Chapter15. Dynamic Programming

Dynamic Programming

- Dynamic Programming is an algorithm design technique for optimization problems: minimizing or maximizing.
- Like divide and conquer, Dynamic Programming solves problems by combining solutions to subproblems.
- Unlike divide and conquer, subproblems are not independent.
  - Subproblems may share subsubproblems,
- Dynamic Programming reduces computation by
  - Solving subproblems in a bottom-up fashion.
  - Storing solution to a subproblem the first time it is solved.
  - Looking up the solution when subproblem is encountered again.
- Key: determine structure of optimal solutions

Dynamic Programming

- Dynamic programming is a way of improving on inefficient divide-and-conquer algorithms.
- By "*inefficient*", we mean that *the same recursive call is made over and over.*
- If same subproblem is solved several times, we can use table to store result of a subproblem the first time it is computed and thus never have to recompute it again.
- Dynamic programming is applicable when the subproblems are dependent, that is, when subproblems share subproblems.
- "Programming" refers to a tabular method

Difference between Dynamic Programming and Divide-and-Conquer

- Using Divide-and-Conquer to solve these problems is inefficient because the same common subproblems have to be solved many times.

- Dynamic Programming will solve each of them once and their answers are stored in a table for future use.

Elements of Dynamic Programming

DP is used to solve problems with the following characteristics:

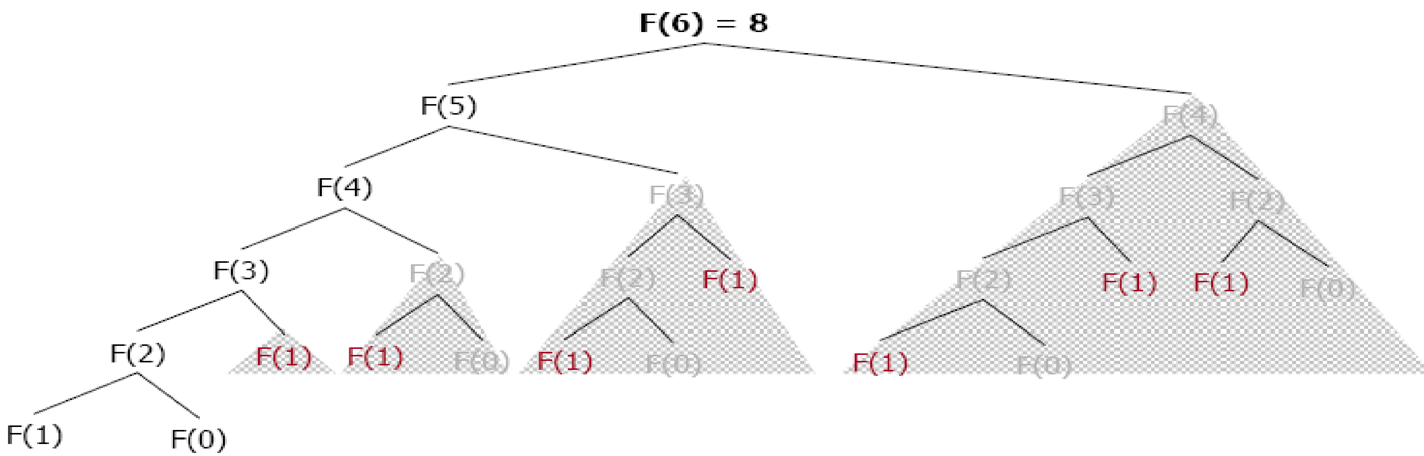
- Simple subproblems
  - We should be able to break the original problem to smaller subproblems that have the same structure
- Optimal substructure of the problems
  - The optimal solution to the problem contains within optimal solutions to its subproblems.
- Overlapping subproblems
  - there exist some places where we solve the same subproblem more than once.

Steps in Dynamic Programming

1. **Characterize** structure of an optimal solution.
2. **Define** value of optimal solution recursively.
3. **Compute** optimal solution values either top-down with caching or bottom-up in a table.
4. **Construct** an optimal solution from computed values. (not always necessary)

Dynamic Programming Example

- <u>Fibonacci Numbers</u>
  - *Fn= Fn-1+ Fn-2* $\qquad n \geq 2$
  - *F0 =0, F1 =1*
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …
- Straightforward recursive procedure is slow!



- We keep calculating the same value over and over!

