Greedy Algorithms

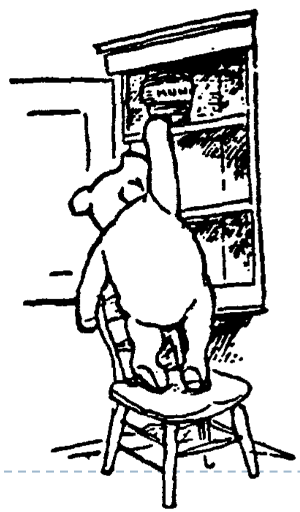
Ying Zhao
Department of Computer Science, Tsinghua University

Outline

- Activity-selection problem (Ch16.1)
- Elements of greedy algorithm (Ch16.2)

Greedy Solutions for Optimization Problems

- Optimization problem: *view the optimal solution as a sequence of choices.*
- In order to get what you want, just start grabbing what looks best.
- The greedy choice: Commit to the selection that looks the "best" (without solving the subproblems first).
- Surprisingly, many important and practical optimization problems can be solved this way.



Greedy Choice

- Example: Knapsack of capacity $W = 5$
  - item weight value

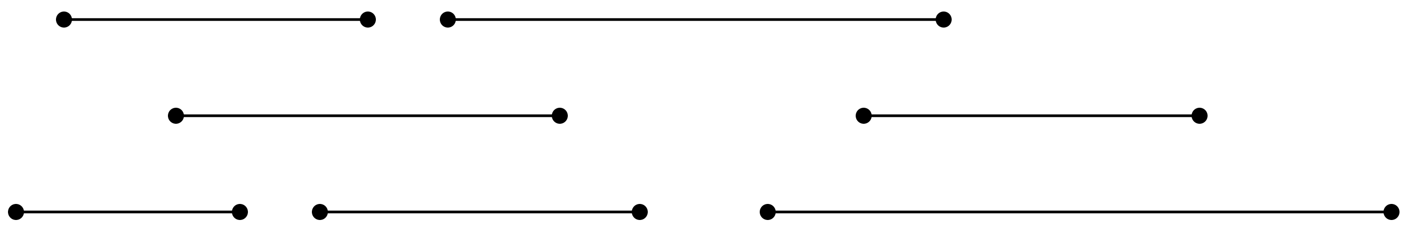
item	weight	value
1	2	$\$12$
2	1	$\$10$
3	3	$\$20$
4	2	$\$15$

  - Greedy choice: take the most valuable item!
- Greedy algorithms may not lead to the optimal solution.
  - *Wrong!* Do not confuse it with heuristic algorithms!

Activity-selection Problem

- Input: Set A of n activities, $a_1, a_2, \dots, a_n$.
  - s_i = start time of activity i.
  - f_i = finish time of activity i.
- Output: Subset S of maximum number of compatible activities.
  - Two activities are compatible, if their intervals don't overlap.

Example:



Subproblems

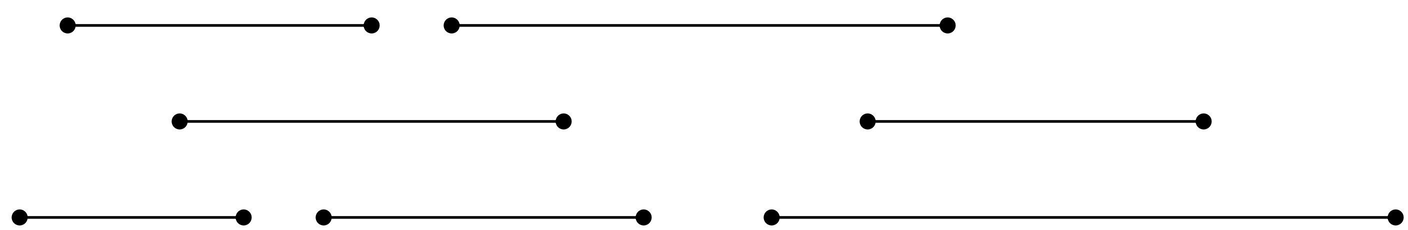
- Two options:
  - Suppose an optimal solution includes activity a_n
  - Suppose an optimal solution does not include activity a_n

- Multiple options:
  - Suppose an optimal solution first picks activity a_1
  - …
  - Suppose an optimal solution first picks activity a_k
  - …
  - Suppose an optimal solution first picks activity a_n

Optimal Substructure

- Suppose an optimal solution does not include activity a_n.
  - Let $A_{1,n}$ be $\{a_1, a_2, \dots, a_n\}$, then the remaining subproblem becomes to find the maximum # of compatible activities from $A_{1,n-1}$.

