Greedy (1)

Oct 18th, 2018

Algorithm Design and Analysis

YUN-NUNG (VIVIAN) CHEN HTTP://ADA.MIULAB.TW

國立臺灣大學 National Taiwan University

Slides credited from Hsueh-I Lu, Hsu-Chun Hsiao, & Michael Tsai

Outline

- Greedy Algorithms
- Greedy #1: Activity-Selection / Interval Scheduling
- Greedy #2: Coin Changing
- Greedy #3: Fractional Knapsack Problem
- Greedy #4: Breakpoint Selection
- Greedy #5: Huffman Codes
- Greedy #6: Task-Scheduling
- Greedy #7: Scheduling to Minimize Lateness

Algorithm Design Strategy

- Do not focus on "specific algorithms"
- But "some strategies" to "design" algorithms

- First Skill: Divide-and-Conquer (各個擊破)
- Second Skill: Dynamic Programming (動態規劃)
- Third Skill: Greedy (貪婪法則)

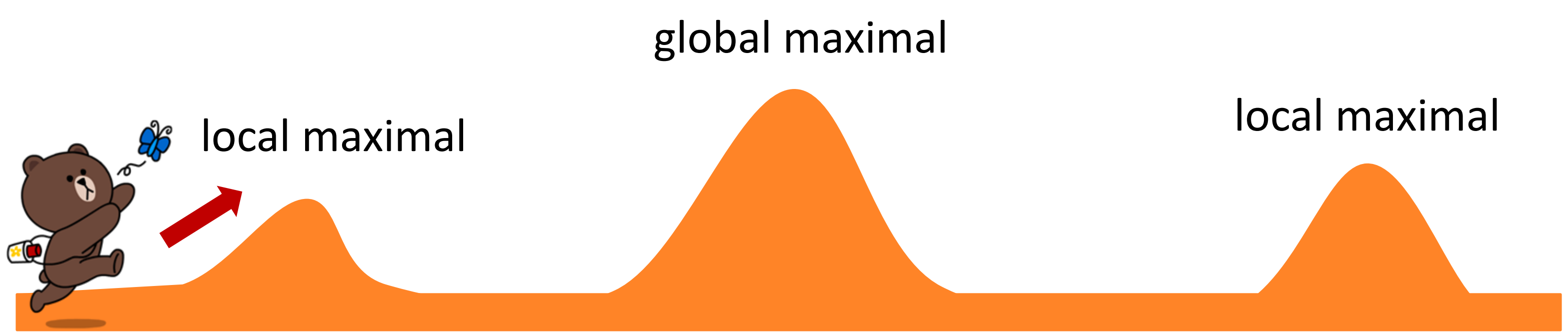
4 Greedy Algorithms

Textbook Chapter 16 – Greedy Algorithms

Textbook Chapter 16.2 – Elements of the greedy strategy

What is Greedy Algorithms?

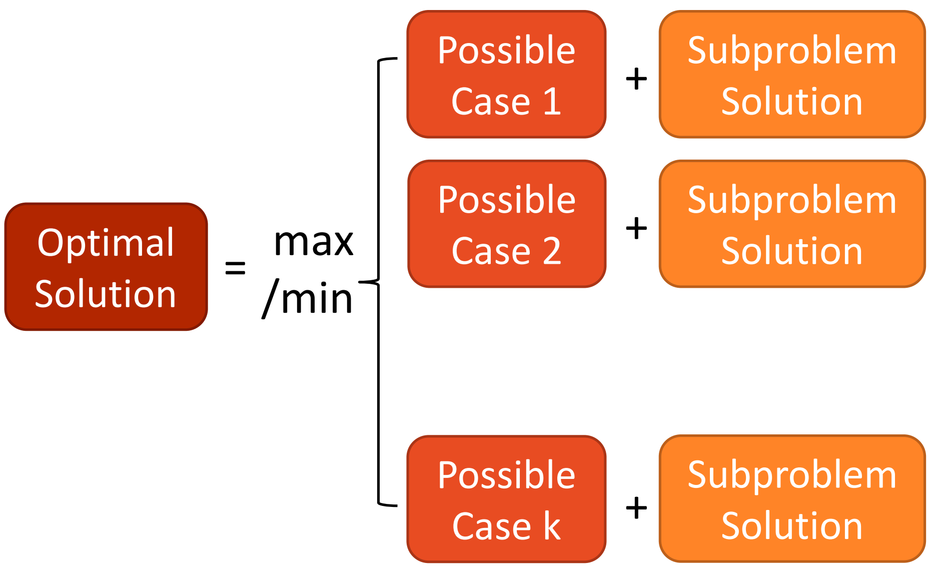
- always makes the choice that looks best at the moment
- makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution
  - not always yield optimal solution; may end up at local optimal



