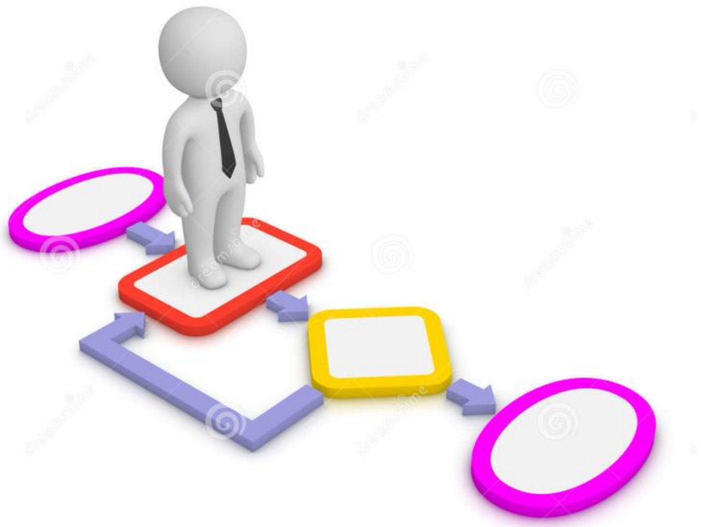


Design & Analysis of Algorithms

Saman Riaz (PhD)
Associate Professor
RSCI, Lahore

Lecture # 10

DYNAMIC PROGRAMMING I

Dynamic Programming

- Dynamic programming is not an algorithm, but a technique like divide and conquer
- Divide and conquer is used for disjoint subproblems however dynamic programming is for overlap subproblems
- Here “Programming” refers to a tabular method, not to writing computer code.
- A dynamic-programming solves each subsubproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time it solves each subsubproblem

Dynamic Programming - Advantages

The main advantages of dynamic programming are:

- *Dynamic Programming can be used to solve optimization problems very efficiently, which cannot be solved efficiently by brute force and divide-and-conquer methods*

- *The implementation of dynamic programming solution is simple. The code consists of mainly nested loops which can be easily coded.*

- *The time and space complexity analysis of dynamic programming is easy and straightforward. Running time can be determined by counting loop cycles.*

Dynamic Programming – Disadvantages

The disadvantages associated with dynamic programming are:

- *There is an overhead of memory requirement to to store tables for the solution of sub problems*

- *The formulation of sub problem structure is difficult*

- *The principle of optimality must hold.*

Dynamic Programming (Cont.)

When developing a dynamic-programming , we follow a sequence of four steps:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Elements of Dynamic Programming

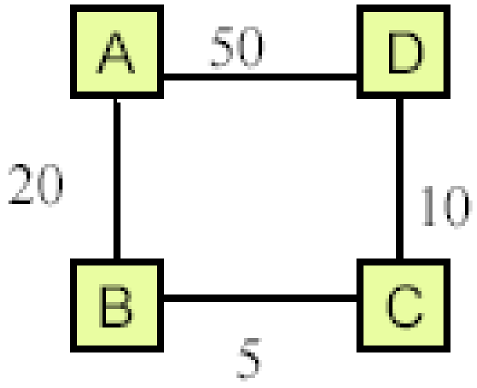
- Optimal sub-structure
  - An optimal solution to a problem contains optimal solutions to sub-problems
  - Dynamic programming uses optimal sub-structure in a bottom-up manner

- Overlapping sub-problems
  - Space of sub-problems must be “small” typically the total number of distinct sub-problems is a polynomial in the input size
  - Recursive algorithm visits same problem again and again
  - Each sub-problem is solved once, solution is stored in table

Optimal sub-structure

- The principle of optimality holds in the case of problem of finding a ***shortest distance*** in a network, but it does not hold good for problem of finding the ***longest distance***

- Consider, for example, the network shown in the figure. The shortest distance A to D is A->B->C->D 35. Further, the shortest distance from A to C is A->B->C=25, and shortest distance from A to B is A->B=20. Thus, principle of ***optimality holds true for each of the sub paths of the shortest distance between A to D.***



- Now consider the longest distance from C to A. It is C->D->A=60. However, the sub path C->D=10 is not the longest distance, because the longest distance from C to D is C->B->A->D=75. Hence, the principle of ***optimality does not hold true***.

Recursive Algorithm

- Takes Exponential time, seen few lectures back!
- Actual sub problems are polynomial (O(n)) but they get repeated
- Sub problems are not INDEPENDENT.
- Sub problems share sub-sub problems.
- We can solve it using Dynamic programming.

