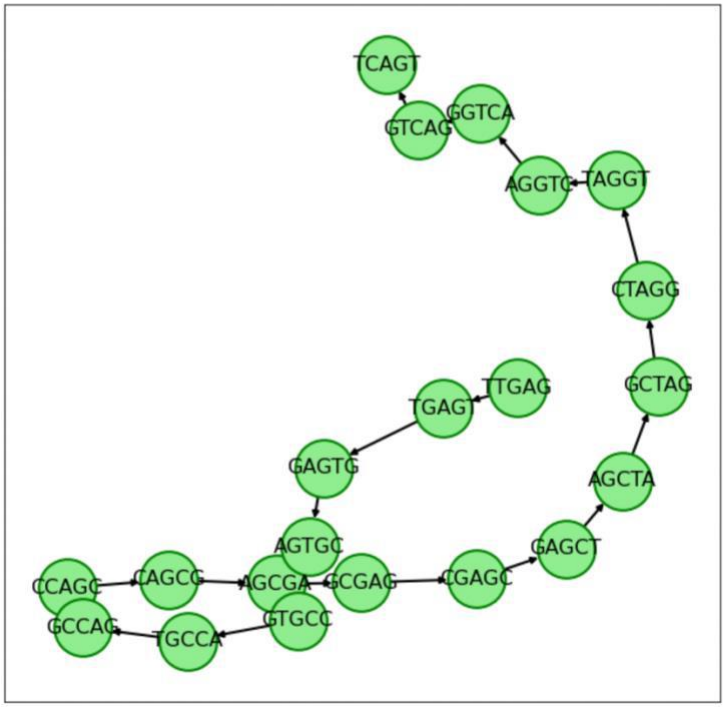
NOTE: when =3, there is no possible successful Hamiltonian path, hence the list returned by the function is empty and the string is not constructed.

K=5

```
input1="TTGAGTGCCAGCGAGCTAGGTCAGT"
a=kmers(input1,5)
```

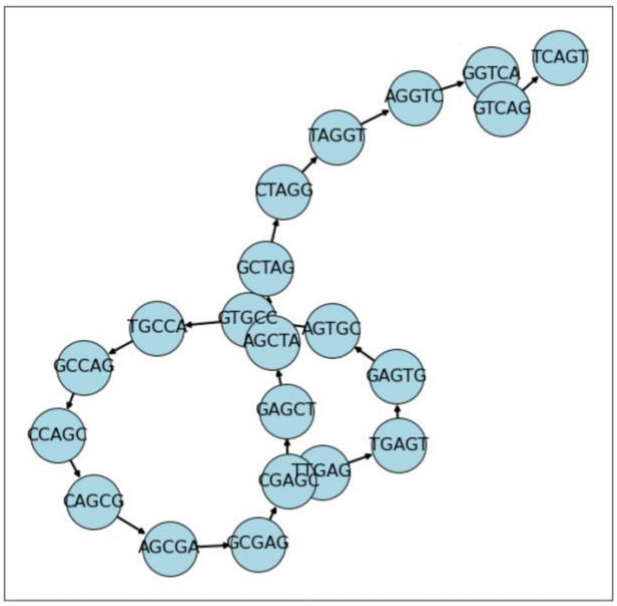
Nodes and Edges of the Constructed Graph :

```
nodes = ['AGCGA', 'GCGAG', 'AGCTA', 'GCTAG', 'AGGTC', 'GGTCA', 'AGTGC', 'GTGCC', 'CAGCG', 'CCAGC', 'CGAGC', 'GAGCT'
, 'CTAGG', 'TAGGT', 'GAGTG', 'GCCAG', 'GTCAG', 'TCAGT', 'TGCCA', 'TGAGT', 'TTGAG']

edges = [('AGCGA', 'GCGAG'), ('GCGAG', 'CGAGC'), ('AGCTA', 'GCTAG'), ('GCTAG', 'CTAGG'), ('AGGTC', 'GGTCA'), ('GGTC
A', 'GTCAG'), ('AGTGC', 'GTGCC'), ('GTGCC', 'TGCCA'), ('CAGCG', 'AGCGA'), ('CCAGC', 'CAGCG'), ('CGAGC', 'GAGCT'), (
'GAGCT', 'AGCTA'), ('CTAGG', 'TAGGT'), ('TAGGT', 'AGGTC'), ('GAGTG', 'AGTGC'), ('GCCAG', 'CCAGC'), ('GTCAG', 'TCAGT
'), ('TGCCA', 'GCCAG'), ('TGAGT', 'GAGTG'), ('TTGAG', 'TGAGT')]
```



Hamiltonian path after string reconstruction:

```
nodes = ['TTGAG', 'TGAGT', 'GAGTG', 'AGTGC', 'GTGCC', 'TGCCA', 'GCCAG', 'CCAGC', 'CAGCG', 'AGCGA', 'GCGAG', 'CGAGC'
, 'GAGCT', 'AGCTA', 'GCTAG', 'CTAGG', 'TAGGT', 'AGGTC', 'GGTCA', 'GTCAG', 'TCAGT']

edges = [('TTGAG', 'TGAGT'), ('TGAGT', 'GAGTG'), ('GAGTG', 'AGTGC'), ('AGTGC', 'GTGCC'), ('GTGCC', 'TGCCA'), ('TGCC
A', 'GCCAG'), ('GCCAG', 'CCAGC'), ('CCAGC', 'CAGCG'), ('CAGCG', 'AGCGA'), ('AGCGA', 'GCGAG'), ('GCGAG', 'CGAGC'), (
'CGAGC', 'GAGCT'), ('GAGCT', 'AGCTA'), ('AGCTA', 'GCTAG'), ('GCTAG', 'CTAGG'), ('CTAGG', 'TAGGT'), ('TAGGT', 'AGGTC
'), ('AGGTC', 'GGTCA'), ('GGTCA', 'GTCAG'), ('GTCAG', 'TCAGT')]
```




```
  Original String   = TTGAGTGCCAGCGAGCTAGGTCAGT
Reconstructed String = TTGAGTGCCAGCGAGCTAGGTCAGT
```

```
Run Time:  0.4512189999995826
```

NOTE: we get the required string when k=5 but not when k=3

2. Each member of the team has to select 50bp long unique segment from the sequence that has been selected from NCBI website. Mention the starting and stopping position of the unique segment selected in the report. (**eg.** Team member 1 can take first 50, Team member 2 can take next 50 and so on.)

Write a program in the programming language you prefer (Java or Python) to carry out string reconstruction based on Hamiltonian path method. (Attempt for k=3 & k=5). Also run the program for the whole sequence you have selected and include the output as well as the total run-time in the report. **(5 marks)**

Amino Acid Sequence selected by our team:

(The highlighted sequence is chosen by me **)**

ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACT
AAGCTATACTAACCCCAGGGTTGGTCAATTTCGTGCCAGCCACCGCGGTCACAC
GATTAACCCAAGTCAATAGAAGCCGGCGTAAAGAGTGTTTTAGATCACCCCCTC
CCCAATAAAGCTAAAACTCACCTGAGTTGTAAAAAACTCCAGTTGACACAAAAT
AGACTACGAAAGTGGCTTTAACATATCTGAACACACAATAGCTAAGACCCAAAC
TGGGATTAGATACCCCACTATGCTTAGCCCTAAACCTCAACAGTTAAATCAACA
AAACTGCTCGCCAGAACACTACGAGCCACAGCTTAAAACTCAAAGGACCTGGCG
GTGCTTCATATCCCTCTAGAGG

```
input1="ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCC"
print(len(input1))
```

```
70
```

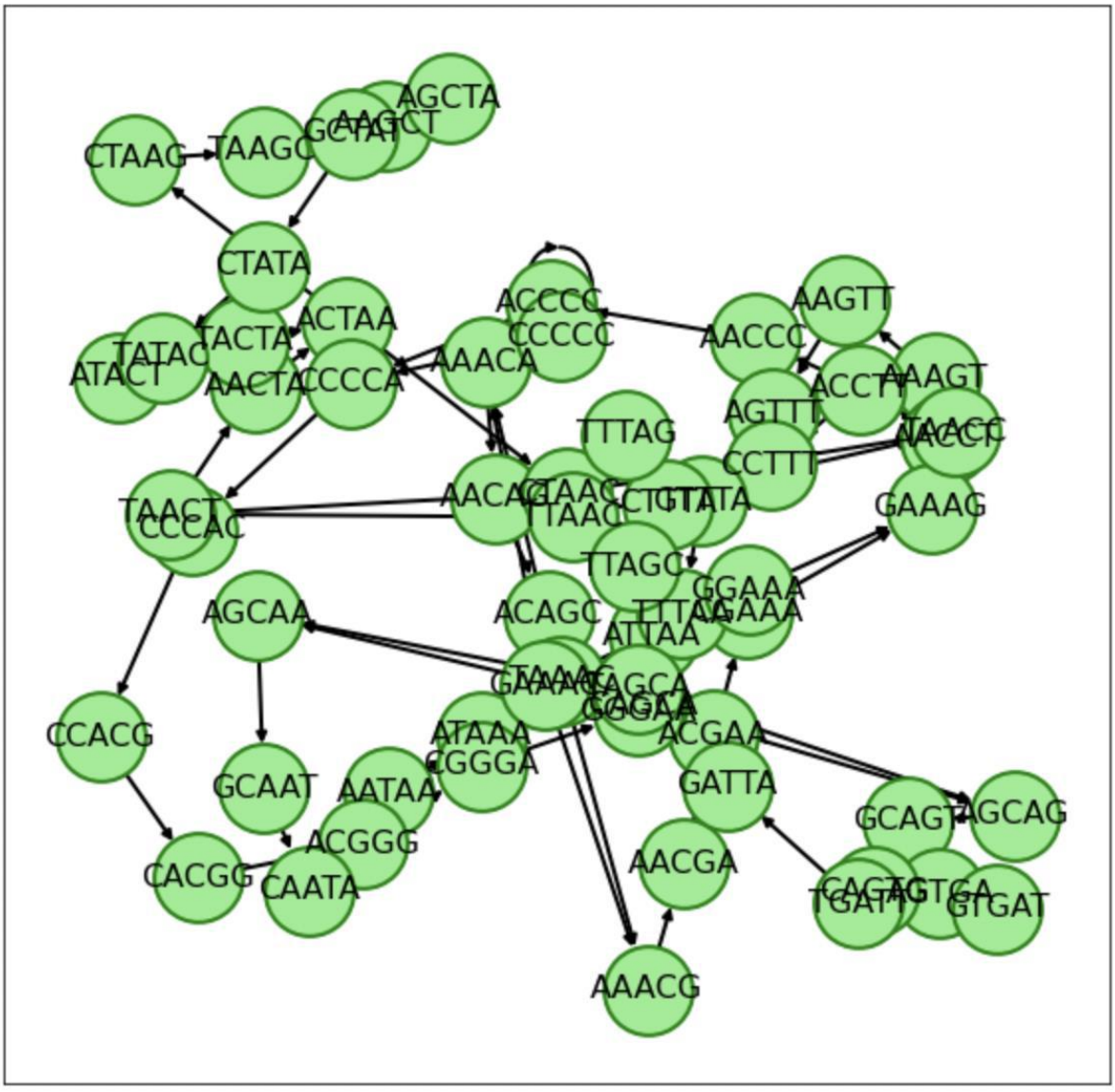
The Code from Q1 is used for This question too

