GUC German University in Cairo

CSEN 703 - Analysis and Design of Algorithms

Lecture 7 - Dynamic Programming II

Dr. Nourhan Ehab
nourhan.ehab@guc.edu.eg

Department of Computer Science and Engineering
Faculty of Media Engineering and Technology

In the Previous Lecture

GUC German University in Cairo

- Dynamic Programming is all about learning from your past. We save the intermediate results for future reference.

In the Previous Lecture

GUC German University in Cairo

- Dynamic Programming is all about learning from your past. We save the intermediate results for future reference.
- DP is used when a problem exhibits optimal substructure and overlapping subproblems.

In the Previous Lecture

- Dynamic Programming is all about learning from your past. We save the intermediate results for future reference.
- DP is used when a problem exhibits optimal substructure and overlapping subproblems.
- Two approaches:
  1. Top-Down Approach: write a regular recursive function, but modify it to remember the values of what it already computed. ⇒ Memoization
  2. Bottom-Up Approach: solve the smaller subproblems first and store their results in a table so that when solving bigger problems we are sure we solved all the prerequisites (which can be acquired from the table). ⇒ Tabulation

DP vs Greedy Algorithms

DP	Greedy
Makes a choice at each step after solving subproblems.	Makes a choice at each step before solving subproblems.
Considers all possibilities.	Considers only one branch.
Always Optimal.	Suboptimal.

More DP is coming!

Outline

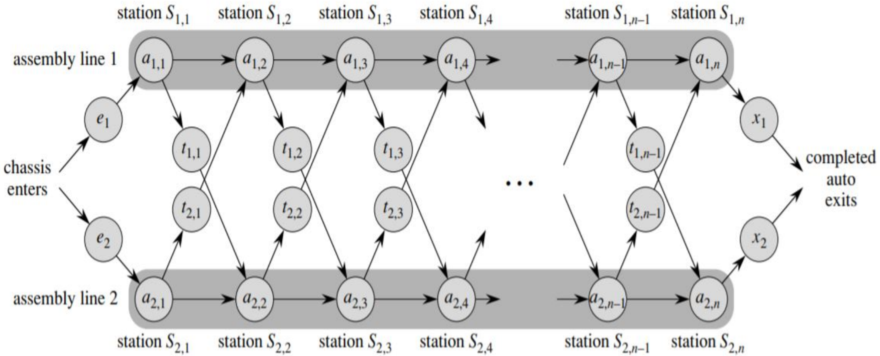
1. Assembly Line Scheduling
2. Longest Common Subsequence
3. Optimal Substructure
4. Recap

Problem Statement

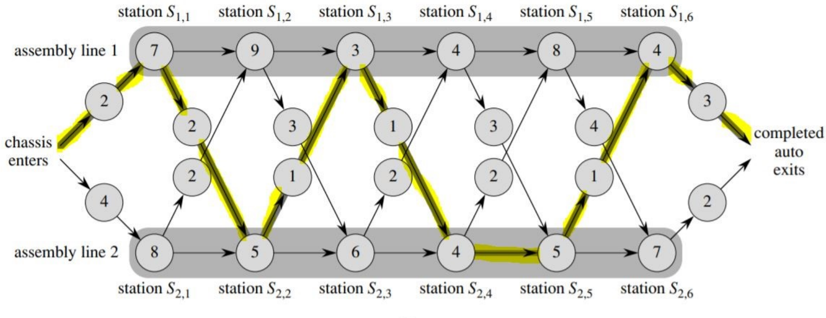
Example

An automotive company produces cars in a factory that has two assembly lines. A vehicle chassis enters an assembly line, and has parts added to it at n different stations. The finished vehicle exits at the end of the line. Normally, once a chassis enters an assembly line, it passes through that line only. The time it takes a vehicle to move across stations in the same assembly line is negligible. For rush orders, however, the chassis still passes through the n stations in order, but the factory manager may switch the vehicle from one assembly line to the other after any station. However, this incurs some transfer time. The problem is to determine which stations to choose from line 1 and which to choose from line 2 in order to minimize the total time through the factory for one vehicle.