Benefit of Dynamic Programming

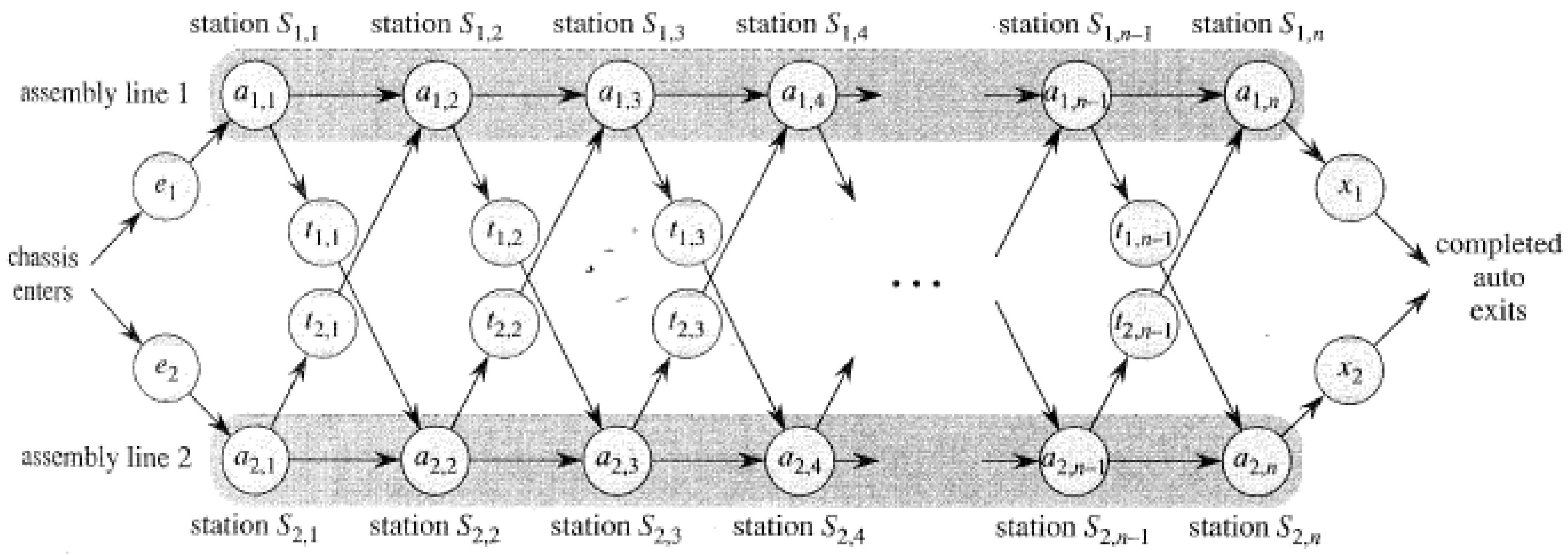
- Run an O(n) time loop, keep a temp variable to store the solution of sub-problems and then reuse them rather then recalculating them.
- So by using dynamic programming we can solve a problem in polynomial time which otherwise was solved in exponential time.

