CS583 Lecture 04
Dynamic Programming

Jyh-Ming Lien

some materials here are based on Prof. Shehu, and Prof. Wang's past lecture notes

Intro

- A term coined by Richard Bellman in the 1940s

(Image from ieee.org. Richard Bellman, 1920 - 1984)

- Some problems solved by dynamic programming
  - Longest increasing subsequences
  - Fibonacci number
  - Knapsack problem
  - All-pairs shortest path problem (Floyd's algorithm)
  - Optimal binary search tree problem
  - Multiplying a sequence of matrices
  - String matching (or DNA sequence matching), where we search for the string closest to the pattern
  - Convex decomposition of polygons
  - ...

Intro

- what kind of problems can be solved using dynamic programming?
  - optimization problems
  - the optimal solution can be obtained from the optimal solutions of the subproblems
  - overlapping subproblems (i.e., the same subproblem may appear many times)

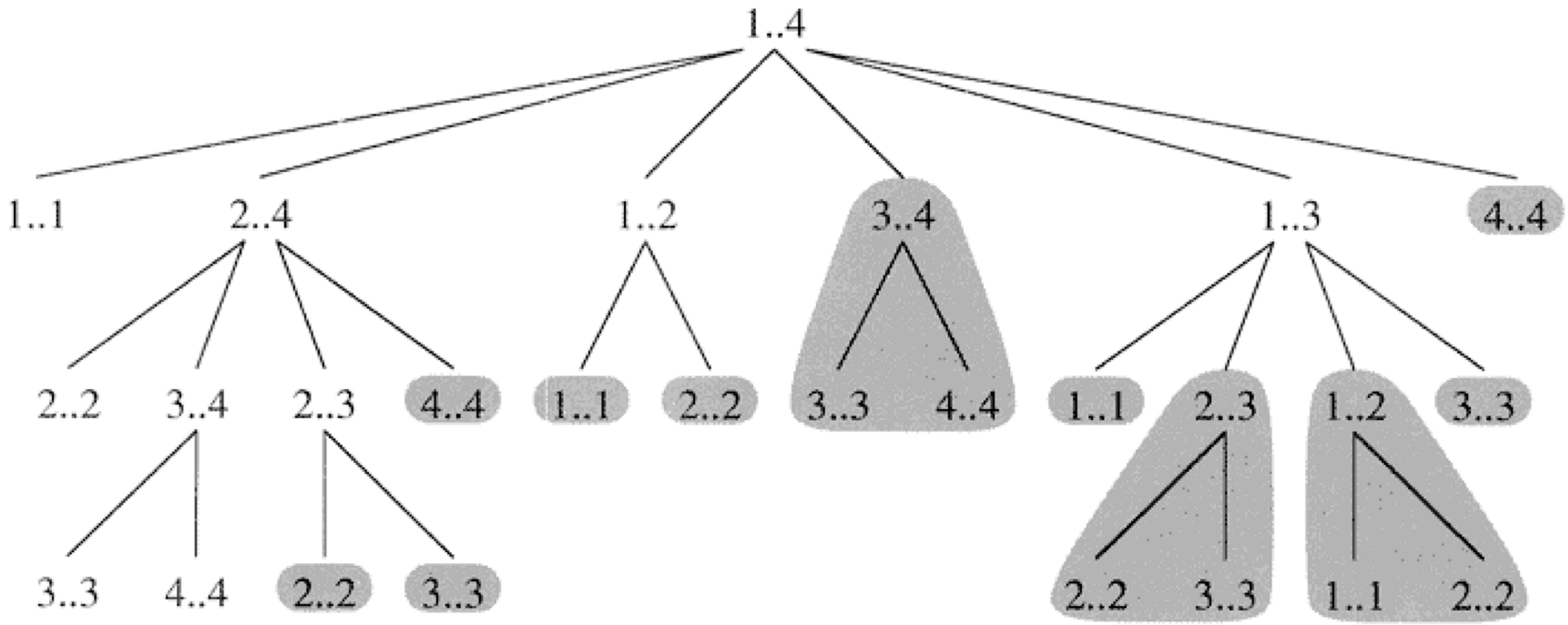
Intro

- How to solve dynamic programming problems?
  - define what your "sub-problem"
  - solve the sub-problems recursively
  - store solutions to the sub-problems (and use them later)

Intro

- matrix chain multiplication

compute $A_1 \times A_2 \times A_3 \times A_4$



LCS

- Longest common sequence (LCS)

Given two sequences $x[1 \ldots m]$ and $y[1 \ldots n]$, find a longest subsequence common to them both

- example: $X = [A, B, C, B, D, A, B]$
  $Y = [B, D, C, A, D, A]$

- Brute force time complexity?

LCS

- Dynamic programming
  - sub-problem $c[i, j] = |\mathrm{LCS}(\mathrm{x}[1 \ldots \mathrm{i}], \mathrm{y}[1 \ldots \mathrm{j}])|$
  - define (and solve) recursively

$$c[i, j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x[i] = y[j] \\ \max\{c[i-1, j], c[i, j-1]\} & \text{otherwise} \end{cases}$$

LCS

- example: $X = [A, B, C, B, D, A, B]$
  $Y = [B, D, C, A, D, A]$

$\{i, j\}$	1	2	3	4	5	6	7
1							
2							
3							
4							
5							
6							

Matrix multiplication

- Given four matrices, $A[50 \times 20]$, $B[20 \times 1]$, $C[1 \times 10]$, $D[10 \times 100]$, we wish to compute $A \times B \times C \times D$.
- If we compute $(((A \times B) \times C) \times D)$, we will perform x multiplications?
- What about $((A \times B) \times (C \times D))$?
- How do we find the best way to group matrices so that the number of multiplications is minimized?

Matrix multiplication

- a pair of parentheses group two matrices
- The final matrix represents the root
- Ex: (((AB)C)D) and ((AB)(CD))

Matrix multiplication

- Example: $A[50 \times 20]$, $B[20 \times 1]$, $C[1 \times 10]$, $D[10 \times 100]$
  compute $A \times B \times C \times D$
  - subproblems
    - with two matrices
    - with three matrices
    - with four matrices

Matrix multiplication

- Sub-problem
  - cost of multiplying matrix i to j

$$C(i,j) = \min_{i \leq k < j} \{C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j\}$$

Matrix multiplication

- example: $A[50 \times 20]$, $B[20 \times 1]$, $C[1 \times 10]$, $D[10 \times 100]$

0	$j = 1$	$j = 2$	$j = 3$	$j = 4$
$i = 1$	0			
$i = 2$		0		
$i = 3$			0	
$i = 4$				0

Optimal BST

- problem:

Given a list of **sorted** values (associated with probabilities), find a binary search tree (BST) whose total weight is minimized

- example: A (0.1), B (0.2), C (0.4), D (0.3)

- What is the time complexity of a brute force algorithm?

Optimal BST

- sub-problem
  - cost of a tree with values from i to j

$$C[i,j] = \min_{i \le k \le j} \{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} p_s$$

Optimal BST

- example: A (0.1), B (0.2), C (0.4), D (0.3)