Example:

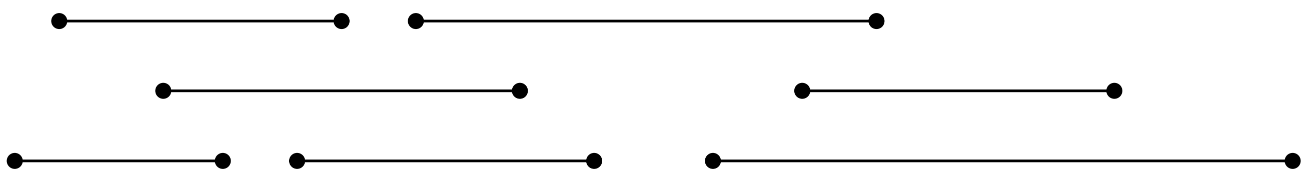


- Suppose an optimal solution includes activity a_n.
  - How to describe a subproblem whose input must be compatible with a_n?
  - $A_{1,n-1} = \{a_1, a_2, \dots, a_{n-1}\}$ does not work.
  - The “two options” way does not work.

Optimal Substructure

- Suppose an optimal solution includes activity a_k.
  - Activities that are compatible with a_k, either starts after a_k finishes, or finish before a_k starts.
  - So we can use two activities to define a subset of activities.

Example:



- Have activities in order.
  - Let a_0 and a_{n+1} be two dummy activities, a_0 finishes before any activity starts and a_{n+1} starts after all activities finish.
  - Suppose activities are sorted by finishing times $f_1 \leq f_2 \leq \ldots \leq f_n$.

Optimal Substructure

- Let $A_{i,j}$ be a subset of activities in A that start after a_i finishes and finish before a_j starts.
  - $A_{0,n+1}$ is the original input A.
- Suppose an optimal solution of $A_{i,j}$ includes activity a_k.
  - This generates two subproblems:
    - Selecting maximum # of compatible activities from $A_{i,k}$.
    - Selecting maximum # of compatible activities from $A_{k,j}$.
- Suppose $S_{i,j}$ is an opt solution to $A_{i,j}$ and $S_{i,j} = \{S_{i,k}, a_k, S_{k,j}\}$, then $S_{i,k}$ and $S_{k,j}$ must be optimal for $A_{i,k}$ and $A_{k,j}$, respectively.
    - Prove by using the cut-and-paste approach.
    - *Key*：the two subproblems are independent!

      Suppose S' is an opt. solution to $A_{i,k}$ and S'' is an opt. solution to $A_{k,j}$, activities in S' and activities in S'' are compatible to one another.

Optimal Substructure

- Suppose S' is an opt. solution to $A_{i,k}$ and S'' is an opt. solution to $A_{k,j}$, activities in S' and activities in S'' are compatible to one another.

Recursive Solution

- Let $c[i, j]$ = size of maximum-size subset of mutually compatible activities in $A_{i,j}$.

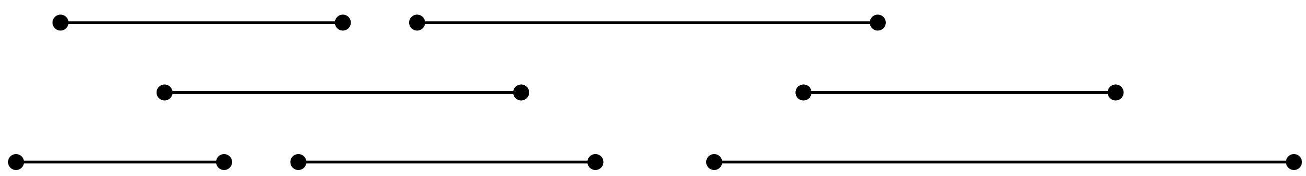
- Suppose an optimal solution of $A_{i,j}$ includes activity a_k.
  - This generates two subproblems:
    - Selecting maximum # of compatible activities from $A_{i,k}$.
    - Selecting maximum # of compatible activities from $A_{k,j}$.

