Outline and Reading

- Task Scheduling
- Fractional Knapsack Problem
Elements of Greedy Strategy

- An greedy algorithm makes a sequence of choices, each of the choices that seems best at the moment is chosen
  - NOT always produce an optimal solution
- Two ingredients that are exhibited by most problems that lend themselves to a greedy strategy
  - Greedy-choice property
  - Optimal substructure
Greedy-Choice Property

- A globally optimal solution can be arrived at by making a locally optimal (greedy) choice
  - Make whatever choice seems best at the moment and then solve the sub-problem arising after the choice is made
  - The choice made by a greedy algorithm may depend on choices so far, but it cannot depend on any future choices or on the solutions to sub-problems

- Of course, we must prove that a greedy choice at each step yields a globally optimal solution
Task Scheduling

- **Given:** a set \( T \) of \( n \) tasks, each having:
  - A start time, \( s_i \)
  - A finish time, \( f_i \) (where \( s_i < f_i \))

- **Goal:** Perform all the tasks using a minimum number of “machines.”

![Diagram showing task scheduling with machines and time intervals]
Task Scheduling Algorithm

- **Greedy choice:** consider tasks by their start time and use as few machines as possible with this order.
  - Run time: $O(n \log n)$.
- **Correctness:** Suppose there is a better schedule.
  - We can use $k-1$ machines
  - The algorithm uses $k$
  - Let $i$ be first task scheduled on machine $k$
  - Task $i$ must conflict with $k-1$ other tasks
  - $K$ mutually conflict tasks
  - But that means there is no non-conflicting schedule using $k-1$ machines

Algorithm **taskSchedule($T$)**

**Input:** set $T$ of tasks w/ start time $s_i$ and finish time $f_i$

**Output:** non-conflicting schedule with minimum number of machines

$m \leftarrow 0$  \{no. of machines\}

while $T$ is not empty
  remove task $i$ w/ smallest $s_i$
  if there’s a machine $j$ for $i$ then
    schedule $i$ on machine $j$
  else
    $m \leftarrow m + 1$
    schedule $i$ on machine $m$
Example

- **Given:** a set \( T \) of \( n \) tasks, each having:
  - A start time, \( s_i \)
  - A finish time, \( f_i \) (where \( s_i < f_i \))
  - \([1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]\) (ordered by start)
- **Goal:** Perform all tasks on min. number of machines
A banquet hall manager has received a list of reservation requests for the exclusive use of her hall for specified time intervals. She wishes to grant the maximum number of reservation requests that have no time overlap conflicts. Help her select the maximum number of conflict-free time intervals.
INPUT: A set $S = \{ I_1, I_2, \ldots, I_n \}$ of $n$ event time-intervals $I_k = \langle s_k, f_k \rangle$, $k = 1..n$, where $s_k = \text{start time of } I_k$, $f_k = \text{finishing time of } I_k$, ($s_k < f_k$).

OUTPUT: A maximum cardinality subset $C \subseteq S$ of mutually compatible intervals (i.e., no overlapping pairs).

Example:

$S = \text{the intervals shown below},$
$C = \text{the blue intervals},$
$|C| = 4.$

$C$ is not the unique optimum. Can you spot another optimum solution?
Some Greedy Heuristics

Greedy iteration step:
From among undecided intervals, select the interval $I$ that looks BEST. Commit to $I$ if it’s conflict-free (i.e., doesn’t overlap with the committed ones so far). Reject $I$ otherwise.

Greedy 1: BEST = earliest start time ($\min s_k$).

Greedy 2: BEST = latest finishing time ($\max f_k$).

Greedy 3: BEST = shortest interval ($\min f_k - s_k$).

Greedy 4: BEST = overlaps with fewest # of undecided intervals.
Activities sorted by finish time

Greedy-Activity-Selector(s, f)
1. $n \leftarrow length[s]$
2. $A \leftarrow \{a_1\}$
3. $i \leftarrow 1$
4. for $m \leftarrow 2$ to $n$
5.     do if $s_m \geq f_i$
6.     then $A \leftarrow A \cup \{a_m\}$
7.         $i \leftarrow m$
8. return $A$
Why it is Greedy?

- Greedy in the sense that it leaves as much opportunity as possible for the remaining activities to be scheduled.
- The greedy choice is the one that maximizes the amount of unscheduled time remaining.
Why this Algorithm is Optimal

- Algorithm has the following properties
  - optimal substructure property
  - The algorithm satisfies the greedy-choice property
- Thus, it is Optimal
Greedy-Choice Property

- Show there is an optimal solution that begins with a greedy choice (with activity 1, which as the earliest finish time)
- Suppose $A \subseteq S$ in an optimal solution
  - Order the activities in $A$ by finish time. The first activity in $A$ is $k$
    - If $k = 1$, the schedule $A$ begins with a greedy choice
    - If $k \neq 1$, show that there is an optimal solution $B$ to $S$ that begins with the greedy choice, activity 1
  - Let $B = A - \{k\} \cup \{1\}$
    - $f_i \leq f_k \implies$ activities in $B$ are disjoint (compatible)
    - $B$ has the same number of activities as $A$
    - Thus, $B$ is optimal
Optimal Substructures

Once the greedy choice of activity 1 is made, the problem reduces to finding an optimal solution for the activity-selection problem over those activities in S that are compatible with activity 1

- Optimal Substructure
- If A is optimal to S, then $A' = A - \{1\}$ is optimal to $S' = \{i \in S: s_i \geq f_i\}$
- Why?
  - If we could find a solution $B'$ to $S'$ with more activities than $A'$, adding activity 1 to $B'$ would yield a solution $B$ to $S$ with more activities than $A$ \(\Rightarrow\) contradicting the optimality of $A$

After each greedy choice is made, we are left with an optimization problem of the same form as the original problem

- By induction on the number of choices made, making the greedy choice at every step produces an optimal solution
A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to sub-problems.

