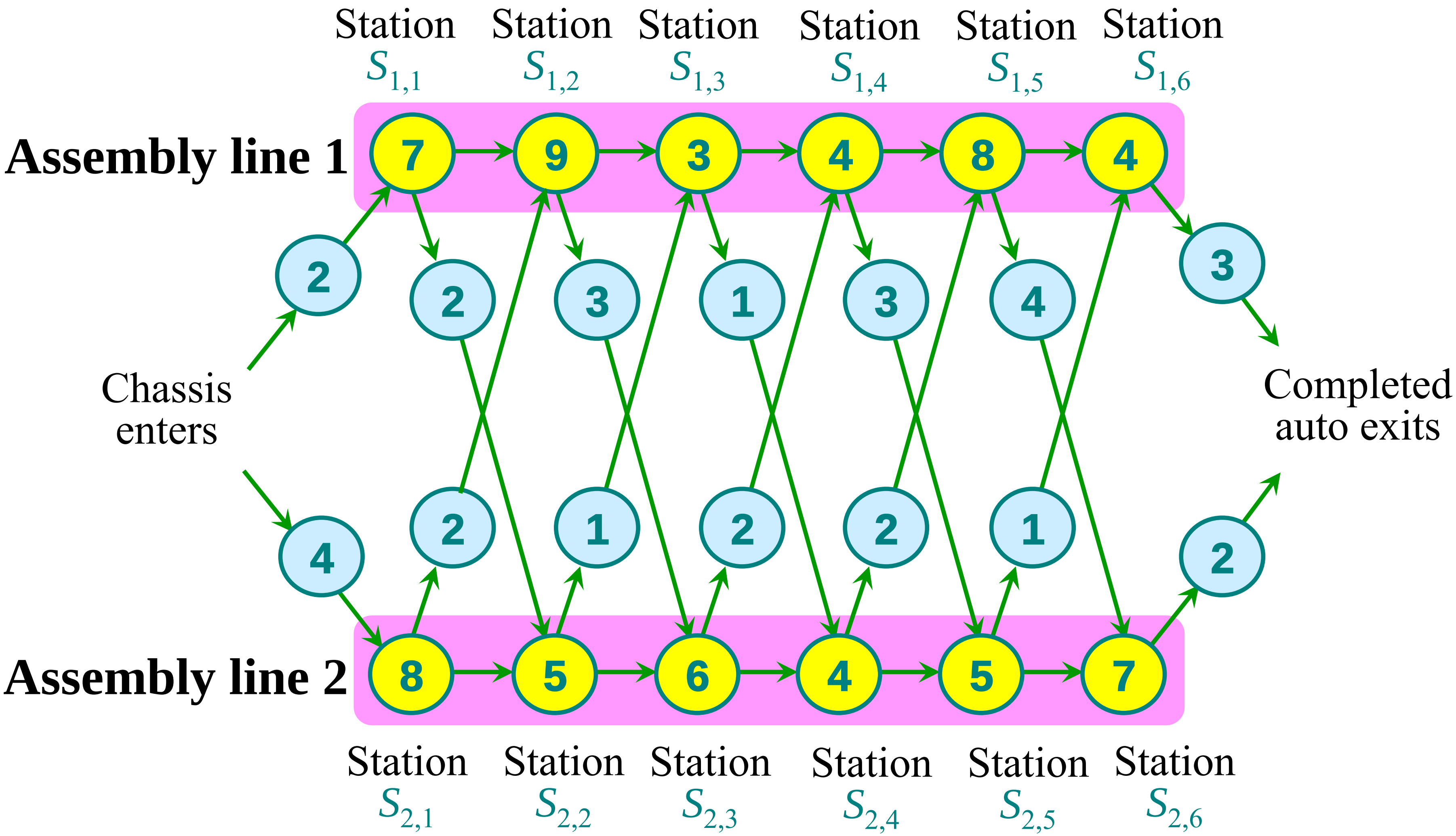
Data Structures and Algorithm

Xiaoqing Zheng
zhengxq@fudan.edu.cn

Dynamic programming

- Dynamic programming is typically applied to ***optimization problems***.
- There can be ***many possible solutions*** in optimization problems.
- Each solution has a value, and we wish to find a solution with the optimal (***minimum*** or ***maximum***) value.

Manufacturing problem



Brute-force

Check every way through a factory and choose the fastest way.

Analysis

- Checking $= O(n)$ time per way.
- 2^n possible ways to choose stations.
- Worst-case running time $= O(n2^n)$

  $=$ exponential time.

It is infeasible!

Structure of manufacturing problem

- An optimal solution to a problem (finding the fastest way though station $S_{i,j}$) contains within it an optimal solution to ***subproblems*** (finding the fastest way through either $S_{1,j-1}$ or $S_{2,j-1}$)
- Suppose that the fastest way through station $S_{1,j}$ is ***either***
  - the fastest way through station $S_{1,j-1}$ and then directly through station $S_{1,j}$, ***or***
  - the fastest way through station $S_{2,j-1}$, a transfer from line 1 to line 1, and then through station $S_{1,j}$.
- Suppose that the fastest way through station $S_{1,j}$ is through station $S_{1,j-1}$. The ***key observation*** is that the chassis must have taken a fastest way from the starting point through station $S_{1,j-1}$.

Recursive solution

$f_i[j]$ denote the fastest possible time to get a chassis from the starting point through station S_{ij}.

e_i denote an entry time for the chassis to enter assembly line i.

x_i denote an exit time for the completed auto to exit assembly line i.

$a_{i,j}$ denote the assembly time required at station S_{ij}.

$t_{i,j}$ denote the time to transfer a chassis away from assembly line i after through station S_{ij}.

Our ***ultimate goal*** is:

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

Recursive solution (cont.)

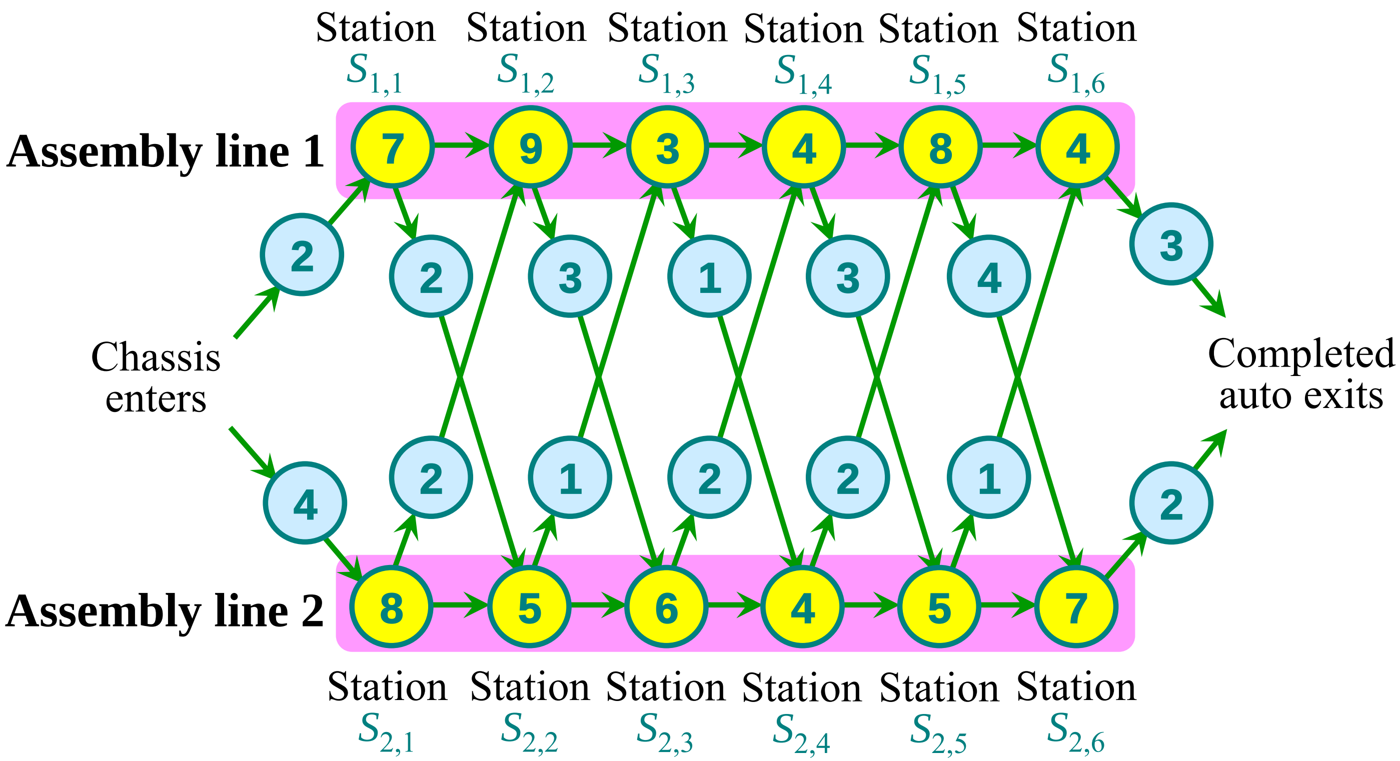
We obtain the ***recursive*** equations

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

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

$l_i[j]$ denote the line number i, whose station $j-1$ is used in a fastest way through station S_{ij}.

