Announcements

• Reading
  – For today, sections 4.1, 4.2, 4.4
  – For next week, sections 4.5, 4.7, 4.8
• Homework 3 is available

Greedy Algorithms

• Solve problems with the simplest possible algorithm
• The hard part: showing that something simple actually works
• Today’s problems (Sections 4.2, 4.3)
  – Homework Scheduling
  – Optimal Caching
  – Subsequence testing

Highlights from Last Lecture

• Interval scheduling
  – Earliest Deadline First
  – Correctness proof: Stay ahead lemma
• Multiprocessor schedule
  – Available processor algorithm
  – Can always schedule with \(d\) processors, where \(d\) is the maximum number of intervals active at any time.

Homework Scheduling

• Tasks to perform
• Deadlines on the tasks
• Freedom to schedule tasks in any order
• Can I get all my work turned in on time?
• If I can’t get everything in, I want to minimize the maximum lateness

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 = \(f_i - d_i\) if \(f_i \geq d_i\)
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>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Lateness 1

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>

Greedy Algorithm

- Earliest deadline first
- Order jobs by deadline
- This algorithm is optimal

Analysis

- Suppose the jobs are ordered by deadlines, \( d_1 \leq d_2 \leq \ldots \leq d_n \)
- A schedule has an inversion if job \( j \) is scheduled before \( i \) where \( j > i \)
- The schedule \( A \) computed by the greedy algorithm has no inversions.
- Let \( O \) be the optimal schedule, we want to show that \( A \) has the same maximum lateness as \( O \)

List the inversions

<table>
<thead>
<tr>
<th>Time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>3</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>4</td>
</tr>
<tr>
<td>( a_3 )</td>
<td>2</td>
</tr>
<tr>
<td>( a_4 )</td>
<td>5</td>
</tr>
<tr>
<td>( a_5 )</td>
<td>12</td>
</tr>
</tbody>
</table>

Lemma: There is an optimal schedule with no idle time

- It doesn’t hurt to start your homework early!
- Note on proof techniques
  - This type of can be important for keeping proofs clean
  - It allows us to make a simplifying assumption for the remainder of the proof
Lemma

• If there is an inversion $i, j$, there is a pair of adjacent jobs $i', j'$ which form an inversion

Interchange argument

• Suppose there is a pair of jobs $i$ and $j$, with $d_i \leq d_j$, and $j$ scheduled immediately before $i$. Interchanging $i$ and $j$ does not increase the maximum lateness.

Proof by Bubble Sort

Determine maximum lateness

Real Proof

• There is an optimal schedule with no inversions and no idle time.
• Let $O$ be an optimal schedule $k$ inversions, we construct a new optimal schedule with $k-1$ inversions
• Repeat until we have an optimal schedule with 0 inversions
• This is the solution found by the earliest deadline first algorithm

Result

• Earliest Deadline First algorithm constructs a schedule that minimizes the maximum lateness

Homework Scheduling

• How is the model unrealistic?
Extensions

- What if the objective is to minimize the sum of the lateness?
  - EDF does not work
- If the tasks have release times and deadlines, and are non-preemptable, the problem is NP-complete
- What about the case with release times and deadlines where tasks are preemptable?

Optimal Caching

- Caching problem:
  - Maintain collection of items in local memory
  - Minimize number of items fetched

Caching example

A, B, C, D, A, E, B, A, D, A, C, B, D, A

Optimal Caching

- If you know the sequence of requests, what is the optimal replacement pattern?
- Note – it is rare to know what the requests are in advance – but we still might want to do this:
  - Some specific applications, the sequence is known
    - Register allocation in code generation
  - Competitive analysis, compare performance on an online algorithm with an optimal offline algorithm

Farthest in the future algorithm

- Discard element used farthest in the future

Correctness Proof

- Sketch
- Start with Optimal Solution O
- Convert to Farthest in the Future Solution F-F
- Look at the first place where they differ
- Convert O to evict F-F element
  - There are some technicalities here to ensure the caches have the same configuration . . .
Subsequence Testing

• Is $a_1a_2...a_m$ a subsequence of $b_1b_2...b_n$?
  – e.g. S,A,G,E is a subsequence of S,T,U,A,R,T,R,E,G,E,S

Greedy Algorithm for Subsequence Testing

Next week