Greedy: move towards max gradient and hope it is global maximum

Algorithm Design Paradigms

- Dynamic Programming
  - has **optimal substructure**
  - make an informed choice after getting optimal solutions to subproblems
  - **dependent** or **overlapping** subproblems



- Greedy Algorithms
  - has **optimal substructure**
  - make a greedy choice before solving the subproblem
  - **no overlapping** subproblems
    - ✓ Each round selects only one subproblem
    - ✓ The subproblem size decreases

Greedy Procedure

1. Cast the optimization problem as one in which we make a choice and remain one subproblem to solve
2. Demonstrate the optimal substructure
   - ✓ Combining an optimal solution to the subproblem via greedy can arrive an optimal solution to the original problem
3. Prove that there is always an optimal solution to the original problem that makes the greedy choice

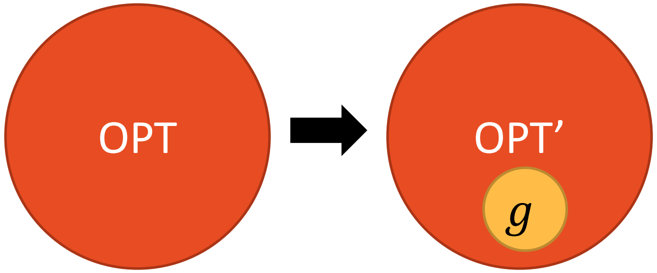
Greedy Algorithms

To yield an optimal solution, the problem should exhibit

1. Optimal Substructure : an optimal solution to the problem contains within it optimal solutions to subproblems
2. Greedy-Choice Property : making locally optimal (greedy) choices leads to a globally optimal solution

Proof of Correctness Skills

- Optimal Substructure : an optimal solution to the problem contains within it optimal solutions to subproblems
- Greedy-Choice Property : making locally optimal (greedy) choices leads to a globally optimal solution
  - Show that it exists an optimal solution that "contains" the greedy choice using **exchange argument**
  - For any optimal solution OPT, the greedy choice g has two cases
    - g is in OPT: done
    - g not in OPT: modify OPT into OPT' s.t. OPT' contains g and is at least as good as OPT



✓ If OPT' is better than OPT, the property is proved by contradiction

✓ If OPT' is as good as OPT, then we showed that there exists an optimal solution containing g by construction