Assembly-line Scheduling

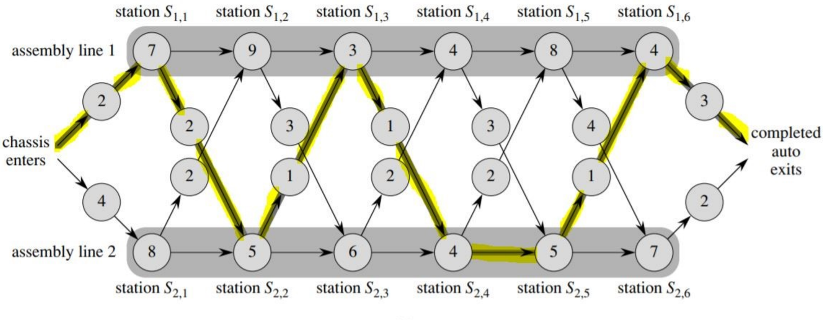


Assembly-line Scheduling



(a)

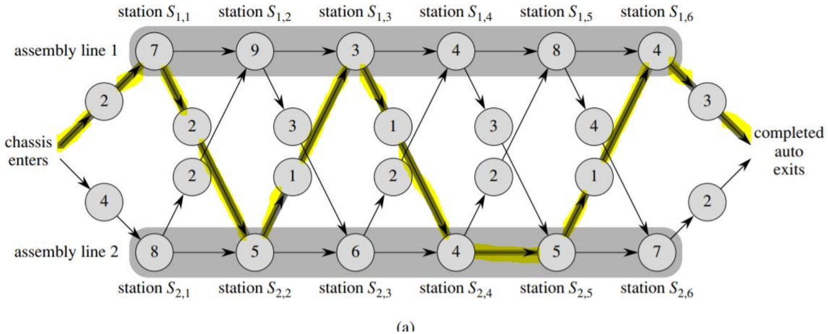
Assembly-line Scheduling



(a)

Complexity is $O(2^n)$ for brute force.

Assembly-line Scheduling



(a)

Complexity is $O(2^n)$ for brute force. Can we use DP?

Optimal Substructure

- What is the fastest way through station $S_{1,j}$?

Optimal Substructure

- What is the fastest way through station $S_{1,j}$?
- If $j = 1$, easy: just determine how long it takes to get through $S_{1,1}$!

Optimal Substructure

- What is the fastest way through station $S_{1,j}$?
- If $j = 1$, easy: just determine how long it takes to get through $S_{1,1}$!
- If $j \geq 2$, there are two choices to get to $S_{1,j}$:
  1. Through $S_{1,j-1}$, then directly to $S_{1,j}$.
  2. Through $S_{2,j-1}$, then transfer over to $S_{1,j}$.

Optimal Substructure

- What is the fastest way through station $S_{1,j}$?
- If $j = 1$, easy: just determine how long it takes to get through $S_{1,1}$!
- If $j \geq 2$, there are two choices to get to $S_{1,j}$:
  1. Through $S_{1,j-1}$, then directly to $S_{1,j}$.
  2. Through $S_{2,j-1}$, then transfer over to $S_{1,j}$.

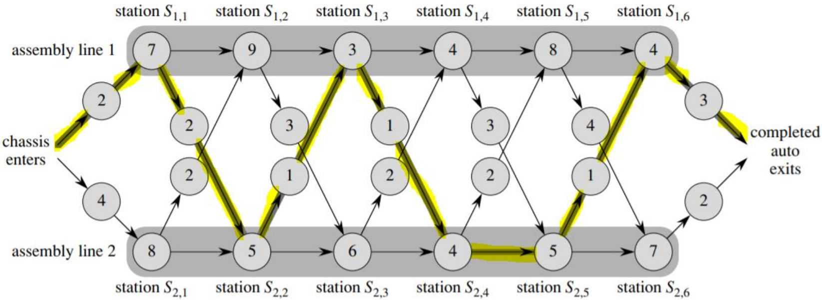
Recursive Formulation

$$f_i[1] = e_1 + a_{i,1} \;\; for \;\; i = 1, 2$$

$$f_1[j] = min\{f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}\}$$

$$f_2[j] = min\{f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}\}$$

Optimal Substructure



(a)

