

Algorithm Design and Analysis Midterm Review

<http://ada.miulab.tw>

slido: #ADA2021



國立臺灣大學
National Taiwan University

Yun-Nung (Vivian) Chen

Announcement

- Homework assignment
 - HW2 due on 11/09 14:20
 - HW2 解答會在死線當天晚上公布
- Midterm announcement
 - Next week!!!

Midterm!!!



- **Date: 11/11 雙十一 (Thursday)**
- **Time: 14:20-17:20** (3 hours)
- **Location: R102 + R103 + R204 + R219** (check the seat assignment before entering the room)
- **Content**
 - Recurrence and Asymptotic Analysis
 - Divide and Conquer
 - Dynamic Programming
 - Greedy
- Based on slides, assignments, and some variations (practice via textbook exercises)
- Format: Yes/No, Multiple-Choice, Short Answer, Prove/Explanation
- Easy: ~65%, Medium: ~25%, Hard: ~10%
- Close book

Tip: exam questions are not all of equal difficulty, move on if you get stuck!

Algorithm Design & Analysis Process

- 1) Formulate a **problem**
- 2) Develop an **algorithm**
- 3) Prove the **correctness**
- 4) Analyze **running time/space** requirement

Design Step

Analysis Step

Algorithm Analysis

- Analysis Skills
 - Prove by contradiction
 - Induction
 - Asymptotic analysis
 - Problem instance
- Algorithm Complexity
 - In the worst case, what is the growth of function an algorithm takes
- Problem Complexity
 - In the worst case, what is the growth of the function the optimal algorithm of the problem takes

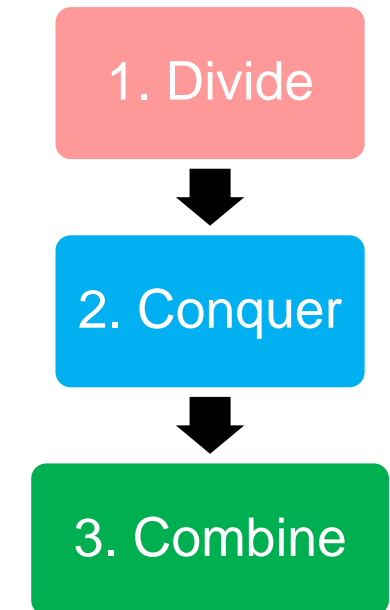
Algorithm Design Strategy

- 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)

Divide-and-Conquer

What is Divide-and-Conquer?

- Solve a problem recursively
- Apply three steps at each level of the recursion
 1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem (比較小的同樣問題)
 2. **Conquer** the subproblems by solving them recursively
If the subproblem sizes are *small enough*
 - then solve the subproblems base case
 - else recursively solve itself recursive case
 3. **Combine** the solutions to the subproblems into the solution for the original problem



How to Solve Recurrence Relations?

1. Substitution Method (取代法)

- Guess a bound and then prove by induction

2. Recursion-Tree Method (遞迴樹法)

- Expand the recurrence into a tree and sum up the cost

3. Master Method (套公式大法/大師法)

- Apply Master Theorem to a specific form of recurrences

Master Theorem

The proof is in Ch. 4.6

divide a problem of size n into a subproblems, each of size $\frac{n}{b}$ is solved in time $T\left(\frac{n}{b}\right)$ recursively

Let $T(n)$ be a positive function satisfying the following recurrence relation

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ a \cdot T\left(\frac{n}{b}\right) + f(n) & \text{if } n > 1, \end{cases}$$

Should follow
this format

where $a \geq 1$ and $b > 1$ are constants.

- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- Case 3: If
 - $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and
 - $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n ,
 then $T(n) = \Theta(f(n))$.



compare $f(n)$ with $n^{\log_b a}$

When to Use D&C?

