Problem 1 (25 points, Problem 16.1-3). Not just any greedy approach to the activity-selection problem produces a maximum- size set of mutually compatible activities. Give an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities do not work. Do the same for the approaches of always selecting the compatible activity that overlaps the fewest other remaining activities and always selecting the compatible remaining activity with the earliest start time.

Solution. Counter example for always selecting the activity with the least duration:

Let $(s_1, f_1) = (3, 5), (s_2, f_2) = (1, 4), (s_3, f_3) = (4, 7)$. The greedy solution is $\{$activity 1$\}$. The optimal solution is $\{$activity 2, activity 3$\}$.

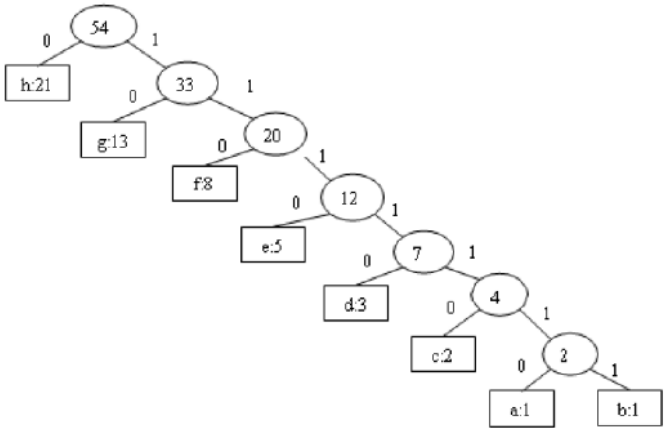
Counter example for always selecting the task with fewest overlaps: Let $(s_1, f_1) = (3, 5), (s_2, f_2) = (0, 2), (s_3, f_3) = (6, 8), (s_4, f_4) = (2, 4), (s_5, f_5) = (4, 6), (s_6, f_6) = (1, 3), (s_7, f_7) = (1, 3), (s_8, f_8) = (5, 7), (s_9, f_9) = (5, 7)$. The Greedy solution is $\{$activity 1, activity 2, activity 3$\}$, optimal solution is $\{$activity 2, activity 3, activity 4, activity 5$\}$.

Problem 2 (25 points, Problem 16.3-3). What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers?

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

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 ith



Symbol	Huffman Code
a	1111110
b	1111111
c	111110
d	11110
e	1110
f	110
g	10
h	0

Figure 1: Problem 16.3-3

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 3 (50 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.

Solution. (a). Given n cents, let $n_q = \lfloor n/25 \rfloor, r_q = n - 25n_q, n_d = \lfloor r_q/10 \rfloor, r_d = r_q - 10n_d, n_k = \lfloor r_d/5 \rfloor, r_k = r_d - 5n_k, n_p = r_k$. 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 $\mathcal{A}_2$. 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 $\mathcal{A}_3$. Otherwise, assume the optimal solution is x_d, x_k, x_p. Then $x_p < 5$; otherwise we can employ $\mathcal{A}_2$ to $5x_k + x_p$ to get a better solution. If $5x_k + x_p < 10$, then (x_d, x_k, x_p) is also the greedy solution to $\mathcal{A}_3$ 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, $\mathcal{A}_3$ 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 $\mathcal{A}_4$. 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 $\mathcal{A}_3$ to get a better solution on the input $10x_d + 5x_k + x_p$. Let $n_q = \lfloor n/25 \rfloor$. If $x_q = n_q$, then this optimal solution is also the solution to algorithm $\mathcal{A}_4$. If $x_q < n_q$, then by $\mathcal{A}_3$, 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 $(x_0, x_1, ..., x_k)$, where x_i indicates the number of coins of denomination c^i. First, $x_i < c$ fore every $i = 1, 2, ..., k-1$. 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 $i = 0, 1, ..., k$. Thus $(x_0, x_1, ..., x_k)$ is the only solution that satisfies the property $x_i < c$ $(i = 1, 2, ..., k-1)$. 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 $numcoins[i]$ be the fewest number of coins to make change for i cents and S be the collection of k distinct coin denominations. Then $numcoins[0] = 0$, and $numcoins[i] = \min_{i \geq c, c \in S} numcoins[i - c] + 1$.

Algorithm 1 Pseudocode for Problem 16.1

Input: S, n
Output: $numcoins[n]$ and $coins$

```
1: numcoins[0] = 0
2: for i = 1 to n do
3:    numcoins[i] = +∞
4:    for c in S do
5:       if c ≤ i and numcoins[i] > numcoins[i − c] + 1 then
6:          numcoins[i] = 1 + numcoins[i − c]
7:       end if
8:    end for
9: end for
10: iter = n
11: let coins be an empty set
12: while iter > 0 do
13:    for c in S do
14:       if c ≤ iter and numcoins[iter] − 1 == numcons[iter − c] then
15:          add c to coins
16:          iter = iter − c
17:       end if
18:    end for
19: end while
```

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