Dynamic programming

Assembly Line Scheduling

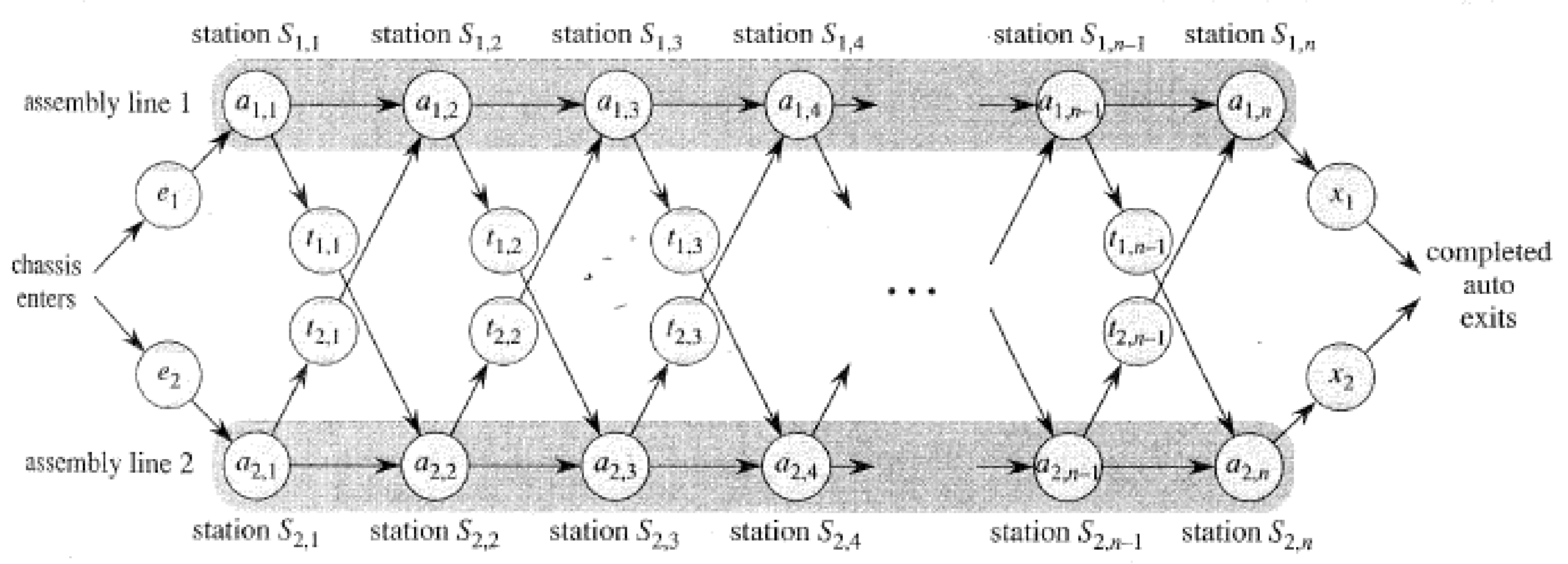
Assembly Line Scheduling

- 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

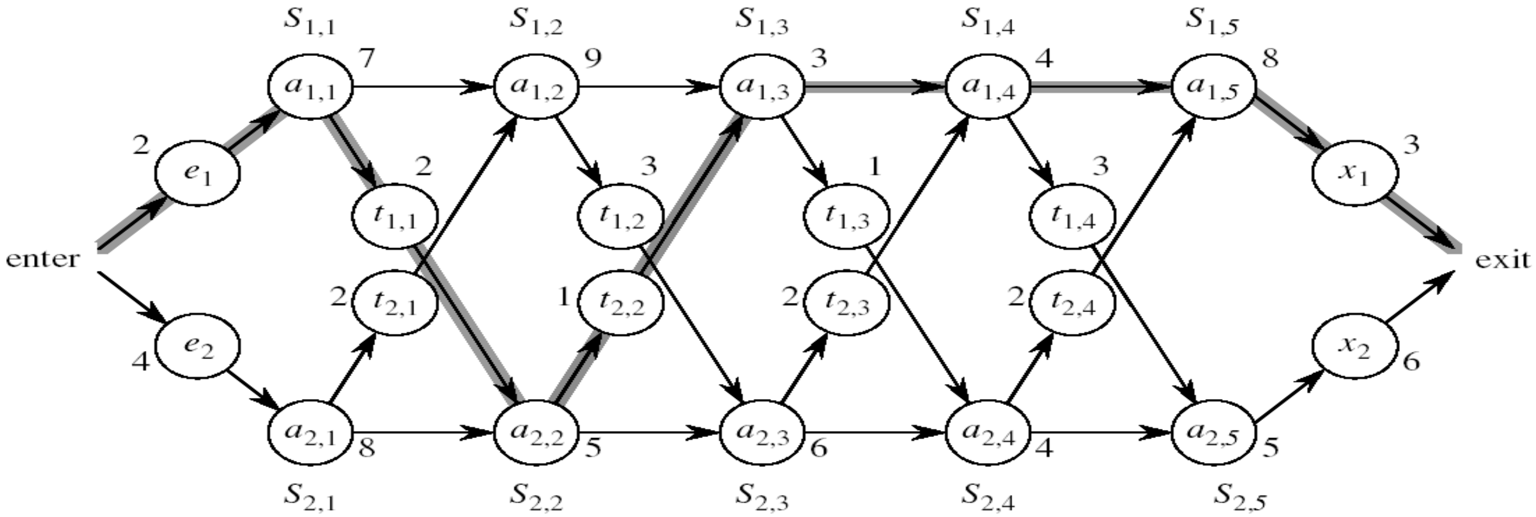
- After going through a station, can either:
  - stay on same line at no cost, or
  - transfer to other line: cost after $S_{i,j}$ is $t_{i,j}$, $j = 1, \ldots, n - 1$



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?