- Analyze the problem about
 - Whether the problem with small inputs can be solved directly
 - Whether subproblem solutions can be combined into the original solution
 - Whether the overall complexity is better than naïve
- If no, then
 - Try to modify it or add more information
 - Try another way for dividing
 - Do not use D&C

Pseudo-Polynomial Time

- Polynomial: polynomial in the **length of the input** (#bits for the input)
- Pseudo-polynomial: polynomial in the **numeric value**
- The time complexity of 0-1 knapsack problem is $\Theta(nW)$
 - n : number of objects
 - W : knapsack's capacity (non-negative integer)
 - polynomial in the numeric value

= pseudo-polynomial in input size
= exponential in the length of the input
- Note: the size of the representation of W is $\log_2 W$
$$= 2^m = m$$

Dynamic Programming

What is Dynamic Programming?

- Dynamic programming, like the divide-and-conquer method, solves problems by combining the solutions to subproblems
 - 用空間換取時間
 - 讓走過的留下痕跡
- “Dynamic”: time-varying
- “Programming”: a *tabular* method

Dynamic Programming: planning over time

Algorithm Design Paradigms

- Divide-and-Conquer
 - partition the problem into **independent** or **disjoint** subproblems
 - repeatedly solving the common subsubproblems
 - more work than necessary
- Dynamic Programming
 - partition the problem into **dependent** or **overlapping** subproblems
 - avoid recomputation
 - ✓ Top-down with memoization
 - ✓ Bottom-up method

Dynamic Programming Procedure

- Apply four steps
 1. Characterize the structure of an optimal solution
 2. **Recursively** define the value of an optimal solution
 3. Compute the value of an optimal solution, typically in a **bottom-up** fashion
 4. Construct an optimal solution from computed information

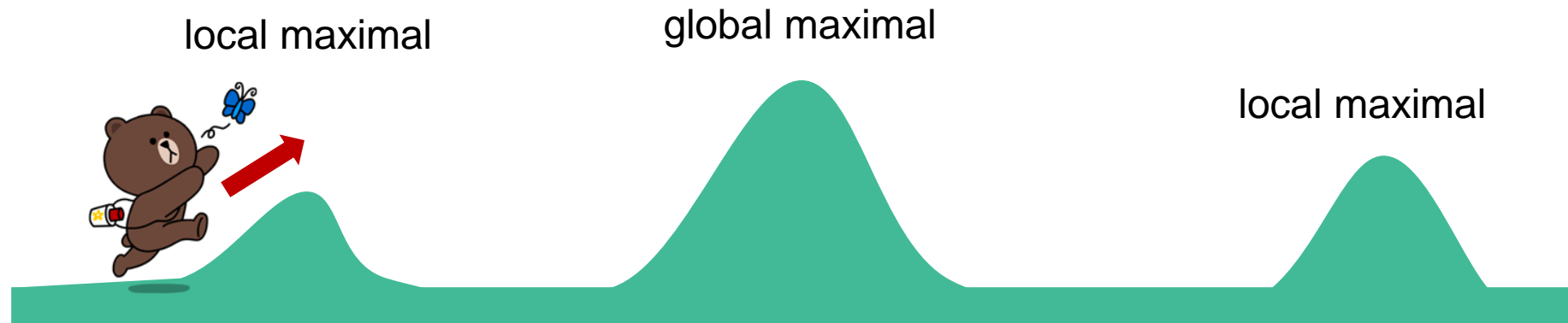
When to Use DP?

- Analyze the problem about
 - Whether subproblem solutions can combine into the original solution
 - When subproblems are overlapping
 - Whether the problem has optimal substructure
 - Common for optimization problem
- Two ways to avoid recomputation
 - Top-down with memoization
 - Bottom-up method
- Complexity analysis
 - Space for tabular filling
 - Size of the subproblem graph

Greedy Algorithms

What is Greedy Algorithms?

