1. **Describe your approach to creating an assembler. Please be a specific as possible in describing your approach and why/how you chose this particular approach. You may also include information about things you initially tried, but that didn't work before you tried something else.**

   I chose the DFS algorithm because of its simplicity and usefulness in graph traversal. DFS allows exploration of the De Bruijn graph, potentially generating contigs from the connected k-mers. The idea was to follow paths in the graph to form sequences that represent parts of the genome. However, my attempt at using DFS was not successful. The assembler failed to produce meaningful contigs, which is evident from the N50 and L50 scores, both being 0.
   I added some print statements to find where is the issue:

```python
print("De Bruijn Graph:")
for node, neighbors in de_bruijn_graph.items():
  print(f"{node}: {neighbors}")

contigs = assemble_contigs(de_bruijn_graph)

print("Assembled Contigs:")
for i, contig in enumerate(contigs, start=1):
  print(f"Contig {i}: {contig}")
```

   I ran *python3 assemble.py reads.txt 3* and got the following results:

```
... 'AG', 'AC', 'AC', 'AT', 'AT', 'AC', 'AA', 'AT', 'AA', 'AA', 'AC', 'AT', 'AG', 'AC',
'AC', 'AG', 'AT', 'AC', 'AA', 'AC', 'AA', 'AG', 'AA', 'AG', 'AA', 'AT', 'AC', 'AC',
'AG', 'AG', 'AG', 'AG', 'AT', 'AG', 'AT', 'AT', 'AT', 'AC', 'AA', 'AG', 'AC', 'AC',
'AG', 'AA', 'AA', 'AC', 'AG', 'AC', 'AA', 'AT', 'AG', 'AT', 'AG', 'AC', 'AT', 'AC',
'AT', 'AC', 'AA', 'AT', 'AG', 'AA', 'AG']
Assembled Contigs:
N50 for assembly: 0
L50 for assembly: 0
```

   I was not able to figure out what the problem was. It could be my DFS implementation because my graph construction worked for the previous assignment. But overall, this indicates that the simplicity of DFS is not sufficient for the complexity of real genomic data.

2. **What modifications to your data structures for storing the De Bruijn graph were necessary? For example, maybe you had to change between storing multi-edges as separate edges to just a single edge with a multiplicity. Maybe you needed to add a new data structure that kept track of which edges corresponded to each read. Etc.**

   In constructing the De Bruijn graph, I utilized a straightforward data structure, which is a defaultdict of lists. Each node in the graph represented a k-mer, and the edges indicated transitions between adjacent k-mers. This approach aimed to keep the representation simple and easy to work with.

3. **How well did you approach work (on both error-free data and data with errors)? Use data to support your conclusions. For example, inclusion of plots or figures (or specific N50, L50 or other scoring metrics) may be useful here. Can you explain why your approach either worked well or worked poorly? Conjectures are also reasonable here if you aren't sure why your approach performed the way it did. At a minimum, your analysis should include your N50 and L50 scores for different values of k for the two datasets you created in part 1 of this assignment (sample c12 r 50 e0.00.txt and sample c12 r 50 e0.01.txt).**

```
ramazania@LAPTOP-F7UE5P2T:/mnt/c/Users/ramaz/Desktop/Comp Bio/HW5$ python3 assemble.py sample_c12_r_50_e0.00.txt 3
N50 for assembly: 0
L50 for assembly: 0
ramazania@LAPTOP-F7UE5P2T:/mnt/c/Users/ramaz/Desktop/Comp Bio/HW5$ python3 assemble.py sample_c12_r_50_e0.00.txt 5
N50 for assembly: 0
L50 for assembly: 0
ramazania@LAPTOP-F7UE5P2T:/mnt/c/Users/ramaz/Desktop/Comp Bio/HW5$ python3 assemble.py sample_c12_r_50_e0.01.txt 3
N50 for assembly: 0
L50 for assembly: 0
ramazania@LAPTOP-F7UE5P2T:/mnt/c/Users/ramaz/Desktop/Comp Bio/HW5$ python3 assemble.py sample_c12_r_50_e0.01.txt 5
N50 for assembly: 0
L50 for assembly: 0
ramazania@LAPTOP-F7UE5P2T:/mnt/c/Users/ramaz/Desktop/Comp Bio/HW5$ python3 assemble.py reads.txt 3
N50 for assembly: 0
L50 for assembly: 0
ramazania@LAPTOP-F7UE5P2T:/mnt/c/Users/ramaz/Desktop/Comp Bio/HW5$ python3 assemble.py reads.txt 5
N50 for assembly: 0
L50 for assembly: 0
ramazania@LAPTOP-F7UE5P2T:/mnt/c/Users/ramaz/Desktop/Comp Bio/HW5$
```

   The N50 and L50 scores consistently being 0 across different k values and datasets suggest that the assembler failed to generate meaningful contigs. This suggests that the DFS-based approach didn't effectively navigate the De Bruijn graph to construct contiguous sequences. This could be because the depth-first search might not capture the complex patterns in the De Bruijn graph, leading to incomplete contigs.

4. What did you learn from this assignment? Please spend some time reflecting on what was interesting or challenging with this assignment and report back.

This assignment taught me a lot about the complexities involved in creating an assembler for genome assembly. I started with a seemingly simple approach, using a DFS algorithm on the De Bruijn graph. The simplicity of DFS, which initially seemed like a good fit, turned out to be a challenge.

The interesting part was observing how the assembler performed on different datasets—both error-free and those with errors. Despite my efforts, the assembler consistently failed to produce meaningful contigs. This was a bit frustrating but also interesting because it made me realize the difficulties of working with real genomic data.

The use of N50 and L50 scores as metrics for evaluating the assembler's performance added a quantitative aspect to the analysis. The scores consistently being 0 highlighted the need for a deeper understanding of the algorithm and data structures. Although I didn't achieve the desired results, the assignment emphasized the importance of persistence and iterative improvement in bioinformatics.

Overall, this assignment was a valuable experience in understanding the challenges of genome assembly and the need for continuous improvement. It emphasized the interdisciplinary nature of bioinformatics, where computational techniques must adapt to the nuances of biological data.