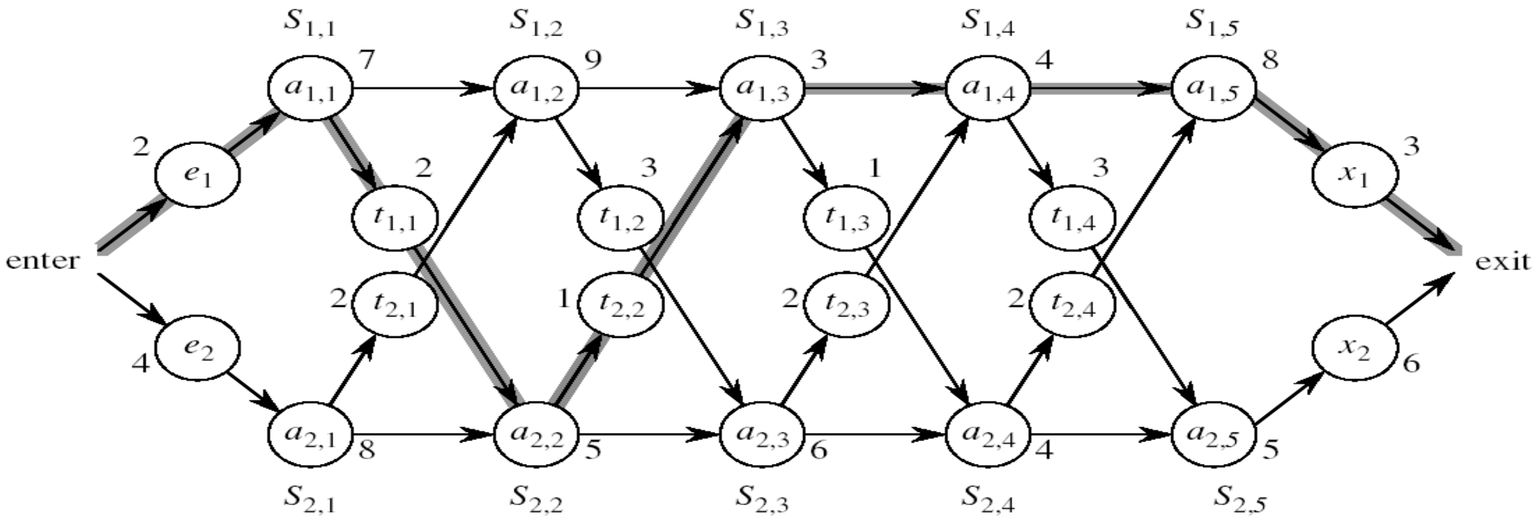


One Solution

- Brute force
  - Enumerate all possibilities of selecting stations
  - Compute how long it takes in each case and choose the best one
  - There are 2^n possible ways to choose stations
  - Infeasible when n is large!!