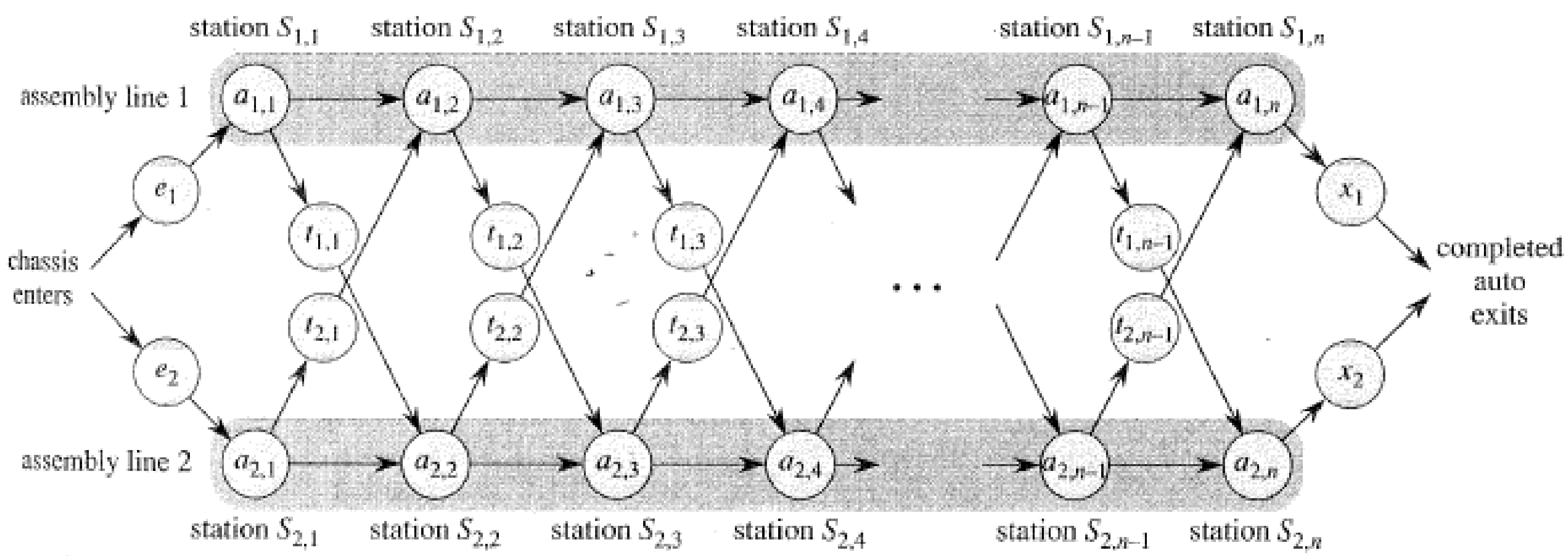
Dynamic Programming Example

- We can calculate *Fn* in ***linear* time** by remembering solutions to the solved subproblems : *dynamic programming*
- Compute solution in a bottom-up fashion
- In this case, only two values need to be remembered at any time

```
Fibonacci(n)
    F_0 ← 0
    F_1 ← 1
    for i ← 2 to n do
        F_i ← F_{i-1} + F_{i-2}
```

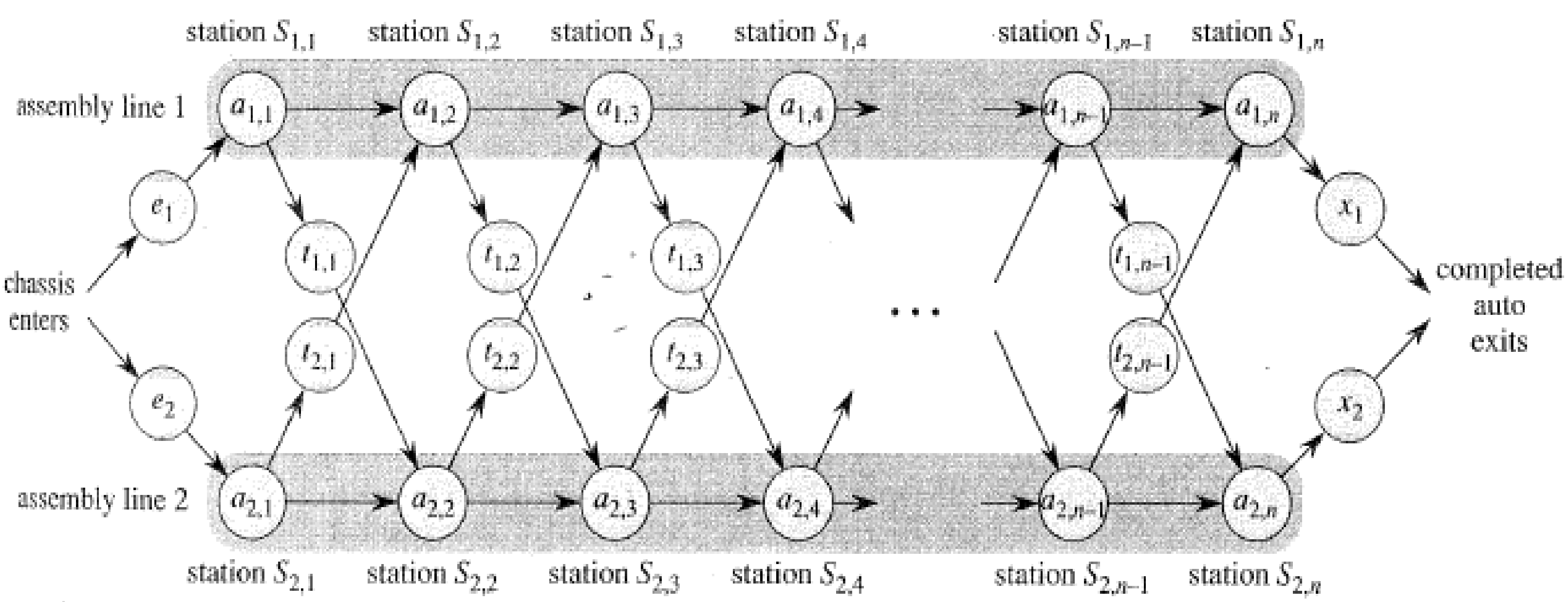
Assembly Line Scheduling

- Automobile factory with two assembly lines
  - Each line has *n* stations: $S_{1,1}, \ldots, S_{1,n}$ and $S_{2,1}, \ldots, S_{2,n}$
  - Corresponding stations $S_{1,j}$ and $S_{2,j}$ perform the same function but can take different amounts of time $a_{1,j}$ and $a_{2,j}$
  - Entry times are: e_1 and e_2; exit times are: x_1 and x_2



