Greedy Algorithms

- A **greedy algorithm** always makes the choice that looks best *at the moment*

- Put another way: Greed makes a *locally* optimal choice in the hope that this choice will lead to a *globally* optimal solution

- Greedy algorithms do **not** always yield optimal solutions, but for some problems they do
Greedy Algorithm

- **When** do we use greedy algorithms?
  - When we need a heuristic for a hard problem
  - When the problem itself is “greedy”

- **Examples of “Greedy” problems:**
  - Minimum Spanning Tree
    - Prim’s and Kruskal’s algorithms follow from “cut property” and “cycle property”, which we’ll see next time
  - Minimum Weight Prefix Codes (Huffman coding)
Properties of Greedy Problems

• **Greedy-choice property**: A globally optimal solution can be arrived at by making a locally optimal (greedy) choice
  – Difficulty is in proving this…

• **Optimal substructure property**: An optimal solution to the problem contains *within it* optimal solutions to subproblems
  – Key ingredient of both DP and Greed
Examples of Greedy Approaches

• Traveling Salesperson Problem
  – What is a greedy approach?

• Knapsack Problem
  – What is a greedy approach?

• Coin Changing Problem
  – What is a greedy approach?

• Graph Coloring, Vertex Cover, K-Center
  – Min #colors needed so that no edge has same-color endpoints
  – Min #vertices to have at least one endpoint of each edge
  – Pick k “centers” out of n points to minimize max point-to-center distance
Examples of Greedy Approaches

• Traveling Salesperson Problem
  – What is a greedy approach?
  – Start somewhere, always go to the nearest unvisited city

• Knapsack Problem
  – What is a greedy approach?
  – Use as much as possible of highest value/weight ratio item

• Coin Changing Problem
  – What is a greedy approach?
  – Use as much as possible of largest denominations first

• Graph Coloring, Vertex Cover, K-Center
  – Use lowest unused color…
  – Add highest-degree vertex…
  – Add new center that is farthest from all existing centers…

“Nearest-Neighbor” ~25% suboptimal for random Euclidean planar pointsets

Optimal for Fractional Knapsack Problem

Optimal for U.S. currency
Problem: Interval Scheduling
Treatment adapted from book by Kleinberg and Tardos

• Interval scheduling.
  – Job $j$ starts at $s_j$ and finishes at $f_j$.
  – Two jobs compatible if they don't overlap.
  – Goal: find maximum subset of mutually compatible jobs.
Interval Scheduling: Greedy Approaches

- **Greedy template**: Consider jobs in some order. Take each job provided that it is compatible with the ones already taken.
  - [Earliest start time] Consider jobs in **ascending order of start time** $s_j$
  - [Earliest finish time] Consider jobs in **ascending order of finish time** $f_j$
  - [Shortest interval] Consider jobs in **ascending order of length** $f_j - s_j$
  - [Fewest conflicts] For each job, count the number of conflicting jobs $c_j$. Schedule in **ascending order of conflicts** $c_j$
Interval Scheduling: Greedy Algorithms

- **Greedy template:** Consider jobs in some order. Take each job provided that it is compatible with the ones already taken.

  - Counterexample for earliest start time
  - Counterexample for shortest interval
  - Counterexample for fewest conflicts
Interval Scheduling: Greedy Algorithm

- Consider jobs in increasing order of finish time
- Take each job provided that it is compatible with the ones already taken (i.e., does not overlap or conflict with any previously scheduled jobs)

Sort jobs by finish times so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

Set of selected jobs

$A \leftarrow \emptyset$

for $j = 1$ to $n$

if (job $j$ compatible with $A$)

$A \leftarrow A \cup \{j\}$

$A \leftarrow A \cup \{j\}$

return $A$
Interval Scheduling
Interval Scheduling
Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11

A B C D E F G H
Interval Scheduling
Interval Scheduling
Interval Scheduling
Interval Scheduling

- A
- B
- C
- D
- E
- F
- G
- H

Time
Interval Scheduling
Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.
- Proof. (by contradiction)
  - Assume greedy is not optimal, and let’s see what happens.
  - Let \( i_1, i_2, \ldots, i_k \) denote jobs selected by greedy
  - Let \( j_1, j_2, \ldots, j_m \) denote jobs selected in optimal solution
  with \( i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r \) for the largest possible value of \( r \)

Greedy: 

OPT: 

why not replace job \( j_{r+1} \) with job \( i_{r+1} \)?
Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.
- Proof. (by contradiction)
  - Assume greedy is not optimal, and let’s see what happens.
  - Let $i_1, i_2, \ldots, i_k$ denote jobs selected by greedy
  - Let $j_1, j_2, \ldots, j_m$ denote jobs selected in optimal solution with
    $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of $r$.

<table>
<thead>
<tr>
<th>Greedy:</th>
<th>OPT:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

job $i_{r+1}$ finishes before $j_{r+1}$

solution still feasible and optimal, but contradicts maximality of $r$
Interval Partitioning

- Interval partitioning.
  - Lecture j starts at $s_j$ and finishes at $f_j$.
  - Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

- This schedule uses 4 classrooms for 10 lectures
Interval Partitioning

• Interval partitioning.
  – Lecture $j$ starts at $s_j$ and finishes at $f_j$
  – Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room

• This schedule uses only 3 classrooms for 10 lectures
Interval Partitioning: **Lower Bound on Optimal Solution**

- **Definition:** The *depth* of a set of open intervals is the maximum number that contain any given time.
- **Key observation.** Number of classrooms needed ≥ depth.
- **Example below:** Depth of intervals = 3 ⇒ schedule is optimal.