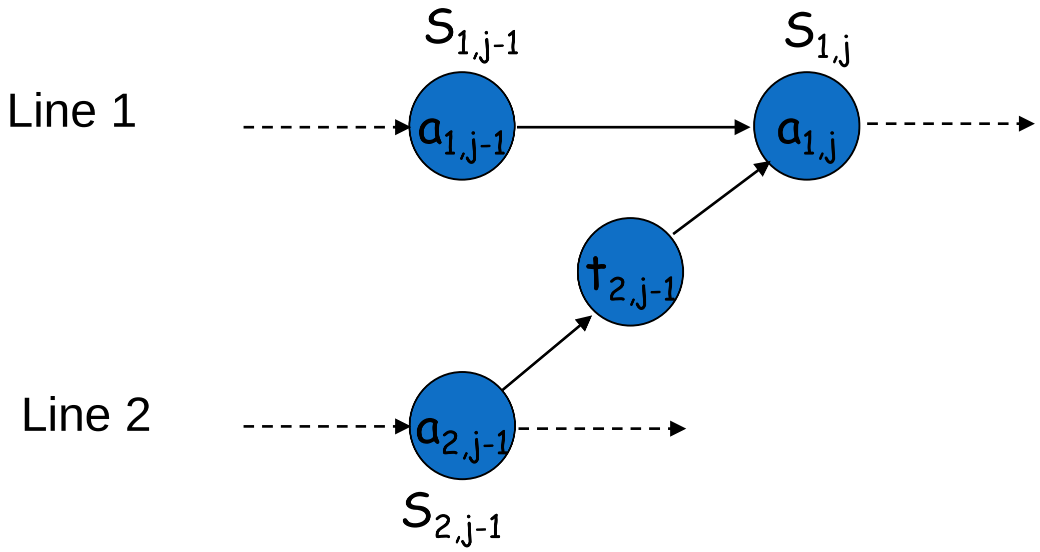
1. Structure of the Optimal Solution

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



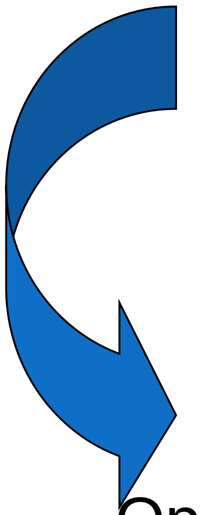
1. Structure of the Optimal Solution

- 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 $S_{1,j}$

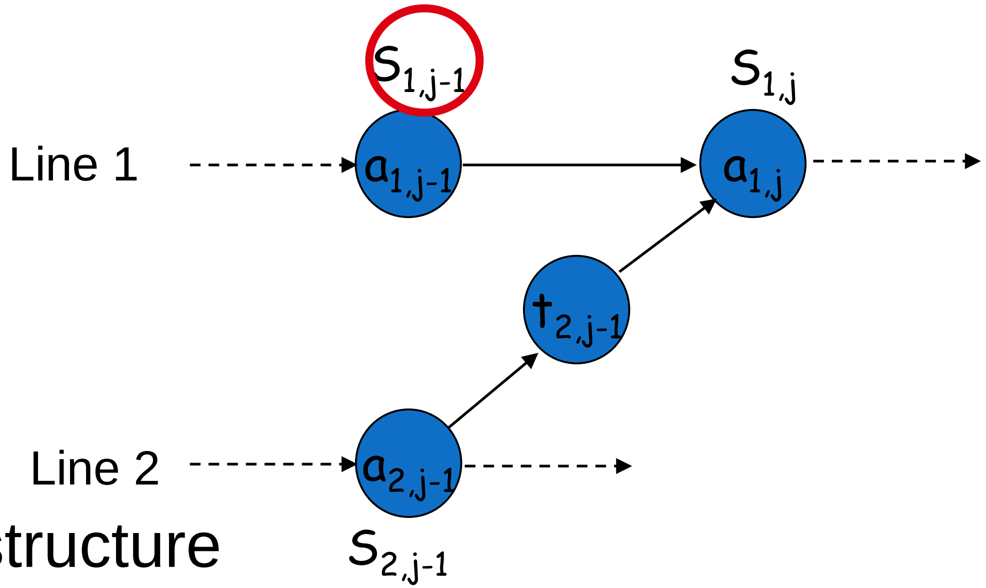


1. Structure of the Optimal Solution

- 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

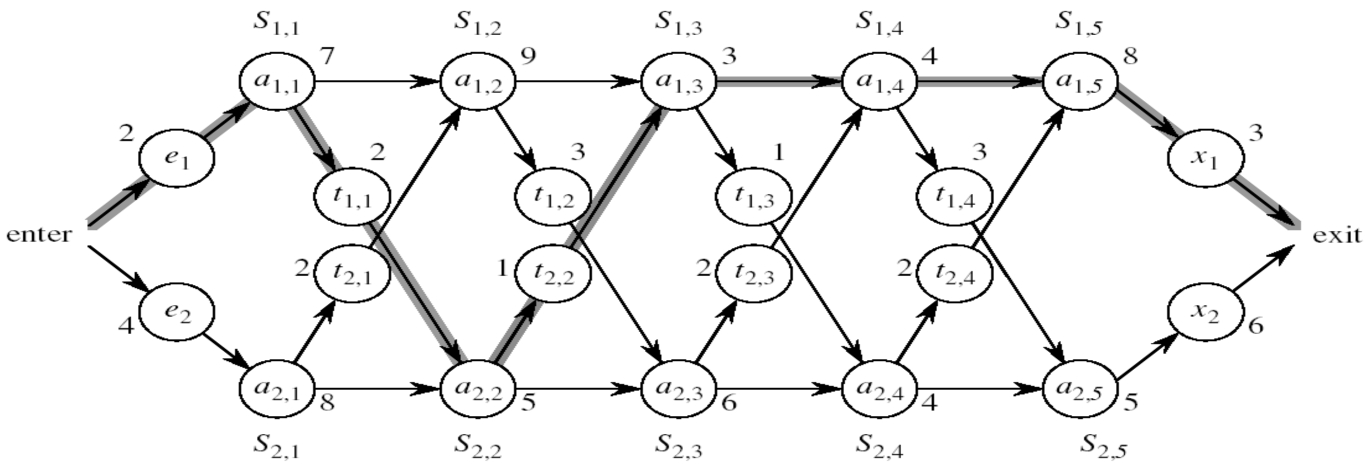
- **Generalization**: an optimal solution to the problem "*find the fastest way through* $S_{1,j}$" contains within it an optimal solution to subproblems: "*find the fastest way through* $S_{1,j-1}$ *or* $S_{2,j-1}$".