K=5

```
input1="ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCC"
a=kmers(input1,5)
```

Nodes and Edges of the Constructed Graph :

```
nodes = ['AAACA', 'AACAG', 'AAACG', 'AACGA', 'AAAGT', 'AAGTT', 'ACAGC', 'AACCC', 'ACCCC', 'AACCT', 'ACCTT', 'ACGAA'
, 'AACTA', 'ACTAA', 'AAGCT', 'AGCTA', 'AGTTT', 'AATAA', 'ATAAA', 'CAGCA', 'CCCCA', 'CCCCC', 'CCTTT', 'CGAAA', 'ACGG
G', 'CGGGA', 'CTAAC', 'CTAAG', 'AGCAA', 'GCAAT', 'AGCAG', 'GCAGT', 'GCTAT', 'AGTGA', 'GTGAT', 'GTTTA', 'TAAAC', 'AT
ACT', 'TACTA', 'ATTAA', 'TTAAC', 'CAATA', 'CACGG', 'CAGTG', 'CCACG', 'CCCAC', 'CTTTA', 'GAAAC', 'GAAAG', 'GGGAA', '
TAACC', 'TAACT', 'TAAGC', 'CTATA', 'TATAC', 'TTTAA', 'TTTAG', 'GATTA', 'GGAAA', 'TGATT', 'TAGCA', 'TTAGC']

edges = [('AAACA', 'AACAG'), ('AACAG', 'ACAGC'), ('AAACG', 'AACGA'), ('AACGA', 'ACGAA'), ('AAAGT', 'AAGTT'), ('AAGT
T', 'AGTTT'), ('ACAGC', 'CAGCA'), ('AACCC', 'ACCCC'), ('ACCCC', 'CCCCA'), ('ACCCC', 'CCCCC'), ('AACCT', 'ACCTT'), (
'ACCTT', 'CCTTT'), ('ACGAA', 'CGAAA'), ('AACTA', 'ACTAA'), ('ACTAA', 'CTAAC'), ('ACTAA', 'CTAAG'), ('AAGCT', 'AGCTA
'), ('AGCTA', 'GCTAT'), ('AGTTT', 'GTTTA'), ('AATAA', 'ATAAA'), ('ATAAA', 'TAAAC'), ('CAGCA', 'AGCAA'), ('CAGCA', '
AGCAG'), ('CCCCA', 'CCCAC'), ('CCCCC', 'CCCCA'), ('CCCCC', 'CCCCC'), ('CCTTT', 'CTTTA'), ('CGAAA', 'GAAAC'), ('CGAA
A', 'GAAAG'), ('ACGGG', 'CGGGA'), ('CGGGA', 'GGGAA'), ('CTAAC', 'TAACC'), ('CTAAC', 'TAACT'), ('CTAAG', 'TAAGC'), (
'AGCAA', 'GCAAT'), ('GCAAT', 'CAATA'), ('AGCAG', 'GCAGT'), ('GCAGT', 'CAGTG'), ('GCTAT', 'CTATA'), ('AGTGA', 'GTGAT
'), ('GTGAT', 'TGATT'), ('GTTTA', 'TTTAA'), ('GTTTA', 'TTTAG'), ('TAAAC', 'AAACA'), ('TAAAC', 'AAACG'), ('ATACT', '
TACTA'), ('TACTA', 'ACTAA'), ('ATTAA', 'TTAAC'), ('TTAAC', 'TAACC'), ('TTAAC', 'TAACT'), ('CAATA', 'AATAA'), ('CACG
G', 'ACGGG'), ('CAGTG', 'AGTGA'), ('CCACG', 'CACGG'), ('CCCAC', 'CCACG'), ('CTTTA', 'TTTAA'), ('CTTTA', 'TTTAG'), (
'GAAAC', 'AAACA'), ('GAAAC', 'AAACG'), ('GAAAG', 'AAAGT'), ('GGGAA', 'GGAAA'), ('TAACC', 'AACCC'), ('TAACC', 'AACCT
'), ('TAACT', 'AACTA'), ('TAAGC', 'AAGCT'), ('CTATA', 'TATAC'), ('TATAC', 'ATACT'), ('TTTAA', 'TTAAC'), ('TTTAG', '
TTAGC'), ('GATTA', 'ATTAA'), ('GGAAA', 'GAAAC'), ('GGAAA', 'GAAAG'), ('TGATT', 'GATTA'), ('TAGCA', 'AGCAA'), ('TAGC
A', 'AGCAG'), ('TTAGC', 'TAGCA')]
```



Hamiltonian path after string reconstruction:

```
All possible hamiltonian paths in the graph =
[['AACCC', 'ACCCC', 'CCCCC', 'CCCCA', 'CCCAC', 'CCACG', 'CACGG', 'ACGGG', 'CGGGA', 'GGGAA', 'GGAAA', 'GAAAC', 'AAAC
A', 'AACAG', 'ACAGC', 'CAGCA', 'AGCAA', 'GCAAT', 'CAATA', 'AATAA', 'ATAAA', 'TAAAC', 'AAACG', 'AACGA', 'ACGAA', 'CG
AAA', 'GAAAG', 'AAAGT', 'AAGTT', 'AGTTT', 'GTTTA', 'TTTAA', 'TTAAC', 'ATTAA', 'GATTA', 'TGATT', 'GTGAT', 'AGTGA', '
CAGTG', 'GCAGT', 'AGCAG', 'TAGCA', 'TTAGC', 'TTTAG', 'CTTTA', 'CCTTT', 'ACCTT', 'AACCT', 'TAACC', 'CTAAC', 'TAACT',
'AACTA', 'ACTAA', 'CTAAG', 'TAAGC', 'AAGCT', 'AGCTA', 'GCTAT', 'CTATA', 'TATAC', 'ATACT', 'TACTA'], ['AACCT', 'ACCT
T', 'CCTTT', 'CTTTA', 'TTTAA', 'TTAAC', 'ATTAA', 'GATTA', 'TGATT', 'GTGAT', 'AGTGA', 'CAGTG', 'GCAGT', 'AGCAG', 'CA
GCA', 'ACAGC', 'AACAG', 'AAACA', 'TAAAC', 'ATAAA', 'AATAA', 'CAATA', 'GCAAT', 'AGCAA', 'TAGCA', 'TTAGC', 'TTTAG', '
GTTTA', 'AGTTT', 'AAGTT', 'AAAGT', 'GAAAG', 'CGAAA', 'ACGAA', 'AACGA', 'AAACG', 'GAAAC', 'GGAAA', 'GGGAA', 'CGGGA',
'ACGGG', 'CACGG', 'CCACG', 'CCCAC', 'CCCCA', 'CCCCC', 'ACCCC', 'AACCC', 'TAACC', 'CTAAC', 'TAACT', 'AACTA', 'ACTAA'
, 'CTAAG', 'TAAGC', 'AAGCT', 'AGCTA', 'GCTAT', 'CTATA', 'TATAC', 'ATACT', 'TACTA'], ['CTAAG', 'TAAGC', 'AAGCT', 'AG
CTA', 'GCTAT', 'CTATA', 'TATAC', 'ATACT', 'TACTA', 'ACTAA', 'AACTA', 'TAACT', 'CTAAC', 'TAACC', 'AACCC', 'ACCCC', '
CCCCC', 'CCCCA', 'CCCAC', 'CCACG', 'CACGG', 'ACGGG', 'CGGGA', 'GGGAA', 'GGAAA', 'GAAAC', 'AAACA', 'AACAG', 'ACAGC',
'CAGCA', 'AGCAA', 'GCAAT', 'CAATA', 'AATAA', 'ATAAA', 'TAAAC', 'AAACG', 'AACGA', 'ACGAA', 'CGAAA', 'GAAAG', 'AAAGT'
, 'AAGTT', 'AGTTT', 'GTTTA', 'TTTAA', 'TTAAC', 'ATTAA', 'GATTA', 'TGATT', 'GTGAT', 'AGTGA', 'CAGTG', 'GCAGT', 'AGCA
G', 'TAGCA', 'TTAGC', 'TTTAG', 'CTTTA', 'CCTTT', 'ACCTT', 'AACCT'], ['TACTA', 'ATACT', 'TATAC', 'CTATA', 'GCTAT', '
AGCTA', 'AAGCT', 'TAAGC', 'CTAAG', 'ACTAA', 'AACTA', 'TAACT', 'CTAAC', 'TAACC', 'AACCC', 'ACCCC', 'CCCCC', 'CCCCA',
'CCCAC', 'CCACG', 'CACGG', 'ACGGG', 'CGGGA', 'GGGAA', 'GGAAA', 'GAAAC', 'AAACA', 'AACAG', 'ACAGC', 'CAGCA', 'AGCAA'
, 'GCAAT', 'CAATA', 'AATAA', 'ATAAA', 'TAAAC', 'AAACG', 'AACGA', 'ACGAA', 'CGAAA', 'GAAAG', 'AAAGT', 'AAGTT', 'AGTT
T', 'GTTTA', 'TTTAA', 'TTAAC', 'ATTAA', 'GATTA', 'TGATT', 'GTGAT', 'AGTGA', 'CAGTG', 'GCAGT', 'AGCAG', 'TAGCA', 'TT
AGC', 'TTTAG', 'CTTTA', 'CCTTT', 'ACCTT', 'AACCT']]

Succesful Hamiltonian Paths =
[]
```