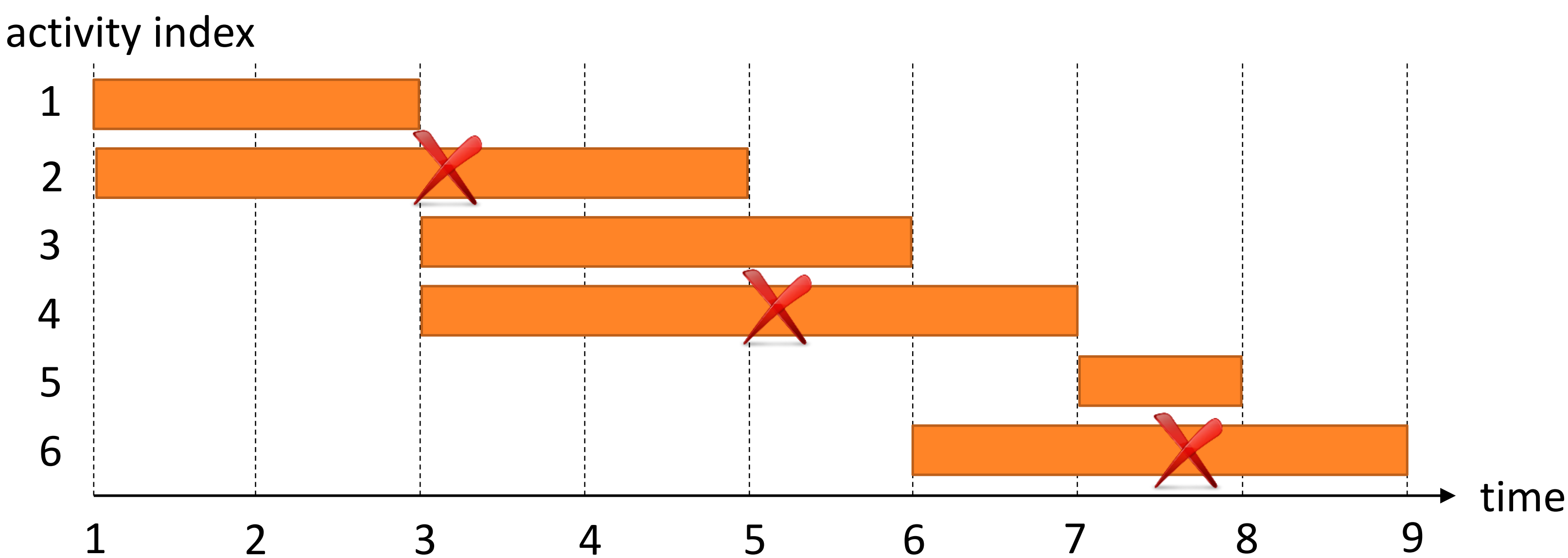
10

Activity-Selection / Interval Scheduling

Textbook Chapter 16.1 – An activity-selection problem

Activity-Selection/ Interval Scheduling

- Input: n activities with start times s_i and finish times f_i (the activities are sorted in monotonically increasing order of finish time $f_1 \leq f_2 \leq \cdots \leq f_n$)
- Output: the maximum number of compatible activities
- Without loss of generality: $s_1 < s_2 < \cdots < s_n$ and $f_1 < f_2 < \cdots < f_n$
  - 大的包小的則不考慮大的 → 用小的取代大的一定不會變差



Weighted Interval Scheduling

Weighted Interval Scheduling Problem
Input: n jobs with $\langle s_i, f_i, v_i \rangle$, $p(j)$ = largest index $i < j$ s.t. jobs i and j are compatible
Output: the maximum total value obtainable from compatible

- Subproblems
  - `WIS(i)`: weighted interval scheduling for the first i jobs
  - Goal: `WIS(n)`
- Dynamic programming algorithm

$$M_i = \begin{cases} 0 & \text{if } i = 0 \\ \max(v_i + M_{p(i)}, M_{i-1}) & \text{otherwise} \end{cases}$$

i	0	1	2	3	4	5	…	n
M[i]								

$$T(n) = \Theta(n)$$

Set $v_i = 1$ for all i to formulate it into the activity-selection problem

Activity-Selection Problem

Activity-Selection Problem
Input: n activities with $\langle s_i, f_i \rangle$, $p(j)$ = largest index $i < j$ s.t. i and j are compatible
Output: the maximum number of activities

- Dynamic programming

$$M_i = \begin{cases} 0 & \text{if } i = 0 \\ \max(1 + M_{p(i)}, M_{i-1}) & \text{otherwise} \end{cases}$$

  - **Optimal substructure** is already proved

- Greedy algorithm

$$M_i = \begin{cases} 0 & \text{if } i = 0 \\ 1 + M_{p(i)} & \text{otherwise} \end{cases}$$