Assembly Line Scheduling

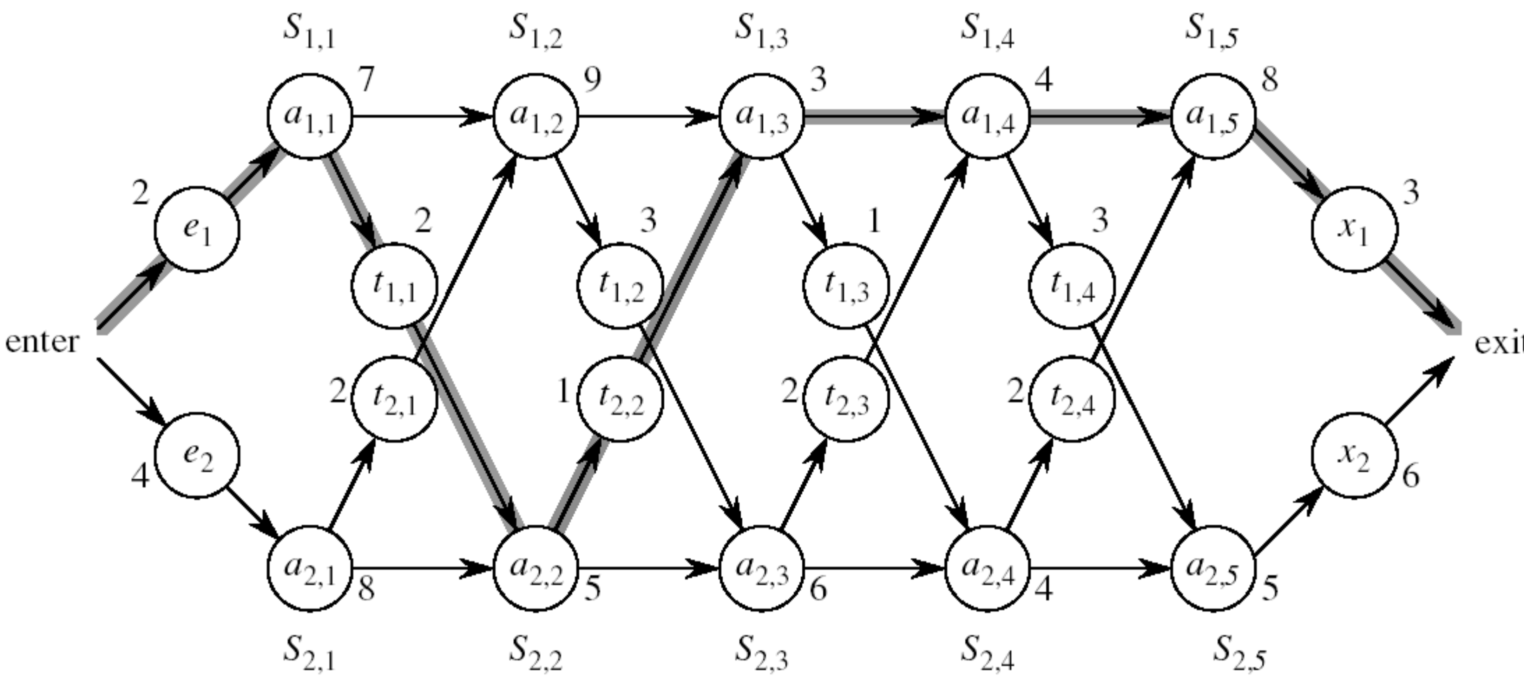
- After going through a station, can either:
  - stay on same line at no cost, or
  - transfer to other line: cost after $S_{i,j}$ is $t_{i,j}$, j = 1, . . . , n - 1



Assembly Line Scheduling

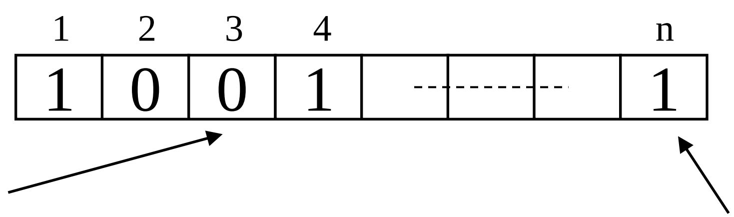
- Problem:

  what stations should be chosen from line 1 and which from line 2 in order to **minimize the total time through the factory for one car?**



One Solution

- Brute force
  - Enumerate all possibilities of selecting stations
  - Compute how long it takes in each case and choose the best one
- Solution:



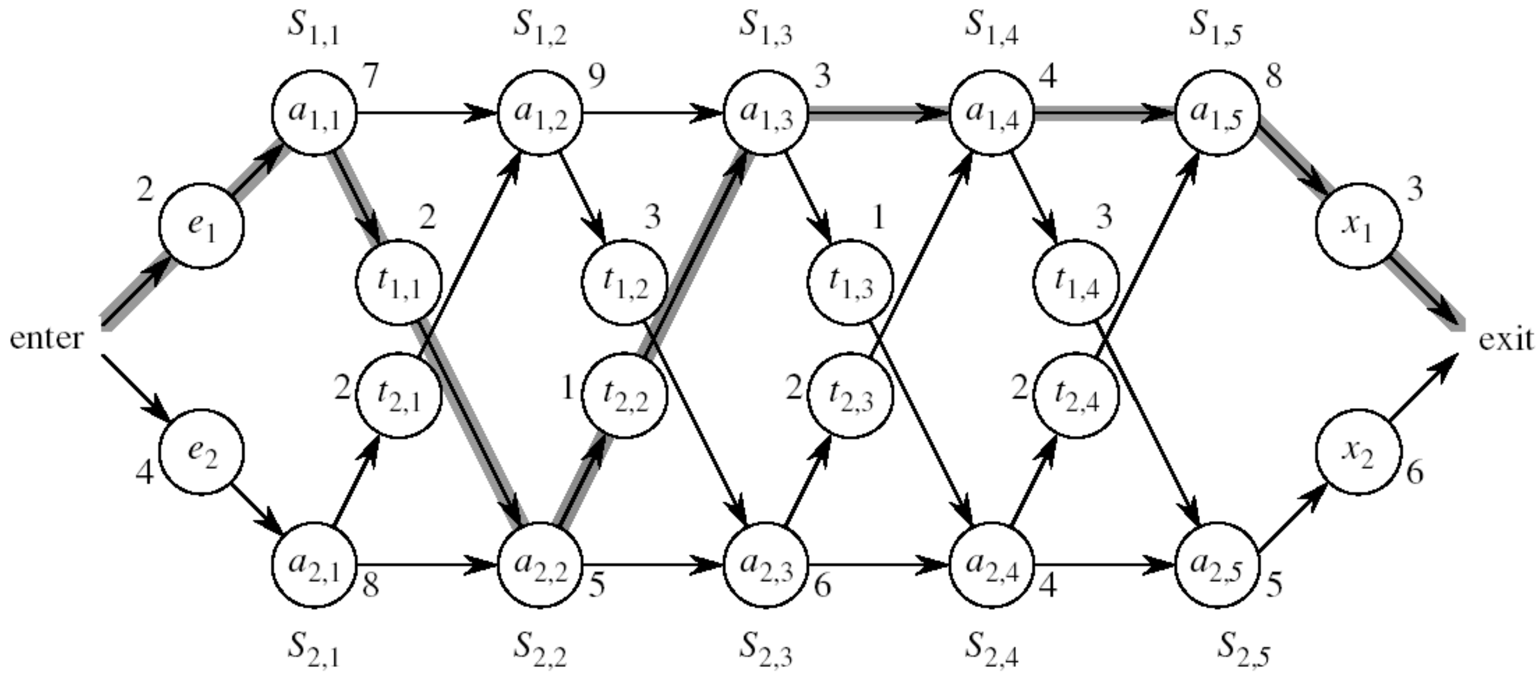
0 if choosing line 2 at step j (= 3)

1 if choosing line 1 at step j (= n)

  - There are 2^n possible ways to choose stations
  - Infeasible when *n* is large!!

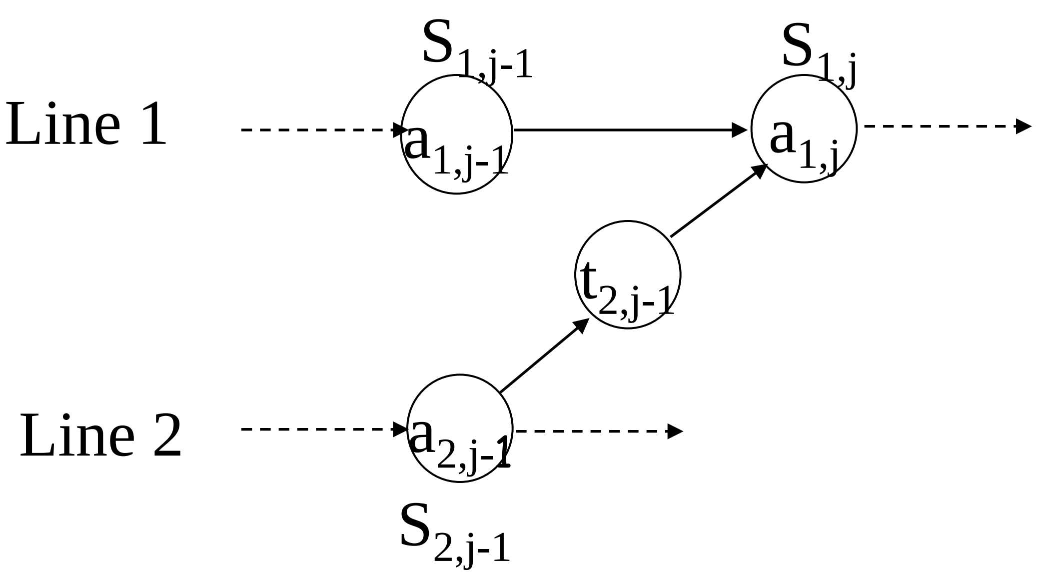
1. Structure of the Optimal Solution

- How do we compute the minimum time of going through a station?