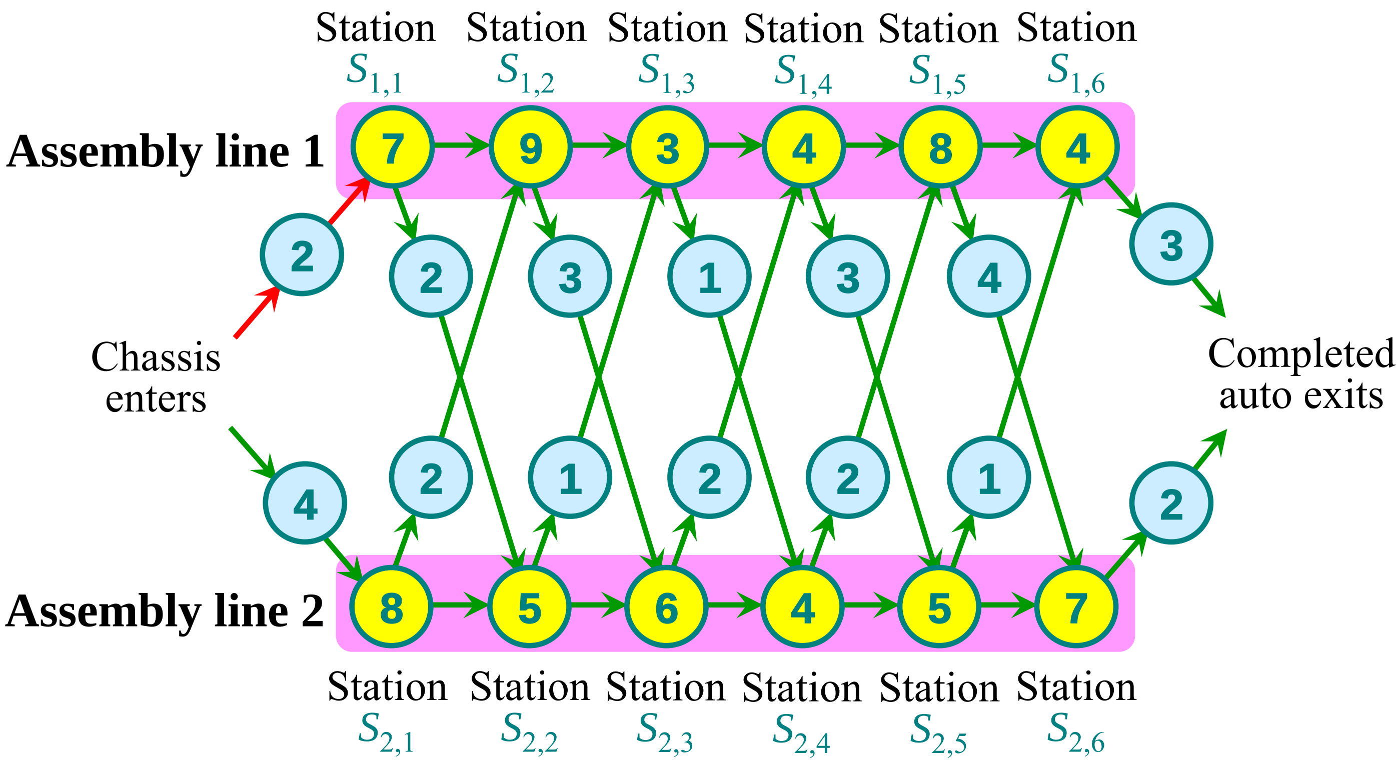
Computing the fastest times



j	**1**	**2**	**3**	**4**	**5**	**6**
$f_1[j]$						
$f_2[j]$						

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

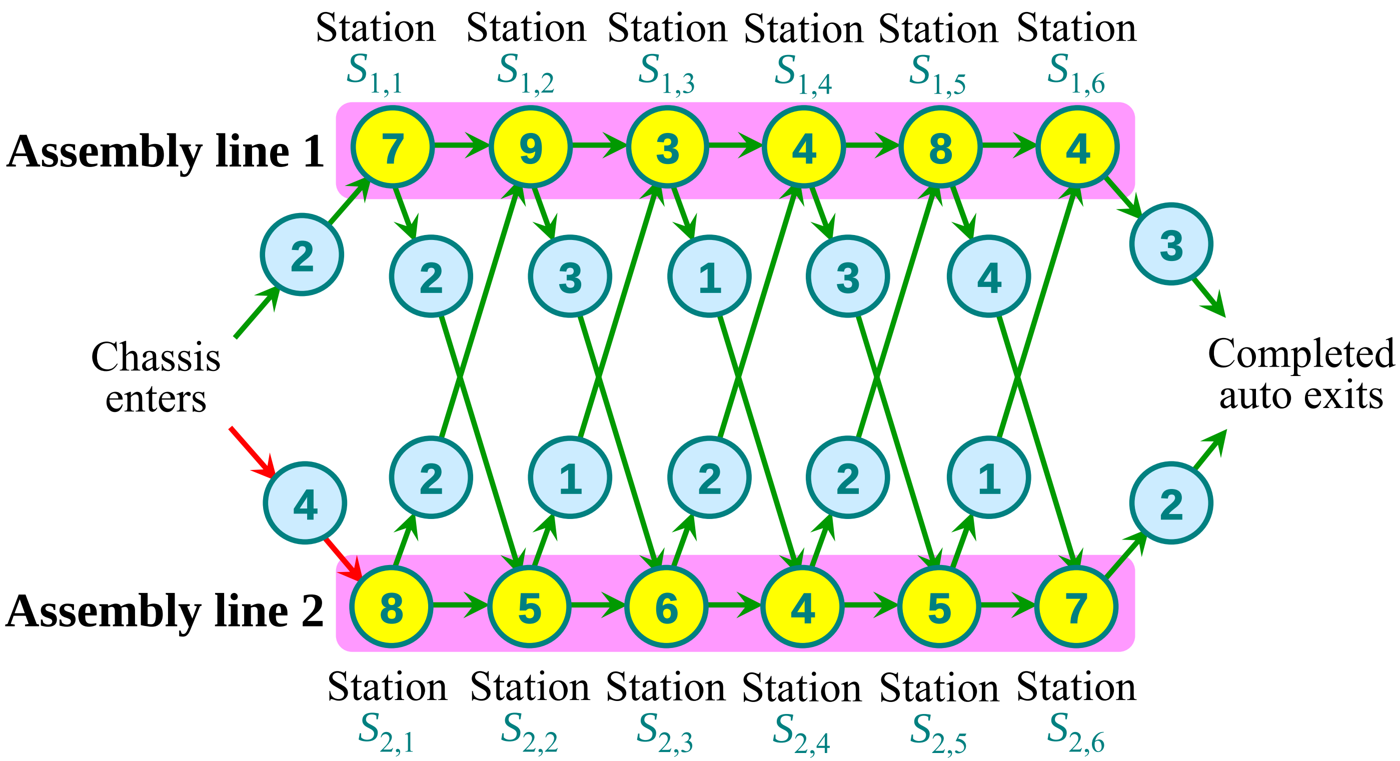
Computing the fastest times



j	1	2	3	4	5	6
$f_1[j]$	9					
$f_2[j]$						

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

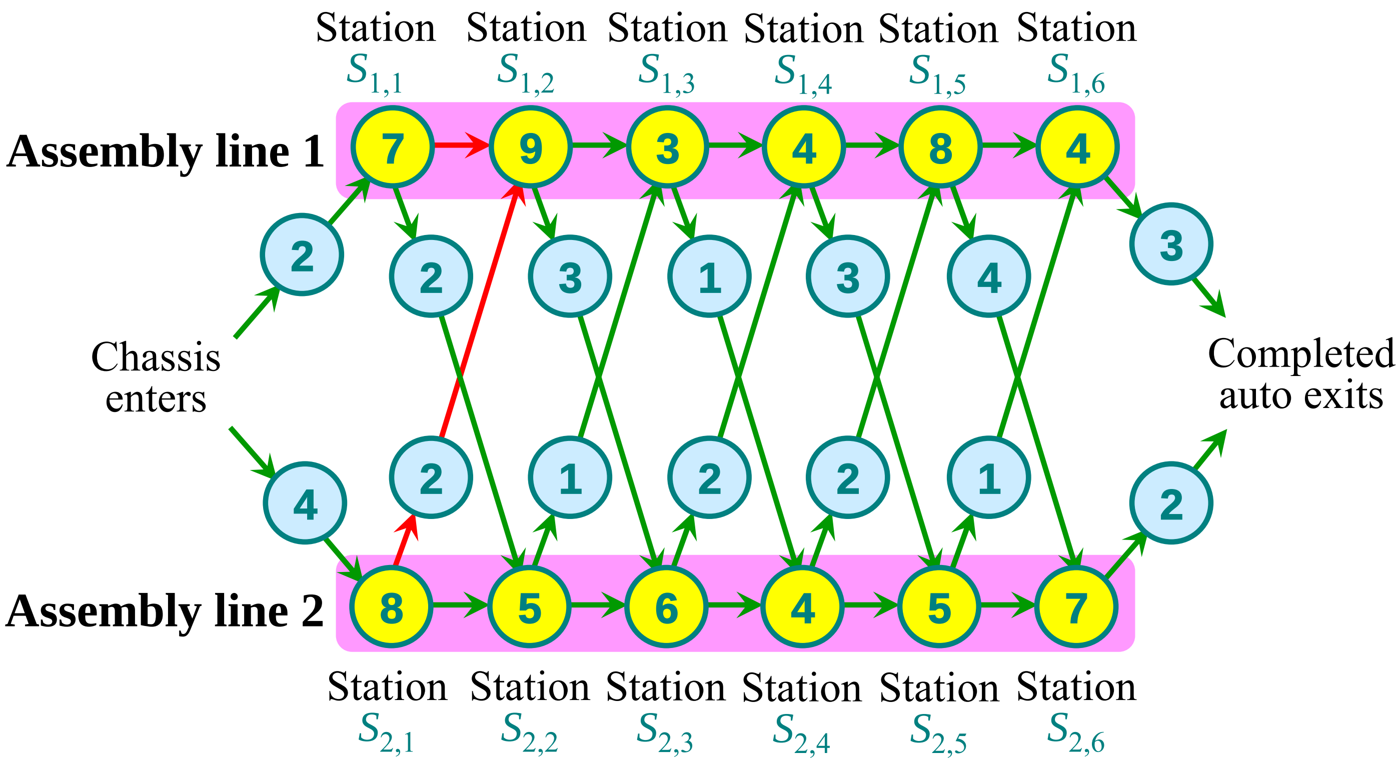
Computing the fastest times



j	1	2	3	4	5	6
$f_1[j]$	9					
$f_2[j]$	12					

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

