Problem 1 (30 points, Problem 16.3-3). What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers?

\[
\begin{align*}
  a & : 1 \\
  b & : 1 \\
  c & : 2 \\
  d & : 3 \\
  e & : 5 \\
  f & : 8 \\
  g & : 13 \\
  h & : 21 \\
\end{align*}
\]

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

Solution. The following figure is the optimal Huffman code for the first 8 numbers: Now, we generalize this the first \( n \) Fibonacci numbers. Let \( f_i \) be the \( i \)th Fibonacci number and \( fc_i \) be the Huffman code for \( f_i \), then \( fc_1 = 1^{n-2}0, fc_2 = 1^{n-1}, fc_i = 1^{n-i}0 \) (\( i = 3, 4, ..., n \)), where \( 1^j \) denotes the concatenation of \( j \) 1’s.

Now let’s prove the correctness. Let \( S_i = \sum_{j=1}^{i} f_j \). We claim that \( S_i < f_{i+2} \).

Let’s prove it by induction.

Base step: for \( j = 1 \), \( S_1 = f_1 < f_3 \) obviously holds.

Inductive step: Let’s assume \( S_j < f_{j+2} \) for all \( j = 1, 2, ..., i-1 \). Now let’s show \( S_i < f_{i+2} \) holds.

\[
S_i = S_{i-1} + f_i < f_{i+1} + f_i = f_{i+2}.
\]

Hence the node with the value \( S_i \) will always combine with \( f_{i+1} \). Therefore, we obtain the Huffman tree similar to the figure.

Problem 2 (60 points, Problem 16-1). Coin changing

Consider the problem of making change for \( n \) cents using the fewest number of coins. Assume that each coin’s value is an integer.

(a). Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

(b). Suppose that the available coins are in the denominations that are powers of \( c \), i.e., the denominations are \( c^0, c^1, ..., c^k \) for some integers \( c > 1 \) and \( k \geq 1 \). Show that the greedy algorithm always yields an optimal solution.
(c). Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of \( n \).

(d). Give an \( O(nk) \)-time algorithm that makes change for any set of \( k \) different coin denominations, assuming that one of the coins is a penny.

\textbf{Solution.} (a). Given \( n \) cents, let \( n_q = \lfloor n/25 \rfloor \), \( n_r = n - 25n_q \), \( n_d = \lfloor r/10 \rfloor \), \( n_5 = r - 10n_d \), \( n_k = \lfloor r_k/5 \rfloor \), \( r_k = r - 5n_k \), \( n_p = r_k \). Thus, we can make change for \( n \) dollars by obtaining \( n_q \) quarters, \( n_d \) dimes, \( n_k \) nickels, and \( n_p \) pennies. This Greedy algorithm requires \( O(1) \) time.

Next, we'll prove the correctness.

We prove it by induction. First, the Greedy algorithm produces optimal solutions for arbitrary \( n \) if there are only nickels and pennies, and let's denote the Greedy algorithm by \( A_2 \). Assume that the optimal solution is \( x_k \) nickels and \( x_p \) pennies. If \( x_p \geq 5 \), then it's not optimal because \( x'_k = x_k + \lfloor x_p/5 \rfloor \), \( x'_p = x_p - 5\lfloor x_p/5 \rfloor \) gives fewer number of coins, contradiction.

Next, we prove that the Greedy algorithm also works if there are only dimes, nickels, and pennies, and we denote this Greedy algorithm by \( A_3 \). Otherwise, assume the optimal solution is \( x_d, x_k, x_p \). Then \( x_p < 5 \); otherwise we can employ \( A_2 \) to \( 3x_k + x_p \) to get a better solution. If \( 5x_k + x_p < 10 \), then \( (x_d, x_k, x_p) \) is also the greedy solution to \( A_2 \) and is optimal. If \( 5x_k + x_p \geq 10 \), which implies \( 5x_k \leq 10 \) since \( x_p < 5 \), then we can get a better solution \( (x_d + \lfloor x_k/2 \rfloor, x_k - 2\lfloor x_k/2 \rfloor, x_p) \). Therefore, \( A_3 \) produces optimal solution.

