Greedy Algorithms
Ch. 16
This Course

• **Sorting**
  – A necessary tool for managing and navigating a variety of datasets.
  – A great context in which to learn computer science
    • Analyze Complexity
    • Prove Correctness

• **Data Structures**
  – Explore variety of structures beyond arrays and linked lists.
  – How you store data affects how easily you can manage and navigate it.

• **Optimization Problems**
  – Learn sophisticated methods that are alternates to brute force approach.

• **Graph Problems**
  – Applicable to wide range of problems (many seemingly unrelated problems map to a graph problem).
  – Another great context in which to learn computer science.
Decision Space
Maximization Problem
(The solution is the one with the highest possible value.)

Greedy
best local part of best global

all viable, equally good solutions
Greedy Approach

• Addresses exponential running time of brute force approach.
• Greedy provides solution BUT not always optimal.
• Proof of Correctness required to prove solution is optimal.
  – Optimal Substructure
  – Greedy Choice Property
• Dynamic Programming can be used to solve these, but far less efficiently.
• Can be hard to distinguish problems solvable with Greedy Method versus those that require Dynamic Programming.
Knapsack Problem: Resource Allocation

\[ E = \text{set of elements} \{ e_1, e_2, \ldots, e_n \} \]
\[ \text{each element } e_i \text{ has weight } w_i \]
\[ \text{each element } e_i \text{ has value } v_i \]

Fill a knapsack with a subset of \( E \) such that value is maximal and weight is \( \leq W \) (weight capacity)

0/1 Knapsack
Take an item (0) or don’t take an item (1)
Solvable with Dynamic Programming

Fractional Knapsack
Take any proportion (fraction) of an item
Solvable with Greedy Algorithm
Knapsack Problem: Resource Allocation

0/1 Knapsack: take the item or don’t take it
m[i,j] = maximum value of set E_i with capacity j

Global Problem: m[n,W]
m[i,j] with choice k = m[i-1, j - k*w_i] + k*v_i
(k = 0 or 1 for take, don’t take item e_i, respectively)

E = set of elements \{ e_1, e_2, ..., e_n \}
E_i = subset of E \{ e_1, e_2, ..., e_i \}
each element e_i has weight w_i
each element e_i has value v_i

Fill a knapsack with a subset of E such that
value is maximal and weight is \leq W (weight capacity)
Knapsack Problem: Resource Allocation

Fractional Knapsack: take a fraction of the item

\[ S = \text{sorted set of elements } \{ e_1, e_2, \ldots, e_n \} \]

such that, for all \( i < j \), \( \frac{v_i}{w_i} \leq \frac{v_j}{w_j} \)

choice \( k = \) weight taken of item \( e_i \)

\[ k = \text{MIN}( j, w_i ) \]

\[ m[i, j] = m[i-1, j-k] + ( \frac{v_i}{w_i} )*k \]

ONLY 1 CHOICE, LEAVES 1 SUBPROBLEM

\[ E = \text{set of elements } \{ e_1, e_2, \ldots, e_n \} \]

each element \( e_i \) has weight \( w_i \)

each element \( e_i \) has value \( v_i \)

Fill a knapsack with a subset of \( E \) such that

value is maximal and weight is \( \leq W \) (weight capacity)
Greedy Strategy Elements

• Show Optimal Substructure

• Define Recursive Solution

• Show Greedy Choice Property
  – Greedy Choice is always part of Globally Optimal Solution
  – Greedy Choice leaves 1 Subproblem

• Create Recursive Algorithm

• Convert Recursive to Iterative
### Activity-Selection: Resource Allocation

<table>
<thead>
<tr>
<th>$i$</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>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

$A$ = ordered set of activities $\{a_1, a_2, ..., a_n\}$
- each activity $a_i$ has start time $s_i$
- each activity $a_i$ has finish time $f_i$

Choose a maximally sized subset of compatible problems.
**Compatible** means that activities are non-overlapping,
such that for all $i \neq j$, $s_j \geq f_i$ OR $s_i \geq f_j$

$m[i, j] =$ maximally sized subset of $A$ within time interval $\{i..j\}$
(only include activities $a_k$ with $s_k \geq i$ and $f_k \leq j$)

$m[i, j]$ with choice $k = m[i, s_k] + m[f_k, j] + 1$
choice $k$ is to include activity $a_k$
<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>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Minimum Time
<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>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Latest Start Time
### Earliest Finish Time

<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>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

The table represents the earliest finish time for a set of activities labeled from 0 to 13.
Activity-Selection Problem

Maximize Number of Tasks

\[
A = \{ a_1, a_2, \ldots, a_n \}
\]

ordered such that for all \( i \neq j \), \( f_i \leq s_j \)

(i.e. earliest finish time)

\( S_{ij} \) = subset of activities that start after \( f_i \) and finish before \( s_j \)

\( c[i,j] \) = maximally-sized subset of compatible activities in \( S_{ij} \)