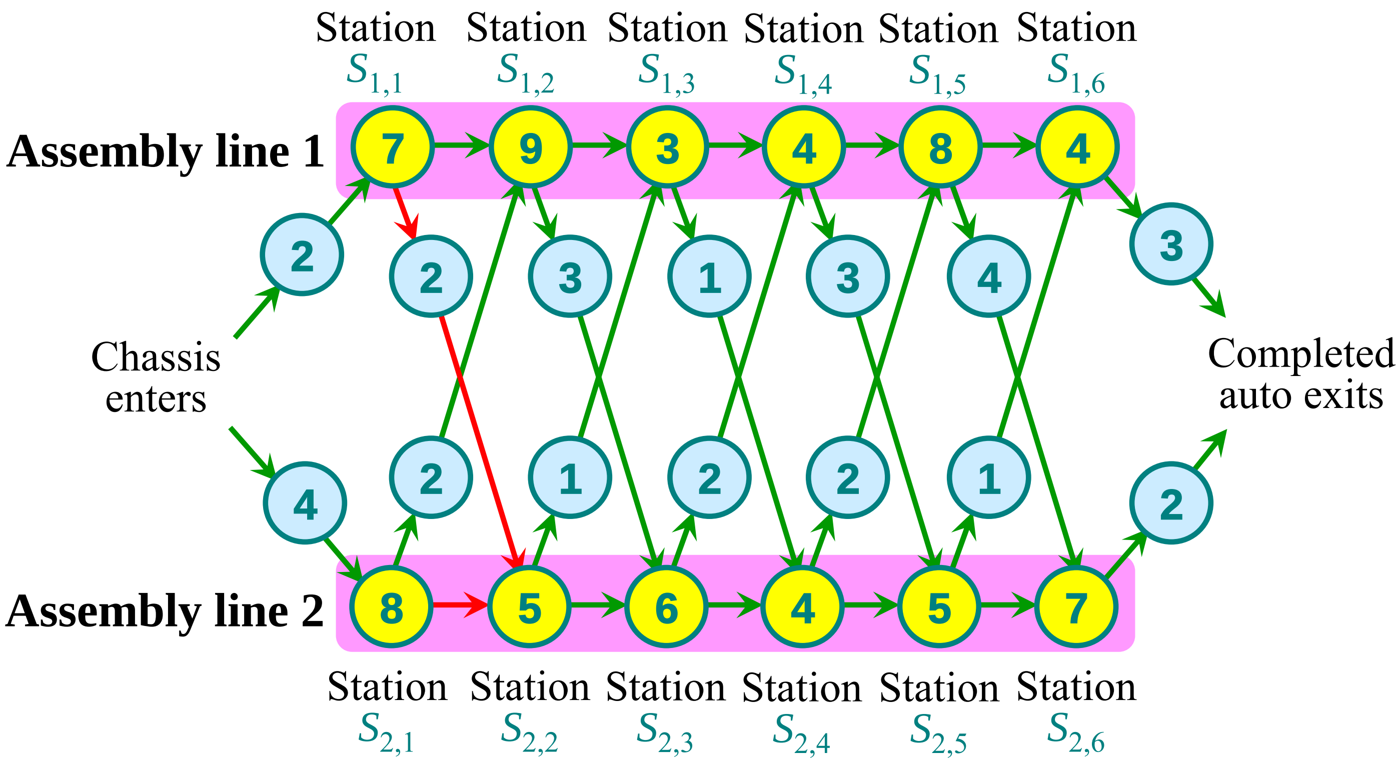
Computing the fastest times



j	1	2	3	4	5	6
$f_1[j]$	9	18				
$f_2[j]$	12					

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

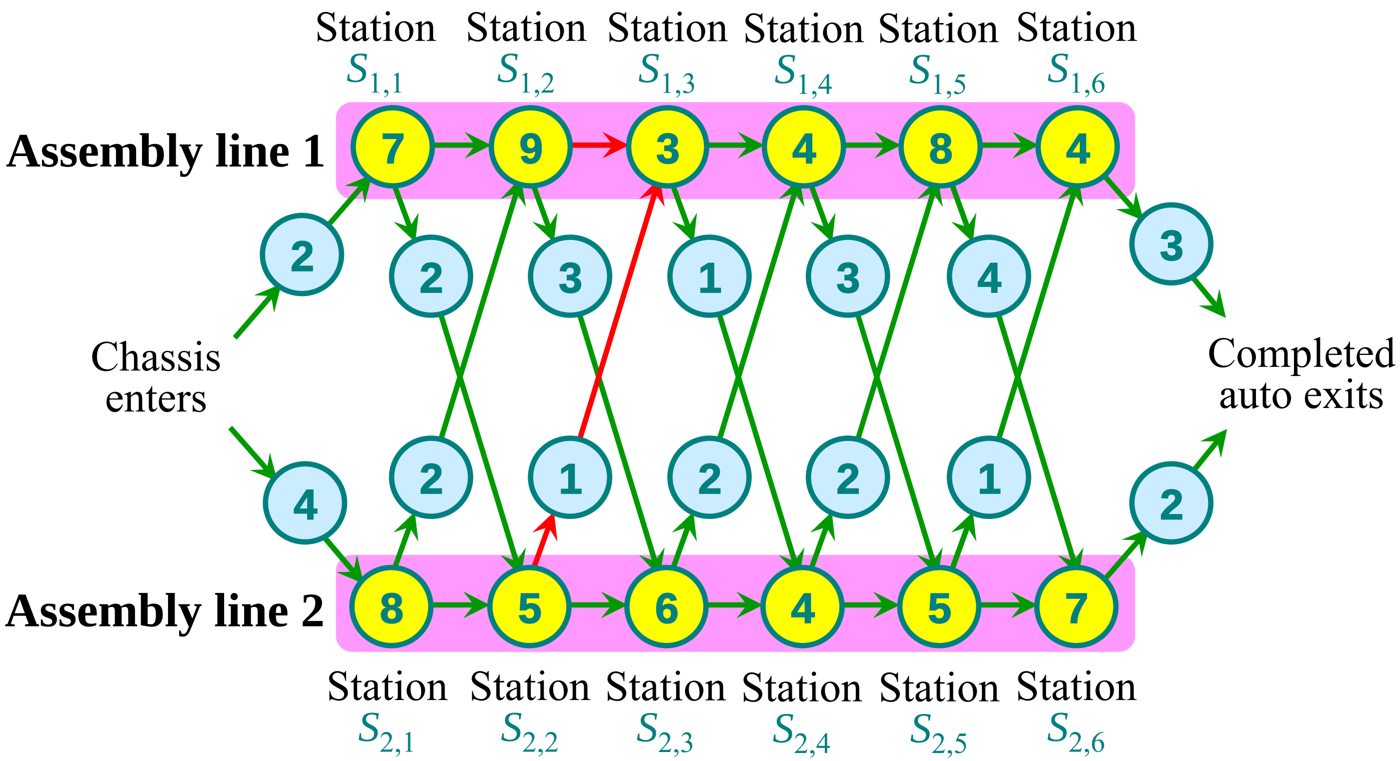
Computing the fastest times



j	1	2	3	4	5	6
$f_1[j]$	9	18				
$f_2[j]$	12	16				

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

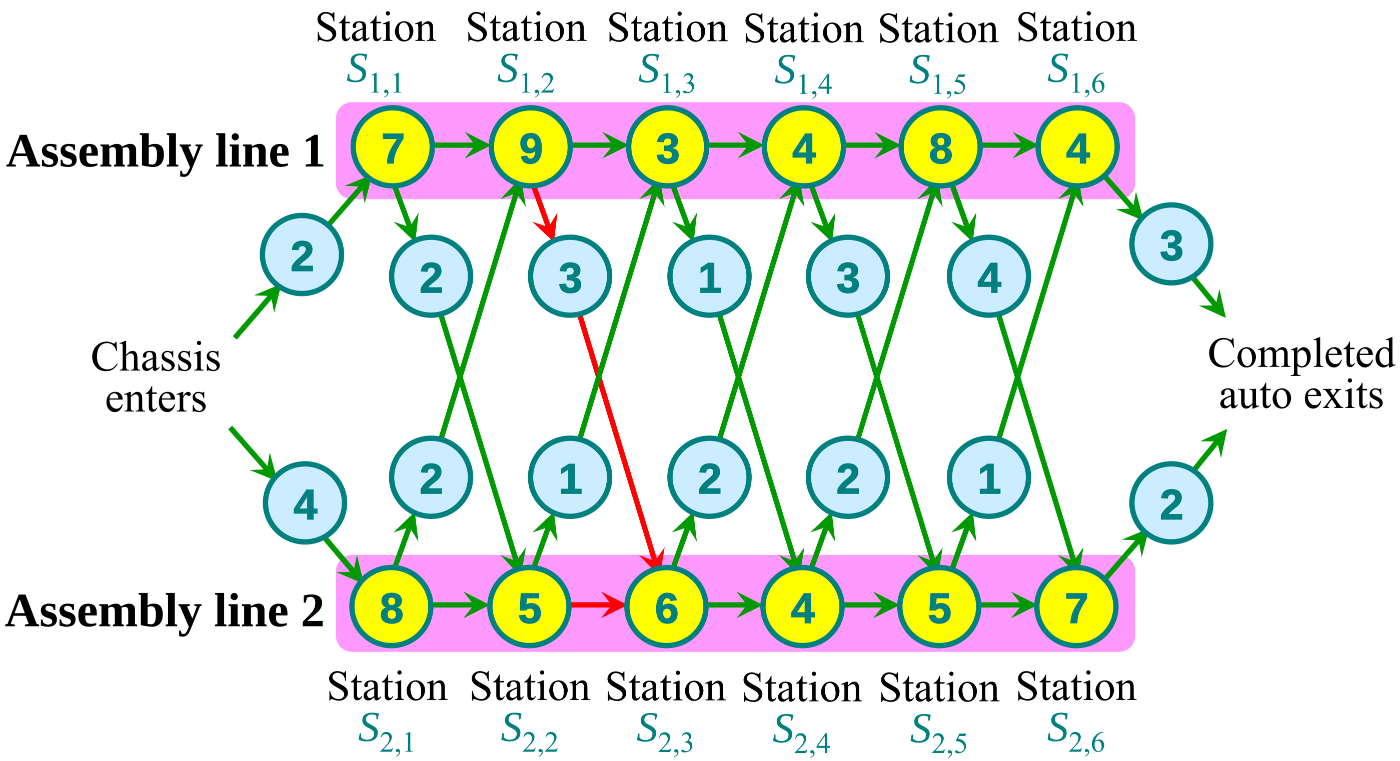
Computing the fastest times



j	1	2	3	4	5	6
$f_1[j]$	9	18	20			
$f_2[j]$	12	16				

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

