Australian National University

COMP3600/6466 Algorithms

Lecture 12

S2 2016

Dr. Hassan Hijazi

Prof. Weifa Liang

Australian National University

Dynamic Programming: Summary

1. Characterise the optimal substructure or show that none exists
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution

   Use memoization or bottom-up DP!

4. Construct an optimal solution from computed information

Australian National University

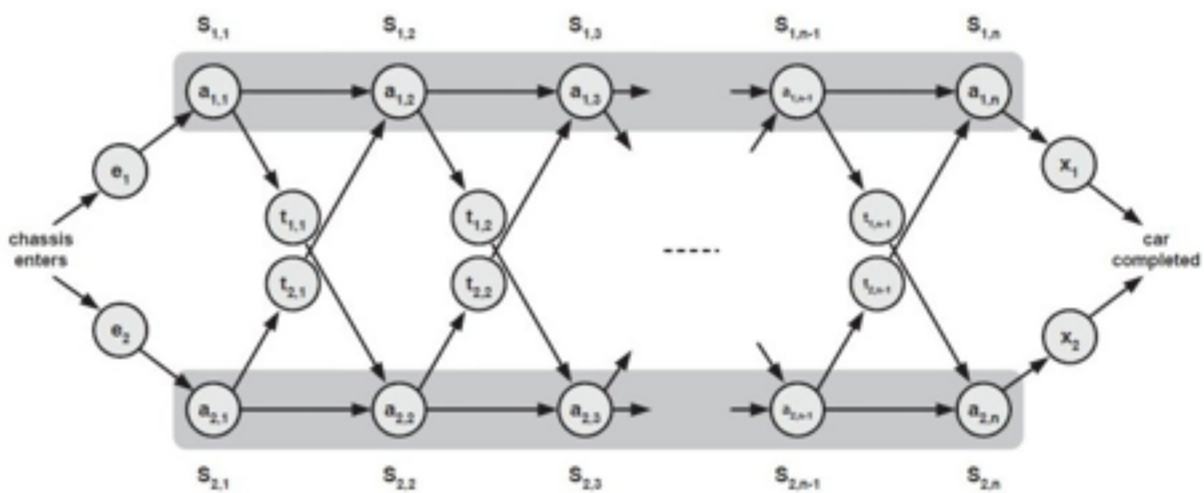
Application 3: Car Assembly

Application 3: Car Assembly

There are two parallel assembly lines for assembling a car.

Each lines can perform the same sequence of operations but at different rates.

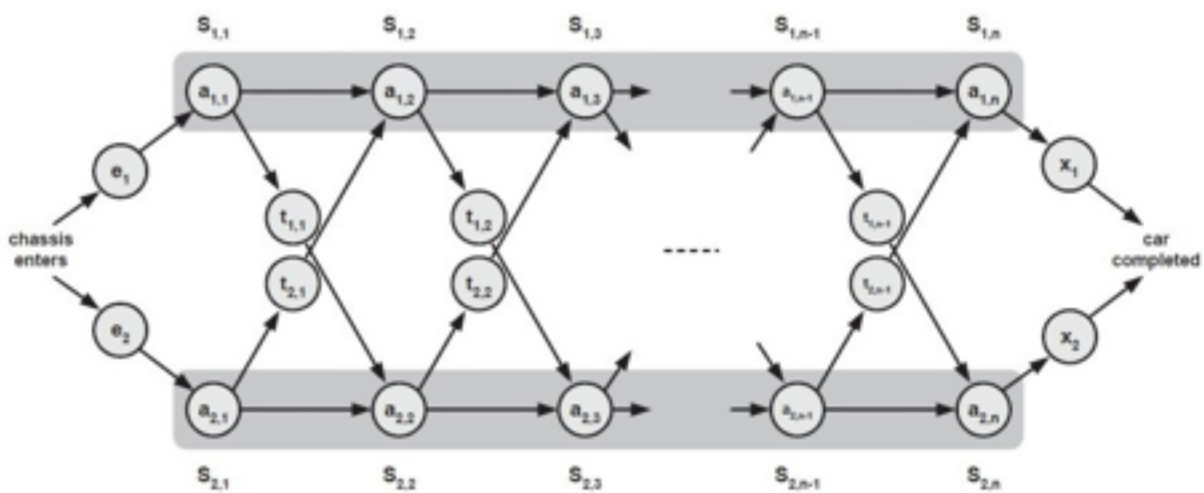
We can transfer the car from one line to the other with a predefined time cost.



