Sharmila Tummala

Homework 1

Question 1:

Euler Circuit

Step1: Verify Euler Theorem

Euler Theorem: A connected undirected graph has Euler circuit if every vertex has even degree.

From the give graph, all the vertices' degrees are:

- → a = 2
- → b = 4
- → c = 4
- → d = 2
- → e = 4
- → f = 4
- → g = 4
- → h = 2
- → i = 4
- → j = 4

Since all the vertices have even degree, the graph will have Euler circuit.

Step2: Apply Hierholzer's Algorithm

In Hierholzer's Algorithm, we begin at vertex and construct a cycle by iteratively following unused edges. After an edge is traversed, it is no longer considered. We keep extending the cycle until all the edges are present whenever we get to a vertex that still has unused edges. This procedure guarantees that the path returns to the initial vertex and that each edge is traversed precisely once.

Finding Euler circuit (1) first greedy partial circuit
Start at vertex **e** and follow unused edges greedily until returning to **e**.
Partial circuit C1:
C1: e → d → j → i → e
Edges used in C1: (e,d), (d,j), (j,i), (i,e)

Finding Euler circuit (2) first vertex in C1 with unused edges
Scan C1 in order (e, d, j, i). The first vertex encountered that still has unused edges is **i**. Start at **i** and build a new partial circuit using unused edges.
Partial circuit C2 (starting at i):
C2: i → f → e → g → i
Edges used in C2: (i,f), (f,e), (e,g), (g,i)
Merge C2 into C1 at vertex i. That is, replace the occurrence of i in C1 with the cycle C2 (starting/ending at i).

Merged circuit after splice (R1):

R1: e → d → j → i → f → e → g → i → e

Finding Euler circuit (3) next vertex with unused edges

Scan R1 in order e, d, j, i, f, e, g, i, e. The first vertex with unused edges is **f**. Start at **f** and build a new partial circuit.

Partial circuit C3 (starting at f):

C3: f → c → h → j → g → b → f

Edges used in C3: (f,c), (c,h), (h,j), (j,g), (g,b), (b,f)

Merge C3 into R1 at vertex f.

Merged circuit after splice (R2):

R2: e → d → j → i → f → c → h → j → g → b → f → e → g → i → e

Finding Euler circuit (4) next vertex with unused edges

Scan R2 in order; find that **b** still has unused edges (to a) that were not yet used. Build the next partial circuit from b.

Partial circuit C4 (starting at b):

C4: b → c → a → b

Edges used in C4: (b,c), (c,a), (a,b)

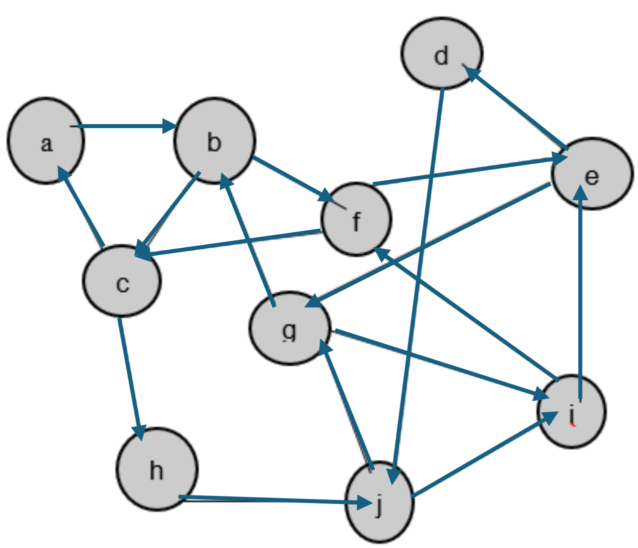
Merge C4 into R2 at vertex b.

Merged circuit after final splice (R3):

R3: e → d → j → i → f → c → h → j → g → b → c → a → b → f → e → g → i → e

Now all edges are used exactly once, so R3 is an Euler circuit.

The final Euler circuit:



Question 2:

Explanation:

- Given a set $X=\{x1,x2,\ldots,xn\}$, the multiset ΔX is:
  $\Delta X=\{|xj-xi| \mid 1\leq i<j\leq n\}$
- Each unordered pair is considered exactly once.
- If multiple pairs give the same difference, that difference appears multiple times in ΔX.
- This is a brute force algorithm, since we check all possible pairs.

