**Greed**

"Greed is good. Greed is right. Greed works. Greed cuts through, clarifies, and captures the essence of the evolutionary spirit."

Gordon Gecko
(Michael Douglas)

---

**Greedy Algorithms**

Some possibly familiar examples:
- Gale-Shapley stable matching algorithm.
- Dijkstra’s shortest path algorithm.
- Prim and Kruskal MST algorithms.
- Huffman codes.
- . . .

---

**Selecting Breakpoints**

Minimizing breakpoints.
- Truck driver going from Princeton to Palo Alto along predetermined route.
- Refueling stations at certain points along the way.
- Truck fuel capacity = C.

Greedy algorithm.
- Go as far as you can before refueling.

---

**Selecting Breakpoints: Greedy Algorithm**

**Greedy Breakpoint Selection Algorithm**

Sort breakpoints by increasing value:

\[ 0 = b_0 < b_1 < b_2 < \ldots < b_n. \]

1. \( S \leftarrow \{0\} \)
2. \( x = 0 \)
3. While \((x \neq b_n)\) do:
   a. Let \( p \) be largest integer such that \( b_p \leq x + C \)
   b. If \( b_p = x \)
      i. Return "no solution"
   c. \( x \leftarrow b_p \)
   d. \( S \leftarrow S \cup \{p\} \)
4. Return \( S \)

Princeton - C - C - C - C - C - C - C - Palo Alto
Selecting Breakpoints

Theorem: greedy algorithm is optimal.

Proof (by contradiction):
- Let $0 = g_0 < g_1 < \ldots < g_p = L$ denote set of breakpoints chosen by greedy and assume it is not optimal.
- Let $0 = f_0 < f_1 < \ldots < f_q = L$ denote set of breakpoints in optimal solution with $f_0 = g_0$, $f_1 = g_1$, \ldots, $f_r = g_r$ for largest possible value of $r$.
- Note: $q < p$.

Greedy: $g_0 \quad g_1 \quad g_2 \quad g_r \quad 6 \quad 7 \quad 8 \quad 9$

OPT: $f_0 \quad f_1 \quad f_2 \quad f_r \quad f_q$

$r = 4$

Activity Selection

Activity selection problem (CLR 17.1).
- Job requests 1, 2, \ldots, n.
- Job j starts at $s_j$ and finishes at $f_j$.
- Two jobs compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.
Activity Selection: Greedy Algorithm

Greedy Activity Selection Algorithm

Sort jobs by increasing finish times so that 
\( f_1 \leq f_2 \leq \ldots \leq f_n \).

\[ S = \emptyset \]

for \( j = 1 \) to \( n \)

if (job \( j \) compatible with \( A \))

\[ S \leftarrow S \cup \{j\} \]

return \( S \)

Activity Selection

Theorem: greedy algorithm is optimal.

Proof (by contradiction):

- Let \( g_1, g_2, \ldots, g_p \) denote set of jobs selected by greedy and assume it is not optimal.
- Let \( f_1, f_2, \ldots, f_q \) denote set of jobs selected by optimal solution with 
  \( f_1 = g_1, f_2 = g_2, \ldots, f_r = g_r \) for largest possible value of \( r \).
- Note: \( r < q \).

Greedy: 

\[
\begin{array}{cccc}
1 & 5 & 8 & 9 \\
\end{array}
\]

\( f_1 = g_1, f_2 = g_2, f_3 = g_3 \)

OPT: 

\[
\begin{array}{cccc}
1 & 5 & 8 & 9 \\
\end{array}
\]

\( r = 3 \)

\( q = 7 \)

Replace 11 with 9
Activity Selection

Theorem: greedy algorithm is optimal.

Proof (by contradiction):
- Let \( g_1, g_2, \ldots, g_p \) denote set of jobs selected by greedy and assume it is not optimal.
- Let \( f_1, f_2, \ldots, f_q \) denote set of jobs selected by optimal solution with \( f_1 = g_1, f_2 = g_2, \ldots, f_r = g_r \) for largest possible value of \( r \).
- Note: \( r < q \).

<table>
<thead>
<tr>
<th>Greedy:</th>
<th>1</th>
<th>5</th>
<th>8</th>
<th>9</th>
<th>13</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT:</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>15</td>
<td>17</td>
</tr>
</tbody>
</table>

p = 6
r = 4
q = 7

Making Change

Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

- Ex. 34¢.

Greedy algorithm.
- At each iteration, add coin of the largest value that does not take us past the amount to be paid.
- Ex. $2.89.