1. Structure of the Optimal Solution

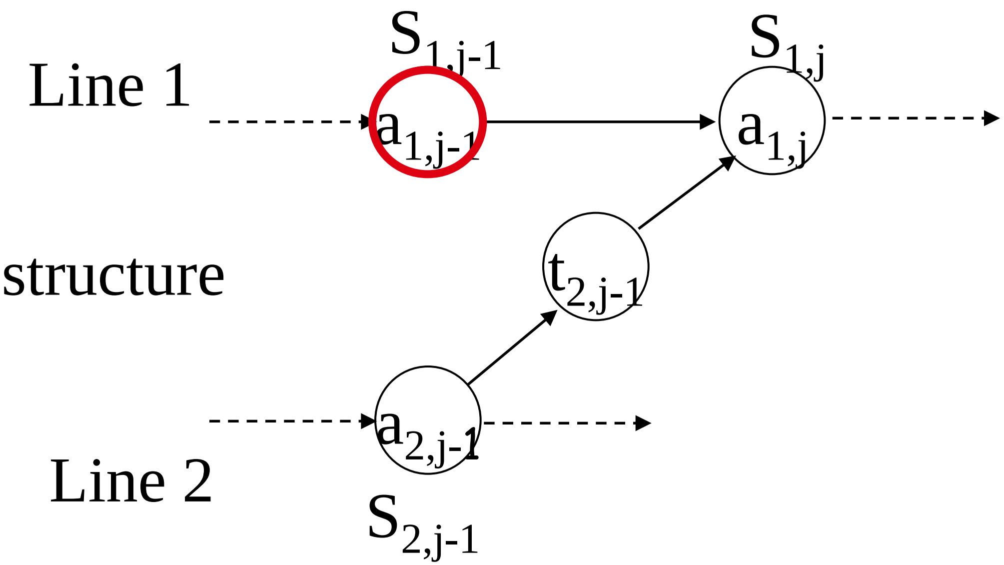
- Let's consider all possible ways to get from the starting point through station $S_{1,j}$
  - We have two choices of how to get to $S_{1, j}$:
    - Through $S_{1, j-1}$, then directly to $S_{1, j}$
    - Through $S_{2, j-1}$, then transfer over to $S_{1, j}$



1. Structure of the Optimal Solution

- Suppose that the fastest way through $S_{1,\,j}$ is through $S_{1,\,j-1}$
  - We must have taken a fastest way from entry through $S_{1,\,j-1}$
  - If there were a faster way through $S_{1,\,j-1}$, we would use it instead
- Similarly for $S_{2,\,j-1}$

Optimal Substructure

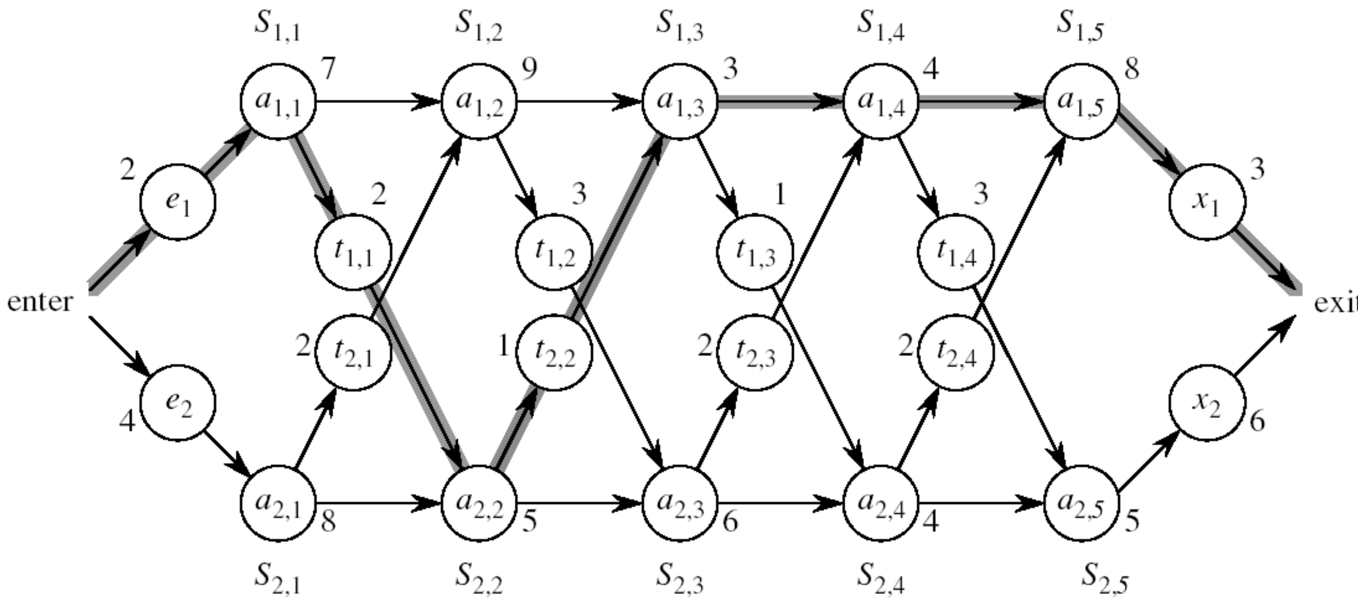


Optimal Substructure

- **Generalization**: an optimal solution to the problem *"find the fastest way through* $S_{1,j}$*"* contains within it an optimal solution to subproblems: *"find the fastest way through* $S_{1,j-1}$ *or* $S_{2,j-1}$*".*
- This is referred to as the optimal substructure property
- We use this property to construct an optimal solution to a problem from optimal solutions to subproblems