Note: Successful Hamiltonian path refers to the paths in which (k-1) suffix is equal to the prefix, when k=5 there are plenty of Hamiltonian paths, but there are no successful ones that is required for string reconstruction, hence there is no string reconstructed

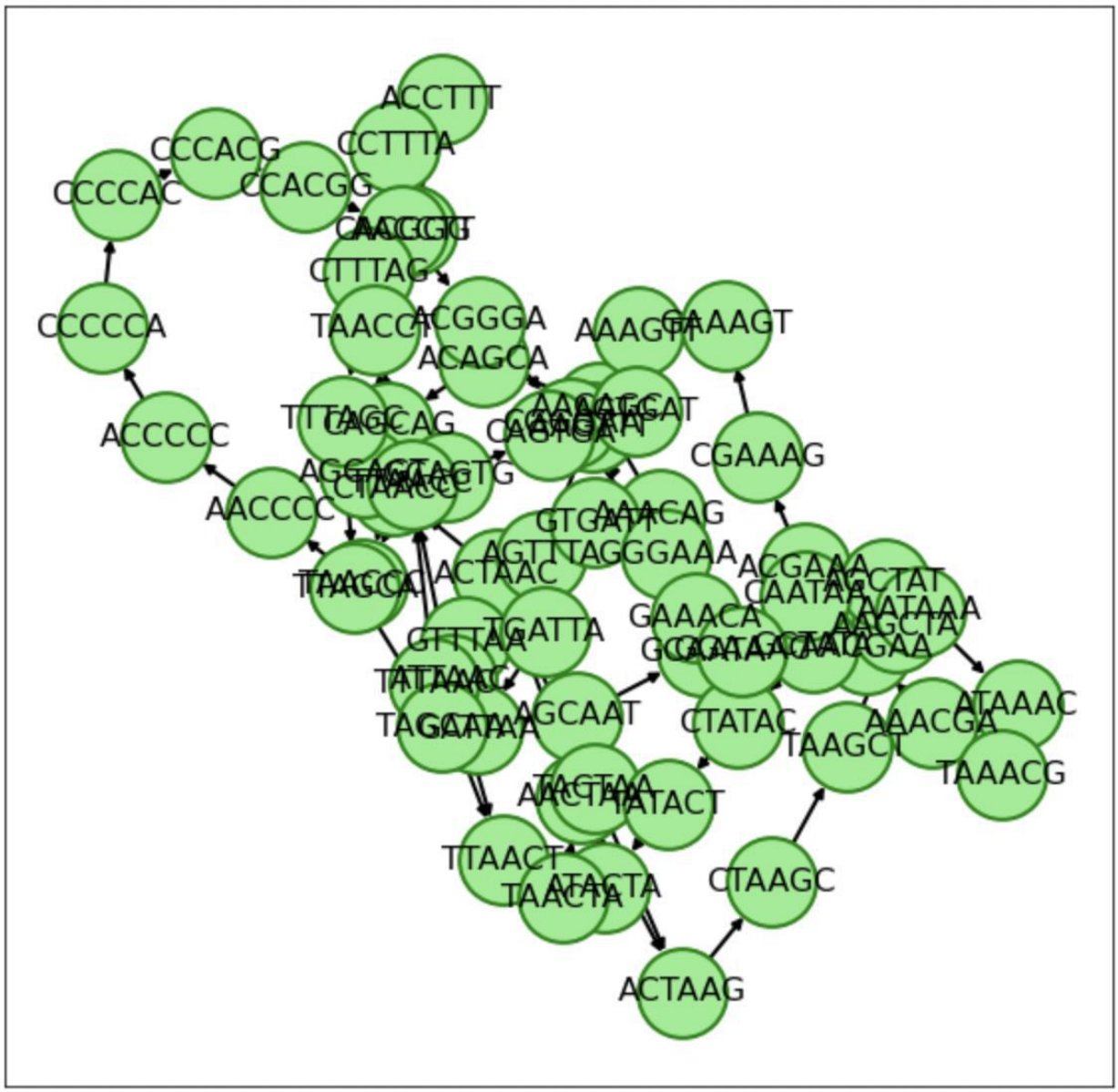
Some more test cases with higher k value is shown below, the results are interesting.

K=6

```
input1="ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCC"
a=kmers(input1,6)
```

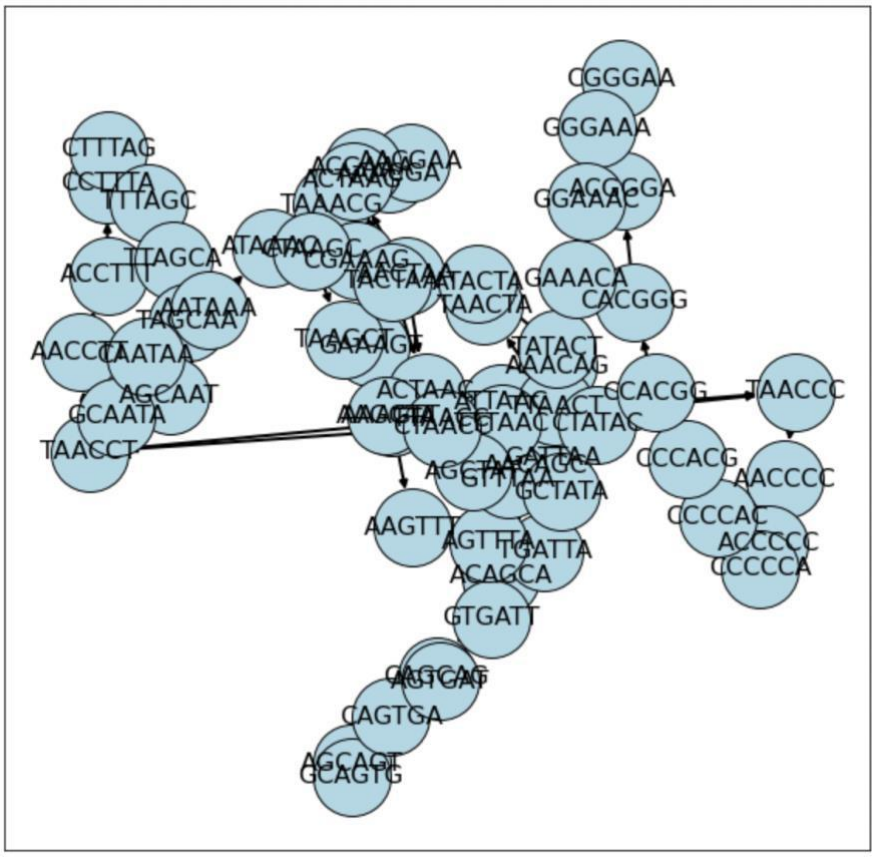
Nodes and Edges of the Constructed Graph :

```
nodes = ['AAACAG', 'AACAGC', 'AAACGA', 'AACGAA', 'AAAGTT', 'AAGTTT', 'ACAGCA', 'AACCCC', 'ACCCCC', 'AACCTT', 'ACCTT
T', 'ACGAAA', 'AACTAA', 'ACTAAC', 'ACTAAG', 'AAGCTA', 'AGCTAT', 'AGTTTA', 'AATAAA', 'ATAAAC', 'CAGCAG', 'CCCCCA', '
CCTTTA', 'CGAAAG', 'ACGGGA', 'CGGGAA', 'CTAACC', 'CTAAGC', 'AGCAAT', 'GCAATA', 'AGCAGT', 'GCAGTG', 'GCTATA', 'AGTGA
T', 'GTGATT', 'GTTTAA', 'TAAACG', 'ATACTA', 'TACTAA', 'ATTAAC', 'TTAACC', 'TTAACT', 'CAATAA', 'CACGGG', 'CAGTGA', '
CCACGG', 'CCCACG', 'CCCCAC', 'CTTTAG', 'GAAAGT', 'GGGAAA', 'TAACCC', 'TAACCT', 'TAAGCT', 'CTATAC', 'TATACT', 'TTTAG
C', 'GAAACA', 'GATTAA', 'GGAAAC', 'TGATTA', 'TTTAAC', 'TAACTA', 'TAGCAA', 'TTAGCA']

edges = [('AAACAG', 'AACAGC'), ('AACAGC', 'ACAGCA'), ('AAACGA', 'AACGAA'), ('AACGAA', 'ACGAAA'), ('AAAGTT', 'AAGTTT
'), ('AAGTTT', 'AGTTTA'), ('ACAGCA', 'CAGCAG'), ('AACCCC', 'ACCCCC'), ('ACCCCC', 'CCCCCA'), ('AACCTT', 'ACCTTT'), (
'ACCTTT', 'CCTTTA'), ('ACGAAA', 'CGAAAG'), ('AACTAA', 'ACTAAC'), ('AACTAA', 'ACTAAG'), ('ACTAAC', 'CTAACC'), ('ACTA
AG', 'CTAAGC'), ('AAGCTA', 'AGCTAT'), ('AGCTAT', 'GCTATA'), ('AGTTTA', 'GTTTAA'), ('AATAAA', 'ATAAAC'), ('ATAAAC',
'TAAACG'), ('CAGCAG', 'AGCAGT'), ('CCCCCA', 'CCCCAC'), ('CCTTTA', 'CTTTAG'), ('CGAAAG', 'GAAAGT'), ('ACGGGA', 'CGGG
AA'), ('CGGGAA', 'GGGAAA'), ('CTAACC', 'TAACCC'), ('CTAACC', 'TAACCT'), ('CTAAGC', 'TAAGCT'), ('AGCAAT', 'GCAATA'),
('GCAATA', 'CAATAA'), ('AGCAGT', 'GCAGTG'), ('GCAGTG', 'CAGTGA'), ('GCTATA', 'CTATAC'), ('AGTGAT', 'GTGATT'), ('GTG
ATT', 'TGATTA'), ('GTTTAA', 'TTTAAC'), ('TAAACG', 'AAACGA'), ('ATACTA', 'TACTAA'), ('TACTAA', 'ACTAAC'), ('TACTAA',
'ACTAAG'), ('ATTAAC', 'TTAACC'), ('ATTAAC', 'TTAACT'), ('TTAACC', 'TAACCC'), ('TTAACC', 'TAACCT'), ('TTAACT', 'TAAC
TA'), ('CAATAA', 'AATAAA'), ('CACGGG', 'ACGGGA'), ('CAGTGA', 'AGTGAT'), ('CCACGG', 'CACGGG'), ('CCCACG', 'CCACGG'),
('CCCCAC', 'CCCACG'), ('CTTTAG', 'TTTAGC'), ('GAAAGT', 'AAAGTT'), ('GGGAAA', 'GGAAAC'), ('TAACCC', 'AACCCC'), ('TAA
CCT', 'AACCTT'), ('TAAGCT', 'AAGCTA'), ('CTATAC', 'TATACT'), ('TATACT', 'ATACTA'), ('TTTAGC', 'TTAGCA'), ('GAAACA',
'AAACAG'), ('GATTAA', 'ATTAAC'), ('GGAAAC', 'GAAACA'), ('TGATTA', 'GATTAA'), ('TTTAAC', 'TTAACC'), ('TTTAAC', 'TTAA
CT'), ('TAACTA', 'AACTAA'), ('TAGCAA', 'AGCAAT'), ('TTAGCA', 'TAGCAA')]
```