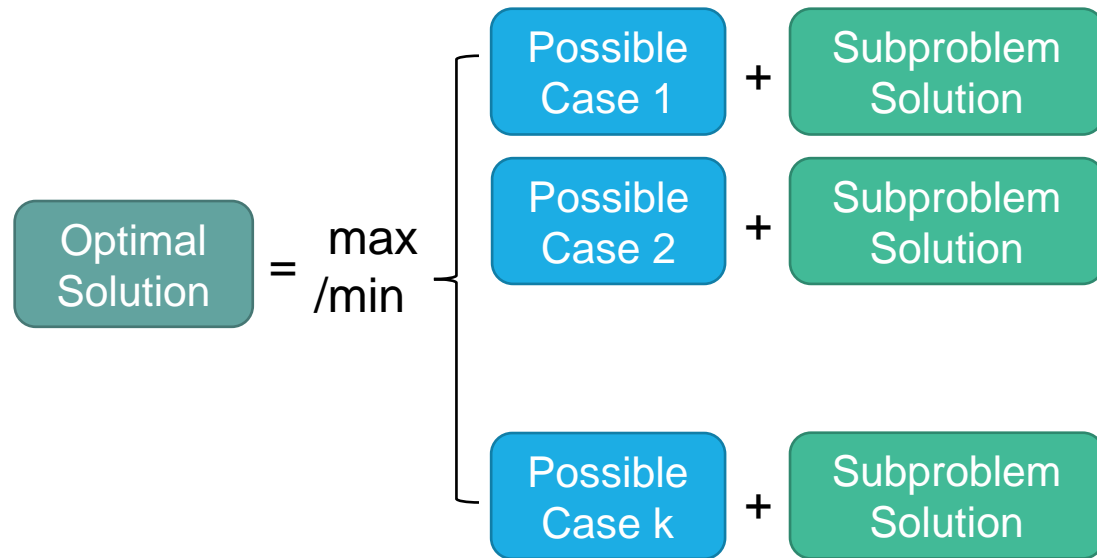
- always makes the choice that looks best at the moment
- makes a **locally optimal** choice in the hope that this choice will lead to a **globally optimal** solution
 - not always yield optimal solution; may end up at local optimal



Algorithm Design Paradigms

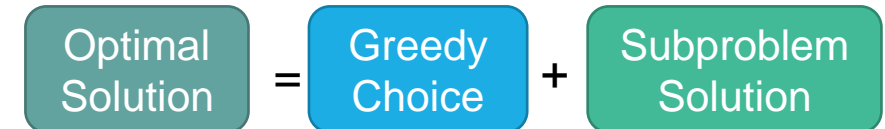
- Dynamic Programming

- 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

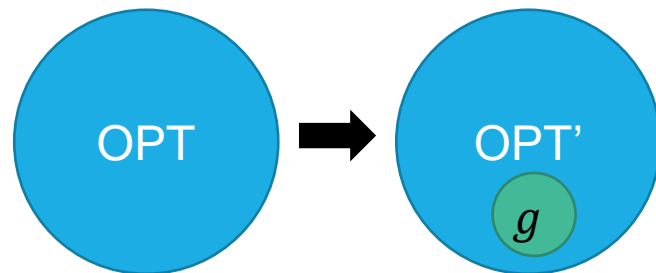


Greedy Procedure

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**

Proof of Correctness Skills

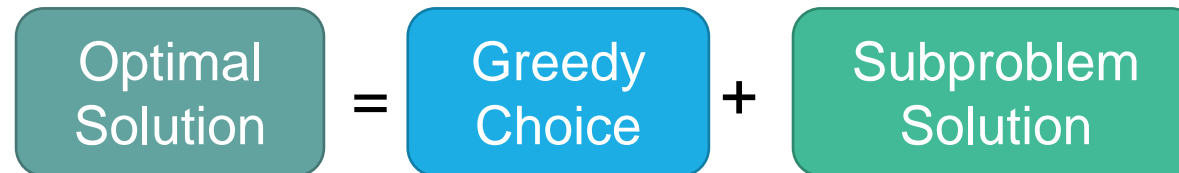
- **Optimal Substructure** : an optimal solution to the problem contains within it optimal solutions to subproblems
- **Greedy-Choice Property** : making locally optimal (greedy) choices leads to a globally optimal solution
 - Show that it exists an optimal solution that “contains” the greedy choice 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

When to Use Greedy?

- Analyze the problem about
 - Whether the problem has optimal substructure
 - Whether we can make a greedy choice and remain only one subproblem
 - Common for optimization problem



Exercises

Kahoot!