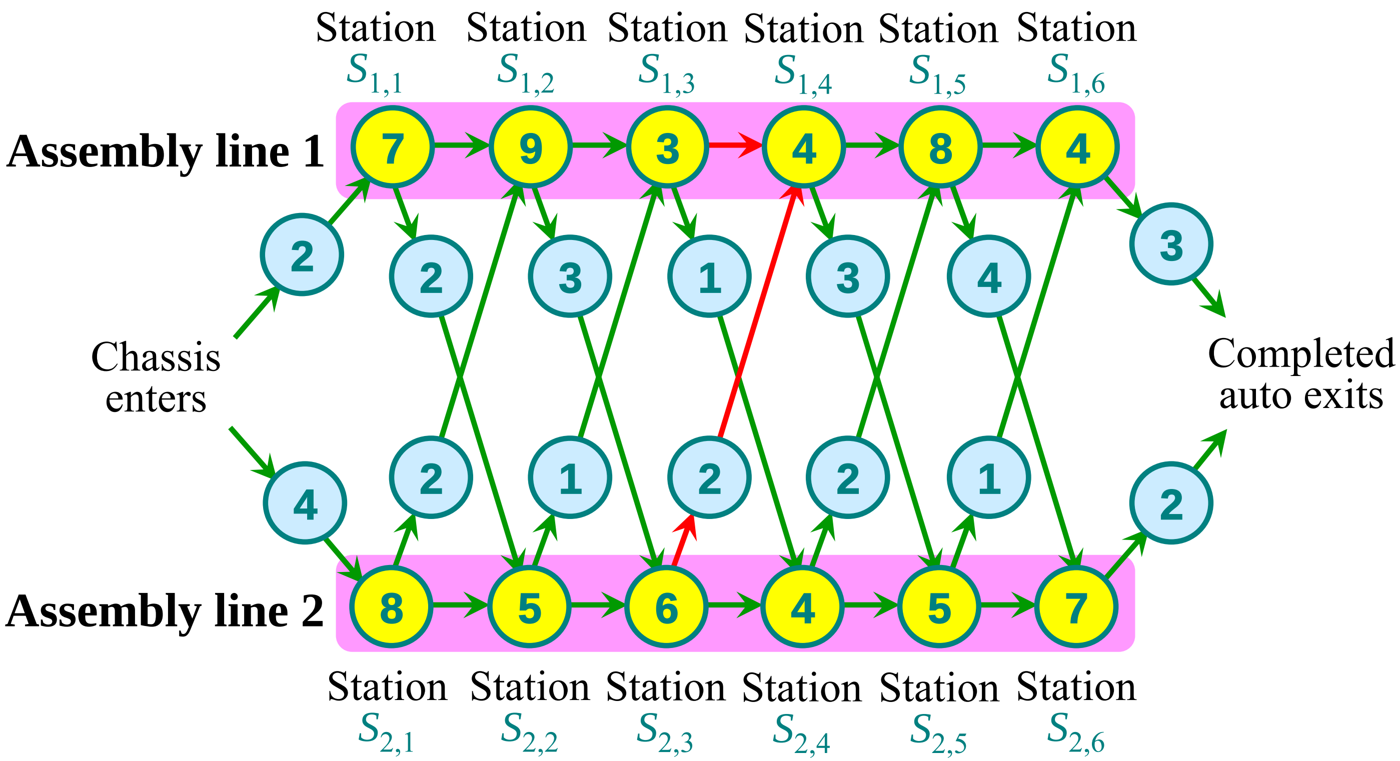
Computing the fastest times



j	1	2	3	4	5	6
$f_1[j]$	9	18	20			
$f_2[j]$	12	16	22			

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

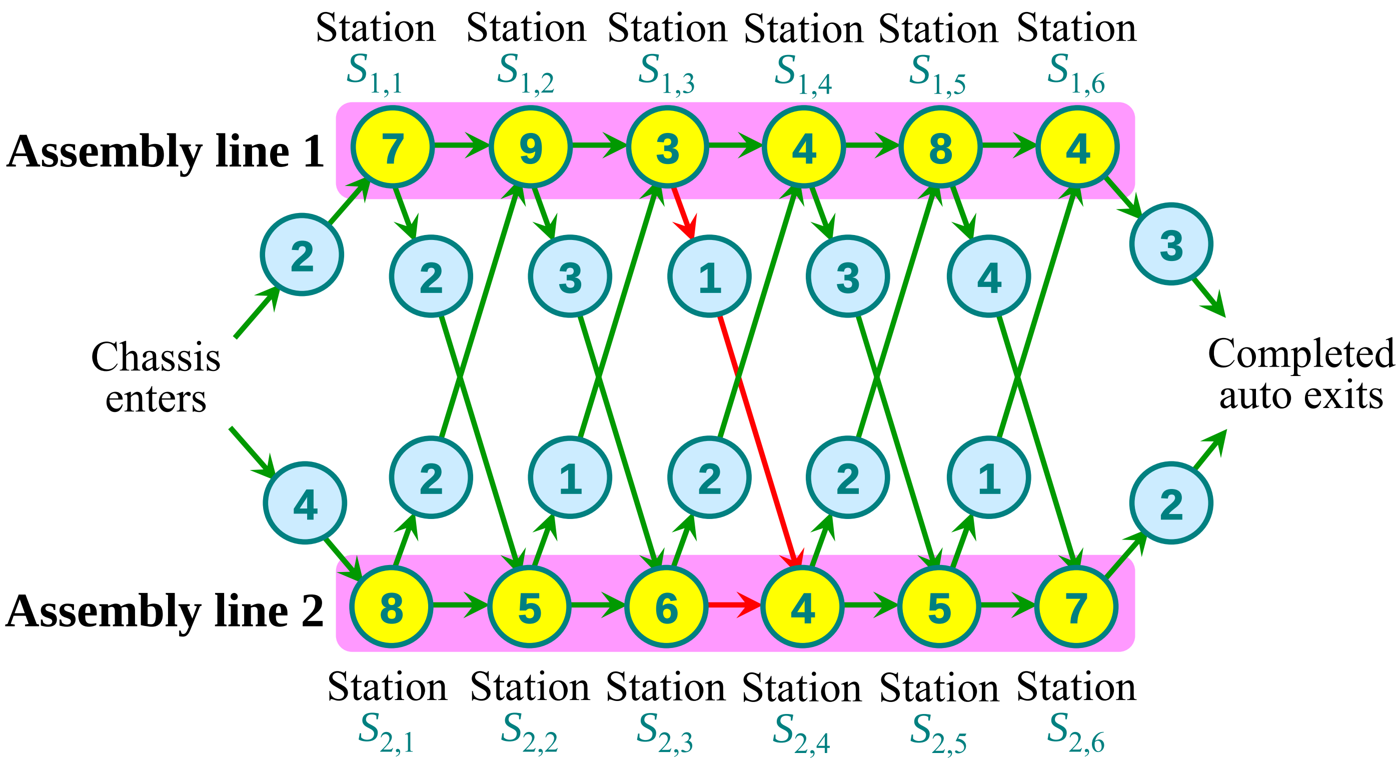
Computing the fastest times



j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24		
$f_2[j]$	12	16	22			

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

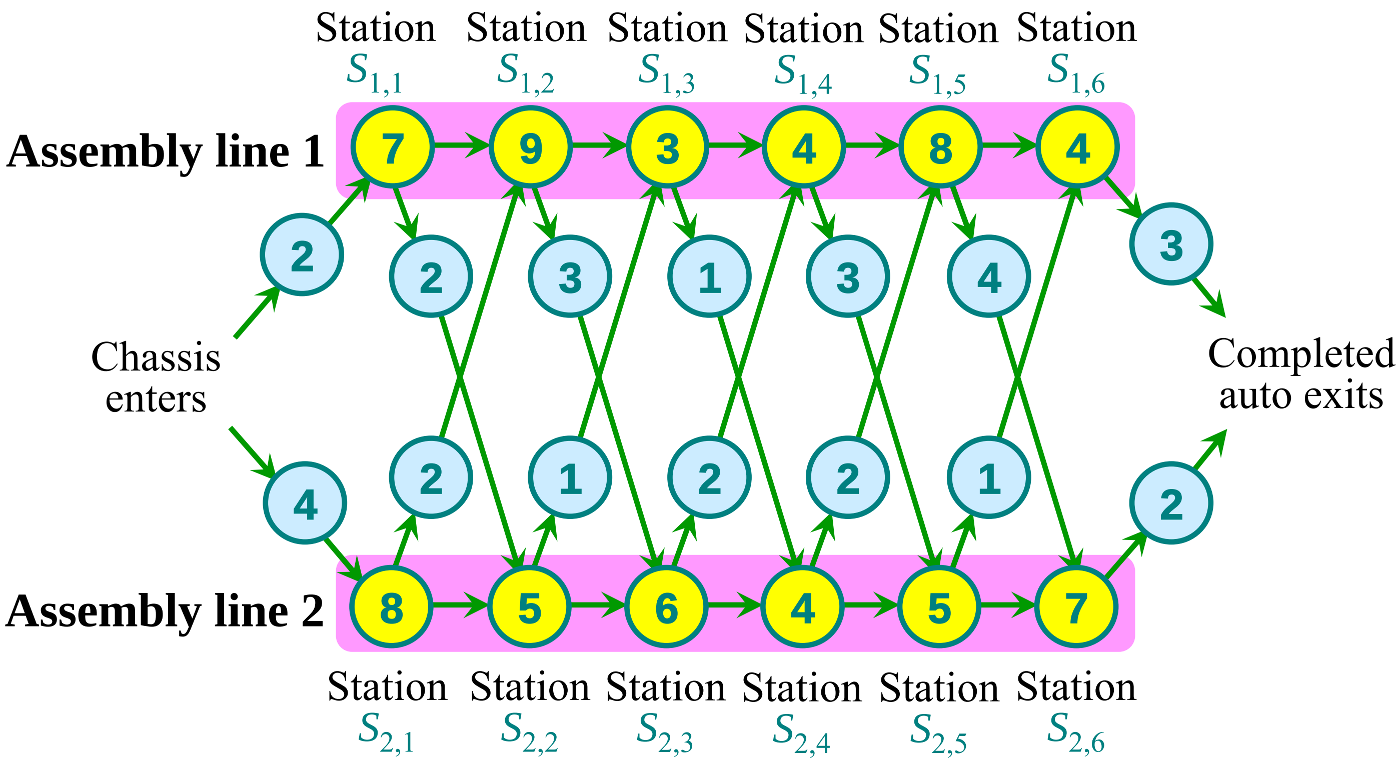
Computing the fastest times



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

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

