Last Time

• Greedy Algorithms
  – Need to show correctness and optimality
  – Proven by Exchange Argument
  – Kruskal, Shortest Job First
Complexities

- We already saw that Kruskal is $O(E \times \log(E))$
- How about SJF?
  - If we have $N$ jobs, it takes $O(N \times \log(N))$ to sort them.
  - After they are sorted, we scan through linearly from smallest job to largest job ($O(N)$).
  - $O(N \times \log(N)) + (O(N)) = O(N \times \log(N))$
  - (There could still be an additional computation factor on top of the $O(N)$ because not every job will run in constant time.)
Greedy Advantages

• Always taking the best choice is usually easy.
  – Just sort the choices?

• Repeatedly taking the next available best choice is usually linear.
  – Sorting is more expensive than linear.

• Much cheaper than exhaustive search.
  – (Just about anything is cheaper than exhaustive search.)
  – Greedy algorithms are much cheaper than most other algorithms.

• Challenge: There is no guaranteed way to recognize problems that can be solved with greedy algorithms.
Egyptian Fractions

• A finite sum of distinct unit functions.
  – Example: \( \frac{43}{48} = \frac{1}{2} + \frac{1}{3} + \frac{1}{16} \)
  – Each fraction has a 1 numerator.
  – Each fraction has a positive denominator.
  – All denominators differ.
Fibonacci’s Greedy Egyptian Expansion

• **Theorem:** Any fraction $\frac{x}{y}$ can be represented as Egyptian Fractions by continuing the expansion $\frac{x}{y} = \frac{1}{[y/x]} + \frac{(-y)\%x}{y[y/x]}$ on the final term of the expansion, stopping when the new final term has a 1 numerator.

• **Example:** $\frac{7}{15} = \frac{1}{3} + \frac{2}{15} = \frac{1}{3} + \frac{1}{8} + \frac{1}{120}$
Fibonacci’s Greedy Egyptian Expansion

• Is it correct?

\[
\begin{align*}
&\frac{1}{[y/x]} + \frac{(-y)\%x}{y[y/x]} \\
&\frac{y}{y[y/x]} + \frac{(-y)\%x}{y[y/x]} \\
&\frac{y+(-y)\%x}{y[y/x]} \\
&\frac{x[y/x]}{y[y/x]} \\
&\frac{x}{y}
\end{align*}
\]
Fibonacci’s Greedy Egyptian Expansion

• Is it optimal?
  – (What is optimal in this case?)
  – Elegant sums? Shortest sums?

\[
\frac{5}{121} = \frac{1}{25} + \frac{1}{757} + \frac{1}{763309} + \frac{1}{873960180913} + \ldots
\]

\[
\frac{5}{121} = \frac{1}{33} + \frac{1}{121} + \frac{1}{363}
\]
Graph Coloring / Vertex Coloring

• Color the vertices of a graph such that no two adjacent vertices share the same color.
Graph Coloring / Vertex Coloring

• Is it correct?
  – Well, yes, if you assign every vertex a new color.

• Is it optimal?
  – “Can we minimize the number of colors required to color the vertices of a graph such that no two adjacent vertices share the same color?”
Greedy Vertex Coloring

• Consider the vertices in some sequence, and the colors in some sequence.
• For each vertex, assign the first available color.

• The vertices of any graph can be ordered in such a way that the greedy algorithm produces an optimal coloring.
  — Determining that order is hard (NP-Hard, in fact).
Heuristics for Ordering

• **Heuristic** – Any approach to problem solving, learning, or discovery that employs a practical method not guaranteed to be optimal/perfect, but sufficient for the immediate goals.

• Start with the vertex with the largest degree and traverse the graph? (Prim-ish)

• Sort vertices by degree and iterate through each? (Kruskal-ish)
Job Scheduling

- **Input** – Jobs with:
  - Release Times $r_1, r_2, \ldots, r_n$
  - Sizes $p_1, p_2, \ldots, p_n$
  - Deadlines $d_1, d_2, \ldots, d_n$

- **Output** – A feasible schedule, if one exists
• **Earliest Deadline First** – Run whichever job has the next deadline, regardless of whether or not a different job is already running.

<table>
<thead>
<tr>
<th>Release Time</th>
<th>Size</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>14</td>
</tr>
</tbody>
</table>
Earliest Deadline First

• Is it correct?
  – Each job runs for an appropriate amount of time.

• Is it optimal?
  – Turns out yes, through a similar argument to SJF.
Subset Sum Problem

• Input:
  – Coin values \( v_1, v_2, \ldots, v_n \)
  – Total value \( L \)

• Output: change for \( L \) cents if possible

• Example: \( \nu = \{7, 2, 11, 9, 16, 13\}, L = 41 \)
• Output: \{11, 16, 14\} or \{2, 9, 16, 14\}
Greedy Subset Sum

• Remember the coin distribution problem from Friday? That algorithm.

• Is it correct?
  – Sure.

• Is it optimal?
  – Heck no.
Any Questions?