Greed is Good?

  “It is not from the benevolence of the butcher, the brewer, or the baker, that we expect our dinner, but from their regard to their own interests.”
• Invisible hand: individuals acting on personal greed produces (nearly) globally optimal results

Interval Scheduling Problem

• Input: $R$, a set of $n$ resource requests:
  \{$(s_0, f_0), (s_1, f_1), \ldots, (s_n, f_n)$\} 
• Output: a subset $S$ of $R$ with no overlapping requests ($s_j > s_i < f_j$ for any $(s_i, f_i), (s_j, f_j) \in S$) such that $|S| \geq |T|$ for any $T \subseteq R$ with no overlapping requests

Greedy Algorithms

• Make the locally best choice “myopically” at each step
  – Need to figure out what “dessert” is
• Hope it leads to a globally good solution
  – Sometimes, can prove it leads to an optimal solution
  – Other times (like phylogeny), non-optimal, but usually okay if you get lucky

Example

$R = \{<0, 3>, <1, 2>, <1, 5>, <2, 5>, <2.5, 4>, <4, 4.5>\}$
Solution
\[ R = \{ <0, 3>, <1, 2>, <1, 5>, <2, 5>, <2.5, 4>, <4, 4.5> \} \]

Brute Force Algorithm
- Try all possible subsets
  - Filter out ones with overlapping intervals
  - Pick the largest subset
- Running time
  - How many subsets? \( 2^n \)
  - Constant work for each subset
  - \( \Theta(2^n) \)

Greedy Approaches
- Need to pick best subset by making myopic decisions, one element at a time
- Many possible criteria for making myopic decision
  - Earliest starting time?
  - Latest ending time?
  - Shortest?

Greedy Approach: Earliest Starting
Not optimal:
\[ |S| < |best| = 3 \]

Greedy Approach: Earliest Finishing
Not optimal:
\[ |S| < |best| = 3 \]

Greedy Approach: Shortest Length

Greedy Approach: Shortest Length
\[ R = \{ <0, 2.5>, <2, 3>, <2.5, 5> \} \]

Not optimal:
\[ |S| < |\text{best}| = 2 \]

Greedy Approach: Pick Earliest Finishing Time

Greedy Algorithm: Running Time Analysis
- Straightforward implementation:
  - Search to find earliest finishing: \( O(n) \)
  - Eliminate matching elements: \( O(n) \)
  - Repeat (up to \( n \) times): \( O(n^2) \)
- Smarter implementation:
  - Sort by finishing time: \( O(n \log n) \)
  - Go through list, selecting if non-overlapped: \( O(n) \)
  - Running time \( \in O(n \log n) \)

Correctness?
- How to prove a greedy algorithm is \textit{non}optimal
  - Find a counterexample: some input where the greedy algorithm does not find the best solution
- How to prove a greedy algorithm is \textit{optimal}
  - By induction: always best up to some size
  - By exchange argument: swapping any element in solution cannot improve result

Proof
- The greedy algorithm produces,
  \[ R = \{ r_0, \ldots, r_{k-1} \} \]
- Suppose there is a better subset,
  \[ Q = \{ q_0, \ldots, q_{k-1}, q_k \} \]
- Sort both by finishing time, so
  \[ f_{r_i} < f_{r_j} \text{ for all } 0 \leq i < j < k \]
  \[ f_{q_i} < f_{q_j} \text{ for all } 0 \leq i < j < k+1 \]

Proof
- \[ R = \{ r_0, \ldots, r_{k-1} \} \]
- \[ Q = \{ q_0, \ldots, q_{k-1}, q_k \} \]
- \[ f_{r_i} < f_{q_j} \text{ for all } 0 \leq i < j < k \]
- \[ f_{q_i} < f_{q_j} \text{ for all } 0 \leq i < j < k+1 \]

Strategy:
1. Prove by induction \( f_{r_i} \leq f_{q_j} \) for all \( i < k \)
2. Then, since \( f_{q_{k-1}} \leq f_{q_k} \) if \( q_k \) is valid, it would have also been added to \( R \).
Induction Proof: $f_{r_i} \leq f_{q_j}$

$R = \{ r_0, \ldots, r_{k-1} \}$  $Q = \{ q_0, \ldots, q_{k-1}, q_k \}$

- Basis: $f_{r_0} \leq f_{q_0}$
  - Greedy algorithm choose $r_0$ as the element with the earliest finishing time
  - So, $f_{r_0} \leq f_j$ for all $j$