Finally, we prove that the Greedy algorithm is correct if there are quarters, dimes, nickels, and pennies, and we denote this algorithm by \( A_4 \). Otherwise, assume the optimal solution is \( x_q, x_d, x_k, x_p \). We know that \( x_k < 2 \) and \( x_p < 5 \); otherwise, we can use algorithm \( A_3 \) to get a better solution on the input \( 10x_d + 5x_k + x_p \). Let \( n_q = \lfloor n/25 \rfloor \). If \( n_q = n_q \), then this optimal solution is also the solution to algorithm \( A_4 \). If \( x_q < n_q \), then by \( A_3 \), each quarters will lead to \( 1 \) more coins, which means that \( (x_q, x_d, x_k, x_p) \) is not optimal.

Altogether, the greedy algorithm yields optimal solution.

(b). Given an optimal solution \( (x_0, x_1, \ldots, x_k) \), where \( x_i \) indicates the number of coins of denomination \( c_i \). First, \( x_i < c \) for every \( i = 1, 2, \ldots, k - 1 \). Suppose we have some \( x_i \geq c \), then we could decrease \( x_i \) by \( c \) and increase \( x_{i+1} \) by \( 1 \). This connection of coins has the same value and has \( c - 1 \) fewer coins, so the original solution must be non-optimal. This configuration of coins is exactly the same as you would get if you kept greedily picking the largest coin possible. Let \( S_i = \sum_{j=1}^{i} c_j (c - 1) \). Then \( S_i < c^{i+1} \) for \( i = 0, 1, \ldots, k \). Thus \( (x_0, x_1, \ldots, x_k) \) is the only solution that satisfies the property \( x_i < c \) for \( i = 1, 2, \ldots, k - 1 \). Therefore, the greedy algorithm always yields an optimal solution.

(c). Let the coin denominations be \( \{1, 3, 4\} \), and the value to make change for be \( 6 \). The greedy solution would result in the collection of coins \( \{1, 1, 4\} \), but the optimal solution is \( \{3, 3\} \).

(d) Let \( \text{numcoins}[i] \) be the fewest number of coins to make change for \( i \) cents and \( S \) be the collection of \( k \) distinct coin denominations. Then \( \text{numcoins}[0] = 0 \), and \( \text{numcoins}[i] = \min_{j \geq c, c \in S} \text{numcoins}[i - c] + 1 \).
Pseudocode:

Algorithm 1 Pseudocode for Problem 16.1

Input: $S, n$
Output: $\text{numcoins}[n]$ and $\text{coins}$

1. $\text{numcoins}[0] = 0$
2. for $i = 1$ to $n$ do
3. 
4. 
5. if $c \leq i$ and $\text{numcoins}[i] > \text{numcoins}[i - c] + 1$ then
6. 
7. 
8. end if
9. end for
10. $\text{iter} = n$
11. let $\text{coins}$ be an empty set
12. while $\text{iter} > 0$ do
13. 
14. 
15. 
16. 
17. 
18. end if
19. end while

The time complexity is $O(nk)$, where $k = |S|$.

Problem 3 (10 points, Problem 22.1-7). The incidence matrix of a directed graph $G = (V, E)$ with no self-loops is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that

$$b[i, j] = \begin{cases} -1 & \text{if edge } j \text{ leaves vertex } i, \\ 1 & \text{if edge } j \text{ enters vertex } i, \\ 0 & \text{otherwise.} \end{cases}$$

Describe what the entries of the matrix product $BB^T$ represent, where $B^T$ is the transpose of $B$.

Solution. The entry $[i, i]$ for some $i$ is the sum of the in and out degrees of the vertex that $i$ corresponds to.

$$BB^T[i, i] = \sum_{j=1}^{E} b_{ij}^2$$

Now consider the entry $[i, j]$, where $i \neq j$.

$$BB^T[i, j] = \sum_{k=1}^{E} b_{ik} b_{jk}$$
Thus $BB^T[i,j]$ is the negation of number of edges between $i$ and $j$, and 0 otherwise.