Hamiltonian path after string reconstruction:

```
nodes = ['AAACAG', 'AACAGC', 'ACAGCA', 'CAGCAG', 'AGCAGT', 'GCAGTG', 'CAGTGA', 'AGTGAT', 'GTGATT', 'TGATTA', 'GATTA
A', 'ATTAAC', 'TTAACC', 'TTAACT', 'TAACCT', 'TAACCC', 'AACCTT', 'ACCTTT', 'CCTTTA', 'CTTTAG', 'TTTAGC', 'TTAGCA', '
TAGCAA', 'AGCAAT', 'GCAATA', 'CAATAA', 'AATAAA', 'ATAAAC', 'TAAACG', 'AAACGA', 'AACGAA', 'ACGAAA', 'CGAAAG', 'GAAAG
T', 'AAAGTT', 'AAGTTT', 'AGTTTA', 'GTTTAA', 'TTTAAC', 'TAACTA', 'AACTAA', 'ACTAAG', 'ACTAAC', 'CTAAGC', 'TAAGCT', '
AAGCTA', 'AGCTAT', 'GCTATA', 'CTATAC', 'TATACT', 'ATACTA', 'TACTAA', 'CTAACC', 'AACCCC', 'ACCCCC', 'CCCCCA', 'CCCCA
C', 'CCCACG', 'CCACGG', 'CACGGG', 'ACGGGA', 'CGGGAA', 'GGGAAA', 'GGAAAC', 'GAAACA']

edges = [('AAACAG', 'AACAGC'), ('AACAGC', 'ACAGCA'), ('ACAGCA', 'CAGCAG'), ('CAGCAG', 'AGCAGT'), ('AGCAGT', 'GCAGTG
'), ('GCAGTG', 'CAGTGA'), ('CAGTGA', 'AGTGAT'), ('AGTGAT', 'GTGATT'), ('GTGATT', 'TGATTA'), ('TGATTA', 'GATTAA'), (
'GATTAA', 'ATTAAC'), ('ATTAAC', 'TTAACC'), ('ATTAAC', 'TTAACT'), ('TTAACC', 'TAACCT'), ('TTAACC', 'TAACCC'), ('TTAA
CT', 'TAACTA'), ('TAACCT', 'AACCTT'), ('TAACCC', 'AACCCC'), ('AACCTT', 'ACCTTT'), ('ACCTTT', 'CCTTTA'), ('CCTTTA',
'CTTTAG'), ('CTTTAG', 'TTTAGC'), ('TTTAGC', 'TTAGCA'), ('TTAGCA', 'TAGCAA'), ('TAGCAA', 'AGCAAT'), ('AGCAAT', 'GCAA
TA'), ('GCAATA', 'CAATAA'), ('CAATAA', 'AATAAA'), ('AATAAA', 'ATAAAC'), ('ATAAAC', 'TAAACG'), ('TAAACG', 'AAACGA'),
('AAACGA', 'AACGAA'), ('AACGAA', 'ACGAAA'), ('ACGAAA', 'CGAAAG'), ('CGAAAG', 'GAAAGT'), ('GAAAGT', 'AAAGTT'), ('AAA
GTT', 'AAGTTT'), ('AAGTTT', 'AGTTTA'), ('AGTTTA', 'GTTTAA'), ('GTTTAA', 'TTTAAC'), ('TTTAAC', 'TTAACC'), ('TTTAAC',
'TTAACT'), ('TAACTA', 'AACTAA'), ('AACTAA', 'ACTAAG'), ('AACTAA', 'ACTAAC'), ('ACTAAG', 'CTAAGC'), ('ACTAAC', 'CTAA
CC'), ('CTAAGC', 'TAAGCT'), ('TAAGCT', 'AAGCTA'), ('AAGCTA', 'AGCTAT'), ('AGCTAT', 'GCTATA'), ('GCTATA', 'CTATAC'),
('CTATAC', 'TATACT'), ('TATACT', 'ATACTA'), ('ATACTA', 'TACTAA'), ('TACTAA', 'ACTAAG'), ('TACTAA', 'ACTAAC'), ('CTA
ACC', 'TAACCT'), ('CTAACC', 'TAACCC'), ('AACCCC', 'ACCCCC'), ('ACCCCC', 'CCCCCA'), ('CCCCCA', 'CCCCAC'), ('CCCCAC',
'CCCACG'), ('CCCACG', 'CCACGG'), ('CCACGG', 'CACGGG'), ('CACGGG', 'ACGGGA'), ('ACGGGA', 'CGGGAA'), ('CGGGAA', 'GGGA
AA'), ('GGGAAA', 'GGAAAC'), ('GGAAAC', 'GAAACA'), ('GAAACA', 'AAACAG')]
```



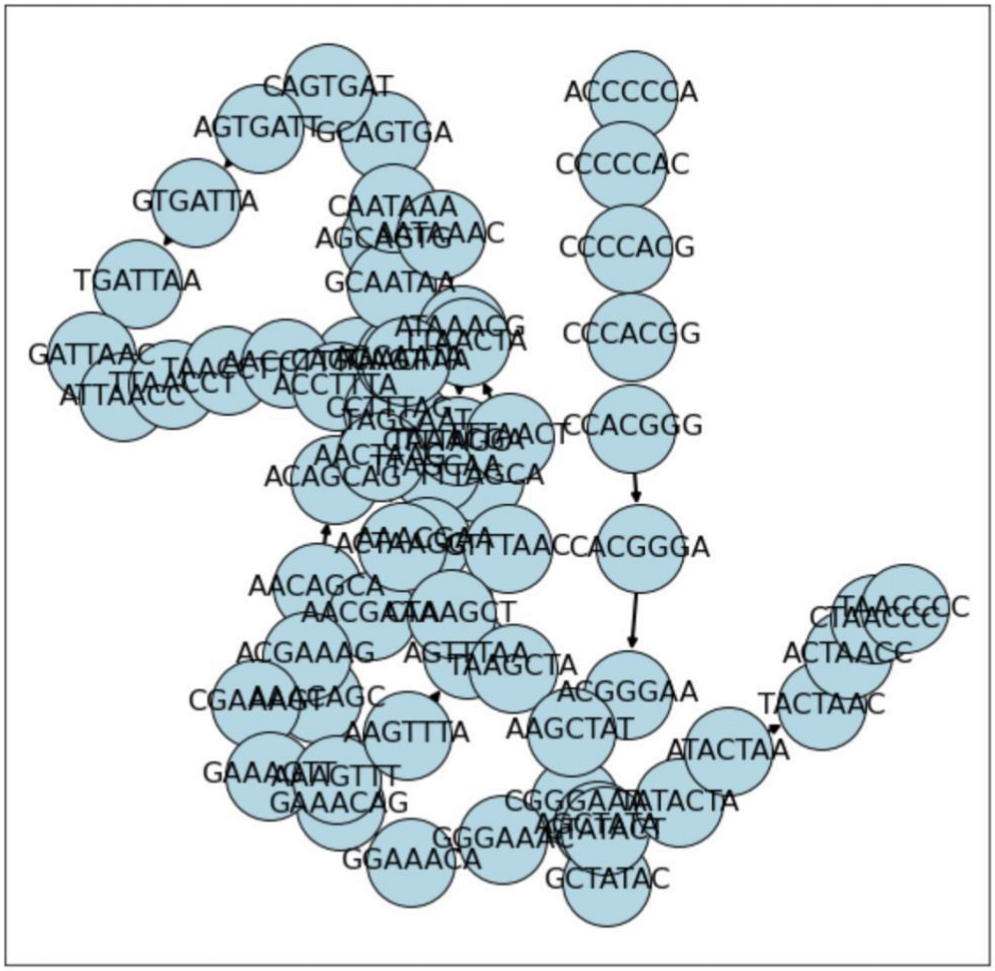

```
  Original String   = ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCC
Reconstructed String = AAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCCACGGGAAACA
Reconstructed String = AAACGAAAGTTTAACCCCCACGGGAAACAGCAGTGATTAACTAAGCTATACTAACCTTTAGCAATAAACG
Reconstructed String = AAAGTTTAACCCCCACGGGAAACAGCAGTGATTAACTAAGCTATACTAACCTTTAGCAATAAACGAAAGT
Reconstructed String = AACCTTTAGCAATAAACGAAAGTTTAACCCCCACGGGAAACAGCAGTGATTAACTAAGCTATACTAACCT
Reconstructed String = AACTAAGCTATACTAACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTA
Reconstructed String = ACTAACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAA
Reconstructed String = AAGCTATACTAACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCT
Reconstructed String = AATAAACGAAAGTTTAACCCCCACGGGAAACAGCAGTGATTAACTAAGCTATACTAACCTTTAGCAATAA
Reconstructed String = ACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCCACGGG
Reconstructed String = AGCAATAAACGAAAGTTTAACCCCCACGGGAAACAGCAGTGATTAACTAAGCTATACTAACCTTTAGCAA
Reconstructed String = AGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCCACGGGAAACAGCAGTGA
Reconstructed String = ATACTAACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACT
Reconstructed String = ATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCCACGGGAAACAGCAGTGATTAA
Reconstructed String = CACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCCACGG
Reconstructed String = CCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCCACG
Reconstructed String = CCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCCAC
Reconstructed String = GAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCCACGGGAAAC
Reconstructed String = GATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCCACGGGAAACAGCAGTGATTA
Reconstructed String = TAGCAATAAACGAAAGTTTAACCCCCACGGGAAACAGCAGTGATTAACTAAGCTATACTAACCTTTAGCA

Run Time:  0.9450985829998899
```