Coin-Changing: Greedy Algorithm

Sort coins denominations by increasing value: \( c_1 < c_2 < \ldots < c_n \).

\[
\text{Greedy Coin-Changing Algorithm}
\]

Sort coins denominations by increasing value: \( c_1 < c_2 < \ldots < c_n \).

\[
\text{Greedy Coin-Changing Algorithm}
\]

S = coins selected.

while (\( x \neq 0 \))
- let \( p \) be largest integer such that \( c_p \leq x \)
  if \( p = 0 \)
    return "no solution found"
- \( x \leftarrow x - c_p \)
- \( S \leftarrow S \cup \{p\} \)
return \( S \)

Is Greedy Optimal for Coin-Changing Problem?

Yes, for U.S. coinage: \( \{c_1, c_2, c_3, c_4, c_5\} = \{1, 5, 10, 25, 100\} \).

Ad hoc proof.
- Consider optimal way to change amount \( c_k \leq x < c_{k+1} \).
- Greedy takes coin \( k \).
- Suppose optimal solution does not take coin \( k \).
  - \( n \) must take enough coins of type \( c_1, c_2, \ldots, c_{k-1} \) to add up to \( x \).

<table>
<thead>
<tr>
<th>( k )</th>
<th>( c_k )</th>
<th>Max # taken by optimal solution</th>
<th>Max value of coins ( 1, 2, \ldots, k ) in any OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 1 )</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>( 5 )</td>
<td>1</td>
<td>( 4 + 5 = 9 )</td>
</tr>
<tr>
<td>3</td>
<td>( 10 )</td>
<td>2</td>
<td>( 20 + 4 = 24 )</td>
</tr>
<tr>
<td>4</td>
<td>( 25 )</td>
<td>3</td>
<td>( 75 + 24 = 99 )</td>
</tr>
<tr>
<td>5</td>
<td>( 100 )</td>
<td>no limit</td>
<td>no limit</td>
</tr>
</tbody>
</table>

2 dimes ⇒ no nickels
Does Greedy Always Work?

US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.
- Ex. 140¢.
- Greedy: 100, 34, 1, 1, 1, 1, 1, 1.
- Optimal: 70, 70.

Characteristics of Greedy Algorithms

Greedy choice property.
- Globally optimal solution can be arrived at by making locally optimal (greedy) choice.
- At each step, choose most “promising” candidate, without worrying whether it will prove to be a sound decision in long run.

Optimal substructure property.
- Optimal solution to the problem contains optimal solutions to subproblems.
  - if best way to change 34¢ is \{25, 5, 1, 1, 1\} then best way to change 29¢ is \{25, 1, 1, 1, 1\}.

Objective function does not explicitly appear in greedy algorithm!

Hard, if not impossible, to precisely define "greedy algorithm."
- See matroids (CLR 17.4), greedoids for very general frameworks.

Minimizing Lateness

Minimizing lateness problem.
- Single resource can process one job at a time.
- n jobs to be processed.
  - job j requires \( p_j \) units of processing time.
  - job j has due date \( d_j \).
- If we assign job j to start at time \( s_j \), it finishes at time \( f_j = s_j + p_j \).
- Lateness: \( l_j = \max \{0, f_j - d_j\} \).
- Goal: schedule all jobs to minimize maximum lateness \( L = \max l_j \).

Minimizing Lateness: Greedy Algorithm

**Greedy Activity Selection Algorithm**

Sort jobs by increasing deadline so that \( d_1 \leq d_2 \leq \ldots \leq d_n \).

\[
t = 0 \\
t_j = t + p_j \\
\text{output intervals } [s_j, f_j]
\]

max lateness = 2
Minimizing Lateness: No Idle Time

Fact 1: there exists an optimal schedule with no idle time.

Fact 2: the greedy schedule has no idle time.

Minimizing Lateness: Inversions

An inversion in schedule S is a pair of jobs i and j such that:
- i < j
- j scheduled before i

Fact 3: greedy schedule \( \Rightarrow \) no inversions.

Fact 4: if a schedule (with no idle time) has an inversion, it has one whose with a pair of inverted jobs scheduled consecutively.

Minimizing Lateness: Proof of Fact 5

An inversion in schedule S is a pair of jobs i and j such that:
- i < j
- j scheduled before i

Swapping two adjacent, inverted jobs does not increase max lateness.
- \( \ell'_{i} = \ell_{k} \) for all \( k \neq i, j \)
- \( \ell'_{i} \leq \ell_{i} \)
- If job j is late:
  \[
  \ell'_{j} = f'_{j} - d_{j} \quad \text{(definition)} \]
  \[
  = f_{i} - d_{j} \quad \text{(j finishes at time } f_{i}) \]
  \[
  \leq f_{i} - d_{i} \quad (i < j) \]
  \[
  \leq \ell_{i} \quad \text{(definition)}
  \]

Theorem: greedy schedule is optimal.