Computing the fastest times



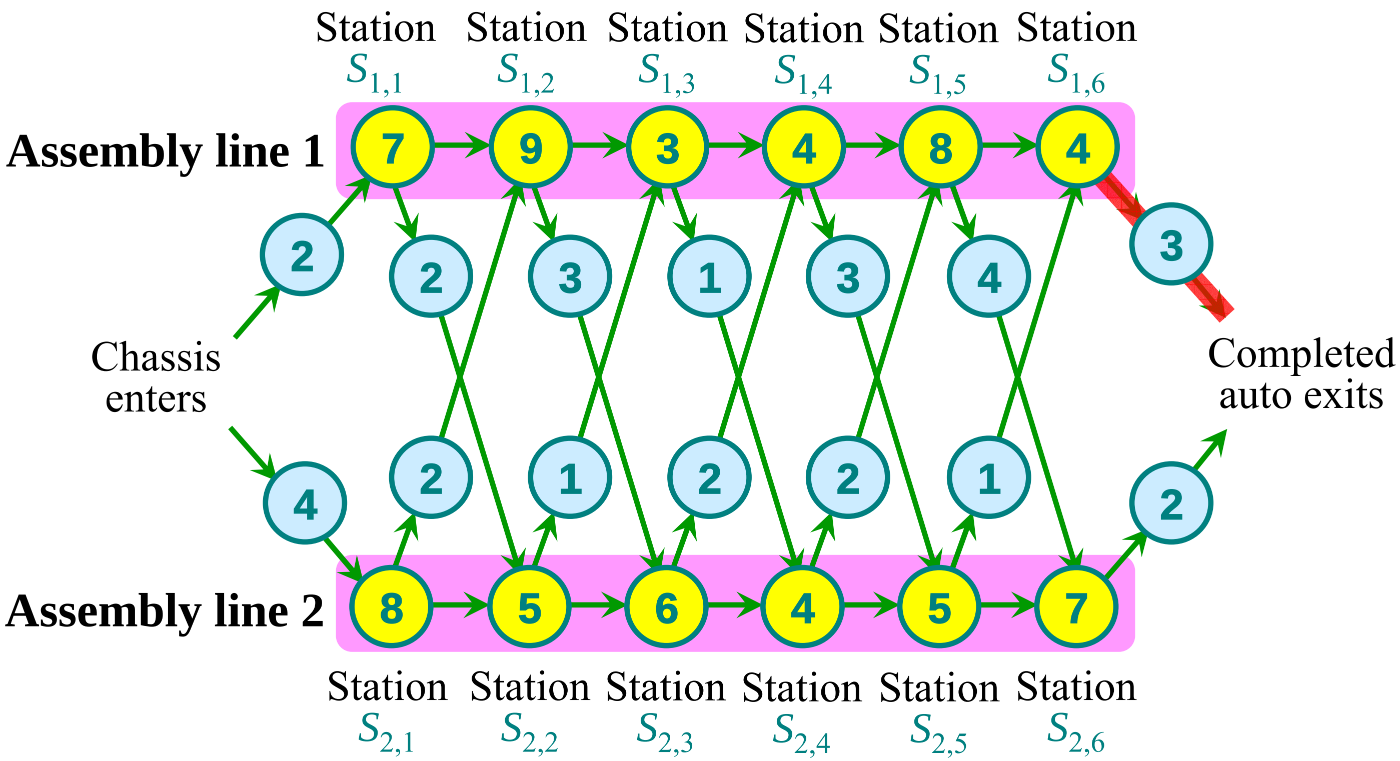
j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

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

$l^* = 1$

Constructing the fastest way



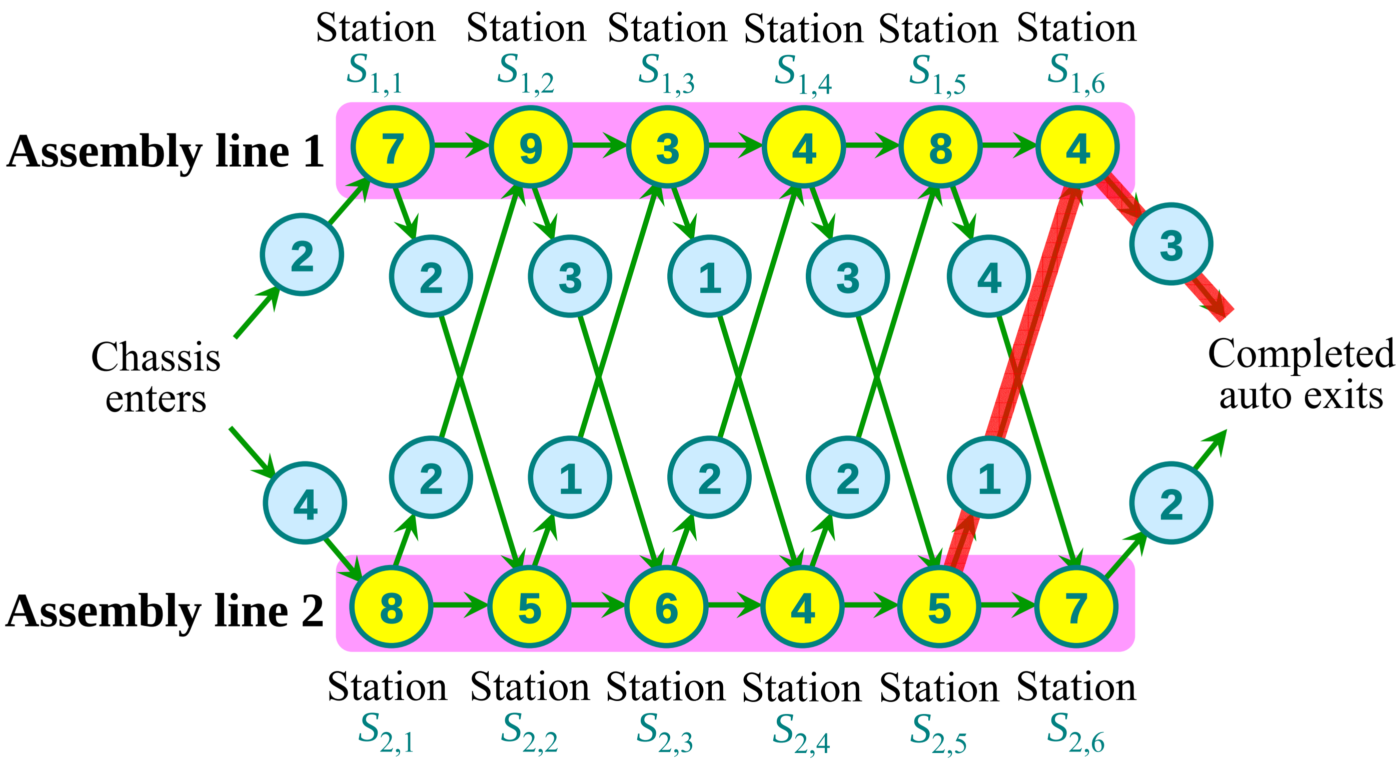
j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

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

$l^* = 1$

Constructing the fastest way



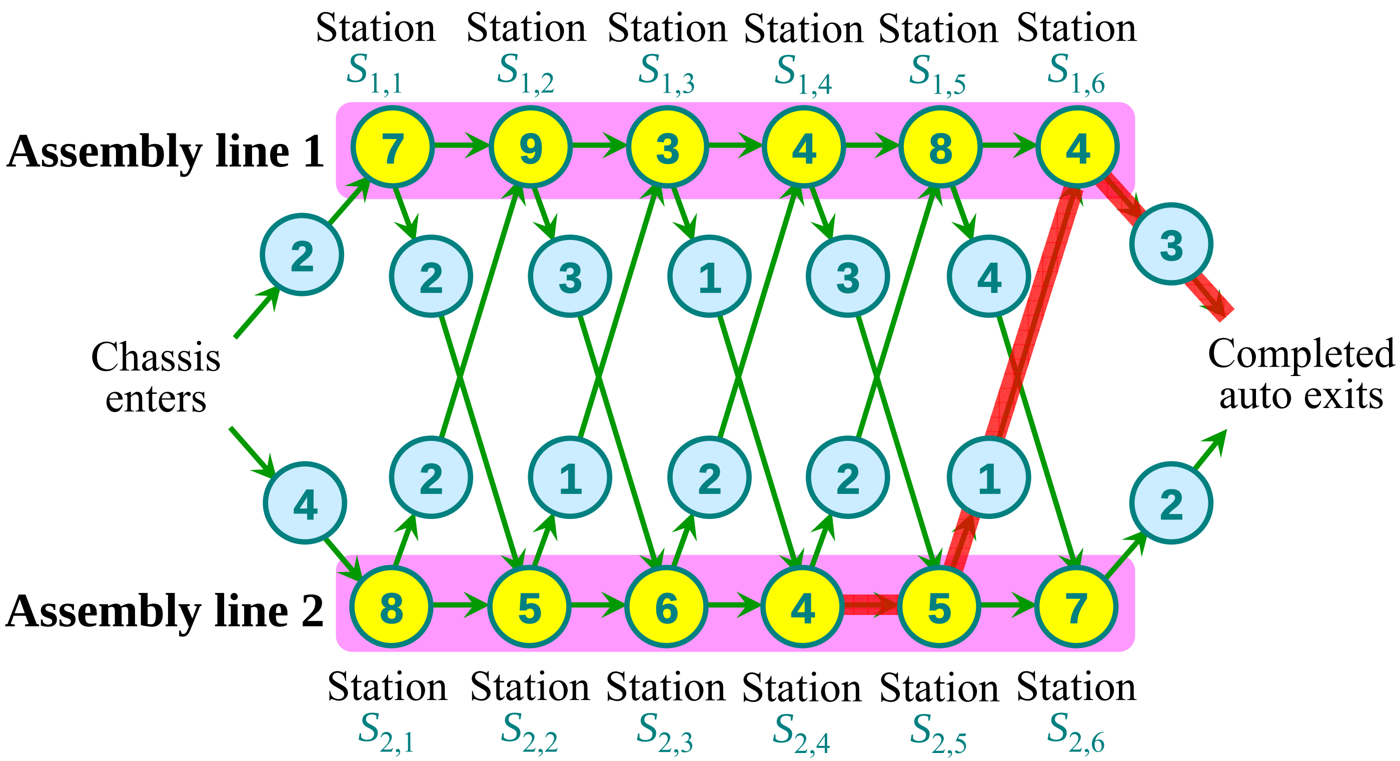
j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

