1. **Greedy Algorithms** (25)

(a) (10) What is an optimal Huffman code for the following set of letters whose frequencies are based on the first 8 Fibonacci numbers?

\[ a : 1 \quad b : 1 \quad c : 2 \quad d : 3 \quad e : 5 \quad f : 8 \quad g : 13 \quad h : 21 \]

Can you generalize your answer to find the optimal code when the frequencies are the first \( n \) Fibonacci numbers?

**Answer Key:** The Huffman code for these characters are: \( C(h) = 1, C(g) = 01, C(f) = 001, C(e) = 0001, C(d) = 00001, C(c) = 000001, C(b) = 0000001, C(a) = 0000000. \) If we switch 0 and 1, we get another code; the codes of \( a \) and \( b \) can also be exchanged.

The generalized case is that the Huffman codes are 1, 01, 001, ..., \( 0^{k-1}1, 0^k \) for the characters from high frequency to low frequency, assuming we have \( k + 1 \) characters in total. The code tree is a linear structure such that each level has exactly one leaf, except the root which has no leaves and the bottom level which has two leaves. The characters with lower frequencies stay at lower level.

(b) (15) Suppose a data file contains a sequence of 8-bit characters such that all 256 characters are about as common: the maximum character frequency is less than twice the minimum character frequency. Prove that Huffman coding in this case also produces 8-bit code for each character.

**Answer Key:** The Huffman algorithm starts with a set of trees of singleton, and then merge two trees with lowest frequencies into a larger tree (called composed trees). We show that if the maximum character frequency is less than twice the minimum character frequency, and we have \( n = 2^k \) characters, then every character has the Huffman code of length \( k \). We prove this by induction on \( k \).

When \( k = 1 \), we have two characters and they both have a Huffman code of length one. The statement is true.

For the inductive case, please take note that singletons will be chosen first before any composed tree in the merge process, because singletons will always have lower frequencies than that of composed trees. That is, if the minimum character frequency is \( m_1 \) and the maximum character frequency is \( m_2 \), than the frequency of any composed tree is at least \( 2m_1 \) and \( 2m_1 > m_2 \) by the assumption. After \( n/2 \) merges, all the \( n \) singletons have been chosen and we have \( n/2 = 2^{k-1} \) composed trees of two characters. If we replace those \( n/2 \) composed trees by \( n/2 = 2^{k-1} \) new characters, they are still satisfy the property that the maximum character frequency is less than twice the minimum character frequency. By the induction hypothesis, these new characters will have Huffman code of length \( k - 1 \). Now replace the new characters by the corresponding composite trees of two leaves in the code tree for \( n/2 \) new characters, we have a Huffman code tree for each old character and the code length will be \( k \).

When \( n = 256 = 2^8 \), we have the solution to the original problem.

2. **Dynamic Programming** (25)

Given two strings of letters, say \( X = x_1x_2 \cdots x_n \) and \( Y = y_1y_2 \cdots y_m \), a common subsequence
of $X$ and $Y$ is a string which appears as subsequence in both $X$ and $Y$. For example, if $X = ACCGGTA$ and $Y = CGTTAG$, then $CGTA$ is a common subsequence of $X$ and $Y$. Please design an algorithm as efficient as possible to compute the length of a longest common subsequence of $X$ and $Y$. You need first to present your idea in terms of a recursive function and then to present your algorithm in pseudo-code and explain briefly each line of the code. You also need to provide a time and space complexity analysis of your algorithm (using the big-O notation).

**Answer Key:** Let $Opt(i, j)$ be the longest length of common subsequence of $x_1 x_2 \cdots x_i$ and $y_1 y_2 \cdots y_j$. Then $Opt(0, j) = Opt(i, 0) = 0$ and

$$Opt(i, j) = \begin{cases} 1 + Opt(i - 1, j - 1) & \text{if } x_i = y_j, \\ \max(Opt(i - 1, j), Opt(i, j - 1)) & \text{if } x_i \neq y_j \end{cases}$$

Let $M$ be an $(n + 1) \times (m + 1)$ array to store $Opt(i, j)$, the pseudo-code will be as follows:

1. for (int $i = 0$; $i <= n$; $i++$) $M[i, 0] = 0$;
2. for (int $j = 0$; $j <= m$; $j++$) $M[0, j] = 0$;
3. for (int $i = 1$; $i <= n$; $i++$)
   4.   for (int $j = 1$; $j <= m$; $j++$)
   5.     if ($X[i] == Y[j]$) $M[i, j] = 1 + M[i-1, j-1]$;
   7.     else $M[i, j] = M[i-1, j]$;
8. return $M[n, m]$;

Line 1 and 2 computes $M$ according to $Opt(0, j) = Opt(i, 0) = 0$.
Line 3-4 are two nested loops for other items of $M$ according to the general case of $Opt(i, j)$.
Line 5 checks if $x_i = y_j$, and let $Opt(i, j) = 1 + Opt(i - 1, j - 1)$.
Line 6-7 computes $\max(Opt(i, j - 1), Opt(i - 1, j))$ for $Opt(i, j)$ when $x_i \neq y_j$.
Line 8 returns the result $Opt(n, m)$.

The time and space complexity is $O(mn)$. The first index of $M$ can be taken modulo 2, thus the space complexity can be reduced to $O(m)$.

3. **Network Flow** (25)

Let $G = (X, Y, E)$ be a bipartite graph, where $X = \{x_1, x_2, x_3, x_4\}$, $Y = \{y_1, y_2, y_3, y_4\}$, and $E = \{(x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2), (x_3, y_3), (x_3, y_4), (x_4, y_4)\}$. We try to find a maximum matching of $G$ using the max-flow algorithm by adding source $s$ and sink $t$ to $G$ with proper capacity for each edge.

(a) (15) Suppose the first two augmenting paths we considered are $s - x_1 - y_1 - t$ and $s - x_3 - y_3 - t$. Please continue to find more augmenting paths until it does not exist.

**Answer Key:** There are two remaining augmenting paths: $s - x_2 - y_1 - x_1 - y_2 - t$ and $s - x_4 - y_3 - x_3 - y_4 - t$.

(b) (10) Please list all the minimum cuts of the corresponding flow network.

**Answer Key:** There are five minimum cuts $(A, B)$:

i. $A = \{s\}$ and $B = V - A$;
ii. $A = \{s, x_4, y_3\}$ and $B = V - A$;
iii. $A = \{s, x_2, y_1, x_4, y_3\}$ and $B = V - A$;
iv. $B = \{t\}$ and $A = V - B$;
v. $B = \{t, x_3, y_4\}$ and $A = V - B$.

4. **Dynamic Programming** (25)

A binary relation $R$ on the set $V$ can be represented by a directed graph $G = (V, E)$ such that, for any $x, y \in V$, $R(x, y)$ is true if and only if $(x, y) \in E$. The transitive closure of $R$, denoted by $R^*$, is a binary relation on $V$ defined as:

(a) If $R(x, y)$, then $R^*(x, y)$;

(b) If $R(x, y)$ and $R^*(y, z)$, then $R^*(x, z)$.

Suppose $G = (V, E)$ is represented by the (boolean) adjacency matrix $M : M[x, y] = 1$ iff $(x, y) \in E$; $M[x, y] = 0$, otherwise. Please design an efficient algorithm which computes the relation $R^*$ and analyze its time and space complexity.

**Answer Key:** The binary relation $R$ is equivalent to the directed graph $G = (V, E)$ where $R(x, y)$ is true iff $(x, y) \in E)$. The transitive closure $R^*$ represents the connectivity relation of $G$, that is, $R^*(x, y)$ is true iff there exists a path from $x$ to $y$ in $G$. Since we have an $O(n^3)$ time and $O(n^2)$ space algorithm to compute the shortest path between any two points in $G$, we may use it to compute $R^*$. The pseudo-code is given below, where $T[x, y] = 1$ iff $R^*(x, y)$ is true.

```plaintext
Alg.computeTransitiveClosure(M, n) {
    for j= 1 to n for i = 1 to n
        T[i,j] = M[i,j];
    for k= 1 to n
        for j= 1 to n for i = 1 to n
            T[i,j] = T[i,j] ∨ (T[i,k] ∧ T[k,j]);
    return T;
}
```

For each $k$, $T[i,j] = T[i,j] ∨ (T[i,k] ∧ T[k,j])$ means that if $T[i,j]$ is true (i.e., there is a path from $i$ to $j$), then $T[i, j]$ is still true; if both $T[i,k]$ and $T[k,j]$ are true, then there is a path from $i$ to $k$ and a path from $k$ to $j$, so there is a path from $i$ to $j$. 