**CS 238 – Assignment 2**

**Name –** Lohith Kumar Bhambore

**Student ID –** 862393065

**Net ID –** Lbham003

***Question 1***

**Solution**

The idea is to use a dp approach only if k is greater than zero. After every insert or delete, decrease the value of k. And once k is zero, all we are left with is just the match or mismatch scores for each element. The dp recurrence when the number of insertions and deletions are bounded by k is given as follows –

Si, j = max {

Si-1, j + w(ai, -) , k > 0

Si, j-1 + w(-, bj) , k > 0

Si-1, j-1 + w(ai, bj) , k > 0 }

This leads to a dp table as shown below, we only compute the values which are a distance k away from the diagonal –

A picture containing line, pattern, rectangle, screenshot

Description automatically generated

Thus, the runtime complexity becomes O((m+n)\*k) where m is the length of A, n is the length of B and k is the bound on the number of insertions and deletions.

***Question 2***

See Colab for code and algorithm – <https://colab.research.google.com/drive/1xoUMHcTVFxX-X8cu40CDFnqDyjsoH8QW?usp=sharing>

I have also shared it below –

**Algorithm**

1. Read the shotgun reads from the fasta file.
2. Create a graph representation of the overlaps between reads.
3. Find the Eulerian path in the graph
4. Stitch the reads together based on the path to obtain the reconstructed genome sequence.

**Code**

def parse\_fasta(file\_path):

reads = []

with open(file\_path, "r") as file:

lines = file.readlines()

for line in lines:

if line.startswith(">"):

reads.append("")

else:

reads[-1] += line.strip()

return reads

def build\_graph(reads):

graph = {}

for i, read in enumerate(reads):

graph[i] = []

for j, other\_read in enumerate(reads):

if i != j:

overlap = find\_overlap(read, other\_read)

if overlap:

graph[i].append(j)

return graph

def find\_overlap(read1, read2):

min\_overlap = min(len(read1), len(read2)) // 2

for i in range(min\_overlap, 0, -1):

if read1.endswith(read2[:i]):

return i

return 0

def find\_hamiltonian\_path(graph):

stack = [0]

path = []

while stack:

node = stack[-1]

if node not in path:

path.append(node)

if len(path) == len(graph):

return path

found\_neighbor = False

for neighbor in graph[node]:

if neighbor not in path:

stack.append(neighbor)

found\_neighbor = True

break

if not found\_neighbor:

stack.pop()

return None

def reconstruct\_genome(reads, path):

genome = reads[path[0]]

for i in range(1, len(path)):

overlap = find\_overlap(reads[path[i - 1]], reads[path[i]])

genome += reads[path[i]][overlap:]

return genome[:1000] # Take the first 1000 basepairs

# Parse the fasta file and extract the reads

reads = parse\_fasta("reads.fasta")

# Build the graph representation

graph = build\_graph(reads)

# Find the Hamiltonian path

hamiltonian\_path = find\_hamiltonian\_path(graph)

if hamiltonian\_path is not None:

# Reconstruct the original genome sequence

genome = reconstruct\_genome(reads, hamiltonian\_path)

print("Reconstructed Genome Sequence:")

print(genome)

print("Length of Reconstructed Genome Sequence:")

print(len(genome))

else:

print("No Hamiltonian path found.")

Output –

Reconstructed Genome Sequence:

AGCCAATAGCAGATATGCCCATACCGCTGTATTCATAGCTTTCTCTACACGGCCTAAAAGCGGTCGACTGCACGGTCGACTGCACGCAGTCTTCCGGAGCCAATAGCAGATATGCCCATACTTATCACGTACGTAGATTCGAATACAAATTCGACAGATGTGGTTTAATGATTCCGCCTCCTATTACAACAGCCCCGAGGATCCTGCACTGAGTCTGAGGAGCTGGGGTGCAAATTAGACGGCCTAAAAGCGGTCGACTGCAAGGTAGGGGTACGTACATGTTTCCCATAGGCAGCGCCTGCCGCTGTTACAACAGCCGACATTGCGACACAATACCAGTTTTTATTGTGTCCATGTACCGCCTAACACTTATCACGTACGTACATGTTTCGGGAGAGAAAGGGGTGATGTTCTGTTATTAGACCGACGCCCCTAATTGGATCAATCAGGGTAGGTCATGGGAGGGGTGATGTTCGAATAAATGGCATATAAGCCCGGATCCGTCCTGTCTGCGACACTGACATGGATCCGTCCTGTCTGCGACGTTTCGGGTCGATAAAGCGTTGTCCGACGCCCCTAATTGGATCAATCCTGATGGTACTCCCCCTTCATTGCGCCCGTTTCCCATGTACCGCCTAGTACAAATTCGACAGATGTGGTTTATTCGATGGGTAGGTCATGGGAGTAGAGTCGGTGAGGAGCTGGGGTGAGTCTTCCTGATGGTACTCCCCCTGCACCATGAACGCGATTGCTAAACATGGATCCGTCCTGTCTGCGACATGGGAGGGGTGATGTTCTGTTATTGGTACTCCCCCTTCATTGGTCGGTAGAGTCGGTGAGGAGCTGGGGTGAGGATTAGCGCCCGTTTCCCATGTACCGCCTAACACTCGTGGTGAGCAGGAAATTATTCGCTTTACTAGTCACGTGCTCTAAAATAGCTTTCTCTACACGATCGAGTTGGGAGGATTAGCGCCCGTTTCCTCCTGGGGTCCGCGATGTC

