NCKU Programming Contest Training Course
Course 8
2013/03/06

Sheng-Chi You(rabbit125)

Jay_s6215@hotmail.com

http://myweb.ncku.edu.tw/~f74986133/Course_8.rar

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

Outline

Dynamic Programming

Longest Increase Sequence(LIS)

Longest Common Subsequence(LCS)

Dynamic Programming

- Dynamic Programming (DP)
  - Dynamic programming is a general algorithm design technique for solving problems defined by or formulated by recursion structure with overlapping substructure.

- Purpose
  - Optimization
  - Avoid duplicate search and calculation
  - Two categories
    - Minimization (maximization) problem
    - Combinatorial problem

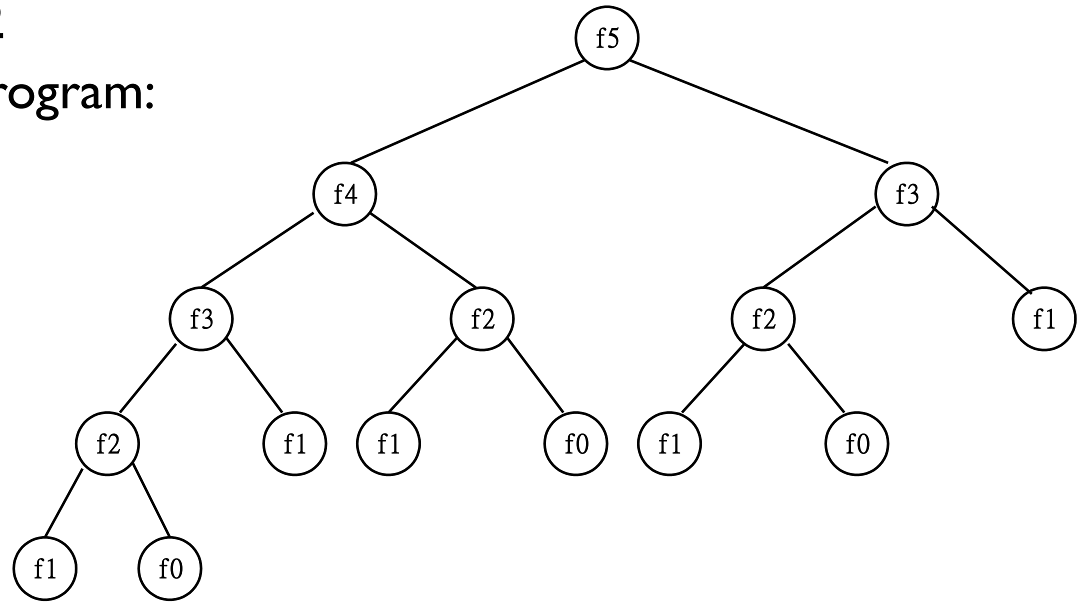
Recall for Fib-Seq

- Fibonacci sequence: 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , …

  $F_i = i \quad \text{if } i \le 1$

  $F_i = F_{i-1} + F_{i-2} \quad \text{if } i \ge 2$

- Solved by a recursive program:



- Much replicated computation is done.
- It should be solved by a simple loop.

Solving Method

- Solving the Fib-Sequence
  - Bottom up design
    - Using the for loop or while loop without recursive stack
  - Top-down design
    - Using the recursive design with memorization for computation

- Top-Down design
  - (1) basic condition
  - (2) if memorized
  - (3) recurrence
  - (4) return the value
- Take a problem for example
  - PKU: 1579 Function Run Fun

Function Run Fun

- How to figure out this is a DP problem ?
  1. We want to find out what value w(a, b, c) is
  2. w(a, b, c) can be generated by the functions at last slide
     - ex, if we want to know w(50, 50, 50), we have to know w(20, 20, 20) at first
  3. The final answer is consist of lots of **subproblems**
     - ex, w(a, b, c) = $\underbrace{\text{w(a-1, b, c) + w(a-1, b-1, c) + w(a-1, b, c-1) - w(a-1, b-1, c-1)}}_{\textbf{subproblems}}$