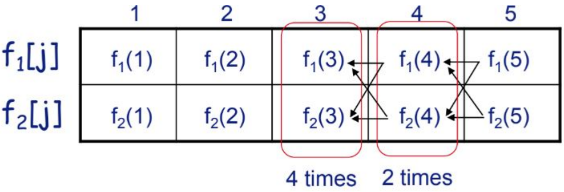
j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

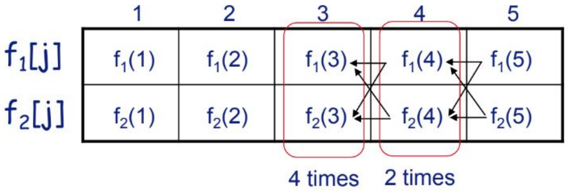
j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

Overlapping Subproblems

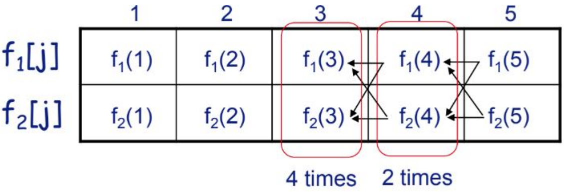


Overlapping Subproblems



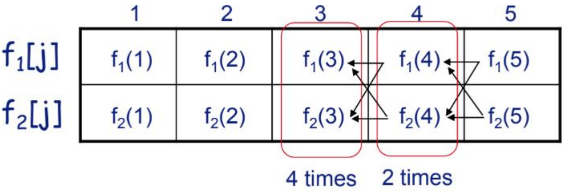
Let $r_i(j)$ be the number of references to $f_i[j]$.

Overlapping Subproblems



Let $r_i(j)$ be the number of references to $f_i[j]$.
Claim: $r_i(j) = 2^{n-j}$

Overlapping Subproblems

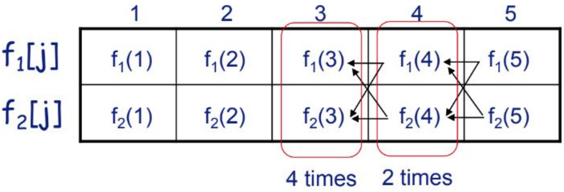


Let $r_i(j)$ be the number of references to $f_i[j]$.
Claim: $r_i(j) = 2^{n-j}$

Proof by Induction:

- Base case (j=n): $2^{n-j} = 2^{n-n} = 1$.

Overlapping Subproblems

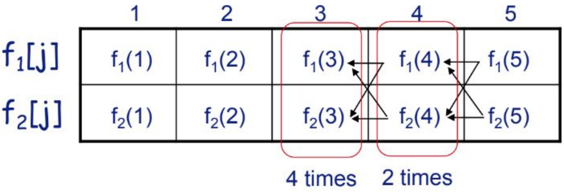


Let $r_i(j)$ be the number of references to $f_i[j]$.
Claim: $r_i(j) = 2^{n-j}$

Proof by Induction:

- Base case (j=n): $2^{n-j} = 2^{n-n} = 1$.
- Induction Hypothesis: $r_i(j+1) = 2^{n-(j+1)}$.