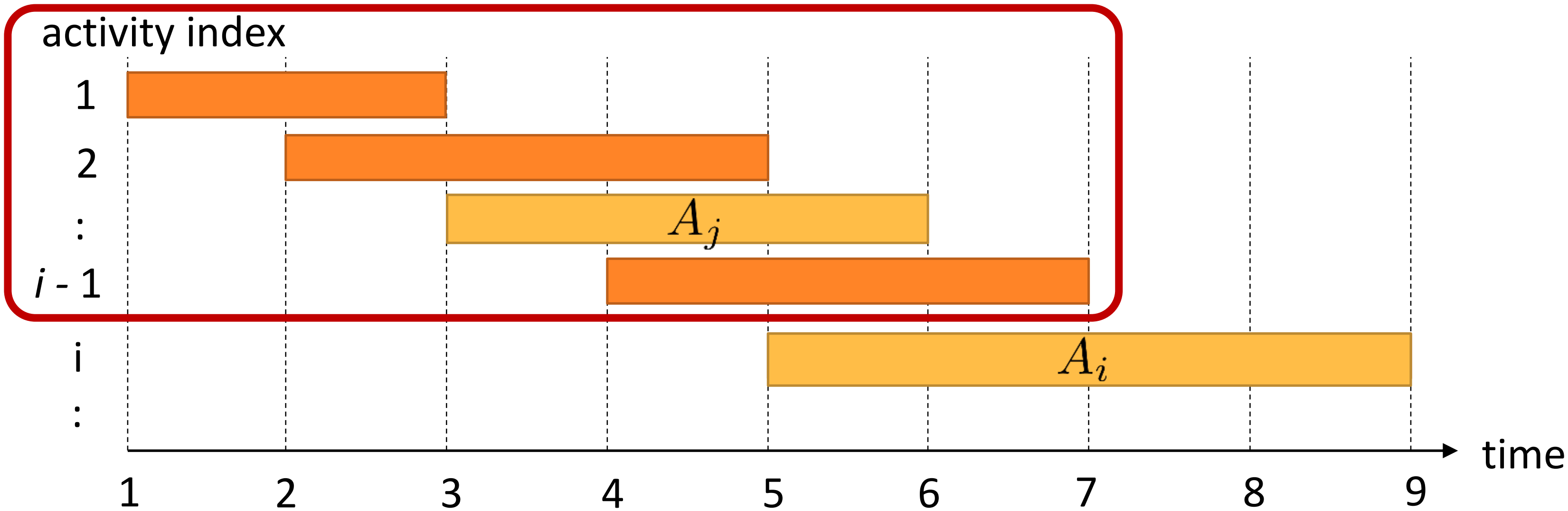
select the i-th activity

Why does the i-th activity must appear in an OPT?

Greedy-Choice Property

- Goal: $1 + M_{p(i)} \geq M_{i-1}$
- Proof
  - Assume there is an OPT solution for the first $i - 1$ activities (M_{i-1})
    - A_j is the last activity in the OPT solution → $M_{i-1} = 1 + M_{p(j)}$
  - Replacing A_j with A_i does not make the OPT worse

$$1 + M_{p(i)} \geq 1 + M_{p(j)} = M_{i-1}$$



Pseudo Code

Activity-Selection Problem
Input: n activities with $\langle s_i, f_i \rangle$, $p(j)$ = largest index $i < j$ s.t. i and j are compatible
Output: the maximum number of activities

```
Act-Select(n, s, f, v, p)
  M[0] = 0
  for i = 1 to n
    if p[i] >= 0
      M[i] = 1 + M[p[i]]
  return M[n]
```

$$T(n) = \Theta(n)$$

```
Find-Solution(M, n)
  if n = 0
    return {}
  return {n} ∪ Find-Solution(p[n])
```

$$T(n) = \Theta(n)$$

Select the **last** compatible one (←) = Select the **first** compatible one (→)

16 Coin Changing

Textbook Exercise 16.1

Coin Changing Problem

- Input: n dollars and unlimited coins with values $\{v_i\}$ (1, 5, 10, 50)
- Output: the minimum number of coins with the total value n

- **Cashier's algorithm**: at each iteration, add the coin with the largest value no more than the current total

Does this algorithm return the OPT?

Step 1: Cast Optimization Problem

Coin Changing Problem
Input: n dollars and unlimited coins with values $\{v_i\}$ (1, 5, 10, 50)
Output: the minimum number of coins with the total value n

- Subproblems
  - `C(i)`: minimal number of coins for the total value i
  - Goal: `C(n)`

Step 2: Prove Optimal Substructure

> **Coin Changing Problem**
> Input: n dollars and unlimited coins with values $\{v_i\}$ (1, 5, 10, 50)
> Output: the minimum number of coins with the total value n

- Suppose OPT is an optimal solution to `C(i)`, there are 4 cases:
  - Case 1: coin 1 in OPT
    - OPT\coin1 is an optimal solution of $\texttt{C(i - v}_1\texttt{)}$
  - Case 2: coin 2 in OPT
    - OPT\coin2 is an optimal solution of $\texttt{C(i - v}_2\texttt{)}$
  - Case 3: coin 3 in OPT
    - OPT\coin3 is an optimal solution of $\texttt{C(i - v}_3\texttt{)}$
  - Case 4: coin 4 in OPT
    - OPT\coin4 is an optimal solution of $\texttt{C(i - v}_4\texttt{)}$

$$C_i = \min_j (1 + C_{i - v_j})$$

Step 3: Prove Greedy-Choice Property

Coin Changing Problem
Input: n dollars and unlimited coins with values $\{v_i\}$ (1, 5, 10, 50)
Output: the minimum number of coins with the total value n

- Greedy choice: select the coin with the largest value no more than the current total
- Proof via contradiction (use the case $10 \leq i < 50$ for demo)
  - Assume that there is no OPT including this greedy choice (choose 10)

    → all OPT use 1, 5, 50 to pay i
    - 50 cannot be used
    - #coins with value 5 < 2 → otherwise we can use a 10 to have a better output
    - #coins with value 1 < 5 → otherwise we can use a 5 to have a better output
  - We cannot pay i with the constraints (at most $5 + 4 = 9$)

21 To Be Continued…

Question?

Important announcement will be sent to @ntu.edu.tw mailbox
& post to the course website

Course Website: http://ada.miulab.tw

Email: ada-ta@csie.ntu.edu.tw