2. A Recursive Solution

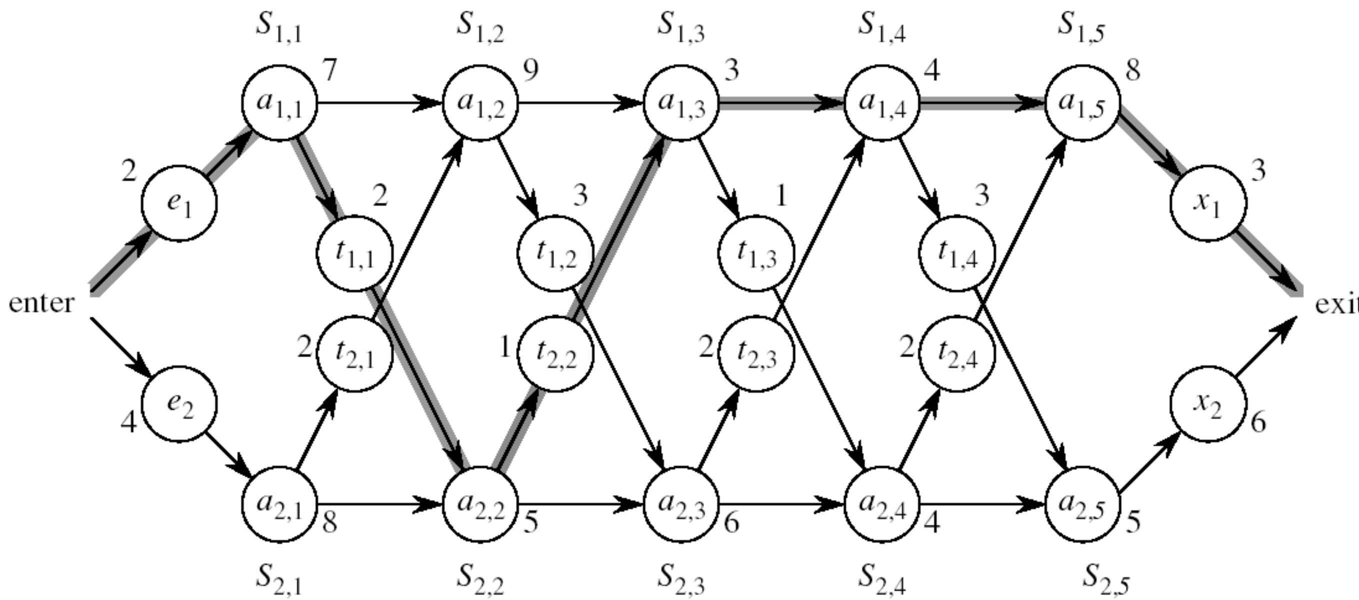
- Define the value of an optimal solution in terms of the optimal solution to subproblems



2. A Recursive Solution (cont.)

- Definitions:
  - f* : the fastest time to get through the entire factory
  - $f_i[j]$: the fastest time to get from the starting point through station $S_{i,j}$

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

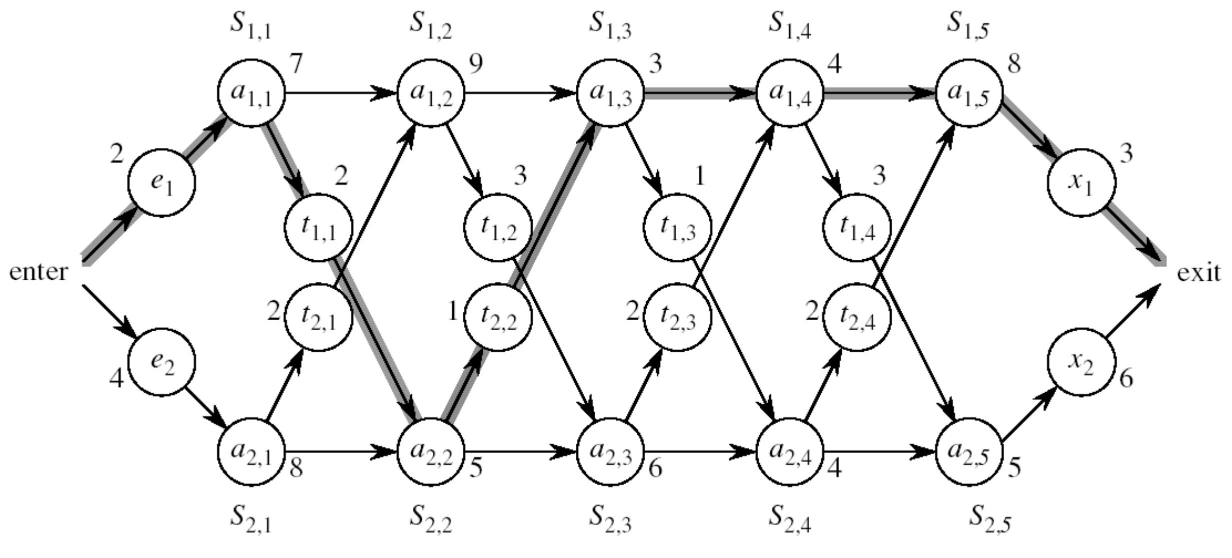


2. A Recursive Solution (cont.)

- Base case: j = 1, i=1,2 (getting through station 1)

