Announcements

• Reading
  – For today, sections 4.1, 4.2, 4.4
  – For January 28, sections 4.5, 4.7, 4.8 (Plus additional material from chapter 5)

• No class January 21

• Homework 2 is due January 21
Highlights from last lecture

- Algorithm runtime
  - Runtime as a function of problem size
  - Asymptotic analysis, (Big Oh notation)

- Graph theory
  - Basic terminology
  - Graph search and breadth first search
  - Two coloring
  - Connectivity
  - Topological search
Greedy Algorithms
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
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, $f(i_r) \leq f(j_r)$

• 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 = $f_i - d_i$ if $f_i \geq d_i$
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>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>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 seem to 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?
Subsequence Testing

Is $a_1 a_2 \ldots a_m$ a subsequence of $b_1 b_2 \ldots b_n$?

E.g. A,B,C,D,A is a subsequence of B,A,B,A,B,D,B,C,A,D,A,C,D,A
Greedy Algorithm for Subsequence Testing
Shortest Paths
Single Source Shortest Path Problem

• Given a graph and a start vertex s
  – Determine distance of every vertex from s
  – Identify shortest paths to each vertex
    • Express concisely as a “shortest paths tree”
    • Each vertex has a pointer to a predecessor on shortest path
Construct Shortest Path Tree from s
Warmup

• If P is a shortest path from s to v, and if t is on the path P, the segment from s to t is a shortest path between s and t

• WHY?
Dijkstra’s Algorithm

S = {}; d[s] = 0; d[v] = infinity for v != s

While S != V

Choose v in V-S with minimum d[v]

Add v to S

For each w in the neighborhood of v

\[ d[w] = \min(d[w], d[v] + c(v, w)) \]

Assume all edges have non-negative cost
Simulate Dijkstra’s algorithm (starting from s) on the graph

<table>
<thead>
<tr>
<th>Round</th>
<th>Vertex Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Who was Dijkstra?

- What were his major contributions?
Edsger Wybe Dijkstra was one of the most influential members of computing science's founding generation. Among the domains in which his scientific contributions are fundamental are

- algorithm design
- programming languages
- program design
- operating systems
- distributed processing
- formal specification and verification
- design of mathematical arguments
Dijkstra’s Algorithm as a greedy algorithm

• Elements committed to the solution by order of minimum distance
Correctness Proof

- Elements in \( S \) have the correct label
- Key to proof: when \( v \) is added to \( S \), it has the correct distance label.
Proof

• Let \( v \) be a vertex in \( V-S \) with minimum \( d[v] \)
• Let \( P_v \) be a path of length \( d[v] \), with an edge \((u,v)\)
• Let \( P \) be some other path to \( v \). Suppose \( P \) first leaves \( S \) on the edge \((x,y)\)
  
  - \( P = P_{sx} + c(x,y) + P_{yv} \)
  - \( \text{Len}(P_{sx}) + c(x,y) \geq d[y] \)
  - \( \text{Len}(P_{yv}) \geq 0 \)
  - \( \text{Len}(P) \geq d[y] + 0 \geq d[v] \)
Negative Cost Edges

- Draw a small example a negative cost edge and show that Dijkstra’s algorithm fails on this example
Bottleneck Shortest Path

• Define the bottleneck distance for a path to be the maximum cost edge along the path
Compute the bottleneck shortest paths
How do you adapt Dijkstra’s algorithm to handle bottleneck distances

• Does the correctness proof still apply?