Function Run Fun

- if (a <= 0 or b <= 0 or c <= 0)
  - w(a, b, c) = 1
- if (a > 20 or b > 20 or c > 20)
  - w(a, b, c) = w(20, 20, 20)
- if (a < b and b < c)
  - w(a, b, c) = w(a, b, c-1) + w(a, b-1, c-1) - w(a, b-1, c)
- otherwise
  - w(a, b, c) = w(a-1, b, c) + w(a-1, b-1, c) + w(a-1, b, c-1) - w(a-1, b-1, c-1)

Just Practice

- Practice
  - PKU: 1579 Function Run Fun

Programming Steps

- Programming Strategy
  - (1) Check the category, min-max problem or combination problem?
  - (2) Check if there is a order(*the optimal order*)
    - This is a very important step!!!
    - Ordered -> optimization with dp
    - In-ordered -> bitmask dp, memorized dp or non-dp problems
  - (3) Think the recurrence formulation
  - (4) Write a program to solve it
    - Top-down
    - Bottom-up
  - (5) Backtrack the optimal path

- Memory Strategy
  - Set up the recorded table

Outline

Dynamic Programming

Longest Increase Sequence(LIS)

Longest Common Subsequence(LCS)

made by electron, free999, rabbit125

LIS Problem

- Longest Increasing Subsequence
  - The longest increasing subsequence problem is to find **a subsequence of a given sequence in which the elements in this subsequence are in sorted order (lowest to highest)**, and in which **the length** of the subsequence **is as long as possible**.
  - The elements in the subsequence are not necessarily to be continuous.
  - Two well-known method to solve this problem are followings:
    - (1) DP by $O(N^2)$
    - (2) Greedy with binary search by O(NlogN)

made by electron, free999, rabbit125

LIS Example

- Example
  - 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15
  - A increasing subsequence is 0, 4, 14, 15
  - The longest increasing subsequence is 0, 2, 6, 9, 13, 15 wit he length six
- Question
  - Is the longest increasing subsequence unique?
  - How should we deal with this problem in different situation by which method?

DP method

- Dynamic programming approach
  - Recall the design strategy
  - (1) Check the category
  - (2) Check the order property
  - (3) Think the recurrence
  - (4) Write and problem with two methods
  - (5) Backtrack the optimal path

DP method

- Rule 1
  - Order?
- Rule 2
  - Category
- Rule 3
  - Given a sequence with *n* elements stored in an array *seq[i]* where *1<=i<=n*.
  - **Define *dp[i]* for representing that the longest length of the increasing subsequence that ended by *seq[i]* from *seq[1]* to *seq[i]*.**
  - So that the recurrence can be formulated as the following:
    - Initialized the dp[i] by 1
    - dp[i] = max(dp[j]+1), where 1<=j<i and seq[i]>seq[i]
  - Also define a pi array *pi[i]* that represent the previous element of the element *i* in the increasing subsequence.

DP method

- Rule 3
  - Example
  - *seq[9] = {9, 5, 2, 8, 7, 3, 1, 6, 4}*
  - Find the dp[i] and pi[i].

initial

seq	9 (1)	5 (2)	2 (3)	8 (4)	7 (5)	3 (6)	1 (7)	6 (8)	4 (9)
dp	1	1	1	1	1	1	1	1	*1*
pi	-1	-1	-1	-1	-1	-1	-1	-1	-1

result

seq	9 (1)	5 (2)	2 (3)	8 (4)	7 (5)	3 (6)	1 (7)	6 (8)	4 (9)
dp	1	1	1	2	2	2	1	3	3
pi	-1	-1	-1	2	2	3	-1	6	6

DP method

- Rule 4
  - Write the program
- Rule 5
  - Trace the result
- Exercise
  - Write a program that find the length of the LIS for a given sequence.
  - Note:
    - Please use the dynamic programming as the practice.
    - The number of the element in the given sequence will not exceed 1000.
- Review
  - Time complexity O(?)
  - Space complexity O(?)
  - Compare with the brute force method.