NOTE: We can notice that there are around 20 successfully reconstructed strings using Hamiltonian path when k=6

When k=7

```
input1="ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCC"
a=kmers(input1,7)
```

Hamiltonian path after string reconstruction:




```
  Original String   = ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCC
Reconstructed String = ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCC

Run Time:  0.8643770000003315
```

NOTE:

When k=7, we get only 1 reconstructed string, however with k=6 we got 20 reconstructed string, A point to be noted here. Further studies can be performed on the same (optimizing the value of k)

When k > 7, we get 1 reconstructed string

K=15 `Run Time:  0.8028017080000609`

K=20 `Run Time:  0.754536625000128`

K=30 `Run Time:  0.7041932920001273`

String reconstruction for the whole String:
ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACT
AAGCTATACTAACCCCAGGGTTGGTCAATTTCGTGCCAGCCACCGCGGTCACAC
GATTAACCCAAGTCAATAGAAGCCGGCGTAAAGAGTGTTTTAGATCACCCCCTC
CCCAATAAAGCTAAAACTCACCTGAGTTGTAAAAAACTCCAGTTGACACAAAAT
AGACTACGAAAGTGGCTTTAACATATCTGAACACACAATAGCTAAGACCCAAAC
TGGGATTAGATACCCCACTATGCTTAGCCCTAAACCTCAACAGTTAAATCAACA
AAACTGCTCGCCAGAACACTACGAGCCACAGCTTAAAACTCAAAGGACCTGGCG
GTGCTTCATATCCCTCTAGAGG

```
input1="ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCAGGGTTGGTCAATTTCGTGCCAGCCACCGCGGTCACAC
print(len(input1))
```

```
400
```

K=10

```
input1="ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCAGGGTTGGTCAATTTCGTGCCAGCCACCGCGGTCACAC
a=kmers(input1,10)
```

```
nodes = ['ACCCCCACGG', 'CCCCCACGGG', 'CCCCACGGGA', 'CCCACGGGAA', 'CCACGGGAAA', 'CACGGGAAAC', 'ACGGGAAACA', 'CGGGA
click to unscroll output; double click to hide CAGCA', 'GAAACAGCAG', 'AAACAGCAGT', 'AACAGCAGTG', 'ACAGCAGTGA', 'CAGCAGTGAT', 'AGCAGT
GATT', 'GCAGTGATTA', 'CAGTGATTAA', 'AGTGATTAAC', 'GTGATTAACC', 'TGATTAACCT', 'GATTAACCTT', 'ATTAACCTTT', 'TTAACCT
TTA', 'TAACCTTTAG', 'AACCTTTAGC', 'ACCTTTAGCA', 'CCTTTAGCAA', 'CTTTAGCAAT', 'TTTAGCAATA', 'TTAGCAATAA', 'TAGCAATA
AA', 'AGCAATAAAC', 'GCAATAAACG', 'CAATAAACGA', 'AATAAACGAA', 'ATAAACGAAA', 'TAAACGAAAG', 'AAACGAAAGT', 'AACGAAAGT
T', 'ACGAAAGTTT', 'CGAAAGTTTA', 'GAAAGTTTAA', 'AAAGTTTAAC', 'AAGTTTAACT', 'AGTTTAACTA', 'GTTTAACTAA', 'TTTAACTAAG
', 'TTAACTAAGC', 'TAACTAAGCT', 'AACTAAGCTA', 'ACTAAGCTAT', 'CTAAGCTATA', 'TAAGCTATAC', 'AAGCTATACT', 'AGCTATACTA'
, 'GCTATACTAA', 'CTATACTAAC', 'TATACTAACC', 'ATACTAACCC', 'TACTAACCCC', 'ACTAACCCCA', 'CTAACCCCAG', 'TAACCCCAGG',
'AACCCCAGGG', 'ACCCCAGGGT', 'CCCCAGGGTT', 'CCCAGGGTTG', 'CCAGGGTTGG', 'CAGGGTTGGT', 'AGGGTTGGTC', 'GGGTTGGTCA', '
GGTTGGTCAA', 'GTTGGTCAAT', 'TTGGTCAATT', 'TGGTCAATTT', 'GGTCAATTTC', 'GTCAATTTCG', 'TCAATTTCGT', 'CAATTTCGTG', 'A
ATTTCGTGC', 'ATTTCGTGCC', 'TTTCGTGCCA', 'TTCGTGCCAG', 'TCGTGCCAGC', 'CGTGCCAGCC', 'GTGCCAGCCA', 'TGCCAGCCAC', 'GC
CAGCCACC', 'CCAGCCACCG', 'CAGCCACCGC', 'AGCCACCGCG', 'GCCACCGCGG', 'CCACCGCGGT', 'CACCGCGGTC', 'ACCGCGGTCA', 'CCG
CGGTCAC', 'CGCGGTCACA', 'GCGGTCACAC', 'CGGTCACACG', 'GGTCACACGA', 'GTCACACGAT', 'TCACACGATT', 'CACACGATTA', 'ACAC
GATTAA', 'CACGATTAAC', 'ACGATTAACC', 'CGATTAACCC', 'GATTAACCCA', 'ATTAACCCAA', 'TTAACCCAAG', 'TAACCCAAGT', 'AACCC
AAGTC', 'ACCCAAGTCA', 'CCCAAGTCAA', 'CCAAGTCAAT', 'CAAGTCAATA', 'AAGTCAATAG', 'AGTCAATAGA', 'GTCAATAGAA', 'TCAATA
GAAG', 'CAATAGAAGC', 'AATAGAAGCC', 'ATAGAAGCCG', 'TAGAAGCCGG', 'AGAAGCCGGC', 'GAAGCCGGCG', 'AAGCCGGCGT', 'AGCCGGC
GTA', 'GCCGGCGTAA', 'CCGGCGTAAA', 'CGGCGTAAAG', 'GGCGTAAAGA', 'GCGTAAAGAG', 'CGTAAAGAGT', 'GTAAAGAGTG', 'TAAAGAGT
GT', 'AAAGAGTGTT', 'AAGAGTGTTT', 'AGAGTGTTTT', 'GAGTGTTTTA', 'AGTGTTTTAG', 'GTGTTTTAGA', 'TGTTTTAGAT', 'GTTTTAGAT
C', 'TTTTAGATCA', 'TTTAGATCAC', 'TTAGATCACC', 'TAGATCACCC', 'AGATCACCCC', 'GATCACCCCC', 'ATCACCCCCT', 'TCACCCCCTC
', 'CACCCCCTCC', 'ACCCCCTCCC', 'CCCCCTCCCC', 'CCCCTCCCCA', 'CCCTCCCCAA', 'CCTCCCCAAT', 'CTCCCCAATA', 'TCCCCAATAA'
, 'CCCCAATAAA', 'CCCAATAAAG', 'CCAATAAAGC', 'CAATAAAGCT', 'AATAAAGCTA', 'ATAAAGCTAA', 'TAAAGCTAAA', 'AAAGCTAAAA',
'AAGCTAAAAC', 'AGCTAAAACT', 'GCTAAAACTC', 'CTAAAACTCA', 'TAAAACTCAC', 'AAAACTCACC', 'AAACTCACCT', '
AACTCACCTG', 'ACTCACCTGA', 'CTCACCTGAG', 'TCACCTGAGT', 'CACCTGAGTT', 'ACCTGAGTTG', 'CCTGAGTTGT', 'CTGAGTTGTA', 'T
GAGTTGTAA', 'GAGTTGTAAA', 'AGTTGTAAAA', 'GTTGTAAAAA', 'TTGTAAAAAA', 'TGTAAAAAAC', 'GTAAAAAACT', 'TAAAAAACTC', 'AA
AAAACTCC', 'AAAAACTCCA', 'AAAACTCCAG', 'AAACTCCAGT', 'AACTCCAGTT', 'ACTCCAGTTG', 'CTCCAGTTGA', 'TCCAGTTGAC', 'CCA
GTTGACA', 'CAGTTGACAC', 'AGTTGACACA', 'GTTGACACAA', 'TTGACACAAA', 'TGACACAAAA', 'GACACAAAAT', 'ACACAAAATA', 'CACA
AAATAG', 'ACAAAATAGA', 'CAAAATAGAC', 'AAAATAGACT', 'AAATAGACTA', 'AATAGACTAC', 'ATAGACTACG', 'TAGACTACGA', 'AGACT
ACGAA', 'GACTACGAAA', 'ACTACGAAAG', 'CTACGAAAGT', 'TACGAAAGTG', 'ACGAAAGTGG', 'CGAAAGTGGC', 'GAAAGTGGCT', 'AAAGTG
GCTT', 'AAGTGGCTTT', 'AGTGGCTTTA', 'GTGGCTTTAA', 'TGGCTTTAAC', 'GGCTTTAACA', 'GCTTTAACAT', 'CTTTAACATA', 'TTTAACA
TAT', 'TTAACATATC', 'TAACATATCT', 'AACATATCTG', 'ACATATCTGA', 'CATATCTGAA', 'ATATCTGAAC', 'TATCTGAACA', 'ATCTGAAC
AC', 'TCTGAACACA', 'CTGAACACAC', 'TGAACACACA', 'GAACACACAA', 'AACACACAAT', 'ACACACAATA', 'CACACAATAG', 'ACACAATAG
C', 'CACAATAGCT', 'ACAATAGCTA', 'CAATAGCTAA', 'AATAGCTAAG', 'ATAGCTAAGA', 'TAGCTAAGAC', 'AGCTAAGACC', 'GCTAAGACCC
', 'CTAAGACCCA', 'TAAGACCCAA', 'AAGACCCAAA', 'AGACCCAAAC', 'GACCCAAACT', 'ACCCAAACTG', 'CCCAAACTGG', 'CCAAACTGGG'
, 'CAAACTGGGA', 'AAACTGGGAT', 'AACTGGGATT', 'ACTGGGATTA', 'CTGGGATTAG', 'TGGGATTAGA', 'GGGATTAGAT', 'GGATTAGATA',
'GATTAGATAC', 'ATTAGATACC', 'TTAGATACCC', 'TAGATACCCC', 'AGATACCCCA', 'GATACCCCAC', 'ATACCCCACT', 'TACCCCACTA', '
ACCCCACTAT', 'CCCCACTATG', 'CCCACTATGC', 'CCACTATGCT', 'CACTATGCTT', 'ACTATGCTTA', 'CTATGCTTAG', 'TATGCTTAGC', 'A
TGCTTAGCC', 'TGCTTAGCCC', 'GCTTAGCCCT', 'CTTAGCCCTA', 'TTAGCCCTAA', 'TAGCCCTAAA', 'AGCCCTAAAC', 'GCCCTAAACC', 'CC
CTAAACCT', 'CCTAAACCTC', 'CTAAACCTCA', 'TAAACCTCAA', 'AAACCTCAAC', 'AACCTCAACA', 'ACCTCAACAG', 'CCTCAACAGT', 'CTC
AACAGTT', 'TCAACAGTTA', 'CAACAGTTAA', 'AACAGTTAAA', 'ACAGTTAAAT', 'CAGTTAAATC', 'AGTTAAATCA', 'GTTAAATCAA', 'TTAA
ATCAAC', 'TAAATCAACA', 'AAATCAACAA', 'AATCAACAAA', 'ATCAACAAAA', 'TCAACAAAAC', 'CAACAAAACT', 'AACAAAACTG', 'ACAAA
ACTGC', 'CAAAACTGCT', 'AAAACTGCTC', 'AAACTGCTCG', 'AACTGCTCGC', 'ACTGCTCGCC', 'CTGCTCGCCA', 'TGCTCGCCAG', 'GCTCGC
CAGA', 'CTCGCCAGAA', 'TCGCCAGAAC', 'CGCCAGAACA', 'GCCAGAACAC', 'CCAGAACACT', 'CAGAACACTA', 'AGAACACTAC', 'GAACACT
ACG', 'AACACTACGA', 'ACACTACGAG', 'CACTACGAGC', 'ACTACGAGCC', 'CTACGAGCCA', 'TACGAGCCAC', 'ACGAGCCACA', 'CGAGCCAC
AG', 'GAGCCACAGC', 'AGCCACAGCT', 'GCCACAGCTT', 'CCACAGCTTA', 'CACAGCTTAA', 'ACAGCTTAAA', 'CAGCTTAAAA', 'AGCTTAAAA
C', 'GCTTAAAACT', 'CTTAAAACTC', 'TTAAAACTCA', 'AAAACTCAAA', 'AAACTCAAAG', 'AACTCAAAGG', 'ACTCAAAGGA', 'CTCAAAGGAC
', 'TCAAAGGACC', 'CAAAGGACCT', 'AAAGGACCTG', 'AAGGACCTGG', 'AGGACCTGGC', 'GGACCTGGCG', 'GACCTGGCGG', 'ACCTGGCGGT'
, 'CCTGGCGGTG', 'CTGGCGGTGC', 'TGGCGGTGCT', 'GGCGGTGCTT', 'GCGGTGCTTC', 'CGGTGCTTCA', 'GGTGCTTCAT', 'GTGCTTCATA',
'TGCTTCATAT', 'GCTTCATATC', 'CTTCATATCC', 'TTCATATCCC', 'TCATATCCCT', 'CATATCCCTC', 'ATATCCCTCT', 'TATCCCTCTA', '
ATCCCTCTAG', 'TCCCTCTAGA', 'CCCTCTAGAG', 'CCTCTAGAGG']
```

```
edges = [('ACCCCCACGG', 'CCCCCACGGG'), ('CCCCCACGGG', 'CCCCACGGGA'), ('CCCCACGGGA', 'CCCACGGGAA'), ('CCCACGGGAA', '
CCACGGGAAA'), ('CCACGGGAAA', 'CACGGGAAAC'), ('CACGGGAAAC', 'ACGGGAAACA'), ('ACGGGAAACA', 'CGGGAAACAG'), ('CGGGAAACA
G', 'GGGAAACAGC'), ('GGGAAACAGC', 'GGAAACAGCA'), ('GGAAACAGCA', 'GAAACAGCAG'), ('GAAACAGCAG', 'AAACAGCAGT'), ('AAAC
AGCAGT', 'AACAGCAGTG'), ('AACAGCAGTG', 'ACAGCAGTGA'), ('ACAGCAGTGA', 'CAGCAGTGAT'), ('CAGCAGTGAT', 'AGCAGTGATT'), (
'AGCAGTGATT', 'GCAGTGATTA'), ('GCAGTGATTA', 'CAGTGATTAA'), ('CAGTGATTAA', 'AGTGATTAAC'), ('AGTGATTAAC', 'GTGATTAACC
'), ('GTGATTAACC', 'TGATTAACCT'), ('TGATTAACCT', 'GATTAACCTT'), ('GATTAACCTT', 'ATTAACCTTT'), ('ATTAACCTTT', 'TTAAC
CTTTA'), ('TTAACCTTTA', 'TAACCTTTAG'), ('TAACCTTTAG', 'AACCTTTAGC'), ('AACCTTTAGC', 'ACCTTTAGCA'), ('ACCTTTAGCA', '
CCTTTAGCAA'), ('CCTTTAGCAA', 'CTTTAGCAAT'), ('CTTTAGCAAT', 'TTTAGCAATA'), ('TTTAGCAATA', 'TTAGCAATAA'), ('TTAGCAATA
A', 'TAGCAATAAA'), ('TAGCAATAAA', 'AGCAATAAAC'), ('AGCAATAAAC', 'GCAATAAACG'), ('GCAATAAACG', 'CAATAAACGA'), ('CAAT
AAACGA', 'AATAAACGAA'), ('AATAAACGAA', 'ATAAACGAAA'), ('ATAAACGAAA', 'TAAACGAAAG'), ('TAAACGAAAG', 'AAACGAAAGT'), (
'AAACGAAAGT', 'AACGAAAGTT'), ('AACGAAAGTT', 'ACGAAAGTTT'), ('ACGAAAGTTT', 'CGAAAGTTTA'), ('CGAAAGTTTA', 'GAAAGTTTAA
'), ('GAAAGTTTAA', 'AAAGTTTAAC'), ('AAAGTTTAAC', 'AAGTTTAACT'), ('AAGTTTAACT', 'AGTTTAACTA'), ('AGTTTAACTA', 'GTTTA
ACTAA'), ('GTTTAACTAA', 'TTTAACTAAG'), ('TTTAACTAAG', 'TTAACTAAGC'), ('TTAACTAAGC', 'TAACTAAGCT'), ('TAACTAAGCT', '
AACTAAGCTA'), ('AACTAAGCTA', 'ACTAAGCTAT'), ('ACTAAGCTAT', 'CTAAGCTATA'), ('CTAAGCTATA', 'TAAGCTATAC'), ('TAAGCTATA
C', 'AAGCTATACT'), ('AAGCTATACT', 'AGCTATACTA'), ('AGCTATACTA', 'GCTATACTAA'), ('GCTATACTAA', 'CTATACTAAC'), ('CTAT
ACTAAC', 'TATACTAACC'), ('TATACTAACC', 'ATACTAACCC'), ('ATACTAACCC', 'TACTAACCCC'), ('TACTAACCCC', 'ACTAACCCCA'), (
'ACTAACCCCA', 'CTAACCCCAG'), ('CTAACCCCAG', 'TAACCCCAGG'), ('TAACCCCAGG', 'AACCCCAGGG'), ('AACCCCAGGG', 'ACCCCAGGGT
'), ('ACCCCAGGGT', 'CCCCAGGGTT'), ('CCCCAGGGTT', 'CCCAGGGTTG'), ('CCCAGGGTTG', 'CCAGGGTTGG'), ('CCAGGGTTGG', 'CAGGG
TTGGT'), ('CAGGGTTGGT', 'AGGGTTGGTC'), ('AGGGTTGGTC', 'GGGTTGGTCA'), ('GGGTTGGTCA', 'GGTTGGTCAA'), ('GGTTGGTCAA', '
GTTGGTCAAT'), ('GTTGGTCAAT', 'TTGGTCAATT'), ('TTGGTCAATT', 'TGGTCAATTT'), ('TGGTCAATTT', 'GGTCAATTTC'), ('GGTCAATTT
C', 'GTCAATTTCG'), ('GTCAATTTCG', 'TCAATTTCGT'), ('TCAATTTCGT', 'CAATTTCGTG'), ('CAATTTCGTG', 'AATTTCGTGC'), ('AATT
TCGTGC', 'ATTTCGTGCC'), ('ATTTCGTGCC', 'TTTCGTGCCA'), ('TTTCGTGCCA', 'TTCGTGCCAG'), ('TTCGTGCCAG', 'TCGTGCCAGC'), (
'TCGTGCCAGC', 'CGTGCCAGCC'), ('CGTGCCAGCC', 'GTGCCAGCCA'), ('GTGCCAGCCA', 'TGCCAGCCAC'), ('TGCCAGCCAC', 'GCCAGCCACC
'), ('GCCAGCCACC', 'CCAGCCACCG'), ('CCAGCCACCG', 'CAGCCACCGC'), ('CAGCCACCGC', 'AGCCACCGCG'), ('AGCCACCGCG', 'GCCAC
CGCGG'), ('GCCACCGCGG', 'CCACCGCGGT'), ('CCACCGCGGT', 'CACCGCGGTC'), ('CACCGCGGTC', 'ACCGCGGTCA'), ('ACCGCGGTCA', '
CCGCGGTCAC'), ('CCGCGGTCAC', 'CGCGGTCACA'), ('CGCGGTCACA', 'GCGGTCACAC'), ('GCGGTCACAC', 'CGGTCACACG'), ('CGGTCACAC
G', 'GGTCACACGA'), ('GGTCACACGA', 'GTCACACGAT'), ('GTCACACGAT', 'TCACACGATT'), ('TCACACGATT', 'CACACGATTA'), ('CACA
