Greedy's application - DP

Huffman code

File coding → compressed → Transmission → DES → decoding → file

file 100,000 characters. a b c d e f \(\Sigma = 6\)
\[\left\lceil \log_2 6 \right\rceil = 3\]

frequency: a b c d e f
45% 13% 12% 11% 9% 1%

0 10 1 11 110 1100

size: 100,000 \times 3 = 3 \times 10^5

\text{VS. } 100,000 \left( 45\% \times 1 + 13\% \times 3 + \ldots \right) = 224,000 \text{ bits}
Huffman codes

For each char \( c \) in the alphabet \( C \), let \( f(c) \) denote the frequency of \( c \) in the file and \( d_T(c) \) denote the code length of \( c \).

The number of bits required to encode a file is thus

\[
\min B(T) = \sum_c f(c) \cdot d_T(c)
\]

Optimization problem: design \( d_T(c) \) to minimize \( B(T) \)

Make a tree: frequency

\[
\begin{array}{ccccccccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} & \text{g} & \text{h} \\
0.01 & 0.1 & 0.11 & 0.04 & 0.4 & 0.2 & 0.1 & 0.04 & \Sigma=1
\end{array}
\]
a tree i frequency: \( \frac{a}{0.05} \), \( \frac{b}{0.01} \), \( \frac{c}{0.11} \), \( \frac{d}{0.04} \), \( \frac{e}{0.4} \), \( \frac{f}{0.2} \), \( \frac{g}{0.1} \), \( \frac{h}{0.04} \), \( \frac{a-d}{0.09} \), \( \frac{a}{0.19} \)

\( a: 0000000 \)
\( b: 0000001 \)
\( c: 00001 \)
\( d: 0001 \)
\( e: 001 \)
\( f: 001 \)
\( g: 010 \)
\( h: 011 \)
Algorithm for Huffman code

1. \( N \leftarrow |C| \)
2. \( Q \leftarrow C \) // build a heap
3. For \( i \leftarrow 1 \) to \( n-1 \)
   do allocate a new code \( z \)
5. \( \text{left}[z] \leftarrow x \leftarrow \text{ExtractMIN}(Q) \)
6. \( \text{right}[z] \leftarrow y \leftarrow \text{ExtractMIN}(Q) \)
7. \( f[z] \leftarrow f[x] + f[y] \) // new node's key
8. \( \text{INSERT}(Q, z) \)
9. Return \( \text{ExtractMIN}(Q) \)

\( Q \): ordinary array/sorted array

\( O(n^2) \)  
\( \text{min-heap} \)  
\( O(n \log n) \)
Dynamic Programming (DP)  
\[ \Rightarrow \text{tabular method, not coding} \]

- apply to optimization
- systematically records answers to subproblems so that they are not recomputed.

e.g. Fibonacci numbers

\[
\begin{align*}
F(1) &= F(0) = 1 \\
F(n) &= F(n-1) + F(n-2)
\end{align*}
\]

```
F(5)  
\_/   
F(4) F(3)  
_/ _/  
F(2) F(1) F(2) F(1)  
/ / 
F(1) F(0) F(1) F(0) F(0) F(1) F(0)  
/ 
F(1) F(0)
```
```python
Fib(n): integer
    if n == 0 or n == 1
        return 1
    else
        return Fib(n-1) + Fib(n-2)
```

```
Fib(n): integer
    if n == 0
        return 1
    F[i] = 1
    for i = 2 to n
        F[i] = F[i-1] + F[i-2]
```

Memoization: a technique used to speed up algorithm by storing results rather than recomputing them.

DP: optimal structure: optimal solution to a problem

Incorporating optimal solutions to subproblems, which we may solve independently.

Independency between subproblems: solution to one subproblem doesn't affect solution to another problem.
Matrix chain multiplication

\[
\begin{pmatrix}
 2 & 1 \\
-1 & 0 \\
3 & 2
\end{pmatrix}
\times
\begin{pmatrix}
4 \\
1
\end{pmatrix}
= 
\begin{pmatrix}
9 \\
-4 \\
14
\end{pmatrix}
\]

\[3 \times 2 	imes 2 \times 1\]

matrix chain multiplication

```
Associative property
```

<table>
<thead>
<tr>
<th>e.g. A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>50x10</td>
<td>10x40</td>
<td>40x30</td>
<td>30x5</td>
</tr>
</tbody>
</table>

\[(A(BC))D : 34,500\]

Step 1: A

\[(AB)C : 50\times40\times30\]

Step 2: (AB)C

\[A(BC)D : 50\times30\times5\]

Step 3: (A(BC))D

\[P \times Q \times R : \# \text{ of multiplications}\]