- Induction: $f_{r_{i+1}} \leq f_{q_{j+1}} \Rightarrow f_{r_i} \leq f_{q_j}$
  - Since $f_{r_{i+1}} \leq f_{q_{j+1}}$, we know $s_{q_j} \geq f_{r_{i+1}}$
  - So, greedy algorithm could choose $q_j$
  - If $f_{r_i} < f_{q_j}$, greedy algorithm would have chosen $f_{q_j}$ instead of $f_{r_i}$

Knapsack Problems

- You have a collection of items, each has a value and weight
- How to optimally fill a knapsack with as many items as you can carry

Scheduling: weight = time, one deadline for all tasks
Budget allocation: weight = cost

General Knapsack Problem

- Input: a set of $n$ items \{<name$_0$, value$_0$, weight$_0$>, ..., <name$_{n-1}$, value$_{n-1}$, weight$_{n-1}$>\}, and maxweight
- Output: a subset of the input items such that the sum of the weights of all items in the output set is $\leq$ maxweight and there is no subset with weight sum $\leq$ maxweight with a greater value sum

Brute Force Knapsack

```python
def knapsack(items, maxweight):
    best = {}
    bestvalue = 0
    for s in allPossibleSubsets(items):
        value = 0
        weight = 0
        for item in s:
            value += item.value
            weight += item.weight
        if weight <= maxweight:
            if value > bestvalue:
                best = s
                bestvalue = value
    return best
```

(Defining and analyzing this might be a good Exam 1 question)

Average size of each subset is $n/2$
(there are as many subsets with size $c$ and of size $n - c$)
Running time $\in \Theta(n^2)$

Dynamic Programming

- Section this week: dynamic programming solution to the knapsack problem
- Running time in $O(maxweight \times n)$
Greedy Knapsack Algorithm

- Repeat until no more items fit:
  - Add the most valuable item that fits

- “Greedy”: always picks the most valuable item that fits first

```
def knapsack_greedy(items, maxweight):
    result = []
    weight = 0
    while True:
        # try to add the best item
        weightleft = maxweight - weight
        bestitem = None
        for item in items:
            if item.weight <= weightleft 
                and (bestitem == None 
                    or item.value > bestitem.value):
                bestitem = item
                if bestitem == None:
                    break
                else:
                    result.append(bestitem)
                    weight += bestitem.weight
        return result
```

Running Time
\[ \in \Theta(n^2) \]

Is Greedy Algorithm Correct?

No.
Proof by counterexample:
Consider input:
items = {"gold", 100, 1 >, 
        "platinum", 110, 3> 
        "silver", 80, 2 >}
maxweight = 3
Greedy algorithm picks {"platinum"}
value = 110, but {"gold", "silver"}
has weight <= 3 and value = 180

Same Weights Knapsack Problem

- Input: a set of \( n \) items \((<\text{name}_0, \text{value}_0>,
\ldots, <\text{name}_{n-1}, \text{value}_{n-1}>)\), each having
weight \( w_i \) and maxweight
- Output: a subset of the input items such that the sum of the weights of all items in the output set is \( \leq \maxweight \) and there is no subset with weight sum \( \leq \maxweight \) with a greater value sum

Greedy Algorithm Correct

- It keeps adding items until maxweight would be exceeded, so the result contains \( k \) items where \( kw \leq \maxweight \) and \( (k+1)w > \maxweight \)
- Hence, cannot add any item (weight \( w \)) without removing another item
- But, any item with value > value of the lowest value item in result would have already been added by greedy algorithm

Subset Sum Problem

- Knapsack problem where \( \text{value} = \text{weight} \)
- Input: set of \( n \) positive integers, \( \{w_0, \ldots, w_{n-1}\} \), maximum weight \( W \)
- Output: a subset \( S \) of the input set such that the sum of the elements of \( S \leq W \) and there is no subset of the input set whose sum is greater than the sum of \( S \) and \( \leq W \)
**Brute Force Subset Sum**

```python
def subsetsum(items, maxweight):
    best = {}
    for s in allPossibleSubsets(items):
        if sum(s) <= maxweight and sum(s) > sum(best):
            best = s
    return best
```

Just like brute force knapsack:
Running time $\in \Theta(n2^n)$

**Greedy Subset Sum?**

- Pick largest item that fits
  - Bad: $I = \{4, 5, 7\}$ $W = 9$
- Pick smallest item
  - Bad: $I = \{4, 5, 7\}$ $W = 7$
- Doesn’t prove there is no myopic criteria that works

Note: Subset Sum is known to be NP-Complete, so finding one would prove $P = NP$

**Charge**

- More greedy algorithm examples in Section this week
- PS3: greedy phylogeny algorithm
  - Not optimal (prove in Question 8)
  - Usually reasonably good (similar to algorithms used in practice)