- **Q.** Does there always exist a schedule equal to depth of intervals?

![Graph showing time intervals and depth](image-url)
Interval Partitioning: Greedy Algorithm

- Greedy algorithm. Consider lectures in increasing order of start time: assign lecture to any compatible classroom.

Sort intervals by starting time so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

d $\leftarrow$ 0 ← number of allocated classrooms

for $j = 1$ to $n$ {
    if (lecture $j$ is compatible with some classroom $k$)
        schedule lecture $j$ in classroom $k$
    else
        allocate a new classroom $d + 1$
        schedule lecture $j$ in classroom $d + 1$
        d $\leftarrow$ d + 1
}
Interval Partitioning: Greedy Analysis

- Observation. Greedy algorithm never schedules two incompatible lectures in the same classroom.

- Theorem. Greedy algorithm is optimal.
- Proof.
  - Let $d =$ number of classrooms that the greedy algorithm allocates.
  - Classroom $d$ is opened because we needed to schedule a job, say $j$, that is incompatible with all $d-1$ other classrooms.
  - Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than $s_j$.
  - Thus, we have $d$ lectures overlapping at time $s_j + \varepsilon$.
  - Key observation $\Rightarrow$ all schedules use $\geq d$ classrooms. ▪
Trees

- A tree is connected and acyclic
- A tree on n nodes has n - 1 edges
- Any connected, undirected graph with |V| - 1 edges is a tree
- An undirected graph is a tree if and only if there is a unique path between any pair of nodes

Fact about trees: Any two of the following properties imply the third:
- Connected
- Acyclic
- |V| - 1 edges
**Minimum Spanning Tree**

Given a connected graph \( G = (V, E) \) with real-valued edge weights, an MST is a subset of the edges \( T \subseteq E \) such that \( T \) is a spanning tree whose sum of edge weights is minimized.

\[
\sum_{e \in T} c_e = 50
\]

Cayley's Theorem: There are \( n^{n-2} \) spanning trees of \( K_n \)

\( \rightarrow \) can't solve by brute force

Greedy MST Algorithms

• **Kruskal’s** algorithm. Start with $T = \emptyset$. Consider edges in ascending order of cost. Insert edge $e$ in $T$ unless doing so would create a cycle.

• **Reverse-Delete** algorithm. Start with $T = E$. Consider edges in descending order of cost. Delete edge $e$ from $T$ unless doing so would disconnect $T$.

• **Prim’s** algorithm. Start with some root node $s$ and greedily grow a tree $T$ from $s$ outward. At each step, add the cheapest edge $e$ to $T$ that has exactly one endpoint in $T$.

Huffman Codes

• **How to transmit English text using binary code?**
• 26 letters + space = alphabet has 27 characters
  – 5 bits per character suffices

• **Observation #1:** Not all characters occur with same frequency
  – Sherlock Holmes, “The Adventure of the Dancing Men”:
    ETAOIN SHRDLU
  – Suggests variable-length encoding

• **Observation #2:** Variable-length code should have prefix property
  – One code word per input symbol
  – No code word is a prefix of any other code word
  – Simplifies decoding process
Greedy Algorithm: Huffman Coding

- **Huffman coding** is based on probability with which symbols appear in a message
  - Goal is to minimize the expected code message length

- **How it works**
  - Create a tree root node for each nonzero symbol frequency, with the frequency as the value of the node
  - REPEAT
    - Find two root nodes with *smallest* value
    - Create a new root node with these two nodes as children, and value equal to the sum of the values of the two children
    - (Until there is only one root node remaining)
Why Huffman is Optimal

• Suppose we have an optimal tree $T$, where $a$, $b$ are siblings at the deepest level, while $x$, $y$ are the two nodes merged by Huffman’s algorithm

• **Swap** $x$ with $a$, $y$ with $b$

• Neither swap can increase cost
  – $x$ and $y$ have minimum weight
  – locations of $a$ and $b$ are at the deepest level
Huffman Coding Example

Example:
Symbol: A E G I M N O R S T U V Y Blank
Frequency: 1 3 2 2 1 2 2 2 2 1 1 1 1 3

(Generic Implementation)
- Place the elements into a min heap (by frequency)
- Remove the first two elements from the heap
- Combine these two elements into one
- Insert the new element back into the heap
Huffman Coding Example

Symbol: A E G I M N O R S T U V Y Blank

Frequency: 1 3 2 2 1 2 2 2 1 1 1 1 3

Step 1:

Step 2:

Step 3:

Step 4:
Huffman Coding Example

Symbol: A E G I M N O R S T U V Y Blank
Frequency: 1 3 2 2 1 2 2 2 2 1 1 1 1 3

Step 5:

Step 6:
Huffman Coding Example

Symbol: A E G I M N O R S T U V Y Blank
Frequency: 1 3 2 2 1 2 2 2 1 1 1 1 3

Step 7

Step 8
Huffman Coding Example

• Step 9

Symbol: A E G I M N O R S T U V Y Blank
Frequency: 1 3 2 2 1 2 2 2 1 1 1 1 3
Huffman Coding Example

Final tree:

Symbol:  A E G I M N O R S T U V Y Blank

Frequency:  1 3 2 2 1 2 2 2 1 1 1 1 3

E.g., code word for “Y” is “10101”

Sum of internal node values = total weighted pathlength of tree = \( \sum W_i \cdot L_i \)

= 4+5+9+2+2+4+7+2+4+4+8+15+24 = 90  (vs. \( \sum W_i \cdot L_i = 120 \) in naïve 5 bit per symbol code)