Application 3: Car Assembly

Problem statement: Assemble cars in a minimal amount of time

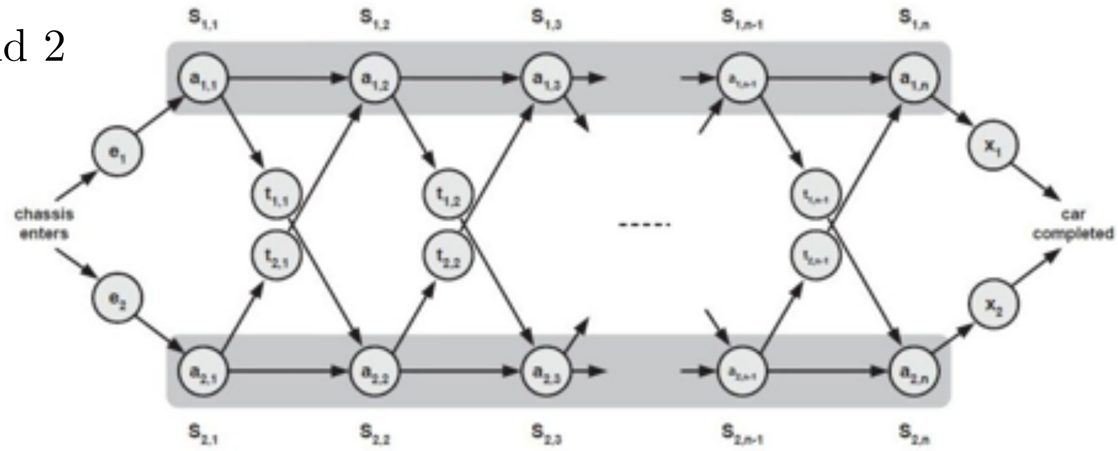
Equivalent to finding a shortest path from the start to the finish



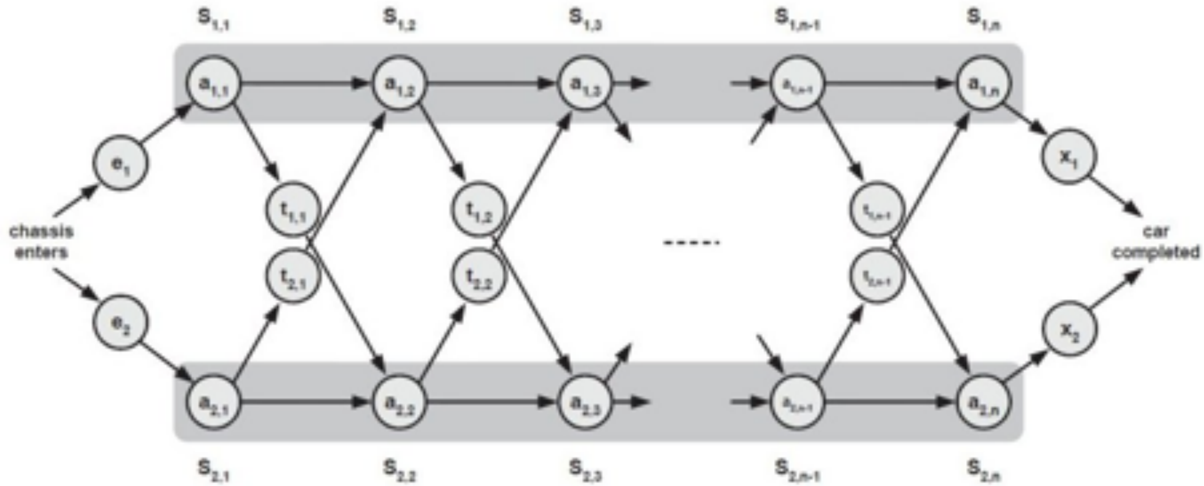
Notations

Each assembly line has n stations. Notation:

- $S_{1,j}$, $S_{2,j}$: jth station in lines 1 and 2
- $a_{1,j}$, $a_{2,j}$: processing times at stations $S_{1,j}$ and $S_{2,j}$
- $t_{1,j}$, $t_{2,j}$: time to transfer from $S_{1,j}$ to $S_{2,j+1}$ and $S_{2,j}$ to $S_{1,j+1}$
- e_1,e_2: entry time for lines 1 and 2
- x_1, x_2: exit time for lines 1 and 2



Brute force enumeration



Two choices at every station: 2^n possible solutions!

