Greedy Algorithm

Ch 16
Hewett

Greedy Algorithm

• Do not always yield optimal solutions, but for many problem they do

Activity selection problem
  Given \( S \), a set of \( n \) activities.
  Let \( s_i \) and \( f_i \) be start and finish times of activity \( i \), respectively.
  Two activities are \textit{compatible} if they do no overlap.
  Find max-size subset \( A \) of compatible activities
• Consider the following $S$. Assume activities are sorted by finish time i.e., $f_1 \leq f_2 \leq \ldots \leq f_n$

\[
\begin{array}{ccc}
1 & \rightarrow & 2 \\
3 & \rightarrow & 4 \\
& \rightarrow & 5 \\
6
\end{array}
\]

• **Optimal substructure**: Let $A$ be opt. sol to $S$ and $A$ begins with activity $k$. Then $A - \{k\}$ must be opt. sol $S' = \{ i \in S : s_i \geq f_k \}$

\[
\begin{array}{cc}
\text{Reason:} & \text{Otherwise,} \exists B \text{ as opt. sol to } S' \text{ s.t. } |B| > |A - \{k\}|.
\end{array}
\]

Since $B$ is a sol of $S' \Rightarrow$ all activities in $B$ are compatible and occur after $f_k$ (thus, no overlap with $k$) $\Rightarrow B \cup \{k\}$ is compatible with larger size than $A$.

Thus, $B \cup \{k\}$ is opt sol of $S$. Contradiction to “$A$ is opt sol to $S$”!

• **Repeated sub-problems**

\[
\begin{array}{ccc}
1 & \rightarrow & 2 \\
3 & \rightarrow & 4 \\
& \rightarrow & 5 \\
6
\end{array}
\]

$S' = \{ i \in S : s_i \geq f_1 \} = \{2, 5, 6\}$

$S'' = \{ i \in S : s_i \geq f_2 \} = \{5, 6\}$

If $2 \in A$ the next sub-problem will have all activities start after $f_2$, i.e., $S''$ otherwise, the next sub-problem will be all activities in the old sub-problem except 2
• Optimal substructure + Repeated sub-problems --> DP
  But there is another method .... Greedy algorithms

• **Greedy choice Property:**
  Local optimal (greedy) choice ==> globally optimal solution
  (local optimal choice does not guarantee global optimal solution)

• For the **activity selection problem:**
  Greedy choice: pick the next compatible activity that has earliest finish time
  I.e., if we assume activities are sorted by their finish times, always pick the first compatible activity found

---

**Example:**

Greedy Algorithm

---

compatible with last activity selected
**Theorem:** Let \( S \) be an activity selection problem and suppose activities are sorted by finish time. Then there exists an optimal solution \( A \subseteq S = \{1, 2, \ldots, n\} \) s.t. \( 1 \notin A \).

**Proof Idea:** Let \( A \) be an optimal solution to \( S \) and \( A \) begins with activity \( k \).

- If \( k = 1 \) then done.
- If \( k \neq 1 \), consider \( B = A - \{k\} \cup \{1\} \).

All activities in \( B \) are compatible (since \( f_1 \leq s_k \) and all activities in \( A - \{k\} \), which are compatible). Also \( |B| = |A - \{k\}| + 1 = |A| \).

Thus, \( B \) is also an optimal solution of \( S \) that starts with activity 1.

For **activity selection problem** - we can show that a greedy choice at each step yields a global optimal solution (by mathematical induction).

---

**Pseudocode**

```
greedy-activity-selector \((s, f)\); \(s\) and \(f\) arrays of start and finish times
1) \(n \leftarrow \text{length}[s]\)
2) \(A \leftarrow \{1\}\); from the theorem
3) \(j \leftarrow 1\)
4) for \(i \leftarrow 2\) to \(n\); activities are ordered by finish time
5) do if \(s_i \geq f_j\); first activity, \(i\) that is compatible with \( j \)
6) then \(A \leftarrow A \cup \{i\}\)
7) \(j \leftarrow i\); keep track of last activity added
8) return \(A\)
```

Analysis: 1)-3, 8) \(O(1)\)

4)-7) \(O(n)\)

Total: linear time complexity
Some characteristics

**Greedy Strategy**
- Topdown
  - Make greedy choice to reduce problem size
- Repeated sub-problems
- Does not guarantee optimal solutions

**Dynamic Programming**
- Bottom up
  - Recursively solve sub-problem
- Overlapping sub-problems
- Does find optimal solutions

**Greedy vs. Dynamic Programming**

**0-1 Knapsack problem**
- A thief robbing a store finds $n$ items, item $i$ worth $v_i$ $\$$ and weighs $w_i$ lbs ($v_i$ and $w_i$ are integers).
- What items should he take to get most valuable load given his knapsack can carry at most $w$ lbs?

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
<th>value</th>
<th>value/weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>item1</td>
<td>10</td>
<td>60</td>
<td>6</td>
</tr>
<tr>
<td>item2</td>
<td>20</td>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td>item3</td>
<td>30</td>
<td>120</td>
<td>4</td>
</tr>
</tbody>
</table>

$w = 50$ lbs

**Greedy choice:** highest value/weight first

1. pick item 1 --> remain weights (50-10) = 40
2. pick item 2 --> remain weight (40-20) = 20

**Greedy solution:** items 1 then 2 giving $160$ in the knapsack

**Optimal solution:** items 2 then 3 giving $220$

**Note:** 0-1 Knapsack problem has dynamic programming elements
Greedy vs. Dynamic Programming

Fractional Knapsack problem

- Same as 0-1 Knapsack except that the thief can take fractions of items instead of binary.

<table>
<thead>
<tr>
<th></th>
<th>weight</th>
<th>value</th>
<th>value/weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>item1</td>
<td>10</td>
<td>60</td>
<td>6</td>
</tr>
<tr>
<td>item2</td>
<td>20</td>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td>item3</td>
<td>30</td>
<td>120</td>
<td>4</td>
</tr>
</tbody>
</table>

\[ w = 50 \text{ lbs} \]

**Greedy choice:** highest value/weight first

1. pick item 1 --> remain weights (50-10) = 40 \[ \$60 \]
2. pick item 2 --> remain weight (40-20) = 20 \[ \$100 \]
3. pick 2/3 of item 3 --> remain 0 lbs \[ \$80 \]

**Greedy solution:** gives a total of \$240. This is an optimal solution. Thus, greedy alg is simpler than DP but may or may not give optimal solutions.