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

$l^* = 1$

Constructing the fastest way



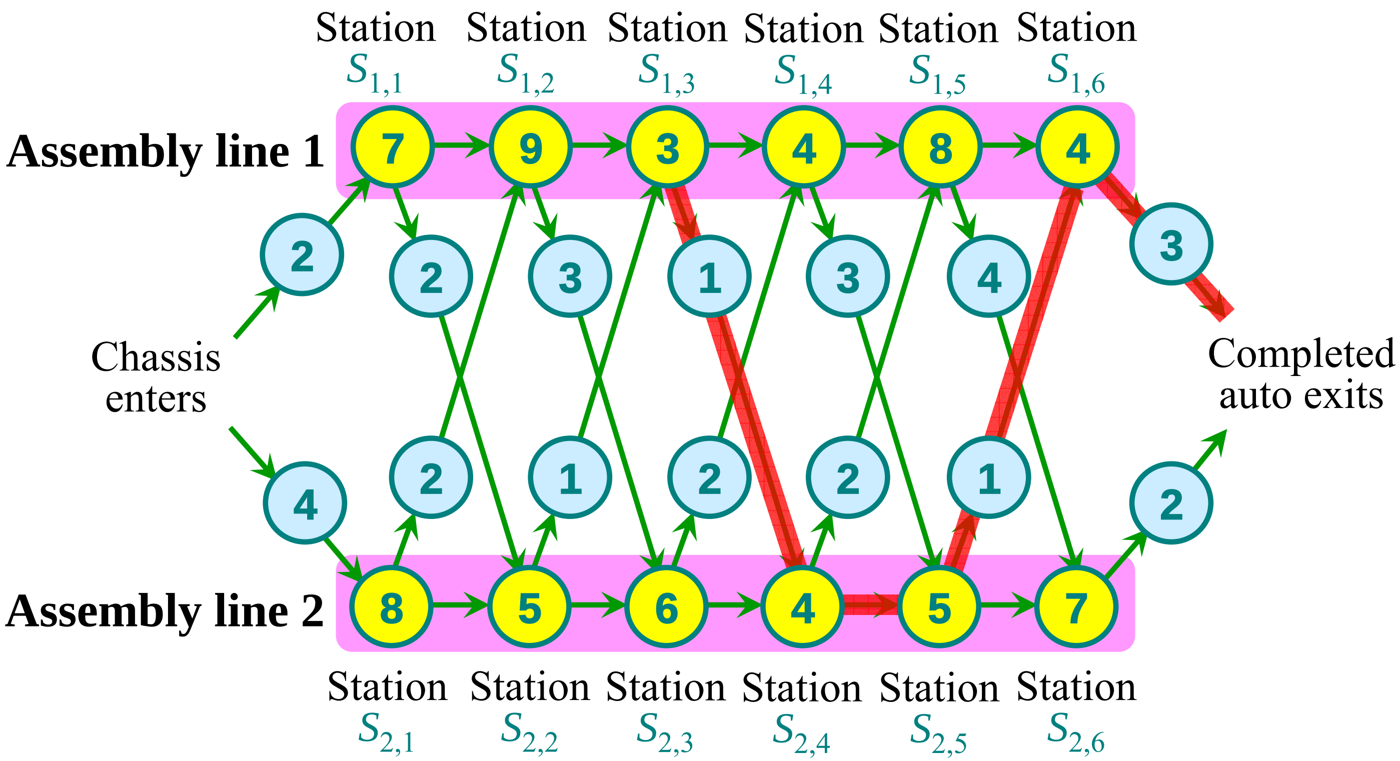
j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

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

$l^* = 1$

Constructing the fastest way



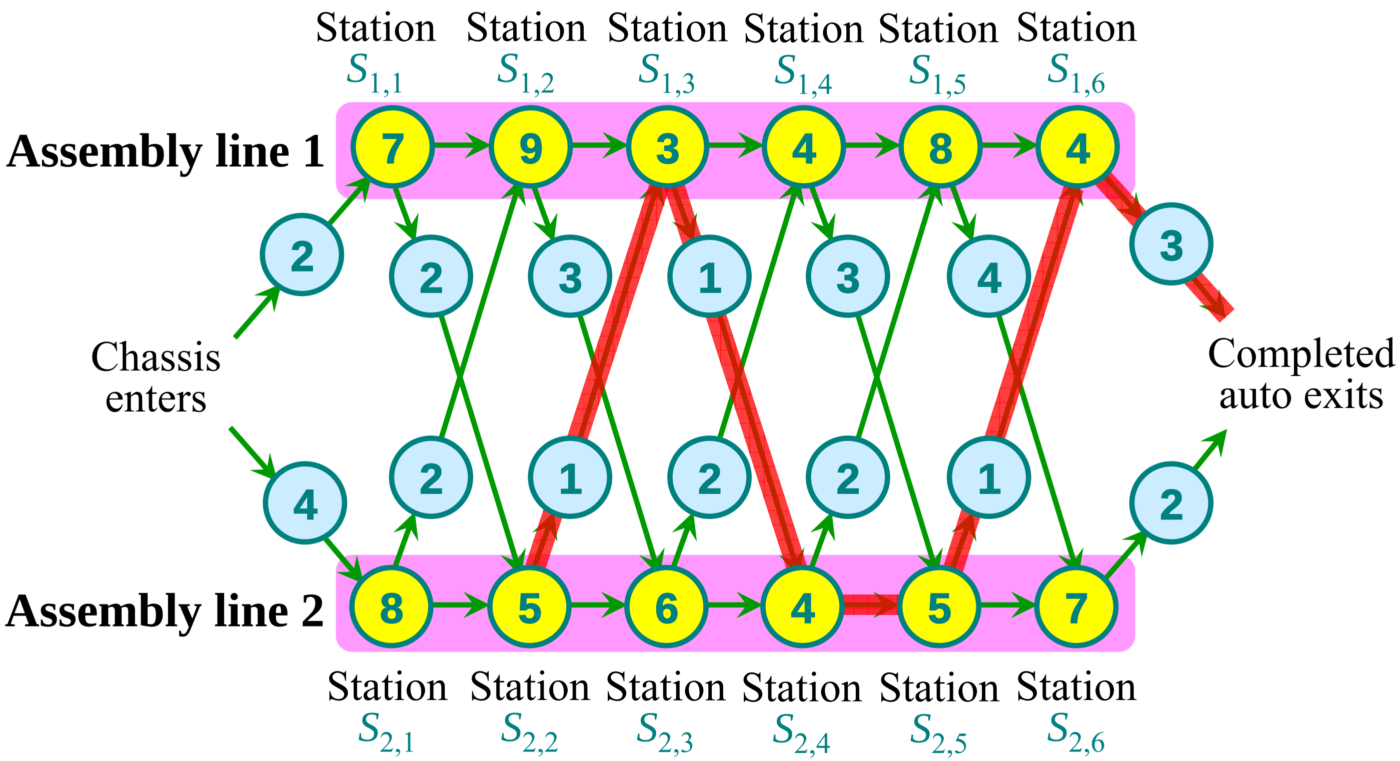
j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

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

$l^* = 1$

Constructing the fastest way



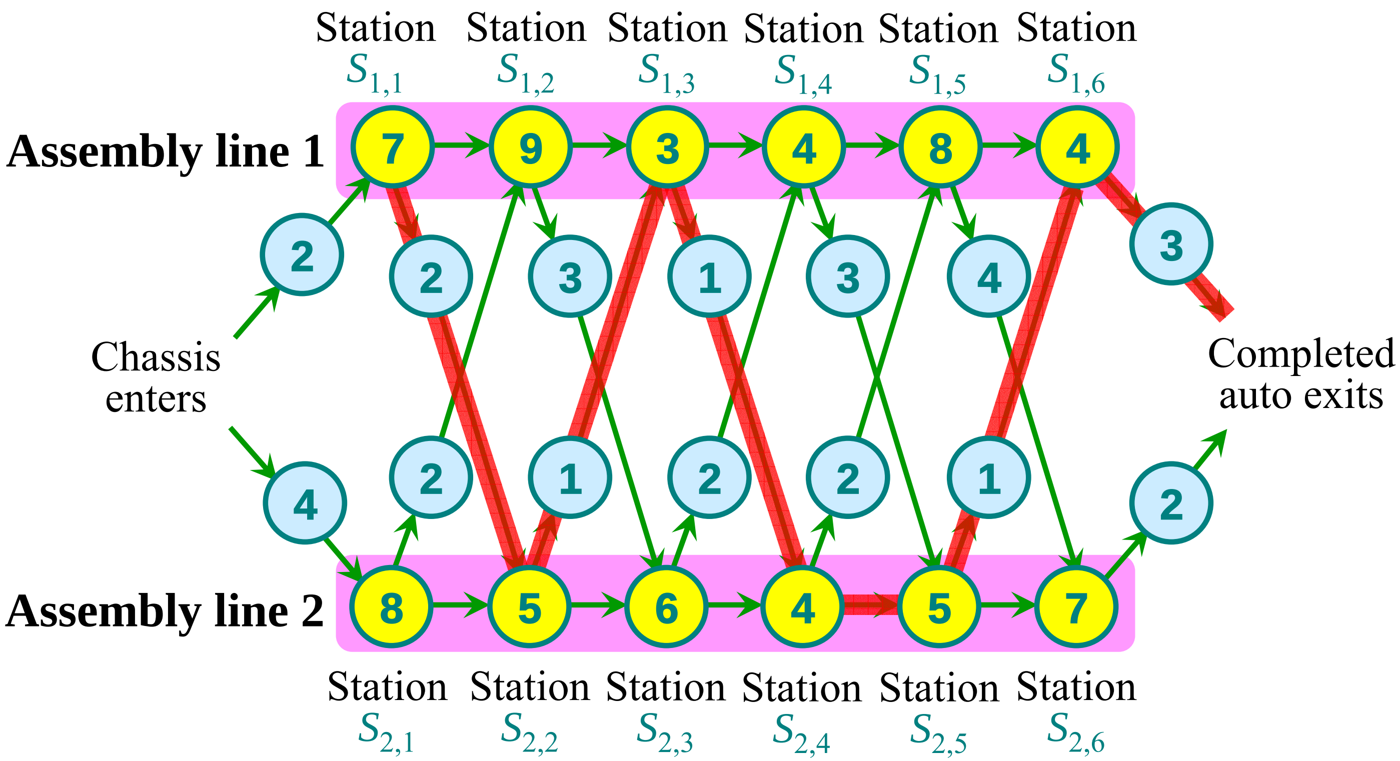
j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