Short Answer Questions

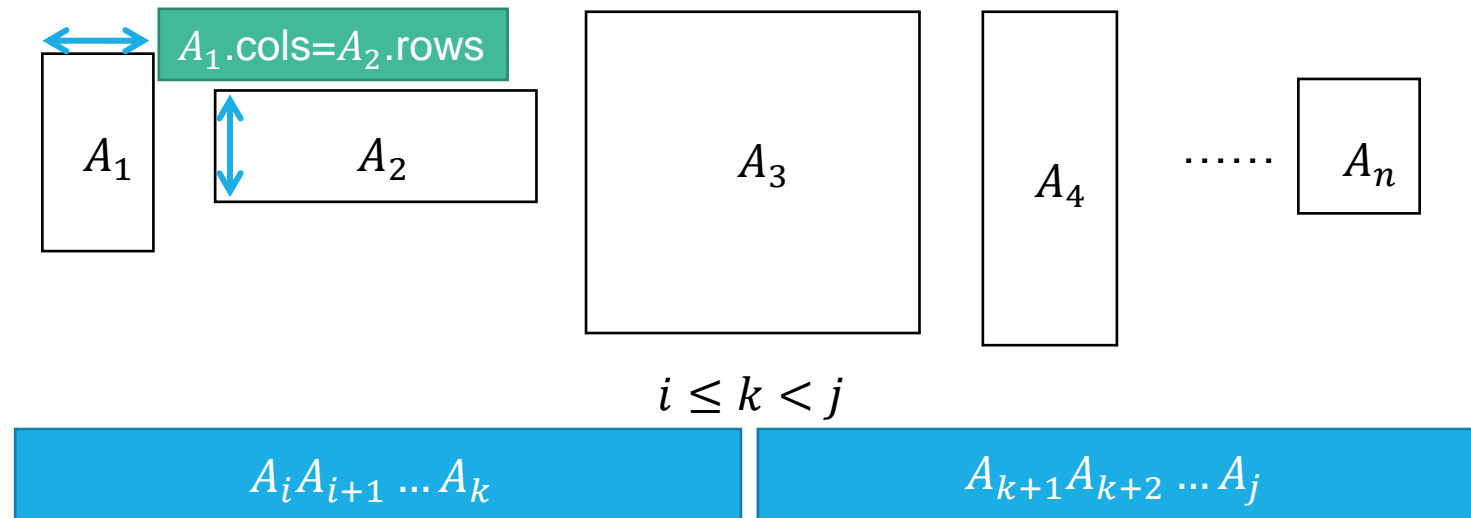
- Given the following recurrence relation, provide a valid traversal order to fill the DP table or justify why no valid traversal exists.

$$A(i, j) = F(A(i - 2, j + 1), A(i + 1, j - 2))$$

- Explain why it is usually easy for a divide-and-conquer algorithm to take advantage of a multi-core computer system.
- Given N items and a bag of capacity W , explain why the $O(NW)$ -time 0/1 knapsack algorithm mentioned in class is pseudo-polynomial.

Matrix-Chain Multiplication









- Input: a sequence of integers l_0, l_1, \dots, l_n
 - l_{i-1} is the number of rows of matrix A_i
 - l_i is the number of columns of matrix A_i
- Output: an order of performing $n - 1$ matrix multiplications in the **maximum** number of operations to obtain the product of $A_1 A_2 \dots A_n$



Q: Does optimal substructure still hold?









Painting

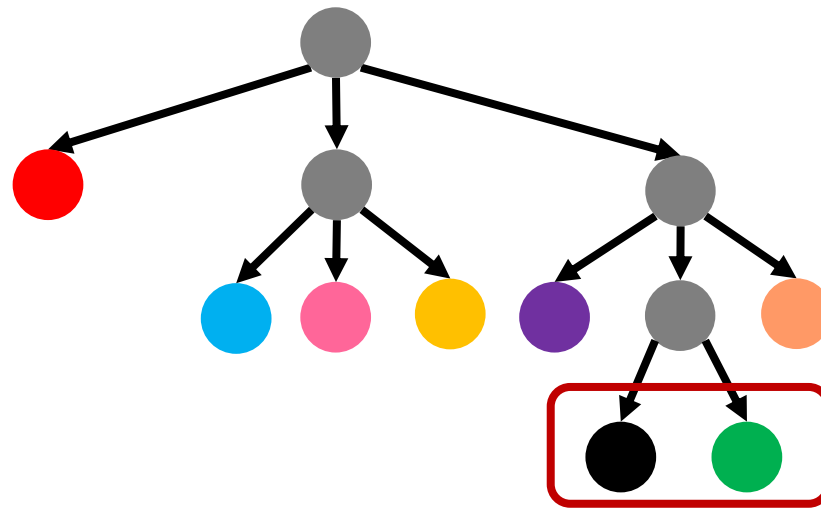
- Put stickers in a single row on each tube to indicate its color.
- There are k types of stickers.
- Tubes with the same color should have the same sticker pattern and should be prefix free.