$\{i, j\}$	0	1	2	3	4
1					
2					
3					
4					
5					

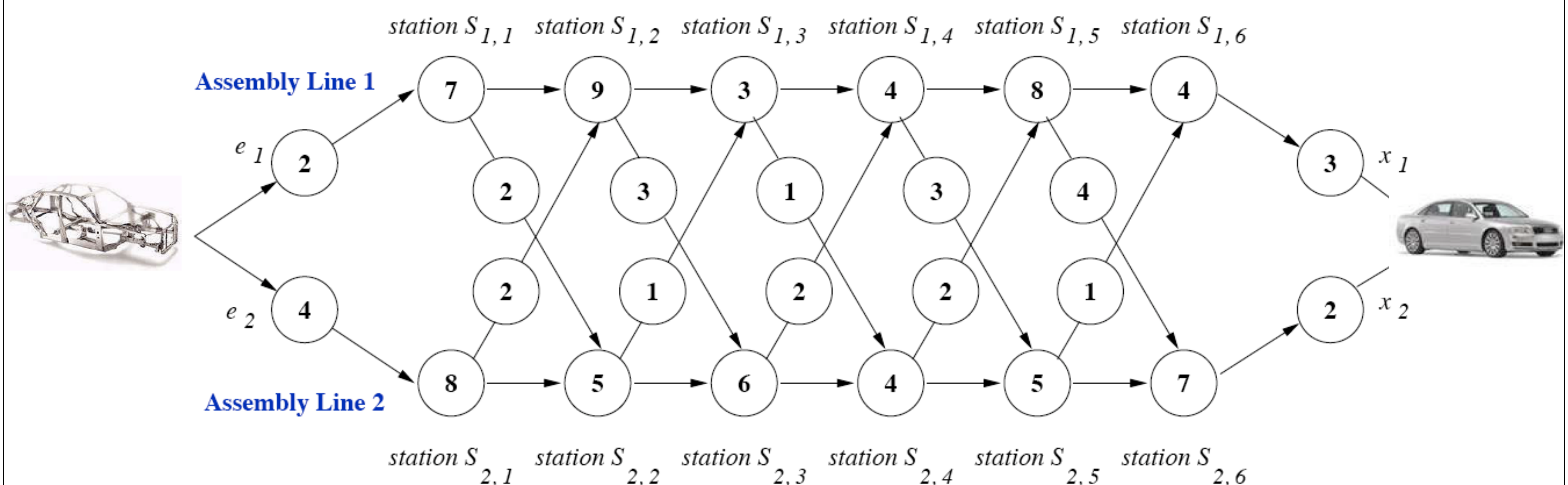
Optimal BST

Algorithm 0.1: $\textsc{OptBST}(A[1 \cdots n])$

$$
\begin{aligned}
&\textbf{for } i \leftarrow \{1 \cdots n\} \\
&\quad \textbf{do } \begin{cases} C[i, i-1] \leftarrow 0 \\ C[i, i] \leftarrow A[i] \end{cases} \\
&\textbf{for } d \leftarrow \{1 \cdots n-1\} \\
&\quad \textbf{do } \begin{cases} \textbf{for } i \leftarrow 1 \cdots n-d \\ \quad \textbf{do } \begin{cases} j \leftarrow i + d \\ min \leftarrow \infty \\ \textbf{for } k \leftarrow \{i \cdots j\} \\ \quad \textbf{do } \begin{cases} \textbf{if } C[i, k-1] + C[k+1, j] < min \\ \quad \textbf{then } min \leftarrow C[i, k-1] + C[k+1, j] \end{cases} \\ C[i, j] \leftarrow min + \sum_{s=i}^{j} A[s] \end{cases} \end{cases}
\end{aligned}
$$

Assembly Line

- schedule an optimal assembly



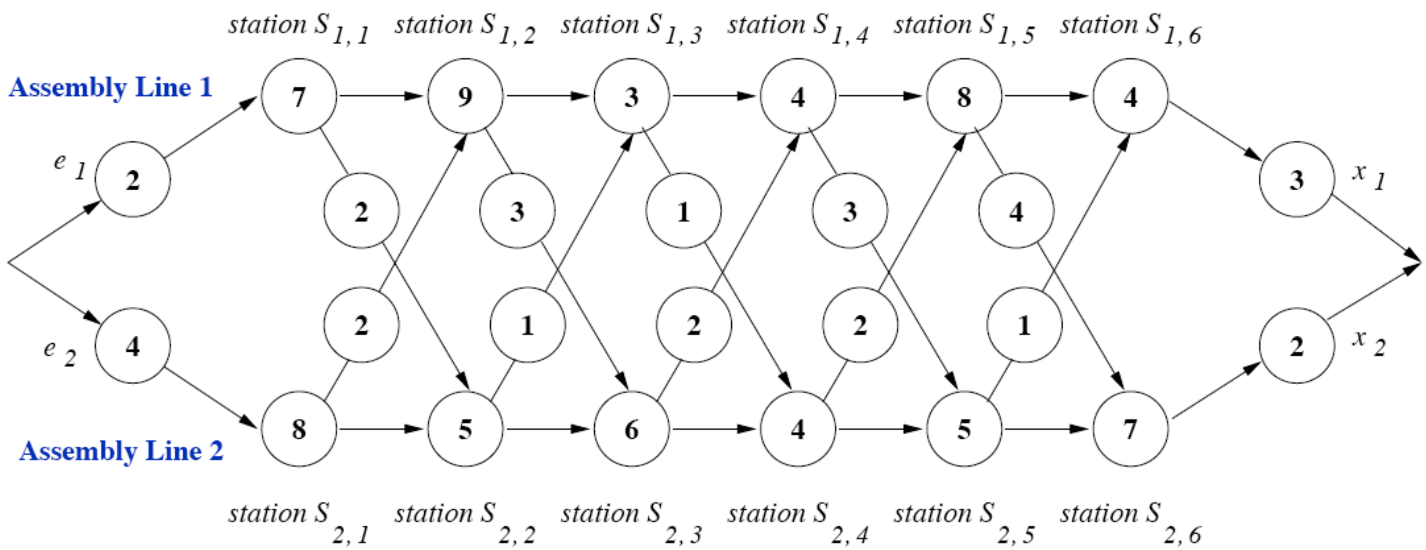
each number is a circle represent delay

Assembly Line

- sub-problems
  - $C(i,j)$ is the optimal solution up to the station $S_{i,j}$
  - define the sub-problem recursively
    - $C(1,j) = \min\{C(1,j-1), C(2,j-1) + t_{2,j}\} + a_{1,j}$
    - $C(2,j) = \min\{C(2,j-1), C(1,j-1) + t_{1,j}\} + a_{2,j}$
  - Final solution
    - $C = \min\{C(1,n) + x_1, C(2,n) + x_2\}$

Assembly Line

- Example



$j =$	1	2	3	4	5	6
$C(1, j)$						
$C(2, j)$						

- $C = \min\{ \qquad , \qquad \} =$