CGATTA', 'ACACGATTAA'), ('ACACGATTAA', 'CACGATTAAC'), ('CACGATTAAC', 'ACGATTAACC'), ('ACGATTAACC', 'CGATTAACCC'), (
'CGATTAACCC', 'GATTAACCCA'), ('GATTAACCCA', 'ATTAACCCAA'), ('ATTAACCCAA', 'TTAACCCAAG'), ('TTAACCCAAG', 'TAACCCAAGT
'), ('TAACCCAAGT', 'AACCCAAGTC'), ('AACCCAAGTC', 'ACCCAAGTCA'), ('ACCCAAGTCA', 'CCCAAGTCAA'), ('CCCAAGTCAA', 'CCAAG
TCAAT'), ('CCAAGTCAAT', 'CAAGTCAATA'), ('CAAGTCAATA', 'AAGTCAATAG'), ('AAGTCAATAG', 'AGTCAATAGA'), ('AGTCAATAGA', '
GTCAATAGAA'), ('GTCAATAGAA', 'TCAATAGAAG'), ('TCAATAGAAG', 'CAATAGAAGC'), ('CAATAGAAGC', 'AATAGAAGCC'), ('AATAGAAGC
C', 'ATAGAAGCCG'), ('ATAGAAGCCG', 'TAGAAGCCGG'), ('TAGAAGCCGG', 'AGAAGCCGGC'), ('AGAAGCCGGC', 'GAAGCCGGCG'), ('GAAG
CCGGCG', 'AAGCCGGCGT'), ('AAGCCGGCGT', 'AGCCGGCGTA'), ('AGCCGGCGTA', 'GCCGGCGTAA'), ('GCCGGCGTAA', 'CCGGCGTAAA'), (
'CCGGCGTAAA', 'CGGCGTAAAG'), ('CGGCGTAAAG', 'GGCGTAAAGA'), ('GGCGTAAAGA', 'GCGTAAAGAG'), ('GCGTAAAGAG', 'CGTAAAGAGT
'), ('CGTAAAGAGT', 'GTAAAGAGTG'), ('GTAAAGAGTG', 'TAAAGAGTGT'), ('TAAAGAGTGT', 'AAAGAGTGTT'), ('AAAGAGTGTT', 'AAGAG
TGTTT'), ('AAGAGTGTTT', 'AGAGTGTTTT'), ('AGAGTGTTTT', 'GAGTGTTTTA'), ('GAGTGTTTTA', 'AGTGTTTTAG'), ('AGTGTTTTAG', '
GTGTTTTAGA'), ('GTGTTTTAGA', 'TGTTTTAGAT'), ('TGTTTTAGAT', 'GTTTTAGATC'), ('GTTTTAGATC', 'TTTTAGATCA'), ('TTTTAGATC
A', 'TTTAGATCAC'), ('TTTAGATCAC', 'TTAGATCACC'), ('TTAGATCACC', 'TAGATCACCC'), ('TAGATCACCC', 'AGATCACCCC'), ('AGAT
CACCCC', 'GATCACCCCC'), ('GATCACCCCC', 'ATCACCCCCT'), ('ATCACCCCCT', 'TCACCCCCTC'), ('TCACCCCCTC', 'CACCCCCTCC'), (
'CACCCCCTCC', 'ACCCCCTCCC'), ('ACCCCCTCCC', 'CCCCCTCCCC'), ('CCCCCTCCCC', 'CCCCTCCCCA'), ('CCCCTCCCCA', 'CCCTCCCCAA
'), ('CCCTCCCCAA', 'CCTCCCCAAT'), ('CCTCCCCAAT', 'CTCCCCAATA'), ('CTCCCCAATA', 'TCCCCAATAA'), ('TCCCCAATAA', 'CCCCA
ATAAA'), ('CCCCAATAAA', 'CCCAATAAAG'), ('CCCAATAAAG', 'CCAATAAAGC'), ('CCAATAAAGC', 'CAATAAAGCT'), ('CAATAAAGCT', '
AATAAAGCTA'), ('AATAAAGCTA', 'ATAAAGCTAA'), ('ATAAAGCTAA', 'TAAAGCTAAA'), ('TAAAGCTAAA', 'AAAGCTAAAG'), ('AAAGCTAAA
A', 'AAGCTAAAAC'), ('AAGCTAAAAC', 'AGCTAAAACT'), ('AGCTAAAACT', 'GCTAAAACTC'), ('GCTAAAACTC', 'CTAAAACTCA'), ('CTAA
AACTCA', 'TAAAACTCAC'), ('CTAAAACTCA', 'TAAAACTCAA'), ('TAAAACTCAC', 'AAAACTCACC'), ('TAAAACTCAA', 'AAAACTCAAA'), (
'AAAACTCACC', 'AAACTCACCT'), ('AAACTCACCT', 'AACTCACCTG'), ('AACTCACCTG', 'ACTCACCTGA'), ('ACTCACCTGA', 'CTCACCTGAG
'), ('CTCACCTGAG', 'TCACCTGAGT'), ('TCACCTGAGT', 'CACCTGAGTT'), ('CACCTGAGTT', 'ACCTGAGTTG'), ('ACCTGAGTTG', 'CCTGA
GTTGT'), ('CCTGAGTTGT', 'CTGAGTTGTA'), ('CTGAGTTGTA', 'TGAGTTGTAA'), ('TGAGTTGTAA', 'GAGTTGTAAA'), ('GAGTTGTAAA', '
AGTTGTAAAA'), ('AGTTGTAAAA', 'GTTGTAAAAA'), ('GTTGTAAAAA', 'TTGTAAAAAA'), ('TTGTAAAAAA', 'TGTAAAAAAC'), ('TGTAAAAAA
C', 'GTAAAAAACT'), ('GTAAAAAACT', 'TAAAAAACTC'), ('TAAAAAACTC', 'AAAAAACTCC'), ('AAAAAACTCC', 'AAAAACTCCA'), ('AAAA
ACTCCA', 'AAAACTCCAG'), ('AAAACTCCAG', 'AAACTCCAGT'), ('AAACTCCAGT', 'AACTCCAGTT'), ('AACTCCAGTT', 'ACTCCAGTTG'), (
'ACTCCAGTTG', 'CTCCAGTTGA'), ('CTCCAGTTGA', 'TCCAGTTGAC'), ('TCCAGTTGAC', 'CCAGTTGACA'), ('CCAGTTGACA', 'CAGTTGACAC
'), ('CAGTTGACAC', 'AGTTGACACA'), ('AGTTGACACA', 'GTTGACACAA'), ('GTTGACACAA', 'TTGACACAAA'), ('TTGACACAAA', 'TGACA
CAAAA'), ('TGACACAAAA', 'GACACAAAAT'), ('GACACAAAAT', 'ACACAAAATA'), ('ACACAAAATA', 'CACAAAATAG'), ('CACAAAATAG', '
ACAAAATAGA'), ('ACAAAATAGA', 'CAAAATAGAC'), ('CAAAATAGAC', 'AAAATAGACT'), ('AAAATAGACT', 'AAATAGACTA'), ('AAATAGACT
A', 'AATAGACTAC'), ('AATAGACTAC', 'ATAGACTACG'), ('ATAGACTACG', 'TAGACTACGA'), ('TAGACTACGA', 'AGACTACGAA'), ('AGAC
TACGAA', 'GACTACGAAA'), ('GACTACGAAA', 'ACTACGAAAG'), ('ACTACGAAAG', 'CTACGAAAGT'), ('CTACGAAAGT', 'TACGAAAGTG'), (
'TACGAAAGTG', 'ACGAAAGTGG'), ('ACGAAAGTGG', 'CGAAAGTGGC'), ('CGAAAGTGGC', 'GAAAGTGGCT'), ('GAAAGTGGCT', 'AAAGTGGCTT
'), ('AAAGTGGCTT', 'AAGTGGCTTT'), ('AAGTGGCTTT', 'AGTGGCTTTA'), ('AGTGGCTTTA', 'GTGGCTTTAA'), ('GTGGCTTTAA', 'TGGCT
TTAAC'), ('TGGCTTTAAC', 'GGCTTTAACA'), ('GGCTTTAACA', 'GCTTTAACAT'), ('GCTTTAACAT', 'CTTTAACATA'), ('CTTTAACATA', '
TTTAACATAT'), ('TTTAACATAT', 'TTAACATATC'), ('TTAACATATC', 'TAACATATCT'), ('TAACATATCT', 'AACATATCTG'), ('AACATATCT
G', 'ACATATCTGA'), ('ACATATCTGA', 'CATATCTGAA'), ('CATATCTGAA', 'ATATCTGAAC'), ('ATATCTGAAC', 'TATCTGAACA'), ('TATC
TGAACA', 'ATCTGAACAC'), ('ATCTGAACAC', 'TCTGAACACA'), ('TCTGAACACA', 'CTGAACACAC'), ('CTGAACACAC', 'TGAACACACA'), (
'TGAACACACA', 'GAACACACAA'), ('GAACACACAA', 'AACACACAAT'), ('AACACACAAT', 'ACACACAATA'), ('ACACACAATA', 'CACACAATAG
'), ('CACACAATAG', 'ACACAATAGC'), ('ACACAATAGC', 'CACAATAGCT'), ('CACAATAGCT', 'ACAATAGCTA'), ('ACAATAGCTA', 'CAATA
GCTAA'), ('CAATAGCTAA', 'AATAGCTAAG'), ('AATAGCTAAG', 'ATAGCTAAGA'), ('ATAGCTAAGA', 'TAGCTAAGAC'), ('TAGCTAAGAC', '
AGCTAAGACC'), ('AGCTAAGACC', 'GCTAAGACCC'), ('GCTAAGACCC', 'CTAAGACCCA'), ('CTAAGACCCA', 'TAAGACCCAA'), ('TAAGACCCA
A', 'AAGACCCAAA'), ('AAGACCCAAA', 'AGACCCAAAC'), ('AGACCCAAAC', 'GACCCAAACT'), ('GACCCAAACT', 'ACCCAAACTG'), ('ACCC
AAACTG', 'CCCAAACTGG'), ('CCCAAACTGG', 'CCAAACTGGG'), ('CCAAACTGGG', 'CAAACTGGGA'), ('CAAACTGGGA', 'AAACTGGGAT'), (
'AAACTGGGAT', 'AACTGGGATT'), ('AACTGGGATT', 'ACTGGGATTA'), ('ACTGGGATTA', 'CTGGGATTAG'), ('CTGGGATTAG', 'TGGGATTAGA
'), ('TGGGATTAGA', 'GGGATTAGAT'), ('GGGATTAGAT', 'GGATTAGATA'), ('GGATTAGATA', 'GATTAGATAC'), ('GATTAGATAC', 'ATTAG
ATACC'), ('ATTAGATACC', 'TTAGATACCC'), ('TTAGATACCC', 'TAGATACCCC'), ('TAGATACCCC', 'AGATACCCCA'), ('AGATACCCCA', '
GATACCCCAC'), ('GATACCCCAC', 'ATACCCCACT'), ('ATACCCCACT', 'TACCCCACTA'), ('TACCCCACTA', 'ACCCCACTAT'), ('ACCCCACTA
T', 'CCCCACTATG'), ('CCCCACTATG', 'CCCACTATGC'), ('CCCACTATGC', 'CCACTATGCT'), ('CCACTATGCT', 'CACTATGCTT'), ('CACT
ATGCTT', 'ACTATGCTTA'), ('ACTATGCTTA', 'CTATGCTTAG'), ('CTATGCTTAG', 'TATGCTTAGC'), ('TATGCTTAGC', 'ATGCTTAGCC'), (
'ATGCTTAGCC', 'TGCTTAGCCC'), ('TGCTTAGCCC', 'GCTTAGCCCT'), ('GCTTAGCCCT', 'CTTAGCCCTA'), ('CTTAGCCCTA', 'TTAGCCCTAA
'), ('TTAGCCCTAA', 'TAGCCCTAAA'), ('TAGCCCTAAA', 'AGCCCTAAAC'), ('AGCCCTAAAC', 'GCCCTAAACC'), ('GCCCTAAACC', 'CCCTA
AACCT'), ('CCCTAAACCT', 'CCTAAACCTC'), ('CCTAAACCTC', 'CTAAACCTCA'), ('CTAAACCTCA', 'TAAACCTCAA'), ('TAAACCTCAA', '
AAACCTCAAC'), ('AAACCTCAAC', 'AACCTCAACA'), ('AACCTCAACA', 'ACCTCAACAG'), ('ACCTCAACAG', 'CCTCAACAGT'), ('CCTCAACAG
T', 'CTCAACAGTT'), ('CTCAACAGTT', 'TCAACAGTTA'), ('TCAACAGTTA', 'CAACAGTTAA'), ('CAACAGTTAA', 'AACAGTTAAA'), ('AACA
GTTAAA', 'ACAGTTAAAT'), ('ACAGTTAAAT', 'CAGTTAAATC'), ('CAGTTAAATC', 'AGTTAAATCA'), ('AGTTAAATCA', 'GTTAAATCAA'), (
'GTTAAATCAA', 'TTAAATCAAC'), ('TTAAATCAAC', 'TAAATCAACA'), ('TAAATCAACA', 'AAATCAACAA'), ('AAATCAACAA', 'AATCAACAAA
'), ('AATCAACAAA', 'ATCAACAAAA'), ('ATCAACAAAA', 'TCAACAAAAC'), ('TCAACAAAAC', 'CAACAAAACT'), ('CAACAAAACT', 'AACAA
AACTG'), ('AACAAAACTG', 'ACAAAACTGC'), ('ACAAAACTGC', 'CAAAACTGCT'), ('CAAAACTGCT', 'AAAACTGCTC'), ('AAAACTGCTC', '
AAACTGCTCG'), ('AAACTGCTCG', 'AACTGCTCGC'), ('AACTGCTCGC', 'ACTGCTCGCC'), ('ACTGCTCGCC', 'CTGCTCGCCA'), ('CTGCTCGCC
A', 'TGCTCGCCAG'), ('TGCTCGCCAG', 'GCTCGCCAGA'), ('GCTCGCCAGA', 'CTCGCCAGAA'), ('CTCGCCAGAA', 'TCGCCAGAAC'), ('TCGC
CAGAAC', 'CGCCAGAACA'), ('CGCCAGAACA', 'GCCAGAACAC'), ('GCCAGAACAC', 'CCAGAACACT'), ('CCAGAACACT', 'CAGAACACTA'), (
'CAGAACACTA', 'AGAACACTAC'), ('AGAACACTAC', 'GAACACTACG'), ('GAACACTACG', 'AACACTACGA'), ('AACACTACGA', 'ACACTACGAG
'), ('ACACTACGAG', 'CACTACGAGC'), ('CACTACGAGC', 'ACTACGAGCC'), ('ACTACGAGCC', 'CTACGAGCCA'), ('CTACGAGCCA', 'TACGA
GCCAC'), ('TACGAGCCAC', 'ACGAGCCACA'), ('ACGAGCCACA', 'CGAGCCACAG'), ('CGAGCCACAG', 'GAGCCACAGC'), ('GAGCCACAGC', '
AGCCACAGCT'), ('AGCCACAGCT', 'GCCACAGCTT'), ('GCCACAGCTT', 'CCACAGCTTA'), ('CCACAGCTTA', 'CACAGCTTAA'), ('CACAGCTTA
A', 'ACAGCTTAAA'), ('ACAGCTTAAA', 'CAGCTTAAAA'), ('CAGCTTAAAA', 'AGCTTAAAAC'), ('AGCTTAAAAC', 'GCTTAAAACT'), ('GCTT
AAAACT', 'CTTAAAACTC'), ('CTTAAAACTC', 'TTAAAACTCA'), ('TTAAAACTCA', 'TAAAACTCAC'), ('TTAAAACTCA', 'TAAAACTCAA'), (
'AAAACTCAAA', 'AAACTCAAAG'), ('AAACTCAAAG', 'AACTCAAAGG'), ('AACTCAAAGG', 'ACTCAAAGGA'), ('ACTCAAAGGA', 'CTCAAAGGAC
'), ('CTCAAAGGAC', 'TCAAAGGACC'), ('TCAAAGGACC', 'CAAAGGACCT'), ('CAAAGGACCT', 'AAAGGACCTG'), ('AAAGGACCTG', 'AAGGA
CCTGG'), ('AAGGACCTGG', 'AGGACCTGGC'), ('AGGACCTGGC', 'GGACCTGGCG'), ('GGACCTGGCG', 'GACCTGGCGG'), ('GACCTGGCGG', '
ACCTGGCGGT'), ('ACCTGGCGGT', 'CCTGGCGGTG'), ('CCTGGCGGTG', 'CTGGCGGTGC'), ('CTGGCGGTGC', 'TGGCGGTGCT'), ('TGGCGGTGC
T', 'GGCGGTGCTT'), ('GGCGGTGCTT', 'GCGGTGCTTC'), ('GCGGTGCTTC', 'CGGTGCTTCA'), ('CGGTGCTTCA', 'GGTGCTTCAT'), ('GGTG
CTTCAT', 'GTGCTTCATA'), ('GTGCTTCATA', 'TGCTTCATAT'), ('TGCTTCATAT', 'GCTTCATATC'), ('GCTTCATATC', 'CTTCATATCC'), (
'CTTCATATCC', 'TTCATATCCC'), ('TTCATATCCC', 'TCATATCCCT'), ('TCATATCCCT', 'CATATCCCTC'), ('CATATCCCTC', 'ATATCCCTCT
'), ('ATATCCCTCT', 'TATCCCTCTA'), ('TATCCCTCTA', 'ATCCCTCTAG'), ('ATCCCTCTAG', 'TCCCTCTAGA'), ('TCCCTCTAGA', 'CCCTC
TAGAG'), ('CCCTCTAGAG', 'CCTCTAGAGG')]
```

