Memoizing

Dynamic Programming in a recursive (top-down) framework

- Whenever you compute the value of a subproblem
  \( f \) becomes \( mf \) plus MEMO table

  \[ mf(x) \]

  \( \text{if } \) MEMO\( [x] \) defined \( \text{then } \{ \)
  \( \text{else } \) MEMO\( [x] \leftarrow f(x) \)
  \( \text{return } (\text{MEMO}[x]) \)

- Replace all calls to \( f \) by calls to \( mf \)
  - includes recursive calls to \( f \) in the body of \( f \)
  - but not the call in \( mf \).

Greedy Algorithm

A series of locally optimum choices produces a globally optimum solution

What is needed:

- Greedy choice property — of all the optimal solutions, at least one contains the initial "greedy" choice.

- Optimal substructure — as with dynamic programming, optimal solutions contain optimal solutions to subproblems.
Greedy Example

Activity Selection
Have a set of possible activities to do
What is the most non-conflicting events I can attend.

Have start and finish times
Order by increasing end time

LIST  A  B  C  D  E  F  G
     2-5 1-6 3-6 6-8 5-9 10-11 10-12

Sched = { A, D, F }

Schedule Algorithm

Sched \leftarrow \emptyset

\textbf{for each ACT in LIST do}
\hspace{1cm} \textbf{if} ACT compatible with Sched
\hspace{1cm} \textbf{then} Sched \leftarrow Sched \cup \{ ACT \}

Complexity: \( O(\text{sort LIST}) \)

Correct?
Let \( \text{LIST} = A_1 A_2 \cdots A_n \)
Greedy Choice

- Suppose \( Sched \) is a maximal legal schedule that doesn't include \( A_1 \) (an activity with earliest finish).
- Let \( A_1 \) be the activity in \( Sched \) with earliest finish.
- Then \( A_1.\text{finish} \geq A_i.\text{finish} \).

So \( Sched' = Sched - \{ A_i \} \cup \{ A_1 \} \) is also a maximum legal schedule.

Optimal Substructure

- Assume \( Sched \) is maximal legal schedule for \( A_1 \ldots A_n \) that contains \( A_1 \).
- Let \( A_1 \) be the first activity with \( A_i.\text{start} \geq A_1.\text{finish} \).
- Then \( Sched - \{ A_1 \} \) is a maximal legal schedule for \( A_i \ldots A_n \).
  - must be legal
  - if \( Sched' \) were a better (larger)
    schedule for \( A_i \ldots A_n \), then
    \( Sched' \cup \{ A_1 \} \) is a better schedule
    to \( A_i \ldots A_n \).
Huffman Codes

◊ A   E   H   N   R   S   T
000 001 010 011 100 101 110 111

THEOREST

Frequencies

◊ A   E   H   N   R   S   T
 7   3   4   3   1   2   2   6
 00 100 01 1010 1010 1011 11

Contribution
14 9   8   12   6   10 12 12 \overline{Z} = 83

Prefix Code
No code word is a prefix of another

11101001001011010011

Every interior node has two children
Greedy Algorithm for Code Generation

Start with \( n \) separate 1-character codes

Merge two codes with lowest frequencies

Frequency of merged code is \( f_1 + f_2 \)

Until there is only one tree.
Huffman Code Example 2

```
7
◊
A S
```

Huffman Code Example 3

```
7
◊
E A S
```

```
12
◊
H N R
```

David Maier
Huffman Code Example 4

Huffman Code Example 5

Why does it work?
Greedy Choice

- Greedy choice
  \[ s_1, s_2 \text{ lowest frequency chains.} \]

\[
\begin{align*}
    & \quad d_1 - \text{depth of } s_1 \\
    & \quad d_2 - \text{depth of } s_2 \\
    & \quad d_{\text{max}} - \text{depth } s_3, s_4
\end{align*}
\]

Greedy Choice 2

\[
f_3 \cdot d_{\text{max}} + f_4 \cdot d_{\text{max}} + f_1 \cdot d_1 + f_2 \cdot d_2
\]

\[
f_1 \cdot d_{\text{max}} + f_2 \cdot d_{\text{max}} + f_3 \cdot d_1 + f_4 \cdot d_2
\]

\[
s_1 - s_3 \text{ switch:} \\
\text{old} \quad (f_3 \cdot d_{\text{max}} + f_1 \cdot d_1) \\
\text{new} \quad -(f_1 \cdot d_{\text{max}} + f_3 \cdot d_1) \\
\text{subtract} \quad \frac{(f_3 - f_1) \cdot d_{\text{max}} + (f_1 - f_3) \cdot d_1}{(f_3 - f_1) \cdot d_{\text{max}} - (f_1 - f_3) \cdot d_1} \\
\quad \geq 0 \quad > 0
\]
Optimal Substructure

Let $S$ be the set of symbols optimal for $S$ and $S'$.

Let $T$ be optimal for $S$ and $T'$ be optimal for $S'$.

For $S = S_1, S_2, S_3$,

$$
S' = S_1, S_2, S_3$$

$$
T' = f_1 + f_2
$$

$$
cost(T) = cost(T') + \frac{f_1}{f_2}
$$

A lower cost tree for $T''$ for $S'$ would give a lower-cost tree for $T$. 