Australian National University

COMP3600/6466 Algorithms

Lecture 13

S2 2016

Dr. Hassan Hijazi

Prof. Weifa Liang

LCS continued

3. Compute the value of an optimal solution

Use memoization or bottom-up DP!

Let $X = \{x_1, x_2, \ldots, x_m\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ be the input sequences.

Let $c[i, j]$ denote the length of an LCS of the sequences X_i and Y_j.

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

Bottom-Up Approach

3. Compute the value of an optimal solution

Use memoization or bottom-up DP!

What is the bottom(smallest) subproblem?

$$(X_1, Y_1)$$

How many unique subproblems?

Two nested for loops

$$n \times m$$

Number of nodes in the subproblem graph

How many sub-subproblems do I need to consider for a given subproblem?

If, elseif, else

At most 3

Number of edges connecting two nodes

Bottom-Up Approach

3. Compute the value of an optimal solution

Use memoization or bottom-up DP!

LCS-LENGTH(X, Y)

```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

$\Theta(1)$
$\Theta(1)$
$\Theta(nm)$
$\Theta(m)$

$\Theta(n)$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} c = c \times nm = \Theta(nm)$$

$$T(n,m) = \Theta(nm)$$

Dynamic Programming

3. Compute the value of an optimal solution

Use memoization or bottom-up DP!

	j	0	1	2	3	4	5	6
i		yj	C	A	C	C	G	G
0	xi							
1	A							
2	C							
3	G							
4	G							

Australian National University

Dynamic Programming

3. Compute the value of an optimal solution

Use memoization or bottom-up DP!

	j	0	1	2	3	4	5	6
i		yj	C	A	C	C	G	G
0	xi	0	0	0	0	0	0	0
1	A	0	0-up	1-diag	1-left	1-left	1-left	1-left
2	C	0	1-diag	1-up	2-diag	2-diag	2-left	2-left
3	G	0	1-up	1-up	2-up	2-up	3-diag	3-diag
4	G	0	1-up	1-up	2-up	2-up	3-diag	4-diag

Dynamic Programming

3. Compute the value of an optimal solution

Use memoization or bottom-up DP!

	j	0	1	2	3	4	5	6
i		yj	C	A	C	C	G	G
0	xi	0	0	0	0	0	0	0
1	A	0	0-up	1-diag	1-left	1-left	1-left	1-left
2	C	0	1-diag	1-up	2-diag	2-diag	2-left	2-left
3	G	0	1-up	1-up	2-up	2-up	3-diag	3-diag
4	G	0	1-up	1-up	2-up	2-up	3-diag	4-diag

Exercise 12.1

3. Compute the value of an optimal solution

Use memoization or bottom-up DP!

	j	0	1	2	3	4	5	6
i		yj	C	A	G	C	A	G
0	xi							
1	A							
2	G							
3	G							
4	G							

Greedy!

greed
/griːd/

noun

intense and selfish desire for something, especially wealth, power, or food.
"mercenaries who had allowed greed to overtake their principles"
synonyms: avarice, acquisitiveness, covetousness, rapacity, graspingness, cupidity, avidity, possessiveness, materialism; More

Greedy Algorithms!

Two key ingredients:

1. Optimal substructure

2. A unique choice per subproblem: The greedy one!

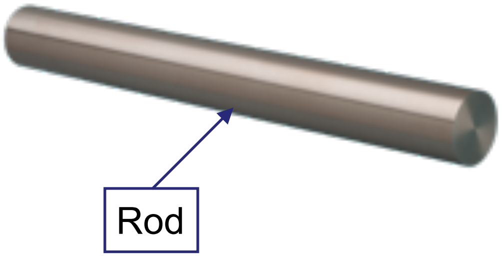
The greedy one:

The one that locally maximises/minimises your objective function

Take a decision that maximises/minimises your objective function NOW, ignore what might happen in the future.

Let’s go back in time

Application 1: Rod Cutting Problem



length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Given a rod of length n meters and a table of prices p_i for $i \in \{1, 2, \ldots, n\}$, determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces.

Australian National University

Application 1: Rod Cutting Problem

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

The greedy decision:

The one that locally maximises/minimises your objective function

To fairly compare all decisions, you first need to convert them in per unit

length i	1	2	3	4	5	6
price p_i	1	5	8	9	10	17
per unit price pu_i	1	2.5	2.66	2.25	2	2.83

$$\text{Let } n = 4$$

Greedy solution = cut 3 then cut 1 with total revenue $= 8 + 1 = 9$.

Optimal solution = cut 2 then cut 2 with total revenue $= 5 + 5 = 10$.

Application 2: Matrix Multiplication

Given the chain $A_1 \cdot A_2 \ldots \cdot A_n$, fully parenthesise/split it in a way that minimises the number of scalar multiplications.

$$((A_1 \cdot A_2) \cdot (A_3 \cdot A_4))$$
$$(A_1 \cdot (A_2 \cdot (A_3 \cdot A_4)))$$
$$(A_1 \cdot ((A_2 \cdot A_3) \cdot A_4))$$
$$(((A_1 \cdot A_2) \cdot A_3) \cdot A_4)$$
$$((A_1 \cdot (A_2 \cdot A_3)) \cdot A_4)$$

Application 2: Matrix Multiplication

Given the chain $A_1 \cdot A_2 \ldots \cdot A_n$, fully parenthesise/split it in a way that minimises the number of scalar multiplications.

The greedy decision:

The one that locally maximises/minimises your objective function

The greedy decision:

Pick the pair that generates the minimal number of operations.

Let $A_1 = 5 \times 4$, $A_2 = 4 \times 2$ and $A_3 = 2 \times 4$.

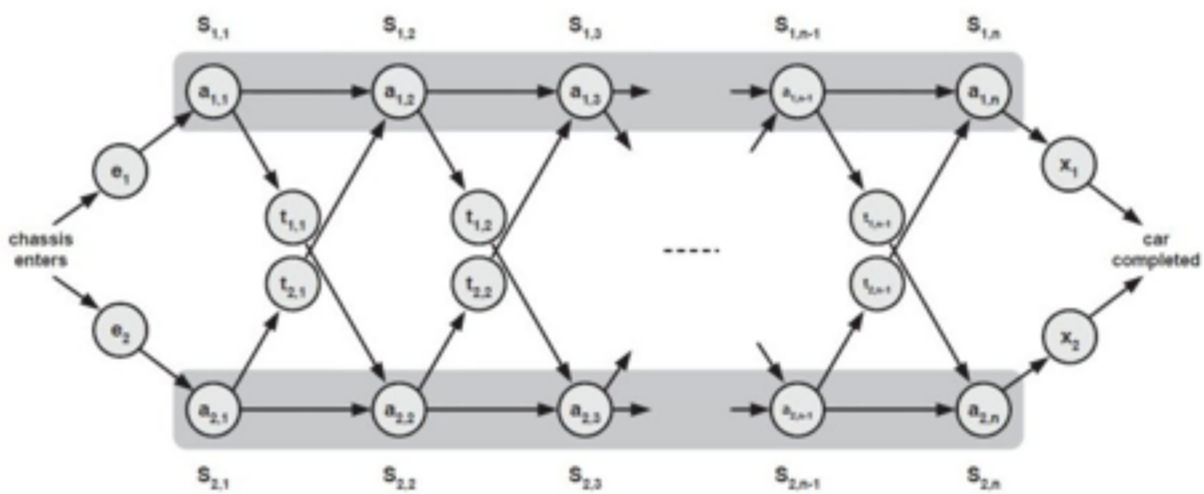
The pair $(A_2 A_3)$ is locally minimal with $4 \times 2 \times 4 = 32$ operations.

But the overall solution $(A_1(A_2 A_3))$ costs $32 + 5 \times 4 \times 4 = 132$ which is not better than $((A_1 A_2) A_3)$ which costs $5 \times 4 \times 2 + 5 \times 2 \times 4 = 40 + 40 = 80$.

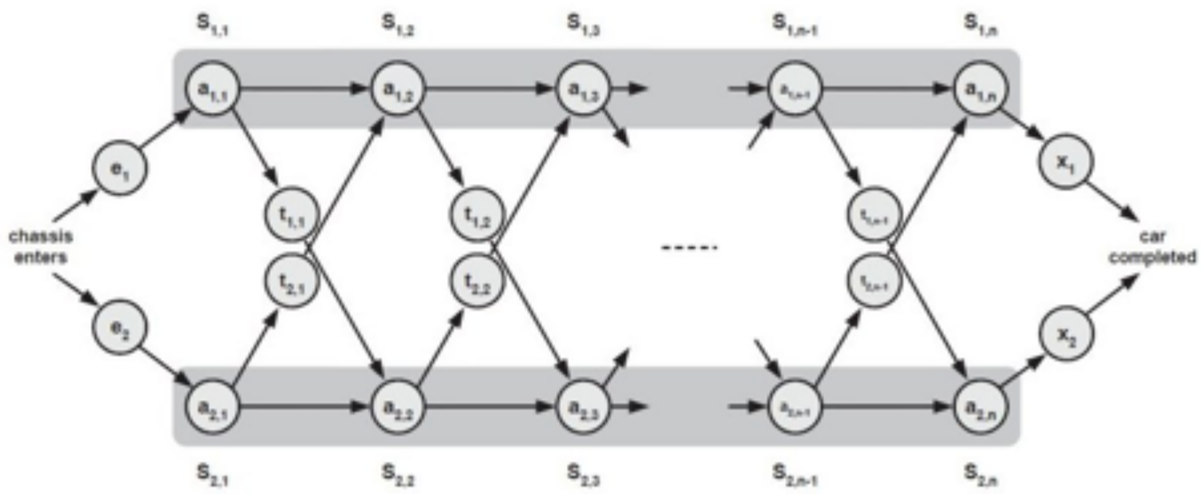
Application 3: Car Assembly

Problem statement: Assemble cars in a minimal amount of time

Equivalent to finding a shortest path from the start to the finish



Application 3: Car Assembly



Consider the case where $e_1 = 1$, $e_2 = 2$, $a_{1,1} = 999999$ and all other costs are 1.

Application 4: DNA Similarity or LCS

Definition: The similarity score of two sequences is the length of the longest common subsequence

Problem statement: Given two DNA strands, compute their similarity score

Example:

S_1 = ACCGGTAA

S_2 = GTCGCTTGT

Common subsequence 1 = CC

Common subsequence 2 = CGGT

Application 4: DNA Similarity or LCS

$S_1 = ACCGGTAA$

$S_2 = GTCGCTTGT$

The greedy decision:

The one that locally maximises/minimises your objective function

The greedy decision:

Iterate on both sequences, every time we find a common element we add it to the solution.

Greedy solution: CGT

Optimal solution: CGGT

Application 5: The Activity-Selection

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Activity i has a **start time** s_i and a **finish time** f_i

Problem: select a maximum-size subset of *mutually compatible* activities.

Activities a_i and a_j are **compatible** if $s_i \geq f_j$ or $s_j \geq f_i$.

Example: The subset $\boxed{\{a_3, a_9, a_{11}\}}$ consists of mutually compatible activities

The subset $\boxed{\{a_1, a_4, a_8, a_{11}\}}$ is a better solution.

The Activity-Selection Problem

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Increasing finish time

Problem: select a maximum-size subset of *mutually compatible* activities.

Optimal substructure?

Introduce activities a_0 with $s_0 = f_0 = 0$ and a_{n+1} with $s_{n+1} = f_{n+1} = f_n$.
Let $S_{i,j}$ denote the set of activities that start after the end of activity a_i and finish before the beginning of activity a_j.

Example: $S_{0,4} = \{a_1\}$, $S_{0,n+1} = \{a_1, \ldots, a_n\}$, $S_{1,11} = \{a_4, a_6, a_7, a_8, a_9\}$.

The Activity-Selection Problem

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Increasing finish time

Problem: select a maximum-size subset of *mutually compatible* activities.

Optimal substructure?

Let A_{ij} denote the maximum set of mutually compatible activities in S_{ij}.

Consider $a_k \in A_{ij}$

It is easy to see that $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$.

$$\Longrightarrow \ |A_{ij}| = |A_{ik}| + |A_{kj}| + 1.$$

The Activity-Selection Problem

a_k is not know in advance!

Let $c[i,j]$ denote the size of an optimal solution to S_{ij}

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

$$\text{Optimal solution value} = c[0, n+1]$$