- 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

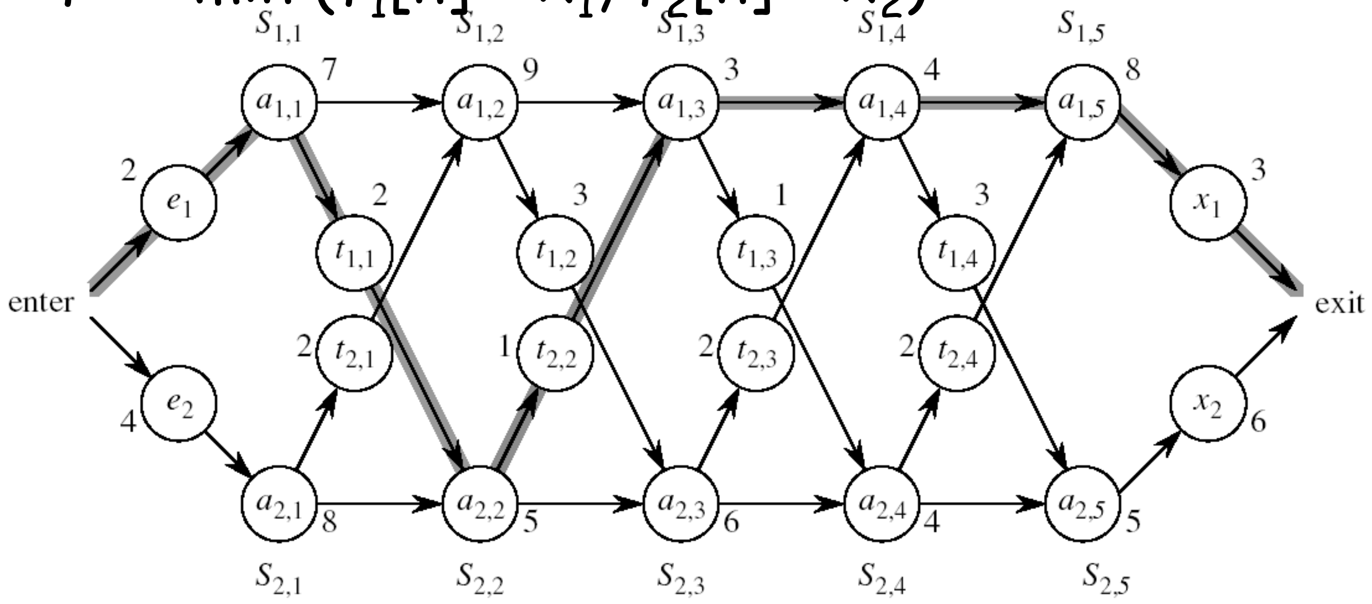
- Define the value of an optimal solution in terms of the optimal solution to subproblems