Hamiltonian path after string reconstruction:
(cant make out anything from the graph due to its huge complexity)

```
  Original String   = ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCAGGGTTGGTCAATTTCGTGCCAG
CCACCGCGGTCACACGATTAACCCAAGTCAATAGAAGCCGGCGTAAAGAGTGTTTTAGATCACCCCCTCCCCAATAAAGCTAAAACTCACCTGAGTTGTAAAAAACTCCAGTTGA
CACAAAATAGACTACGAAAGTGGCTTTAACATATCTGAACACACAATAGCTAAGACCCAAACTGGGATTAGATACCCCACTATGCTTAGCCCTAAACCTCAACAGTTAAATCAAC
AAAACTGCTCGCCAGAACACTACGAGCCACAGCTTAAAACTCAAAGGACCTGGCGGTGCTTCATATCCCTCTAGAGG

Reconstructed String = ACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAAACGAAAGTTTAACTAAGCTATACTAACCCCAGGGTTGGTCAATTTCGTGCCA
GCCACCGCGGTCACACGATTAACCCAAGTCAATAGAAGCCGGCGTAAAGAGTGTTTTAGATCACCCCCTCCCCAATAAAGCTAAAACTCACCTGAGTTGTAAAAAACTCCAGTTG
ACACAAAATAGACTACGAAAGTGGCTTTAACATATCTGAACACACAATAGCTAAGACCCAAACTGGGATTAGATACCCCACTATGCTTAGCCCTAAACCTCAACAGTTAAATCAA
CAAAACTGCTCGCCAGAACACTACGAGCCACAGCTTAAAACTCAAAGGACCTGGCGGTGCTTCATATCCCTCTAGAGG

 Run Time:  4.216611208000359
```

Note: If K < 10, it does not give any string , And when k > 10 we get one desired reconstructed string

End