Overlapping Subproblems



Let $r_i(j)$ be the number of references to $f_i[j]$.
Claim: $r_i(j) = 2^{n-j}$

Proof by Induction:

- Base case (j=n): $2^{n-j} = 2^{n-n} = 1$.
- Induction Hypothesis: $r_i(j+1) = 2^{n-(j+1)}$.
- Induction Step: $r_i(j) = r_1(j+1) + r_2(j+1)$

$$r_i(j) = 2^{n-(j+1)} + 2^{n-(j+1)} = 2^{n-(j+1)+1} = 2^{n-j}$$

DP Solution by Tabulation

FASTEST-WAY(a, t, e, x, n)

$f_1[1] \leftarrow e_1 + a_{1,1}$
$f_2[1] \leftarrow e_2 + a_{2,1}$
for $j \leftarrow 2$ to n
    do if $f_1[j-1] + a_{1,j} \leq f_2[j-1] + t_{2,j-1} + a_{1,j}$
        then $f_1[j] \leftarrow f_1[j-1] + a_{1,j}$
            $l_1[j] \leftarrow 1$
        else $f_1[j] \leftarrow f_2[j-1] + t_{2,j-1} + a_{1,j}$
            $l_1[j] \leftarrow 2$
    if $f_2[j-1] + a_{2,j} \leq f_1[j-1] + t_{1,j-1} + a_{2,j}$
        then $f_2[j] \leftarrow f_2[j-1] + a_{2,j}$
            $l_2[j] \leftarrow 2$
        else $f_2[j] \leftarrow f_1[j-1] + t_{1,j-1} + a_{2,j}$
            $l_2[j] \leftarrow 1$
if $f_1[n] + x_1 \leq f_2[n] + x_2$
    then $f^* = f_1[n] + x_1$
        $l^* = 1$
    else $f^* = f_2[n] + x_2$
        $l^* = 2$

Reconstructing Solution

PRINT-STATIONS(l, n)

$i \leftarrow l^*$
print "line " i ", station " n
for $j \leftarrow n$ **downto** 2
    do $i \leftarrow l_i[j]$
        print "line " i ", station " $j - 1$

Reconstructing Solution

```
PRINT-STATIONS(l, n)
i ← l*
print "line " i ", station " n
for j ← n downto 2
        do i ← l_i[j]
            print "line " i ", station " j − 1
```

line 1, station 6
line 2, station 5
line 2, station 4
line 1, station 3
line 2, station 2
line 1, station 1

Complexity reduces to just $O(n)$ using DP!

Outline

The LCS Problem

- A common problem that occurs in biological setting is to compare strands of DNA.

The LCS Problem

- A common problem that occurs in biological setting is to compare strands of DNA.
- A strand of DNA consists of a string of molecules called bases, where the possible bases are adenine, guanine, cytosine, and thymine.

The LCS Problem

- A common problem that occurs in biological setting is to compare strands of DNA.
- A strand of DNA consists of a string of molecules called bases, where the possible bases are adenine, guanine, cytosine, and thymine.
- The comparison is done by finding common bases in each sequence but not necessarily consecutive.

The LCS Problem

- A common problem that occurs in biological setting is to compare strands of DNA.
- A strand of DNA consists of a string of molecules called bases, where the possible bases are adenine, guanine, cytosine, and thymine.
- The comparison is done by finding common bases in each sequence but not necessarily consecutive.
- For example:
  S1= ACCGGTCGAGTGCGCGGAAGCCGGCCGAA
  S2= GTCGTTCGGAATGCCGTTGCTCCTGTAAA

Problem Statement

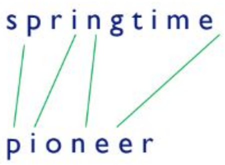
Example

Given two sequences $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$, find the longest common subsequence (LCS) of X and Y.

Problem Statement

Example

Given two sequences $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$, find the longest common subsequence (LCS) of X and Y.



Brute Force solution is $O(n2^m)$!

Optimal Substructure

- At every step, we have two choices: either to include one element in the subsequence or exclude one element from the subsequence.

Optimal Substructure

- At every step, we have two choices: either to include one element in the subsequence or exclude one element from the subsequence.
- Case 1: $x_i = y_j$ $(X = ABDE, Y = ZBE)$
- Case 2: $x_i \neq y_j$ $(X = ABD, Y = ZB)$

Optimal Substructure

GUC German University in Cairo

- At every step, we have two choices: either to include one element in the subsequence or exclude one element from the subsequence.
- Case 1: $x_i = y_j$ $(X = ABDE, Y = ZBE)$
- Case 2: $x_i \neq y_j$ $(X = ABD, Y = ZB)$

Recursive Formulation

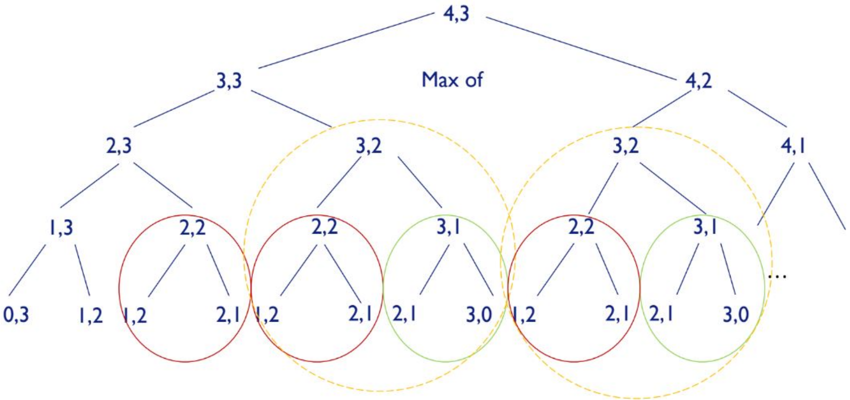
$$c[i,j] = 0 \ if \ i = 0 \ or \ j = 0$$

$$c[i,j] = 1 + c[i-1, j-1] \ if \ x_i = y_j$$

$$c[i,j] = max\{c[i-1,j], c[i,j-1]\} \ if \ x_i \neq y_j$$

Overlapping Subproblems

Recursion tree for LCS of Bozo and Bat.



DP Solution - Tabulation

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

Prints BCBA

DP Solution - Tabulation

```
LCS-LENGTH(X, Y)
m ← length[X]
n ← length[Y]
for i ← 1 to m
        do c[i, 0] ← 0
for j ← 0 to n
        do c[0, j] ← 0
for i ← 1 to m
        do for j ← 1 to n
                do if x_i = y_j
                        then c[i, j] ← c[i − 1, j − 1] + 1
                             b[i, j] ← "↖"
                        else if c[i − 1, j] ≥ c[i, j − 1]
                                then 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
```