Pseudocode:

Algorithm BruteForceDeltaX(X)
Input: A set X = {x1, x2, …, xn} of integers
Output: The multiset ΔX of all pairwise differences

```
Sort X into nondecreasing order
ΔX ← empty multiset
FOR i ← 1 TO n - 1 DO
    FOR j ← i + 1 TO n DO
        d ← X[j] - X[i]
        INSERT d INTO ΔX
    END FOR
END FOR
RETURN ΔX
```

Example:

Let
$X=\{0,4,7,10\}$
Compute pairwise differences (j>i):

- 4-0 = 4
- 7-0 = 7
- 10-0 = 10
- 7-4 = 3
- 10-4 = 6
- 10-7 = 3

So the multiset is:
$\Delta X=\{3,3,4,6,7,10\}$

Correctness

- Every pair is included once, and duplicates are preserved (two "3"s appear).

Complexity Analysis

- **Sorting step:** Sorting n integers takes O (n log n).
- **Nested loops:**
  - The outer loop runs n-1times.
  - The inner loop runs decreasingly from n-1down to 1.
  - Total iterations = n(n-1)/2, which is $O(n^2)$
  - Each iteration does constant-time work (subtraction and insertion).
- Total running time:

  $O(n \log n) + O(n^2) = O(n^2)$

Output size check:
ΔX itself contains $n(n-1)/2=O(n^2)$ elements, so no algorithm can be faster than $O(n^2)$ in general. This proves the brute force algorithm is asymptotically optimal.

CBI_home1 (1)

September 18, 2025

[1]:

```python
from collections import Counter
import re

def read_fasta(filename):
    """Read a FASTA file and return the concatenated sequence (only A/C/G/T)."""
    with open(filename, 'r') as f:
        lines = [line.strip() for line in f if line.strip()]
    if lines[0].startswith(">"):
        lines = lines[1:]   # skip header line
    seq = ''.join(lines).upper()
    seq = re.sub(r'[^ACGT]', '', seq)  # keep only A,C,G,T
    return seq

def count_codons(seq):
    """Count non-overlapping codons in the sequence (frame 0)."""
    counts = Counter()
    for i in range(0, len(seq) - 2, 3):
        codon = seq[i:i+3]
        if len(codon) == 3:
            counts[codon] += 1
    # Ensure all 64 codons appear (even if zero)
    bases = ['A','C','G','T']
    for a in bases:
        for b in bases:
            for c in bases:
                counts.setdefault(a+b+c, 0)
    return counts

def main():
    fasta_file = "sequence (1).fasta"
    seq = read_fasta(fasta_file)
    counts = count_codons(seq)

    # Print to screen in required format
    for codon in sorted(counts.keys()):
        print(f"{codon}   {counts[codon]}")
```

```python
    # Also save to file
    with open("codon_counts_output.txt", "w") as out:
        for codon in sorted(counts.keys()):
            out.write(f"{codon}    {counts[codon]}\n")

    print("\nCodon counts saved to codon_counts_output.txt")

if __name__ == "__main__":
    main()
```

```
AAA   37539
AAC   26349
AAG   20454
AAT   27522
ACA   18535
ACC   24472
ACG   27244
ACT   15015
AGA   18433
AGC   29696
AGG   18679
AGT   14964
ATA   24069
ATC   29562
ATG   26335
ATT   27975
CAA   23363
CAC   19401
CAG   35517
CAT   25647
CCA   28619
CCC   17808
CCG   34569
CCT   18521
CGA   25823
CGC   48735
CGG   34240
CGT   27185
CTA   9869
CTC   14923
CTG   35051
CTT   21272
GAA   27786
GAC   20901
GAG   15490
GAT   29516
GCA   30122
GCC   36763
```

```
GCG   49644
GCT   29014
GGA   21062
GGC   37594
GGG   17900
GGT   24124
GTA   18903
GTC   20605
GTG   19652
GTT   26209
TAA   24092
TAC   19168
TAG   10363
TAT   24411
TCA   27491
TCC   21256
TCG   25783
TCT   18708
TGA   26880
TGC   29233
TGG   27737
TGT   18737
TTA   23956
TTC   27835
TTG   23251
TTT   37572

Codon counts saved to codon_counts_output.txt
```

[ ]:

```

```