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

$l^* = 1$

Constructing the fastest way



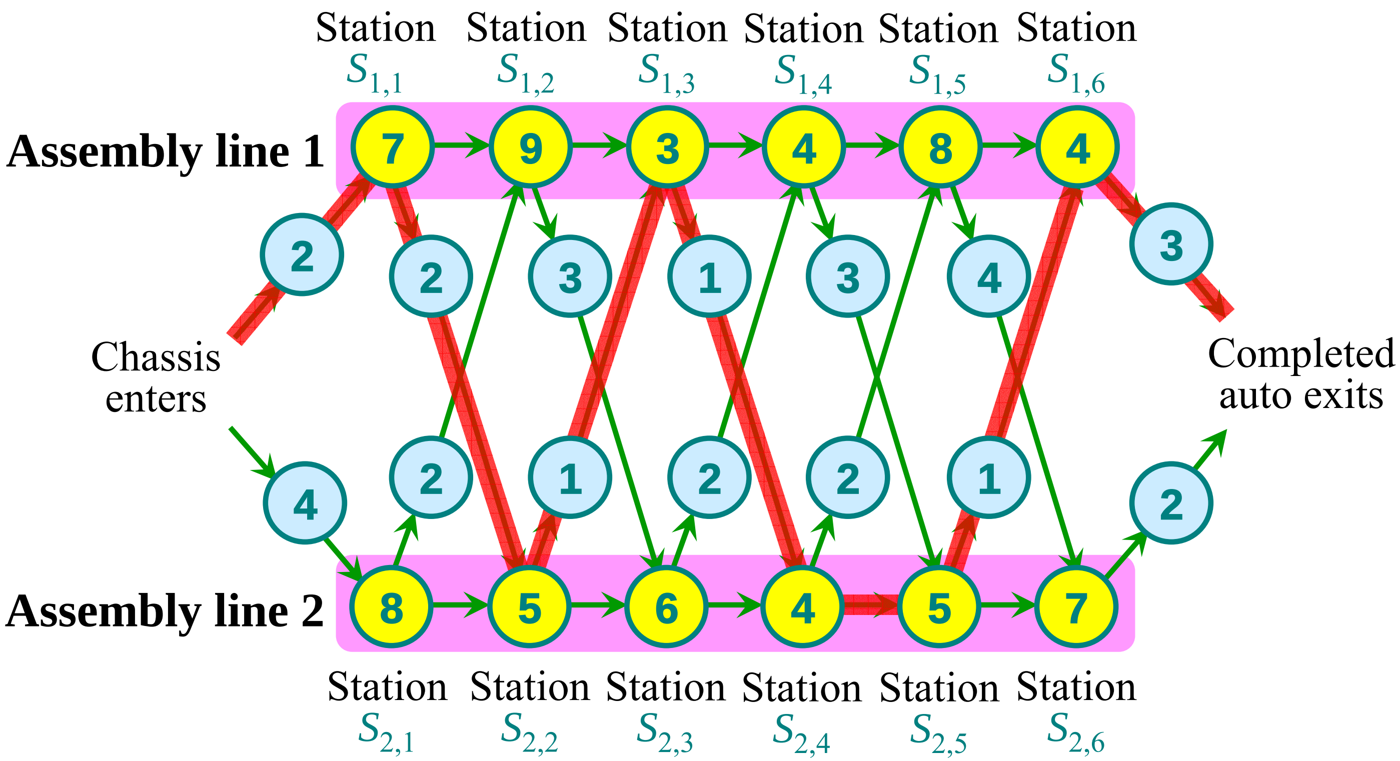
j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

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

$l^* = 1$

Constructing the fastest way



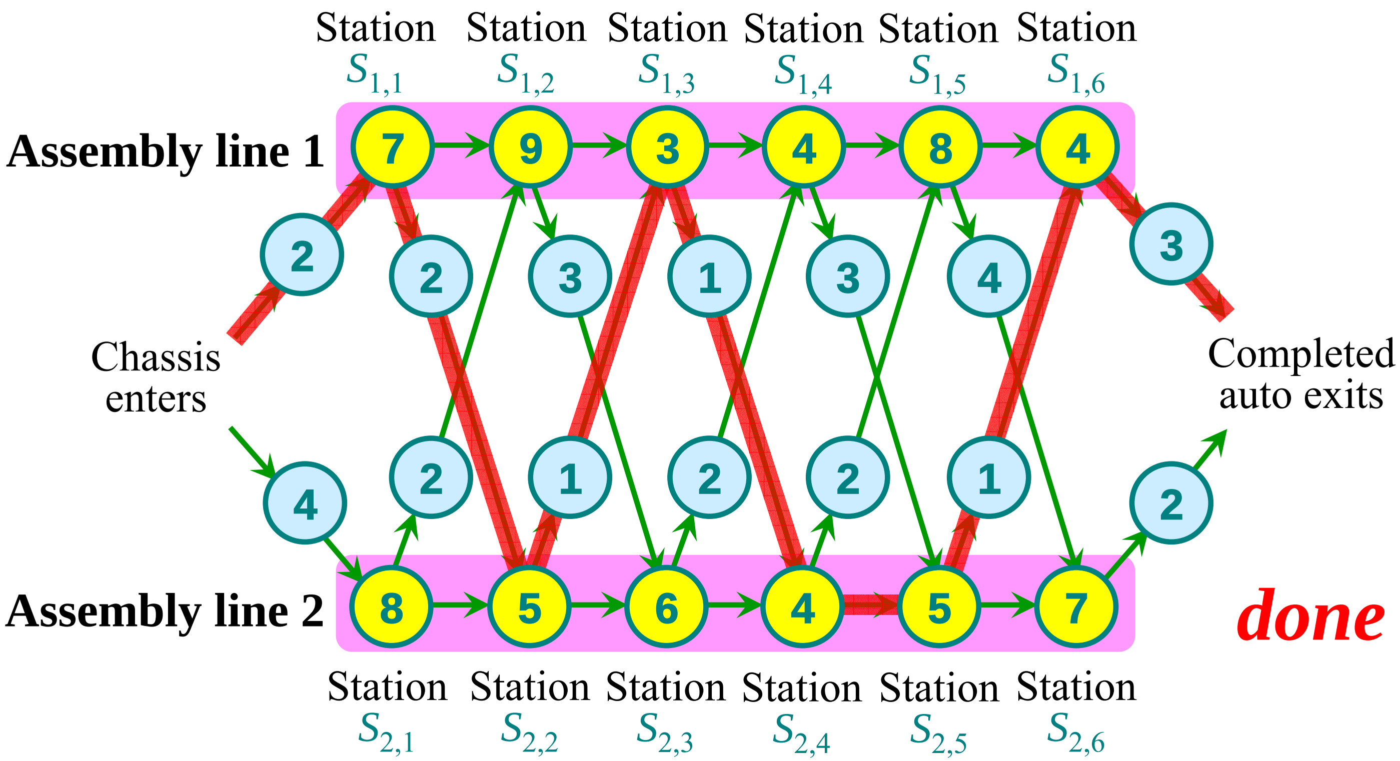
j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

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

$l^* = 1$

Constructing the fastest way



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

$l^* = 1$

Matrix multiplication

$$\left.\begin{array}{l}\textbf{Input: } A = [a_{ik}],\ B = [b_{kj}].\\[1ex] \textbf{Output: } C = [c_{ij}] = A \cdot B.\end{array}\right\}$$

for $i \leftarrow 1$ **to** $rows[A]$
  do for $j \leftarrow 1$ **to** $columns[B]$
    do $c_{ij} \leftarrow 0$
      for $k \leftarrow 1$ **to** $columns[A]$
        do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Number of scalar multiplications

$= rows[A] \times columns[A] \times columns[B]$

Matrix-chain multiplication

A_1: 10×100,
A_2: 100×5,
A_3: 5×50.

$((A_1 A_2) A_3)$

$$\left.\begin{array}{l} 10 \times 100 \times 5 = 5{,}000 \\ 10 \times 5 \times 50 \quad = 2{,}500 \end{array}\right\} \Rightarrow 5{,}000 + 2{,}500 = \mathbf{7{,}500}$$

$(A_1 (A_2 A_3))$

$$\left.\begin{array}{l} 100 \times 5 \times 50 = 25{,}000 \\ 10 \times 100 \times 5 = 50{,}000 \end{array}\right\} \Rightarrow 25{,}000 + 50{,}000 = \mathbf{75{,}000}$$

First parenthesization is **10** *times faster.*

Matrix-chain multiplication

Given a chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1 A_2 \ldots A_n$ in a way that minimizes the number of scalar multiplications.

- We are not actually multiplying matrices. Our goal is only to determine an order for multiplying matrices that has the lowest cost.
- Typically, the time invested in determining this optimal order is more than paid for by the time saved later on when actually performing the matrix multiplications

Join

student [<u>学生学号</u> 学生姓名]
course [<u>课程名称</u> 教师姓名]
grade [<u>学生学号</u> <u>课程名称</u> 成绩]
teacher [<u>教师姓名</u> 教师职称]

⇩

[<u>学生学号</u> 学生姓名 <u>课程名称</u> 成绩 教师姓名 教师职称]

Join

course

课程名称	教师姓名
Web应用基础	许劭
数据结构与算法	司马徽
Java程序设计	李先隆

teacher

教师姓名	教师职称
许劭	讲师
司马徽	教授
李先隆	副教授

Cartesian Product

课程名称	教师姓名	教师姓名	教师职称
Web应用基础	许劭	许劭	讲师
Web应用基础	许劭	司马徽	教授
Web应用基础	许劭	李先隆	副教
数据结构与算法	司马徽	许劭	讲师
数据结构与算法	司马徽	司马徽	教授
数据结构与算法	司马徽	李先隆	副教
Java程序设计	李先隆	许劭	讲师
Java程序设计	李先隆	司马徽	教授
Java程序设计	李先隆	李先隆	副教

where course.教师姓名＝teacher.教师姓名

课程名称	教师姓名	教师职称
Web应用基础	许劭	讲师
数据结构与算法	司马徽	教授
Java程序设计	李先隆	副教

Join

grade

学生学号	课程名称	成绩
200701	Web应用基础	86
200702	数据结构与算法	88
200703	Web应用基础	95
200704	Web应用基础	76
200705	数据结构与算法	90
200706	Java程序设计	68
200707	Java程序设计	45
200708	Web应用基础	82
200709	Java程序设计	85

temporary1

课程名称	教师姓名	教师职称
Web应用基础	许劭	讲师
数据结构与算法	司马徽	教授
Java程序设计	李先隆	副教

grade join temporary1 on grade.课程名称=temporary1.课程名称

学生学号	课程名称	成绩	教师姓名	教师职称
200701	Web应用基础	86	许劭	讲师
200702	数据结构与算法	88	司马徽	教授
200703	Web应用基础	95	许劭	讲师
200704	Web应用基础	76	许劭	讲师
200705	数据结构与算法	90	司马徽	教授
200706	Java程序设计	68	李先隆	副教授
200707	Java程序设计	45	李先隆	副教授
200708	Web应用基础	82	许劭	讲师
200709	Java程序设计	85	李先隆	副教授