$$f_1[1] = e_1 + a_{1,1}$$

$$f_2[1] = e_2 + a_{2,1}$$

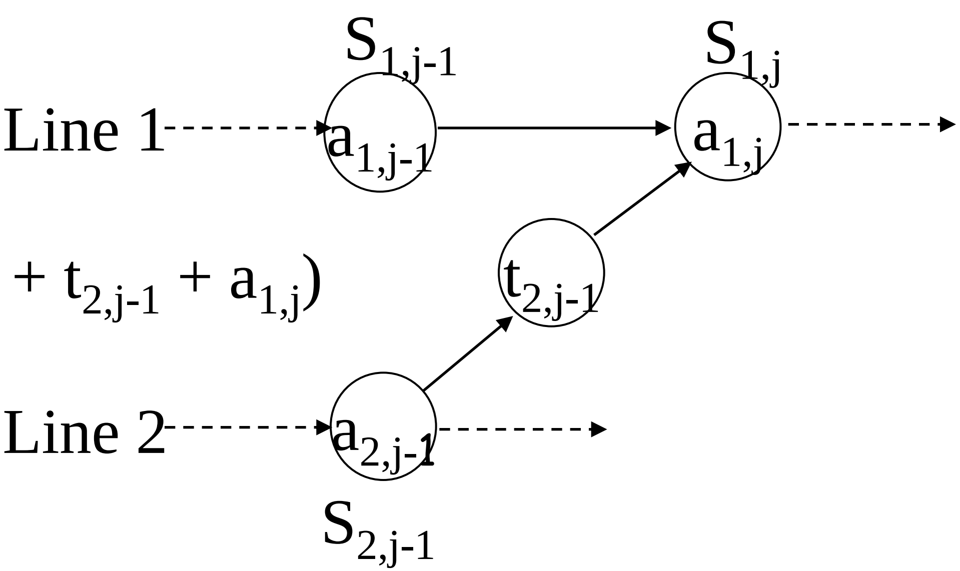


2. A Recursive Solution (cont.)

- General Case: $j = 2, 3, \ldots, n$, and $i = 1, 2$
- Fastest way through $S_{1,\,j}$ is either:
  - the way through $S_{1,\,j-1}$ then directly through $S_{1,\,j}$, or

    $f_1[j - 1] + a_{1,j}$
  - the way through $S_{2,\,j-1}$, transfer from line 2 to line 1, then through $S_{1,\,j}$

    $f_2[j - 1] + t_{2,j-1} + a_{1,j}$



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

2. A Recursive Solution (cont.)

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

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

3. Computing the Optimal Solution

$$\mathbf{f^* = \min(f_1[n] + x_1, f_2[n] + x_2)}$$

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

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

	1	2	3	4	5
$f_1[j]$	$f_1(1)$	$f_1(2)$	$f_1(3)$	$f_1(4)$	$f_1(5)$
$f_2[j]$	$f_2(1)$	$f_2(2)$	$f_2(3)$	$f_2(4)$	$f_2(5)$