- Optimal Substructure

\[
c[i, j] = \begin{cases} 
0 & \text{if } S_{ij} = \emptyset, \\
\max_{a_k \in S_{ij}} \{ c[i, k] + c[k, j] + 1 \} & \text{if } S_{ij} \neq \emptyset.
\end{cases}
\]
Activity-Selection Problem and Greedy Choice Property

Maximize Number of Tasks

\[ A = \{ a_1, a_2, \ldots, a_n \} \]
ordered such that for all \( i \neq j \), \( f_i \leq s_j \)
(i.e. earliest finish time)

\[ S_{ij} = \text{subset of activities that start after } f_i \text{ and finish before } s_j \]
\[ c[i,j] = \text{maximally-sized subset of compatible activities in } S_{ij} \]

\[
c[i, j] = \begin{cases} 
0 & \text{if } S_{ij} = \emptyset, \\
\max_{a_k \in S_{ij}} \{ c[i, k] + c[k, j] + 1 \} & \text{if } S_{ij} \neq \emptyset. 
\end{cases}
\]

consider \( k = \text{activity with earliest finish time.} \)

leaves empty subproblem.

\( a_k \) part of a global optimal solution
Theorem 16.1

3rd Ed Version

Consider any nonempty subproblem $S_k$, and let $a_m$ be an activity in $S_k$ with the earliest finish time. Then $a_m$ is included in some maximum-size subset of mutually compatible activities of $S_k$.

2nd Ed Version

Consider any nonempty subproblem $S_{ij}$, and let $a_m$ be the activity in $S_{ij}$ with the earliest finish time:

$$f_m = \min \{ f_k : a_k \in S_{ij} \}$$

1. Activity $a_m$ is used in some maximum-size subset of mutually compatible activities of $S_{ij}$.

2. Subproblem $S_{im}$ is empty, so that choosing $a_m$ leaves the subproblem $S_{mk}$ as the only one that may be nonempty.
Greedy Strategy Elements

• Show Optimal Substructure
• Define Recursive Solution

• Show Greedy Choice Property
  – Greedy Choice is always part of Globally Optimal Solution
  – Greedy Choice leaves 1 Subproblem

• Create Recursive Algorithm

• Convert Recursive to Iterative
Recursive Activity Selection Algorithm

• Define and Construct Recursively

FIRST CALL:
RECURSIVE-ACTIVITY-SELECTOR( s, f, 0, n )

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]  // find the first activity in S_k to finish
3     m = m + 1
4  if m ≤ n
5    return \{a_m\} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6 else return ∅
```
Iterative Activity Selection Algorithm

• Reconstruct as Iterative

**NEED SORTING ALGORITHM**

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1 n = s.length
2 A = {a₁}
3 k = 1
4 for m = 2 to n
5   if s[m] ≥ f[k].
6     A = A ∪ {aₘ}
7     k = m
8 return A
```

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1 m = k + 1
2 while m ≤ n and s[m] < f[k]  // find the first activity in Sₖ to finish
3   m = m + 1
4 if m ≤ n
5   return {aₘ} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6 else return ∅
```
Prefix Code
no codeword is also a prefix of some other codeword
(i.e. the initial word is unambiguous)

Huffman Code
An optimal prefix code generated using greedy algorithm.

\[ B(T) = \sum_{c \in C} c \cdot \text{freq} \cdot d_T(c) \]
Greedy Strategy Elements

• Show Optimal Substructure

• Define Recursive Solution

• Show Greedy Choice Property
  – Greedy Choice is always part of Globally Optimal Solution
  – Greedy Choice leaves 1 Subproblem

• Create Recursive Algorithm

• Convert Recursive to Iterative
Huffman Code

An optimal prefix code generated using greedy algorithm.

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n - 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT (Q, z)
9 return EXTRACT-MIN(Q)  // return the root of the tree
```
Let \( C \) be an alphabet in which each character \( c \) in \( C \) has frequency \( c.\text{freq} \).

Let \( x \) and \( y \) be two characters in \( C \) having the lowest frequencies.

Then there exists an optimal prefix code for \( C \) in which the codewords for \( x \) and \( y \) have the same length and differ only in the last bit.

**Lemma 16.2**

Let \( C \) be an alphabet in which each character \( c \) in \( C \) has frequency \( c.\text{freq} \).

Let \( x \) and \( y \) be two characters in \( C \) having the lowest frequencies.

Then there exists an optimal prefix code for \( C \) in which the codewords for \( x \) and \( y \) have the same length and differ only in the last bit.
Lemma 16.3
Let $C$ be a given alphabet with frequency $c.freq$ defined for each character $c$ in $C$.

Let $x$ and $y$ be two characters in $C$ with minimum frequency.

Let $C'$ be the alphabet $C$ with the characters $x$ and $y$ removed and a new character $z$ added so that $C' = C - \{x,y\} \cup \{z\}$.

Define $freq$ for $C'$ as for $C$, except that $z.freq = s.freq + y.freq$.

Let $T'$ be any tree representing an optimal prefix code for the alphabet $C'$.

Then the tree $T$, obtained from $T'$ by replacing the leaf node for $z$ with an internal node having $x$ and $y$ as children, represents an optimal prefix code for the alphabet $C$. 