Dynamic Programming: How To

1. Characterise the optimal substructure or show that none exists

The problem has an optimal substructure:
If S_{ij} is on the optimal path, then both the subpaths leading to and leaving from S_{ij} are optimal subpaths

Optimal Substructure Proof: How To

The Cut and Paste method

> The problem has an optimal substructure:
> If S_{ij} is on the optimal path, then both the subpaths leading to and leaving from S_{ij} are optimal subpaths

Proof.

1. Assume that the subpath leading to S_{ij} is not optimal

2. If I replace this subpath with an optimal one, I can improve the cost of the overall solution, which contradicts the optimality assumption on the global path.

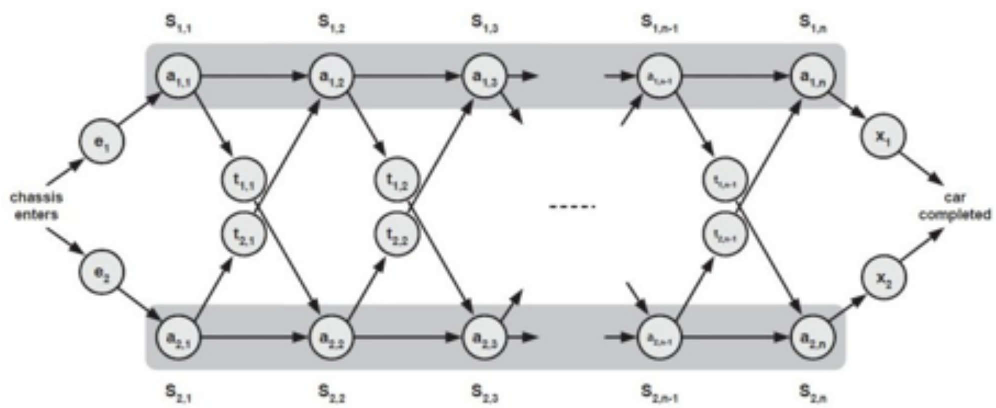
Dynamic Programming: How To

2. Recursively define the value of an optimal solution

Let $f_1(j)$ (resp. $f_2(j)$) denote the fastest way to get a chassis from the starting point to station S_{1j} (resp. S_{2j}).

$$f_1(j) = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1, \\ \min\left(f_1(j-1) + a_{1,j},\ f_2(j-1) + t_{2,j-1} + a_{1,j}\right) & \text{if } j \geq 2. \end{cases}$$

$$f_2(j) = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1, \\ \min\left(f_2(j-1) + a_{2,j},\ f_1(j-1) + t_{1,j-1} + a_{2,j}\right) & \text{if } j \geq 2. \end{cases}$$



Exercise 11.4

Write an iterative dynamic programming algorithm that computes the optimal solution and its value.

$$f^* = \min\left(f_2(n) + x_2,\ f_1(n) + x_1\right)$$

Application 4: DNA

https://www.youtube.com/watch?v=uXdzuz5Q-hs

Application 4: DNA

DNA consists of a string of molecules called ***bases***, where the possible bases are adenine, guanine, cytosine, and thymine.

We can express a strand of DNA as a sequence of letters over the finite set {A,C,G,T}.

S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA

Application 4: DNA

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

Equivalent Problem:
Given two sequences, find the Longest Common Subsequence (LCS)

$\boxed{\text{How many possible common subsequences?}}$

Number of subsequences in the shortest sequence

$$2^{\min(m,n)}$$

For every element in the shortest sequence you have 2 possible choices: include it in the subsequence or not

Dynamic Programming

1. Characterise the optimal substructure or show that none exists

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

Let $Z = \{z_1, z_2, \ldots, z_k\}$ be any LCS of X and Y.

If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.

Proof. If $z_k \neq x_m$ then we could append x_m to Z to obtain a common subsequence of X and Y of length $k+1$, contradicting the supposition that Z is a longest common subsequence of X and Y. If Z_{k-1} is not an LCS of X_{m-1} and Y_{n-1}, then using the cut-and-paste method, we can improve Z by replacing Z_{k-1} with an LCS.

Dynamic Programming

1. Characterise the optimal substructure or show that none exists

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

Let $Z = \{z_1, z_2, \ldots, z_k\}$ be any LCS of X and Y.

If $x_m \neq y_n$, then there are two options:

1. $z_k \neq x_m$ which implies that Z_k is an LCS of X_{m-1} and Y_n.
2. $z_k \neq y_n$ which implies that Z_k is an LCS of X_m and Y_{n-1}.

Proof. Cut-and-paste method.

