

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:

$$\rightarrow a = 2$$

$$\rightarrow b = 4$$

$$\rightarrow c = 4$$

$$\rightarrow d = 2$$

$$\rightarrow e = 4$$

$$\rightarrow f = 4$$

$$\rightarrow g = 4$$

$$\rightarrow h = 2$$

$$\rightarrow i = 4$$

$$\rightarrow j = 4$$

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

Step2: Apply Hierholzer's Algorithm

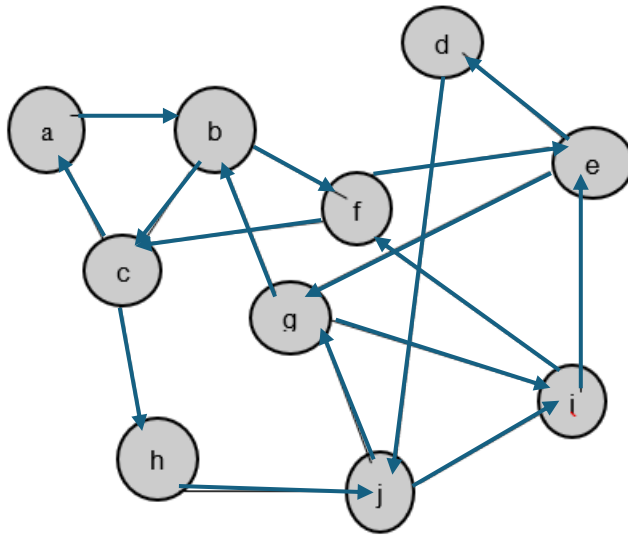
In Hierholzer's Algorithm, we begin at vertex a and construct a cycle by iteratively following unused edges. After an edge is traversed, it is no longer taken into account. We keep extending the cycle until all of the edges are present whenever we get to a vertex that still has unused edges. This procedure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

Starting at a

The path goes like below:

$a \rightarrow b \rightarrow f \rightarrow e \rightarrow d \rightarrow j \rightarrow i \rightarrow e \rightarrow g \rightarrow i \rightarrow f \rightarrow c \rightarrow h \rightarrow j \rightarrow g \rightarrow b \rightarrow c \rightarrow a$

The final Euler circuit:



Question 2:

Explanation:

- Given a set $X = \{x_1, x_2, \dots, x_n\}$, the multiset ΔX is:
 $\Delta X = \{|x_j - x_i| \mid 1 \leq i < j \leq n\}$
- 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 $X = \{x_1, x_2, \dots, x_n\}$ of integers

Output: The multiset ΔX of all pairwise differences

1. Sort X into nondecreasing order
2. $\Delta X \leftarrow$ empty multiset
3. FOR $i \leftarrow 1$ TO $n - 1$ DO
4. FOR $j \leftarrow i + 1$ TO n DO
5. $d \leftarrow X[j] - X[i]$
6. INSERT d INTO ΔX
7. END FOR
8. END FOR
9. 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 $n-1$ times.
 - The inner loop runs decreasingly from $n-1$ down 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]: 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]}")
```

```

# 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

[]: