Problem 1. (25 points [greedy])

Draw the Huffman tree and find the optimal prefix code for the symbols in the following frequency table

Answer:

<table>
<thead>
<tr>
<th>symbol</th>
<th>frequency</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>1010</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>1011</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>1001</td>
</tr>
<tr>
<td>E</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>10000</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>10001</td>
</tr>
</tbody>
</table>

```
          D
           1
          /  
         0    1
        /     /
      E      6
      /  
     0    0
    /    /
  45 1  15
    /  
A   1  9
    /      /
  25 1  0
    /  
  0   0
```

Problem 2. (25 points [greedy])

In the activity selection problem we discussed in class, we are given a set of activities $S$, where each activity $a$ is defined by a start time $s_a$ and a finish time $f_a$; the objective is to produce the largest subset of non-conflicting activities. The greedy algorithm first sorts the activities in $S$ by finish time, then schedules first the activity in $S$ that has the earliest finish time (greedy choice). Prove that there exists an optimal solution that begins with a greedy choice. Complete the proof.

Answer: See slides.

Problem 3. (25 points [dynamic programming])
In this problem you are asked to traceback the solution of an instance of 0-1 knapsack with \( n = 6 \) items, with weights \( w_1 = 2, w_2 = 4, w_3 = 6, w_4 = 3, w_5 = 5, w_6 = 3 \), benefits \( b_1 = 1, b_2 = 3, b_3 = 5, b_4 = 4, b_5 = 4, b_6 = 2 \), and a knapsack size \( W = 11 \).

Recall that we define the table \( P[i, k] \) as the maximum profit possible using items \( \{i, i+1, \ldots, n\} \) and residual (knapsack) capacity \( k \). The table is filled according to the following rules:

\[
P[i, k] = \begin{cases} 
0 & \text{if } i = n \text{ and } w_n > k \\
b_n & \text{if } i = n \text{ and } w_n \leq k \\
P[i + 1, k] & \text{if } i < n \text{ and } w_i > k \\
\max\{P[i + 1, k], P[i + 1, k - w_i] + b_i\} & \text{if } i < n \text{ and } w_i \leq k 
\end{cases}
\]

1. Draw the traceback pointers for the optimal solution(s) from the cell in bold.

2. Write the optimal solution(s): which items are selected? Compute that total benefit and the total weight of the optimal solution(s).

**Answer:** Here is the traceback pointers from cell (1,11):

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>

We have two optimal solutions, namely \( x_1 = (1, 0, 1, 1, 0, 0) \) and \( x_2 = (0, 0, 0, 1, 1, 1) \). The total weight for \( x_1 \) is \( 2 + 6 + 3 = 11 \) and the total benefit is \( 1 + 5 + 4 = 10 \). The total weight for \( x_2 \) is \( 3 + 5 + 3 = 11 \) and the total benefit is \( 4 + 4 + 2 = 10 \).

**Problem 4.** (25 points [dynamic programming])

Consider the weighted version of the activity selection problem we discussed in class, where now tasks have benefits. You are given an array \( A \) of \( n \) tasks described by start time \( s_i \), finish time \( f_i \), and benefit \( b_i \), which are all integers. Tasks are sorted in increasing order of \( s_i \) in \( A \). Give a dynamic programming algorithm that computes the largest possible total benefit for a subset of non-conflicting tasks in \( A \). Your algorithm should run in \( O(n^2) \) time (explain why).

For instance, if \( A = \{(1, 4, 5), (1, 3, 6), (2, 3, 2), (4, 5, 1), (4, 6, 3), (5, 6, 3)\} \) where each triple is \((s_i, f_i, b_i)\), the highest total benefit is 10 (see figure on the right).

**Hints:** Define \( P[i] \) to be the largest total benefit for a subset of tasks \( \{1, 2, \ldots, i\} \) which includes task \( i \). The structure of the recurrence relation for \( P[i] \) is similar to the one for longest increasing subsequence.
**Answer:** The tasks are sorted by start times in $A$. The recurrence relation to fill the table $P$ is the following

$$P[i] = \begin{cases} b_1 & \text{if } i = 1 \\ b_i + \max_{1 \leq j < i} \{P[j] : s_i \geq f_j\} & \text{otherwise} \end{cases}$$

where we assume that max of an empty set returns 0. The condition $s_i \geq f_j$ ensures that task $i$ can be scheduled without conflict with task $j$. In that case, we fetch the optimal way to schedule $\{1, 2, \ldots, j\}$ by getting $P[j]$. Here is the implementation:

$$P[1] \leftarrow b_1$$

for $i \leftarrow 2$ to $n$

$\quad max \leftarrow 0$

for $j \leftarrow 1$ to $i$

$\quad$ if $s_i \geq f_j$ and $P[j] > max$ then $max \leftarrow P[j]$

$\quad P[i] \leftarrow b_i + max$

The time complexity is $O(n^2)$, while the space complexity is $O(n)$. 