Dynamic Programming

2. Recursively define the value of an optimal solution

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

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

Dynamic Programming

3. Compute the value of an optimal solution

Algorithm 1 RecLCS(X,Y,i,j)

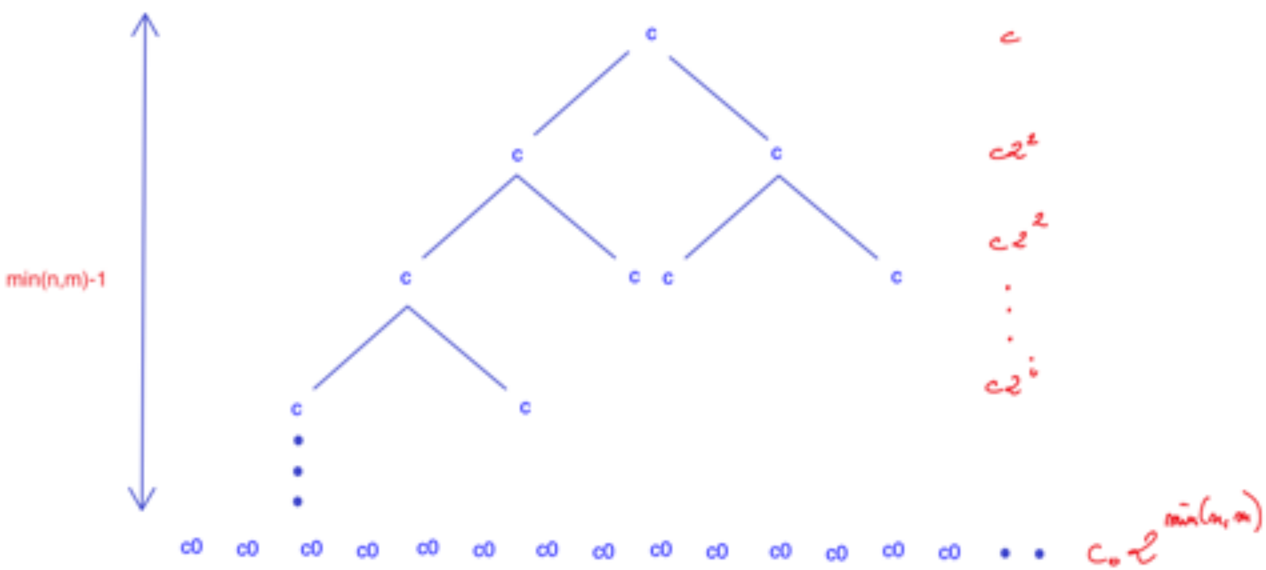
1: **if** $i = 0$ or $j = 0$ **then return** 0
2: **if** $x_i = y_j$ **then return** RecLCS(X,Y,$i-1$,$j-1$)+1
3: **return** $\max(\text{RecLCS}(X, Y, i-1, j), \text{RecLCS}(X, Y, i, j-1))$

Give a tight asymptotic bound on the running time of RecLCS

$$\begin{cases} T(n,m) = c_0 & \text{if } n = 0 \text{ or } m = 0, \\ T(n,m) = c + T(n-1, m-1) & \text{if } x_n = y_m, \\ T(n,m) = c + T(n-1, m) + T(n, m-1) & \text{otherwise.} \end{cases}$$

Dynamic Programming

$$\begin{cases} T(n,m) = c_0 & \text{if } n = 0 \text{ or } m = 0, \\ T(n,m) = c + T(n-1, m-1) & \text{if } x_n = y_m, \\ T(n,m) = c + T(n-1,m) + T(n,m-1) & \text{otherwise.} \end{cases}$$



Dynamic Programming

$$\begin{cases} T(n,m) = c_0 & \text{if } n = 0 \text{ or } m = 0, \\ T(n,m) = c + T(n-1, m-1) & \text{if } x_n = y_m, \\ T(n,m) = c + T(n-1, m) + T(n, m-1) & \text{otherwise.} \end{cases}$$

The worst case running time at level i of the recursion tree is:

$$c \times 2^i, \; i \in \{0, 1, \ldots, \min(n,m) - 1\}$$

The total running time of intermediate levels:

$$c \sum_{i=0}^{\min(n,m)-1} 2^i = c\left(2^{\min(n,m)} - 1\right) = \Theta\left(2^{\min(n,m)}\right)$$

The number of leafs in the recursion tree is $2^{\min(n,m)}$.

$$T(n,m) = \Theta\left(2^{\min(n,m)}\right)$$

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?

$$n \times m$$

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

At most 2

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
```

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							

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	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	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							