Greedy Method

- Greedy Method
  - An efficient algorithm based on binary search.
  - *Given the sequence seq[9] = {9, 5, 2, 8, 7, 3, 1, 6, 4}*
  - http://www.csie.ntnu.edu.tw/~u91029/LongestIncreasingSubsequence.html

							6	4
			8	7	3	3	3	3
9	5	2	2	2	2	1	1	1

Greedy Method

- Exercise
  - Write a program that find the length of the LIS for a given sequence.
  - Note:
    - Please use the dynamic programming as the practice.
    - The number of the element in the given sequence will exceed 1000.

- Review
  - Time complexity O(?)
  - Space complexity O(?)

Just Practice

- Practice
  - NOJ 30 LIS Problem

Outline

Dynamic Programming

Longest Increase Sequence(LIS)

Longest Common Subsequence(LCS)

LCS Problem

- Longest Common Subsequence
  - The longest increasing subsequence problem is to find a **common subsequence** of two given sequences in which the elements in this common subsequence **are appear in both original sequences**, and in which **the length** of the subsequence **is as long as possible**.
  - The elements in the subsequence are not necessarily to be continuous.
  - Two well-known method to solve this problem are followings:
    - (1) DP by $O(N^2)$
    - (2) Greedy with binary search by O(NlogN)

Example

- Common Subsequence Example

president

providence

president

providence

- Longest Common Subsequence

president

providence

DP method

- Dynamic programming approach
  - Recall the design strategy
  - (1) Check the category
  - (2) Check the order property
  - (3) Think the recurrence
  - (4) Write and problem with two methods
  - (5) Backtrack the optimal path

DP method

- Rule 1
  - Order?
- Rule 2
  - Category
- Rule 3
  - Let A=$a_1 a_2 \ldots a_m$ and $B=b_1 b_2 \ldots b_n$.
  - $len(i, j)$: the length of an LCS between $a_1 a_2 \ldots a_i$ and $b_1 b_2 \ldots b_j$
  - With proper initializations, $len(i, j)$ can be computed as follows.

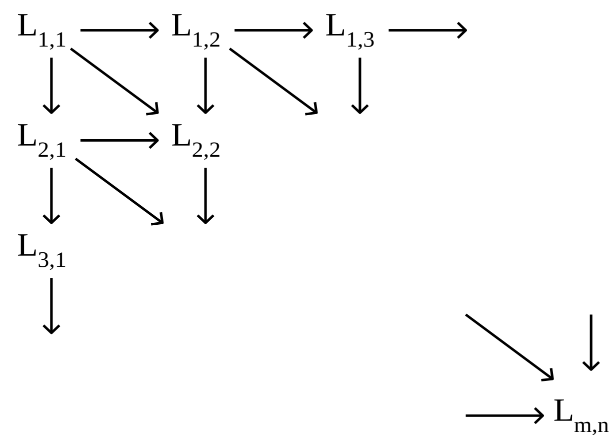
$$len(i,j)=\begin{cases}0 & \text{if } i=0 \text{ or } j=0,\\ len(i-1,j-1)+1 & \text{if } i,j>0 \text{ and } a_i=b_j,\\ \max(len(i,j-1),len(i-1,j)) & \text{if } i,j>0 \text{ and } a_i\neq b_j.\end{cases}$$