Color	red	pink	orange	yellow	green	blue	purple	black
								
#Tubes	25	15	12	19	7	12	8	2

- Minimize the total number of stickers put on all tubes
- 3-ary prefix tree (each node can have at most k children).

3-arry Huffman Coding

Color	red	pink	orange	yellow	green	blue	purple	black
								
#Tubes	25	15	12	19	7	12	8	2



- The total length is

$$25 \cdot 1 + (12 + 15 + 19 + 8 + 12) \cdot 2 + (7 + 2) \cdot 3 = 184$$

考古題 Practice 1

1. Maximum Subarray of a Circular Infinite Sequence (2015 midterm) Recall that a maximum subarray of A is a contiguous subarray a_s, \dots, a_t of A such that $\sum_{s \leq i \leq t} a_i$ is maximized over all s and t , $0 \leq s \leq t$.

Given a *circular infinite sequence* $A = \langle a_0, a_1, a_2, \dots \rangle$ in which $a_i = a_j$ if $i = j \pmod n$, please answer the following questions.

1. Suppose $\sum_{0 \leq i < n} a_i > 0$. What is the length of the maximum subarray of A ? Briefly explain your answer.
2. Suppose $\sum_{0 \leq i < n} a_i < 0$. Please briefly explain why the length of any maximum subarray is at most n .
3. Please design an algorithm to find a maximum subarray of the circular infinite sequence A in $O(n \log n)$ time. Can you reduce the running time of your algorithm to $O(n)$? Please justify the correctness and running time of your algorithm.

考古題 Practice 2

2. Fair Division of Christmas Gifts (2014 midterm) Christmas is approaching. You're helping Santa Claus to distribute gifts to children.

For ease of delivery, you are asked to divide n gifts into two groups such that the weight difference of these two groups is minimized. The weight of each gift is a positive integer. Please design an algorithm to find an optimal division minimizing the value difference. The algorithm should find the minimal weight difference as well as the groupings in $O(nS)$ time, where S is the total weight of these n gifts. Briefly justify the correctness of your algorithm.

Hint: This problem can be converted into making one set as close to $S/2$ as possible.

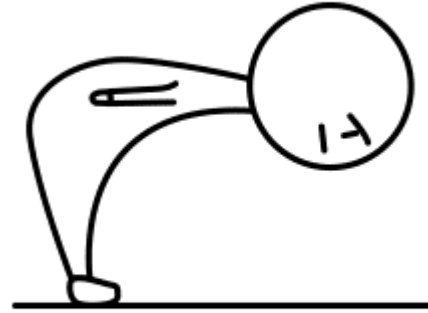
考古題 Practice 3

3. Zombie Apocalypse (2016 midterm) Due to a zombie virus outbreak, some cities have been occupied by zombies and are no longer safe. You and your survivor team need to travel through several cities to get to a far away shelter.

There are n cities forming a line topology. You are at city 1 now and the shelter is at city n . The location of city i is $L[i]$, and $L[i] < L[j] \forall 1 \leq i < j \leq n$. $z[i] = 1$ indicates city i has been occupied by zombies; otherwise, $z[i] = 0$ indicates the city is still safe to stop at night.

If you plan to move at most 100km a day, and you need to rest at a safe city at night, please design a greedy algorithm to pick the cities for resting at night so that you can arrive at the shelter as soon as possible. Your algorithm should run in $O(n)$ time. Please show that your algorithm has the greedy choice property.

slido



下台 - 鞠躬
Thank you~



What do you want to say to me?

① Start presenting to display the poll results on this slide.



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