Length of Reconstructed Genome Sequence:

1000

***Question 3***

See Colab for code and algorithm – <https://colab.research.google.com/drive/1xoUMHcTVFxX-X8cu40CDFnqDyjsoH8QW?usp=sharing>

I have also shared it below –

**Algorithm**

1. Parse the fasta file and extract the shotgun reads.

2. Build a de Bruijn graph using the shotgun reads.

3. Find the Eulerian path in the de Bruijn graph.

4. Concatenate the nodes of the Eulerian path to obtain the reconstructed genome sequence.

The key part is choosing the appropriate value for k in the de Bruijn graph approach can be a challenging task, as it depends on various factors such as read length, sequencing errors, and genome complexity.

**Code**

def parse\_fasta\_file(file\_path):

sequences = []

with open(file\_path, 'r') as file:

lines = file.readlines()

for line in lines:

line = line.strip()

if line.startswith('>'):

sequences.append('')

else:

sequences[-1] += line

return sequences

def build\_de\_bruijn\_graph(sequences, k):

graph = {}

for sequence in sequences:

for i in range(len(sequence) - k + 1):

kmer = sequence[i:i+k]

prefix = kmer[:-1]

suffix = kmer[1:]

if prefix in graph:

graph[prefix].append(suffix)

else:

graph[prefix] = [suffix]

return graph

def find\_eulerian\_path(graph):

start\_node = list(graph.keys())[0]

current\_node = start\_node

path = [current\_node]

while True:

if current\_node not in graph:

break

next\_node = graph[current\_node].pop()

if len(graph[current\_node]) == 0:

del graph[current\_node]

current\_node = next\_node

path.append(current\_node)

return path

def reconstruct\_genome(sequences, k):

graph = build\_de\_bruijn\_graph(sequences, k)

eulerian\_path = find\_eulerian\_path(graph)

reconstructed\_genome = eulerian\_path[0]

for node in eulerian\_path[1:]:

reconstructed\_genome += node[-1]

return reconstructed\_genome[:1000] # Take the first 1000 basepairs

# Provide the path to the fasta file

fasta\_file = 'reads.fasta'

# Parse the fasta file and extract the shotgun reads

sequences = parse\_fasta\_file(fasta\_file)

# Reconstruct the original genome sequence

reconstructed\_sequence = reconstruct\_genome(sequences, k=5) # Adjust the value of k as needed

print("Reconstructed Genome Sequence using Eulerian Path approach:")

print(reconstructed\_sequence)

print("Length of Reconstructed Genome Sequence:")

print(len(reconstructed\_sequence))

**Output**

Reconstructed Genome Sequence using Eulerian Path approach:

AGCCCATTACGTGATTCCGTGCTCTACCAGTTTCGGGAGAGATGGTCGGTGAAGTTTCATTGGTACTCCCCTTCATCTAAAACCGATCTGTTATTAGACCCCCGGCCCGTACATGTTCTGCCGTCCTGTCTGCCGCCGCTGCGAATGGATCCGGAGATGGTTGGGAGTGGAGAAAGGCATATAAGCCCGGAATACCCATAGGCCAATCCGGAGTGGAGGATTACTAGTCACGTACGAAAGCCAATCCGGCCAATAAGCGCCTGCGAATCCGCCTCCTAGACATGGCAGCGCAAGGCTTTACAACAGTGGGAGTGGGTACATGGTTGGGTGATTCGGGAGGGGTAGAGCCGACAAAACAGCTTCAATCCGGAGAGATATAAAACAGTTATTGTGTCCCATGTTTTATCTGCGACACGTGCTCTAAATGGCGCCCGTTTCCGGAGTGGAGTGATGTATAAGCCCGTGCTCTAAATGGTTATCTGCGAATCCGGAGATTAGCGTTTCGAATAGCAGATAAAGGTAGGTCGGTGAATCCGGAGTTAATGGGGTATTATGCCGTCCTGTACGTGCTCTACACAAGGTACTAGTCCGGCCCATTGGTCACGTACCGATATGATTTGCAAGGCATATAGAGATGTTTCTCTCGTTGGTTATCTGCGACATTGGGTAGGGGTATAGCTTTACTAGTCATTGCGACACTGAATAAAAACCGCCCGTAGAGGCCCGGAGCTTCAGCTGCGAATGGTCGGGTCGATCCGTTTCGGGTCGGTGCAAGGCATATAGACATAGGATCATGTGGTTTCGGTGATCGATTAGAGGCTTTACTCCCCTTCAGCTTTACTAGTCACGTGAGCTGGGAGTGGGTGATGGCAGAGGATCTACCAGTCATAGACAACAGCCGCCGACGTGTCTGTCCCATTAGCGCCTATTCATTGCGACATAGGAGGCGACACAAATTACGACATTGCAAAAACCGACACTGGATTATCATCGACATTCG

Length of Reconstructed Genome Sequence:

1000