DP method

- Rule 4

procedure *LCS-Length(A, B)*

```
for i ← 0 to m do len(i,0) = 0
for j ← 1 to n do len(0,j) = 0
for i ← 1 to m do
    for j ← 1 to n do
        if a_i = b_j then [ len(i,j) = len(i-1,j-1)+1 ; prev(i,j) = "↖" ]
                else if len(i-1,j) ≥ len(i,j-1)
                    then [ len(i,j) = len(i-1,j) ; prev(i,j) = "↑" ]
                    else [ len(i,j) = len(i,j-1) ; prev(i,j) = "←" ]
return len and prev
```



DP method

- Rule 4
  - The dp result of the two string, "providence" and "president"

i	j 0	1 p	2 r	3 o	4 v	5 i	6 d	7 e	8 n	9 c	10 e
0	0	0	0	0	0	0	0	0	0	0	0
1 p	0	↖ 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
2 r	0	↑ 1	↖ 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2
3 e	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↖ 3	← 3	← 3	↖ 3
4 s	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↑ 3	↑ 3	↑ 3	↑ 3
5 i	0	↑ 1	↑ 2	↑ 2	↑ 2	↖ 3	← 3	↑ 3	↑ 3	↑ 3	↑ 3
6 d	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↖ 4	← 4	← 4	← 4	← 4
7 e	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↖ 5	← 5	← 5	↖ 5
8 n	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↖ 6	← 6	← 6
9 t	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↑ 6	↑ 6	↑ 6

DP method

- Rule 5
  - Trace the path

procedure *Output-LCS(A, prev, i, j)*

1 **if** $i = 0$ **or** $j = 0$ **then return**

2 **if** $prev(i, j) = \text{“} \nwarrow \text{”}$ **then** $\begin{cases} Output-LCS(A, prev, i-1, j-1) \\ \text{print} \quad a_i \end{cases}$

3 **else if** $prev(i, j) = \text{“} \uparrow \text{”}$ **then** *Output-LCS(A, prev, i-1, j)*

4 **else** *Output-LCS(A, prev, i, j-1)*

DP method

- Rule 5
  - The result "priden" of the two string, "providence" and "president"

i	j 0	1 p	2 r	3 o	4 v	5 i	6 d	7 e	8 n	9 c	10 e
0	0	0	0	0	0	0	0	0	0	0	0
1 p	0	↖ 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
2 r	0	↑ 1	↖ 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2
3 e	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↖ 3	← 3	← 3	↖ 3
4 s	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↑ 3	↑ 3	↑ 3	↑ 3
5 i	0	↑ 1	↑ 2	↑ 2	↑ 2	↖ 3	← 3	↑ 3	↑ 3	↑ 3	↑ 3
6 d	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↖ 4	← 4	← 4	← 4	← 4
7 e	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↖ 5	← 5	← 5	↖ 5
8 n	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↖ 6	← 6	← 6
9 t	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↑ 6	↑ 6	↑ 6

DP method

- More Example
  - Given string A = bacad, string B = accbadcb
  - The dp table can be optimized as the following figure and the longest common string can be backtraced by the table.

B

			a	c	c	b	a	d	c	b
		0	0	0	0	0	0	0	0	0
	b	0	0	0	0	1	1	1	1	1
	a	0	1	1	1	1	2	2	2	2
A	c	0	1	2	2	2	2	2	3	3
	a	0	1	2	2	2	3	3	3	3
	d	0	1	2	2	2	3	4	4	4

DP method

- Exercise
  - Write a program that find the length of the LCS for two given sequences.
  - Note:
    - Please use the dynamic programming as the practice.
    - The number of the element in the given sequence will not exceed 1000.

- Review
  - Time complexity O(?)
  - Space complexity O(?)
  - Compare with the brute force method.

Just Practice

- Practice
  - NOJ 31 LCS Problem

Homework 8

- UVA (**total 23 problems**)
  - 103, 108, 111, 231, 437, 481, 497, 507, 531, 836, 10066, 10131, 10192, 10252, 10405, 10534, 10635, 10684, 10723, 10755, 10827, 10949, 11582

Thank You For Attention!

made by electron, free999, rabbit125