- $c[i, j] = \begin{cases} 0 & \text{if } A_{i,j} = \emptyset \\ \max\limits_{i<k<j} \{c[i, k] + c[k, j] + 1\} & \text{if } A_{i,j} \neq \emptyset \end{cases}$

When the greedy choice DOES work?

- Problems exhibit optimal substructure.
  - an optimal solution to the problem contains within it optimal solutions to subproblems.
- Problems also exhibit the **greedy-choice** property.
  - **greedy-choice property**: there is a **global optimal solution** that contains the greedy choice.
  - Otherwise, taking the greedy choice means *WRONG!*

Example:



In-class Exercise

- Give a greedy-choice candidate for the activity selection problem.

Greedy-choice Property

- The problem also exhibits the greedy-choice property.
  - Assume activities are sorted by finishing times，
    $$f_1 \le f_2 \le \ldots \le f_n$$
  - Greedy-choice: the smallest finish time in $A_{i,j}$
  - Greedy-choice property: There is an optimal solution to the subproblem $A_{i,j}$, that includes the activity with the smallest finish time in set $A_{i,j}$

Greedy-choice Property

- Therefore, we can first make this greedy choice and then solve the remaining subproblems.
- Combine the greedy choice and the solution to the subproblem to get the overall solution. The following recurrence can be simplified.

$$c[i,j] = \begin{cases} 0 & \text{if } A_{i,j} = \emptyset \\ \max\limits_{i<k<j}\{c[i,k] + c[k,j] + 1\} & \text{if } A_{i,j} \neq \emptyset \end{cases}$$

Top-down Algorithm

Assuming activities are sorted by finish time.

Greedy-Activity-Selector (s, f)

$n \leftarrow length[s]$

$S \leftarrow \{a_1\}$

$i \leftarrow 1$

for $m \leftarrow 2$ **to** n //earliest to finish in $S_{i,n+1}$

  if $s_m \geq f_i$ //check compatibility

  then $S \leftarrow S \cup \{a_m\}$

    $i \leftarrow m$

Return S

Initial Call: `Greedy-Activity-Selector(s,f)`

Complexity: $\theta(n)$

Example

k	0	1	2	3	4	5	6	7	8	9	10	11	12
s_k	---	1	3	0	5	3	5	6	8	8	2	12	---
f_k	0	4	5	6	7	8	9	10	11	12	13	14	∞

Design Points of GA

- Cast the optimization problem as one in which we make a choice and are left with subproblems to solve.
- Derive a recurrence:
  - Prove that there's always an optimal solution that makes the greedy choice, so that the greedy choice is always safe.
  - Prove optimal substructure: show that greedy choice and optimal solution to subproblem $\Rightarrow$ optimal solution to the problem.
- Implement a top-down solution:
  - make the greedy choice and solve the remaining problem.
- May have to preprocess input to put it into the greedy order.
  - Example: Sorting activities by finish time
  - Running time: $O(n \lg n)$ for sorting, and solve a recurrence of $T(n)$

Comparison with DP

	Greedy Algorithm	Dynamic Programming
“Recursive” nature	Yes	Yes
Combine solutions to subproblems	Yes	Yes
Partition subproblems	Making a greedy choice at a time	Making one choice at a time
Overlapping subproblems	No	Yes
Primarily for optimization	Yes	Yes
Optimal substructure	Yes (to develop a recurrence)	Yes (to develop a recurrence)
Preprocessing	Usually sorting	
Top-down vs. Bottom-up	Top-down	Bottom-up (but…)
Characteristic running time	Often dominated by $n \lg n$ sort	The space of subproblems

About Final

- Time: 19:00-21:00, Dec 24, 2020 (Beijing Time)
  - Same settings as the midterm for on-line students
  - 19:00-19:30 check identity & get ready
  - 19:30-21:00 final exam (closed book & notes)
- Review Section:
  - Next Monday, Dec 21, 2020
  - A quick review & comments on HW13-15 and sample exam.
- Office-hours:
  - 9:00am-11:00am, Dec 23, 2020 (Beijing Time) @my office east-main building 8-204, or @the class Zhumu conference room.
  - 19:00-21:00, Dec 23, 2020 (Beijing Time) @the class Zhumu conference room.
- HW15 & Programming Assignment #2 are due on Jan 5, 2021.

Thank you!