Greedy Algorithms – Cont'd

Making Change

Example: Making Change

- Input
  - Positive integer n
- Task
  - Compute the minimum number of minimal multisets of coins from C = $\{d_1, d_2, d_3, \ldots, d_k\}$ such that the sum of all coins chosen equals n
- Example
  - n = 73, C = {1, 3, 6, 12, 24}
  - Solution: 3 coins of size 24, 1 coin of size 1

Dynamic Programming Solution 1

- Subsolutions: T(j) for $0 \leq j \leq n$
- Recurrence relation
  - $T(n) = \min_i (T(i) + T(n-i))$
  - $T(d_i) = 1$
  - Linear array of values to compute
    - Time complexity of computing each entry?

Dynamic Programming Solution 2

- Subsolutions: T(j) for $0 \leq j \leq n$
- Recurrence relation
  - $T(n) = \min_i (T(n-d_i) + 1)$
    - There has to be a "first/last" coin
  - $T(d_i) = 1$
  - Linear array of values to compute
    - Time complexity of computing each entry?

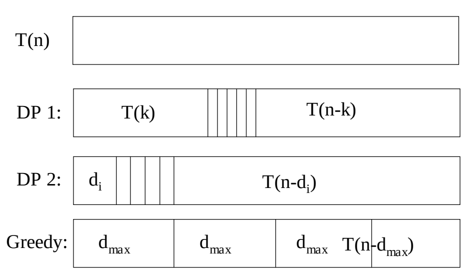
Greedy Solution

- From dynamic programming 2:

$$T(n) = \min_i (T(n-d_i) + 1)$$

- Key observation
  - For many (but not all) sets of coins, the optimal choice for the first/last coin d_i will always be the maximum possible d_i
  - That is, $T(n) = T(n-d_{max}) + 1$ where d_{max} is the largest $d_i \leq n$
- Algorithm
  - Choose largest d_i smaller than n and recurse

Comparison



Example 1: Making Change Proof 1

- Greedy is optimal for coin set C = {1, 3, 9, 27, 81}
- Structural property of any optimal solution:
  - In any optimal solution, the number of coins of denomination 1, 3, 9, and 27 must be at most 2.
  - Why?
- This structural property immediately leads to the fact that the greedy solution must be optimal
  - Why?

Example 1: Making Change Proof 2

- Greedy is optimal for coin set C = {1, 3, 9, 27, 81}
- Let S be an optimal solution and G be the greedy solution
- Let A_k denote the number of coins of size k in solution A
- Let *kdiff* be the largest value of k s.t. $G_k \neq S_k$
- Claim 1: $G_{kdiff} > S_{kdiff}$. Why?
- Claim 2: For some $d_i < d_{kdiff}$, we should have $S_i \geq 3$. Why?
- Claim 3: We can create a better solution than S by performing a "swap". What swap?
- These three claims imply *kdiff* does not exist and G_k is optimal.

Proof that Greedy is NOT optimal

- Consider the following coin set
  - C = {1, 3, 6, 12, 24, 30}
- Prove that greedy will not produce an optimal solution
- What about the following coin set?
  - C = {1, 5, 10, 25, 50}

Greedy Technique

- When trying to solve a problem, make a local greedy choice that optimizes progress towards global solution and recurse
- Implementation/running time analysis is typically straightforward
  - Often implementation involves use of a sorting algorithm or a data structure to facilitate identification of next greedy choice
- Proof of optimality is typically the hard part

Proofs of Optimality

- We will often prove some structural properties about an optimal solution
  - Example: Every optimal solution to the activity selection problem has a task with earliest end time
- We will often prove that **an** optimal solution is the one generated by the greedy algorithm
  - If we have an optimal solution that does not obey the greedy constraint, we can “swap” some elements to make it obey the greedy constraint
- Always consider the possibility that greedy is not optimal and consider counter-examples

Exercise: Minimizing Sum of Completion Times

- Input
  - Set of n jobs with lengths x_i
- Task
  - Schedule these jobs on a single processor so that the sum of all job completion times are minimized
- Example
  - {2, 1, 3}
  - Solution:
    - Completion times: 3, 1, 6 for a sum of 10
- Develop a greedy strategy and prove it is optimal

Questions

- What is the running time of your algorithm?
- Does it ever make sense to preempt a job? That is, start a job, interrupt it to run a second job (and possibly others), and then finally finish the first job?
- Can you develop a swapping proof of optimality for your algorithm?