Reconstruct Solution

```
PRINT-LCS(b, X, i, j)
if i = 0 or j = 0
    then return
if b[i, j] = "↖"
    then PRINT-LCS(b, X, i − 1, j − 1)
         print x_i
elseif b[i, j] = "↑"
    then PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

Reconstruct Solution

PRINT-LCS(b, X, i, j)

if $i = 0$ or $j = 0$
    then return
if $b[i, j] =$ "↖"
    then PRINT-LCS$(b, X, i-1, j-1)$
        print x_i
elseif $b[i, j] =$ "↑"
    then PRINT-LCS$(b, X, i-1, j)$
else PRINT-LCS$(b, X, i, j-1)$

Complexity reduces to $O(mn)$ with $O(m+n)$ for PRINT-LCS!

Outline

1. Assembly Line Scheduling
2. Longest Common Subsequence
3. Optimal Substructure
4. Recap

Back to Optimal Substructure

- Does optimal substructure apply to all optimization problems?

Back to Optimal Substructure

- Does optimal substructure apply to all optimization problems?
  No!

Back to Optimal Substructure

- Does optimal substructure apply to all optimization problems? No!
- All the examples we saw so far do!

Back to Optimal Substructure

- Does optimal substructure apply to all optimization problems? No!
- All the examples we saw so far do!
- What about finding the shortest path from u to v in an un-weighted graph?

Back to Optimal Substructure

- Does optimal substructure apply to all optimization problems? No!
- All the examples we saw so far do!
- What about finding the shortest path from u to v in an un-weighted graph?
  - Any path from u to v must contain an intermediate vertex, say w.

Back to Optimal Substructure

- Does optimal substructure apply to all optimization problems? No!
- All the examples we saw so far do!
- What about finding the shortest path from u to v in an un-weighted graph?
  - Any path from u to v must contain an intermediate vertex, say w.
  - We can decompose the path $u \longrightarrow_p v$ into subpaths $u \longrightarrow_{p_1} w \longrightarrow_{p_2} v$.

Back to Optimal Substructure

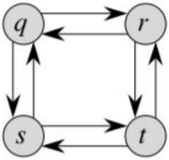
- Does optimal substructure apply to all optimization problems? No!
- All the examples we saw so far do!
- What about finding the shortest path from u to v in an un-weighted graph?
  - Any path from u to v must contain an intermediate vertex, say w.
  - We can decompose the path $u \longrightarrow_p v$ into subpaths $u \longrightarrow_{p_1} w \longrightarrow_{p_2} v$.
  - If p is optimal then p_1 and p_2 are shortest paths as well from u to w and from w to v.

Un-weighted Longest Simple Path

- The problem is to find a simple path (not cyclic) from u to v consisting of the most edges.

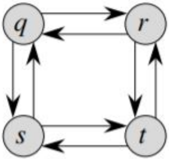
Un-weighted Longest Simple Path

- The problem is to find a simple path (not cyclic) from u to v consisting of the most edges.
- Does this problem exhibits optimal substructure?



Un-weighted Longest Simple Path

- The problem is to find a simple path (not cyclic) from u to v consisting of the most edges.
- Does this problem exhibits optimal substructure? No!



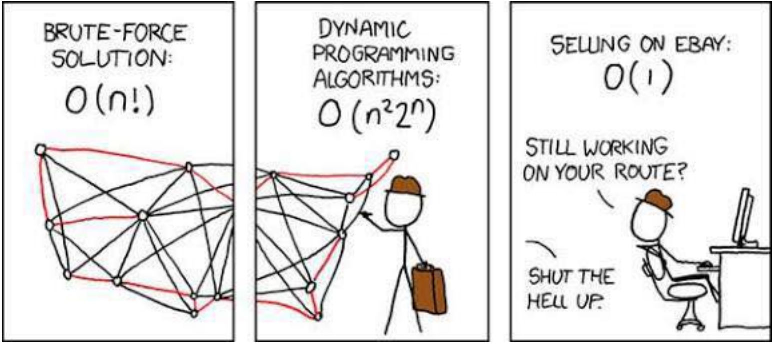
The longest path from q to t is $q \longrightarrow r \longrightarrow t$, but $q \longrightarrow r$ and $r \longrightarrow t$ are NOT the longest paths from q to r and from r to t.

Conclusion

Key Takeaway

DP is only useful when the problem exhibits both optimal substructure and overlapping subproblems.

One Last DP Meme



Outline

1. Assembly Line Scheduling
2. Longest Common Subsequence
3. Optimal Substructure
4. Recap

Points to Take Home

1. Assembly Line Scheduling.
2. Longest Common Subsequence.
3. Not all problems exhibit the optimal substructure property.
4. Reading Material:
    - Introduction to Algorithms, 2nd edition. Chapter 15, Section 15.1.
    - Introduction to Algorithms. Chapter 15, Section 15.4.

Next Lecture: Graph Algorithms I!

Due Credits

The presented material is based on:

1. Previous editions of the course at the GUC due to Dr. Wael Aboulsaadat, Dr. Haythem Ismail, Dr. Amr Desouky, and Dr. Carmen Gervet.
2. Stony Brook University's Analysis of Algorithms Course.
3. MIT's Introduction to Algorithms Course.
4. Stanford's Design and Analysis of Algorithms Course.