4 times 2 times

- Solving top-down would result in exponential running time

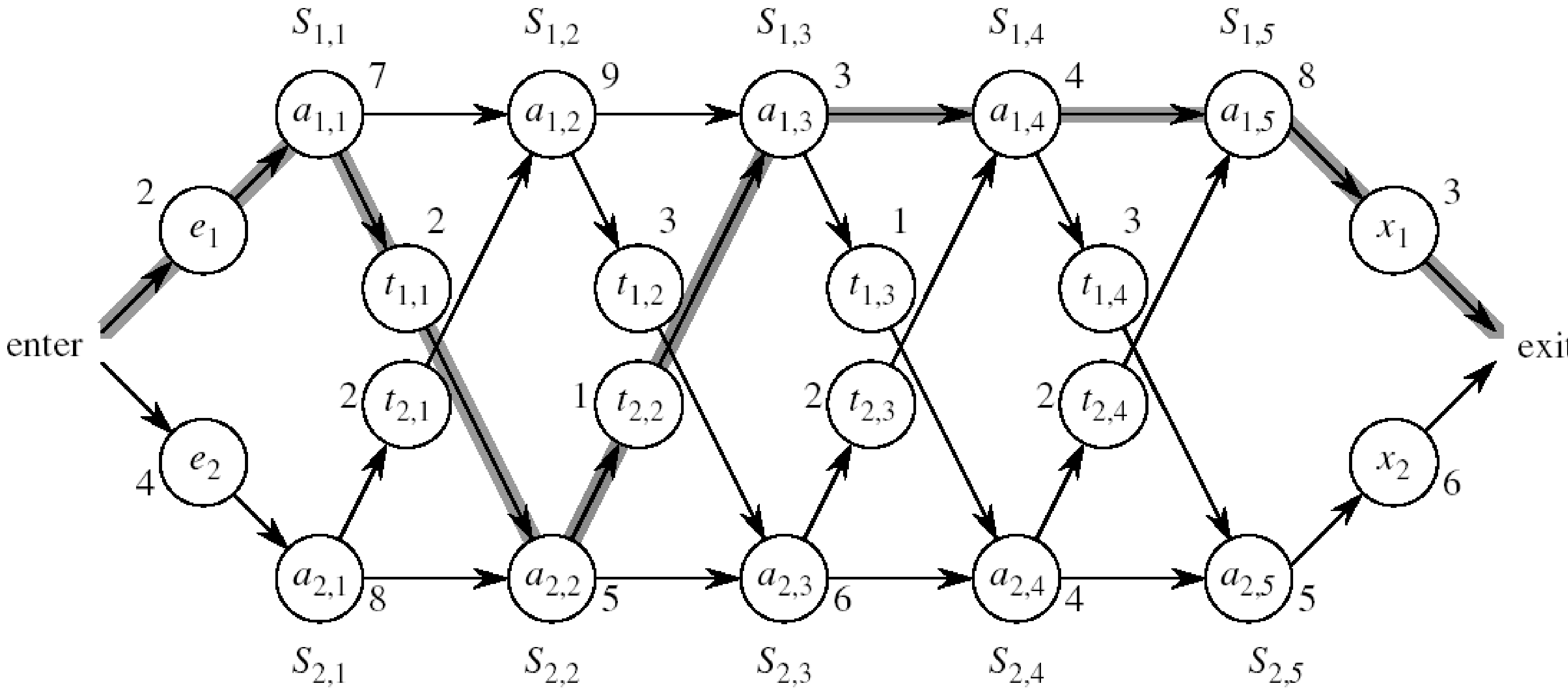
3. Computing the Optimal Solution

- For $j \geq 2$, each value $f_i[j]$ depends only on the values of $f_1[j - 1]$ and $f_2[j - 1]$
- Idea: compute the values of $f_i[j]$ as follows:

in increasing order of j →

	1	2	3	4	5
$f_1[j]$					
$f_2[j]$					

- Bottom-up approach
  - First find optimal solutions to subproblems
  - Find an optimal solution to the problem from the subproblems



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

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

$f^* = 35^{[1]}$

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

$f_1[1] \leftarrow e_1 + a_{1,1}$
$f_2[1] \leftarrow e_2 + a_{2,1}$

Compute initial values of f_1 and f_2

for $j \leftarrow 2$ **to** n

O(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 $f_1[j] \leftarrow f_2[j-1] + t_{2,\, j-1} + a_{1,\, j}$
            $l_1[j] \leftarrow 2$

Compute the values of $f_1[j]$ and $l_1[j]$

    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 $f_2[j] \leftarrow f_1[j-1] + t_{1,\, j-1} + a_{2,\, j}$
            $l_2[j] \leftarrow 1$

Compute the values of $f_2[j]$ and $l_2[j]$

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

if $f_1[n] + x_1 \le f_2[n] + x_2$

    then $f^* = f_1[n] + x_1$

        $l^* = 1$

    else $f^* = f_2[n] + x_2$

        $l^* = 2$

Compute the values of the fastest time through the entire factory

4. Construct an Optimal Solution

Alg.: PRINT-STATIONS(l, n)

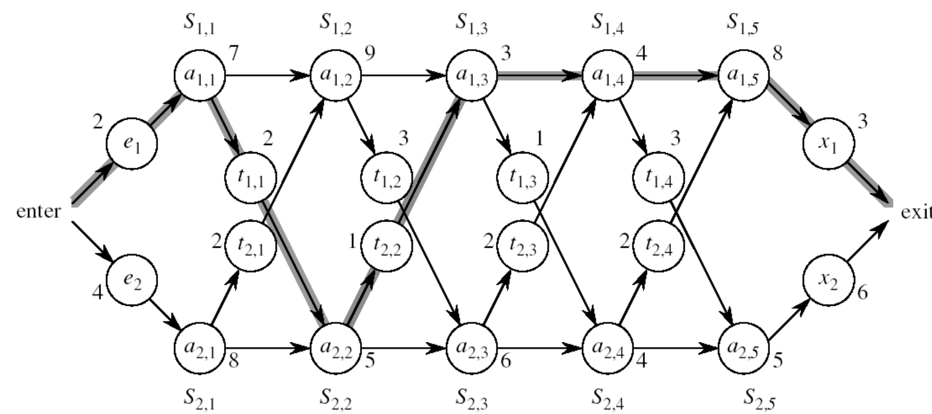
i ← l*

print "line " i ", station " n

for j ← n **downto** 2

  do i ← $l_i[j]$

   print "line " i ", station " j - 1



	1	2	3	4	5
$f_1[j]/l_1[j]$	9	18[1]	20[2]	24[1]	32[1]
$f_2[j]/l_2[j]$	12	16[1]	22[2]	25[1]	30[2]

l* = 1