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 $S_{i,j}$

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

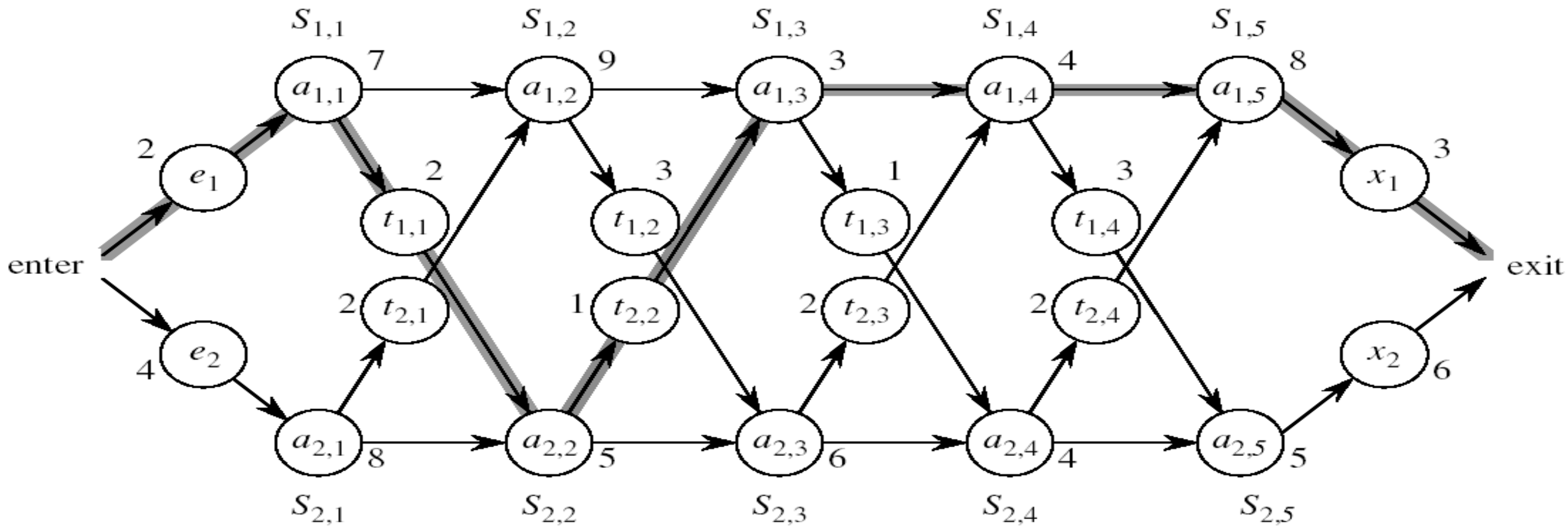


2. A Recursive Solution (cont.)

- <u>Base case</u>: j = 1, i=1,2 (getting through station 1)

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

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

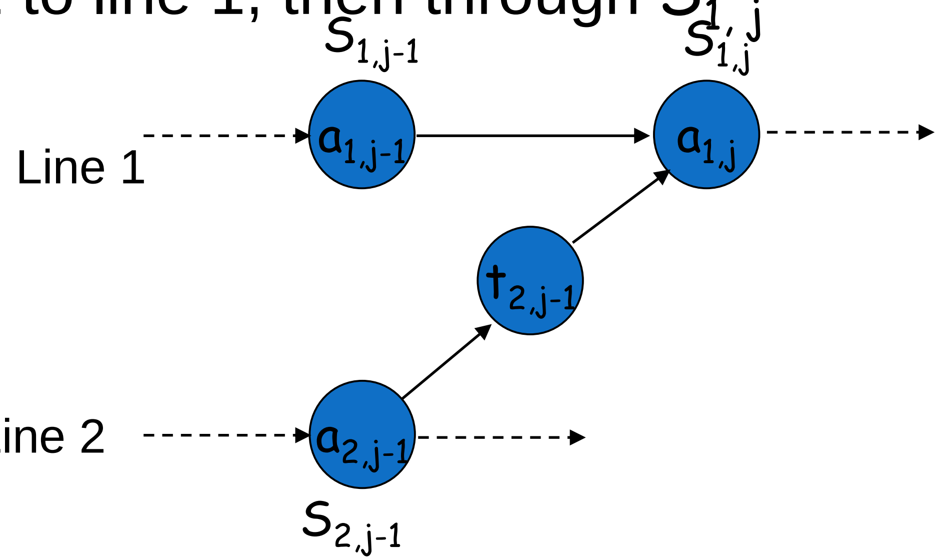


2. A Recursive Solution (cont.)

- <u>General Case</u>: $j = 2, 3, \ldots, n$, 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

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

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

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

	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

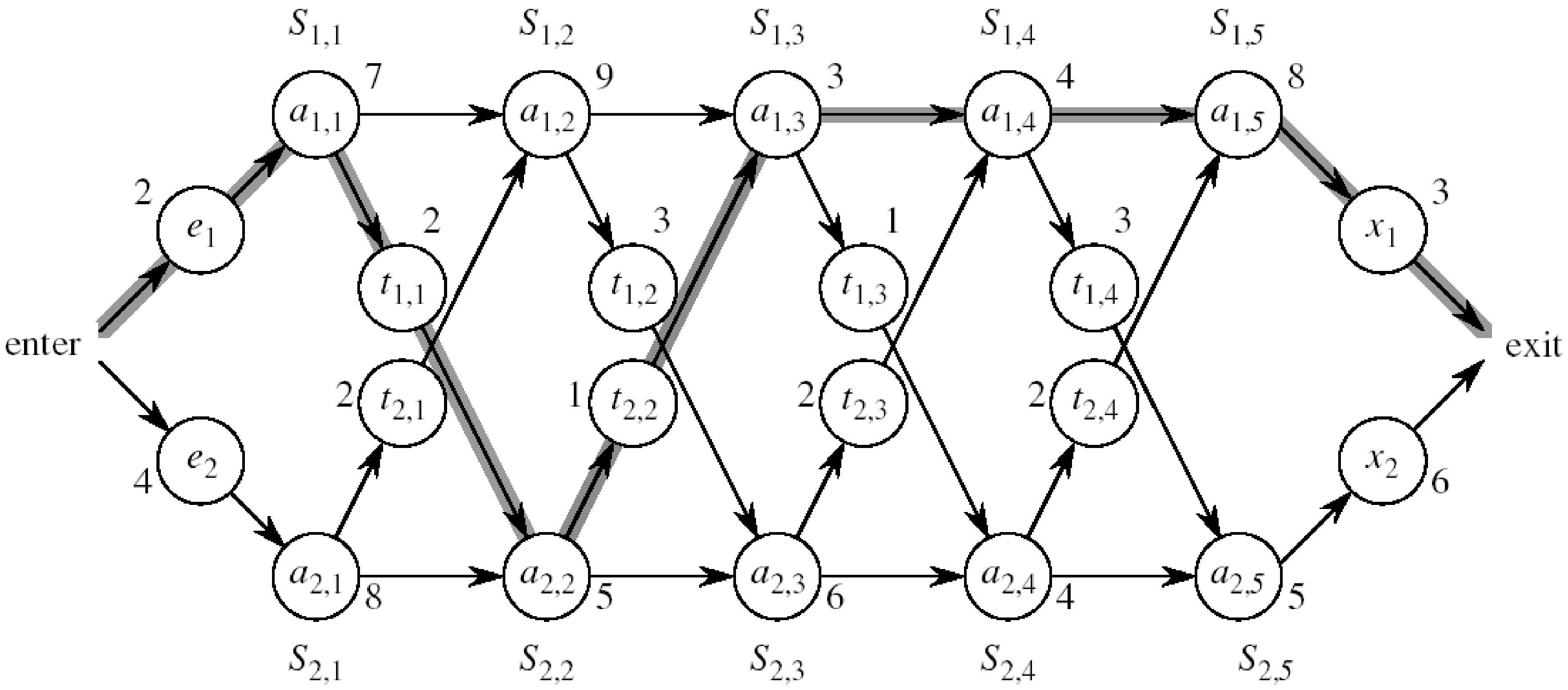
- For $j \geq 2$, each value $f_i[j]$ depends only on the values of $f_1[j - 1]$ and $f_2[j - 1]$
- Idea: compute the values of $f_i[j]$ as follows:

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

Example



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

	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

  do if $f_1[j-1] + a_{1,j} \le f_2[j-1] + t_{2,j-1} + a_{1,j}$

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

    then $f_2[j] \leftarrow f_2[j-1] + a_{2,j}$

      $l_2[j] \leftarrow 2$

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

      $l_2[j] \leftarrow 1$

O(N)

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

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

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

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

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

    $l^* = 1$

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

    $l^* = 2$

Compute the values of the fastest time through the entire factory

4. Construct an Optimal Solution

Alg.: PRINT-STATIONS(l, n)

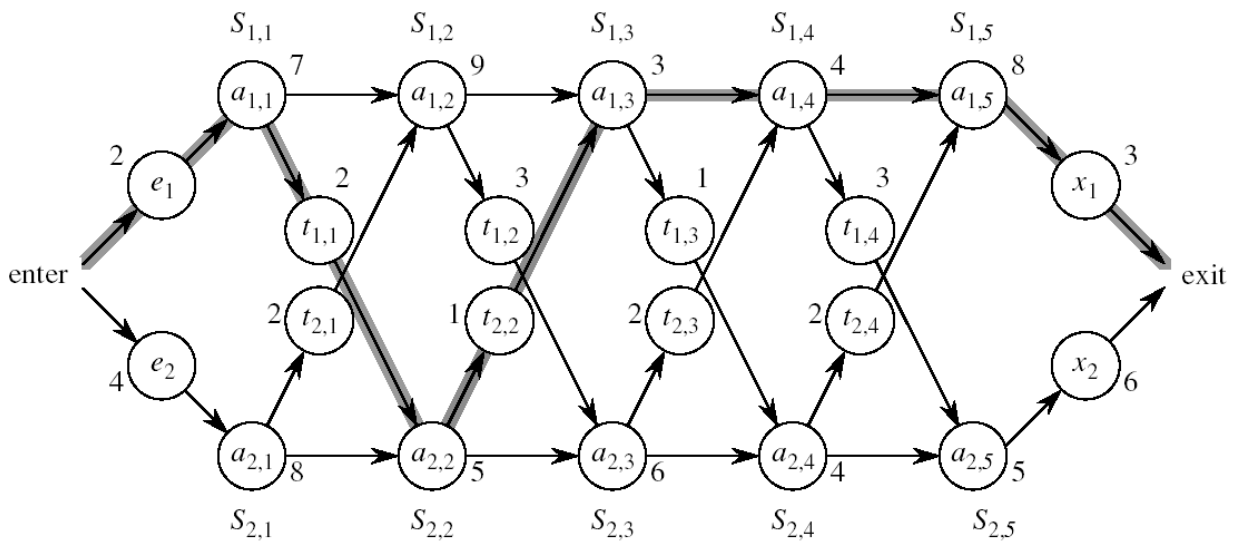
i ← l*

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

Quiz 5

- Using KMP algorithm-for-pattern-searching

  Text : aabbcraabdjtabbabbabdjrk

Pattern :abbabbabdj