- If an optimal solution $A$ to $S$ begins with activity 1, then $A' = A - \{1\}$ is optimal to $S' = \{i \in S: s_i \geq f_1\}$
Knapsack Problem

- One wants to pack \( n \) items in a luggage
  - The \( i \)th item is worth \( v_i \) dollars and weighs \( w_i \) pounds
  - Maximize the value but cannot exceed \( W \) pounds
  - \( v_i, w_i, W \) are integers

- 0-1 knapsack \( \rightarrow \) each item is taken or not taken

- Fractional knapsack \( \rightarrow \) fractions of items can be taken

- Both exhibit the optimal-substructure property
  - 0-1: If item \( j \) is removed from an optimal packing, the remaining packing is an optimal packing with weight at most \( W-w_j \)
  - Fractional: If \( w \) pounds of item \( j \) is removed from an optimal packing, the remaining packing is an optimal packing with weight at most \( W-w \) that can be taken from other \( n-1 \) items plus \( w_j \) – \( w \) of item \( j \)
The Fractional Knapsack Problem

- **Given**: A set $S$ of $n$ items, with each item $i$ having
  - $b_i$ - a positive benefit
  - $w_i$ - a positive weight
- **Goal**: Choose items with maximum total benefit but with weight at most $W$.
- If we are allowed to take fractional amounts, then this is the fractional knapsack problem.
  - In this case, we let $x_i$ denote the amount we take of item $i$
  - **Objective**: maximize
    \[
    \sum_{i \in S} b_i \left( \frac{x_i}{w_i} \right)
    \]
  - **Constraint**:
    \[
    \sum_{i \in S} x_i \leq W
    \]
Example

- Given: A set $S$ of $n$ items, with each item $i$ having
  - $b_i$ - a positive benefit
  - $w_i$ - a positive weight
- Goal: Choose items with maximum total benefit but with volume at most $W$. 

<table>
<thead>
<tr>
<th>Items</th>
<th>Volume</th>
<th>Benefit</th>
<th>Value ($ per ml$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4 ml</td>
<td>$12</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8 ml</td>
<td>$32</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2 ml</td>
<td>$40</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>6 ml</td>
<td>$30</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1 ml</td>
<td>$50</td>
<td>50</td>
</tr>
</tbody>
</table>

Solution:
- 1 ml of 5
- 2 ml of 3
- 6 ml of 4
- 1 ml of 2

"knapsack"
The Fractional Knapsack Algorithm

- Greedy choice: Keep taking item with highest value (benefit to weight ratio)
  - Use a heap-based priority queue to store the items, then the time complexity is \( \mathcal{O}(n \log n) \).

- Correctness: Suppose there is a better solution
  - there is an item \( i \) with higher value than a chosen item \( j \) (i.e., \( v_i < v_j \)), if we replace some \( j \) with \( i \), we get a better solution
  - Thus, there is no better solution than the greedy one

Algorithm \( \text{fractionalKnapsack}(S, W) \)

Input: set \( S \) of items w/ benefit \( b_i \) and weight \( w_i \); max. weight \( W \)

Output: amount \( x_i \) of each item \( i \) to maximize benefit with weight at most \( W \)

for each item \( i \) in \( S \)
  \[ x_i \leftarrow 0 \]
  \[ v_i \leftarrow b_i / w_i \quad \{\text{value}\} \]

\( w \leftarrow 0 \quad \{\text{current total weight}\} \)

while \( w < W \)
  remove item \( i \) with highest \( v_i \)
  \[ x_i \leftarrow \min\{w_i, W - w\} \]
  \[ w \leftarrow w + \min\{w_i, W - w\} \]
Greedy Algorithm for Fractional Knapsack problem

- Fractional knapsack can be solvable by the greedy strategy
  - Compute the value per pound $\frac{v_i}{w_i}$ for each item
  - Obeying a greedy strategy, take as much as possible of the item with the greatest value per pound.
  - If the supply of that item is exhausted and there is still more room, take as much as possible of the item with the next value per pound, and so forth until there is no more room
  - $O(n \log n)$ (we need to sort the items by value per pound)
  - Greedy Algorithm?
  - Correctness?
O-1 knapsack is harder!

- 0-1 knapsack cannot be solved by the greedy strategy
  - Unable to fill the knapsack to capacity, and the empty space lowers the effective value per pound of the packing
  - We must compare the solution to the sub-problem in which the item is included with the solution to the sub-problem in which the item is excluded before we can make the choice
Why is 0/1 Knapsack sub-optimal?

- Can you define a counter-example to Greedy-choice property?