Greedy Algorithms

- Solve problems with the simplest possible algorithm
- The hard part: showing that something simple actually works
- Pseudo-definition
  - An algorithm is Greedy if it builds its solution by adding elements one at a time using a simple rule

CSE 421
Algorithms
Richard Anderson
Lecture 7
Greedy Algorithms

Scheduling Theory

- Tasks
  - Processing requirements, release times, deadlines
- Processors
- Precedence constraints
- Objective function
  - Jobs scheduled, lateness, total execution time

Interval Scheduling

- Tasks occur at fixed times
- Single processor
- Maximize number of tasks completed
- Tasks \{1, 2, \ldots N\}
- Start and finish times, s(i), f(i)

What is the largest solution?

Greedy Algorithm for Scheduling

Let $T$ be the set of tasks, construct a set of independent tasks $I$, $A$ is the rule determining the greedy algorithm

$I = \{\}$
While ($T$ is not empty)

Select a task $t$ from $T$ by a rule $A$
Add $t$ to $I$
Remove $t$ and all tasks incompatible with $t$ from $T$
Simulate the greedy algorithm for each of these heuristics:

Schedule earliest starting task

Schedule shortest available task

Schedule task with fewest conflicting tasks

Greedy solution based on earliest finishing time

Example 1

Example 2

Example 3

Theorem: Earliest Finish Algorithm is Optimal
- Key idea: Earliest Finish Algorithm stays ahead
- Let \( A = \{i_1, \ldots, i_k\} \) be the set of tasks found by EFA in increasing order of finish times
- Let \( B = \{j_1, \ldots, j_m\} \) be the set of tasks found by a different algorithm in increasing order of finish times
- Show that for \( r \leq \min(k, m) \), \( f(i_r) \leq f(j_r) \)

Stay ahead lemma
- \( A \) always stays ahead of \( B \), \( f(i) \leq f(j) \)
- Induction argument
  - \( f(i_1) \leq f(j_1) \)
  - If \( f(i_{r-1}) \leq f(j_{r-1}) \) then \( f(i_r) \leq f(j_r) \)

Completing the proof
- Let \( A = \{i_1, \ldots, i_k\} \) be the set of tasks found by EFA in increasing order of finish times
- Let \( O = \{j_1, \ldots, j_m\} \) be the set of tasks found by an optimal algorithm in increasing order of finish times
- If \( k < m \), then the Earliest Finish Algorithm stopped before it ran out of tasks

Scheduling all intervals
- Minimize number of processors to schedule all intervals
How many processors are needed for this example?

---

Prove that you cannot schedule this set of intervals with two processors

---

Depth: maximum number of intervals active

---

Algorithm

- Sort by start times
- Suppose maximum depth is $d$, create $d$ slots
- Schedule items in increasing order, assign each item to an open slot
- Correctness proof: When we reach an item, we always have an open slot

---

Scheduling tasks

- Each task has a length $t_i$ and a deadline $d_i$
- All tasks are available at the start
- One task may be worked on at a time
- All tasks must be completed
- Goal minimize maximum lateness
  - Lateness = $t_i - d_i$ if $t_i >= d_i$

---

Example

<table>
<thead>
<tr>
<th>Time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Lateness 1

<table>
<thead>
<tr>
<th>Time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Lateness 3

<table>
<thead>
<tr>
<th>Time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Lateness 3
Determine the minimum lateness

<table>
<thead>
<tr>
<th>Time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
</tbody>
</table>

To be continued . . .