Join

student

学生学号	学生姓名
200701	曹操
200702	郭嘉
…	…

temporary2

学生学号	课程名称	成绩	教师姓名	教师职称
200701	Web应用基础	86	许劭	讲师
200702	数据结构与算法	88	司马徽	教授
…	…	…	…	…

student join temporary2 on student.学生学号=temporary2.学生学号

学生学号	学生姓名	课程名称	成绩	教师姓名	教师职称
200701	曹操	Web应用基础	86	许劭	讲师
200702	郭嘉	数据结构与算法	88	司马徽	教授
200703	贾诩	Web应用基础	95	许劭	讲师
200704	刘备	Web应用基础	76	许劭	讲师
200705	诸葛亮	数据结构与算法	90	司马徽	教授
200706	关羽	Java程序设计	68	李先隆	副教授
200707	张飞	Java程序设计	45	李先隆	副教授
200708	孙权	Web应用基础	82	许劭	讲师
200709	周瑜	Java程序设计	85	李先隆	副教授

Brute-force

$P(n)$: denote the number of alternative parenthesizations of a sequence of n matrices.

We obtain the recurrence

$$P(n) = \begin{cases} 1 & \text{if } n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2. \end{cases}$$

This recurrence is the sequence of ***Catalan numbers***, which grows as $\Omega(4^n / n^{3/2})$.

It is infeasible!

Structure of an optimal parenthesization

- Any parenthesization of the product $A_i \ldots A_j$ must split the product between A_k and A_{k+1} for some integer k in the range $i \leq k < j$. For some k, we first compute the matrices $A_i \ldots A_k$ and $A_{k+1} \ldots A_j$ and then multiply them together to produce the final product $A_i \ldots A_j$.
- Suppose that an optimal parenthesization of $A_i \ldots A_j$ splits the product between A_k and A_{k+1}. Then the parenthesization of the "prefix" subchain $A_i \ldots A_k$ within this optimal parenthesization of $A_i \ldots A_j$ must be an optimal parenthesization of $A_i \ldots A_k$.
- We can build an optimal solution to an instance of the matrix-chain multiplication problem by splitting the problem into two ***subproblems***, finding optimal solutions to subproblem, and then combining these optimal subproblem solutions.

Recursive solution

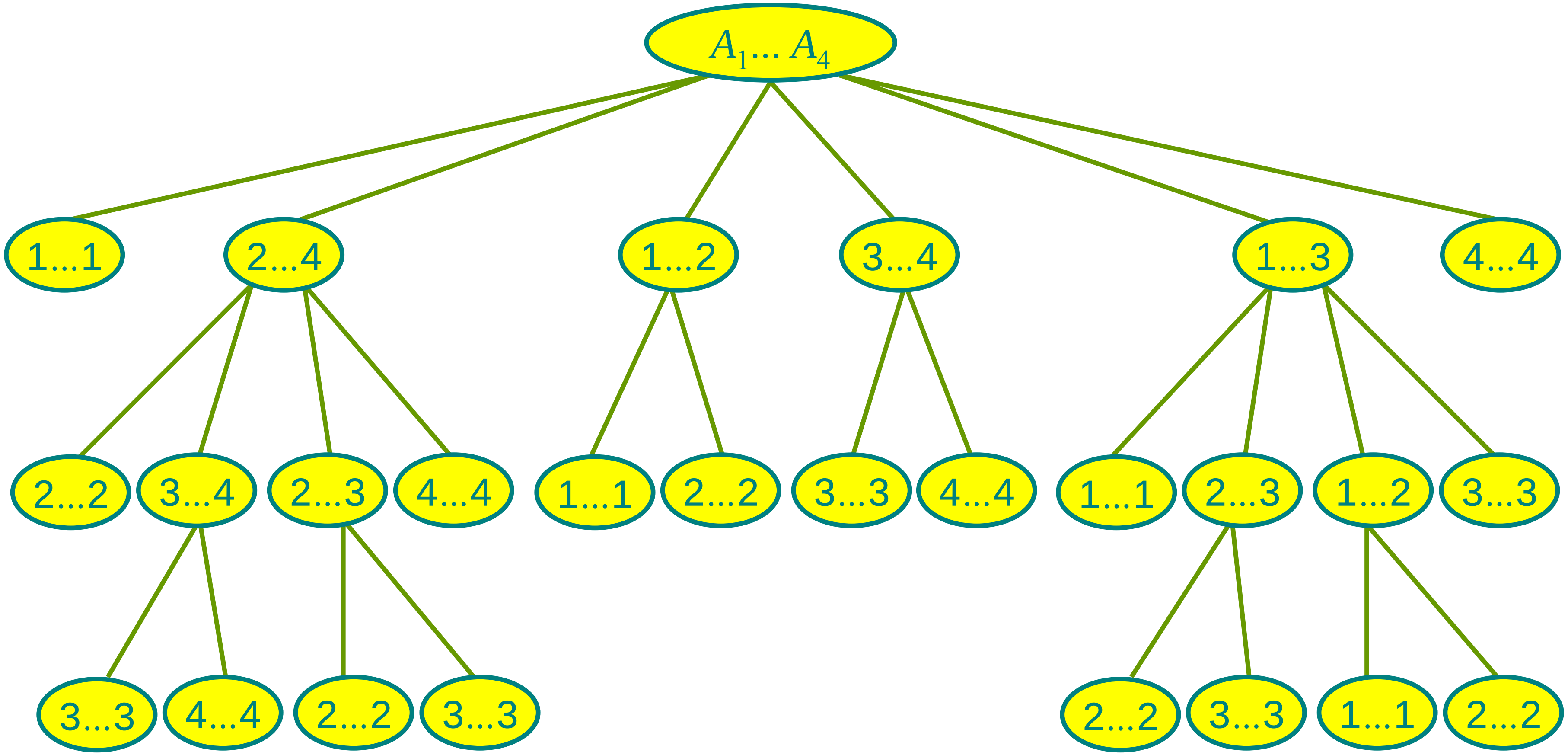
$m[i,j]$ denote the minimum number of scalar multiplications needed to compute the matrix $A_i \ldots A_j$.

We obtain the ***recursive*** equations

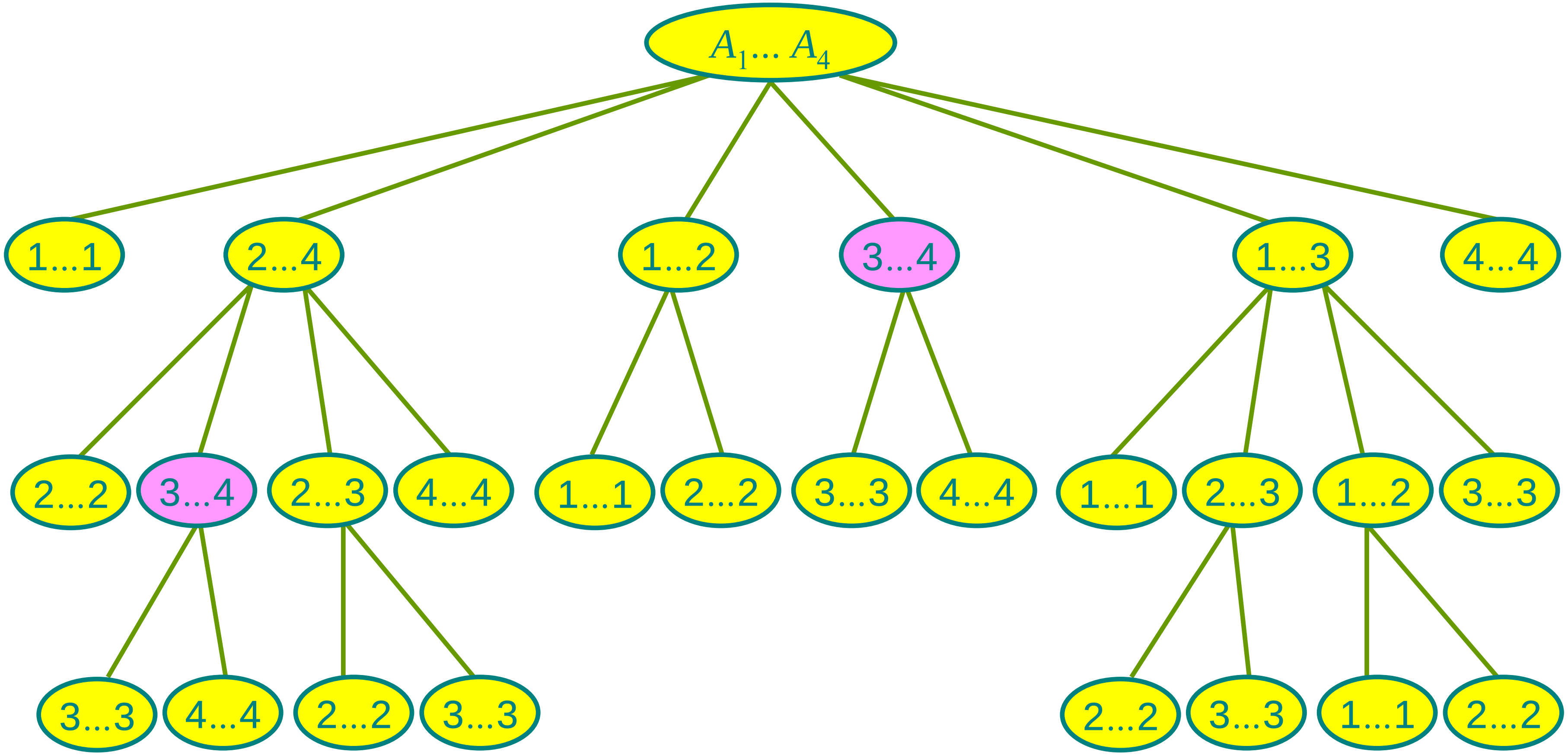
$$m[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j. \end{cases}$$

Our goal is $m[1,n]$.

Recursion tree

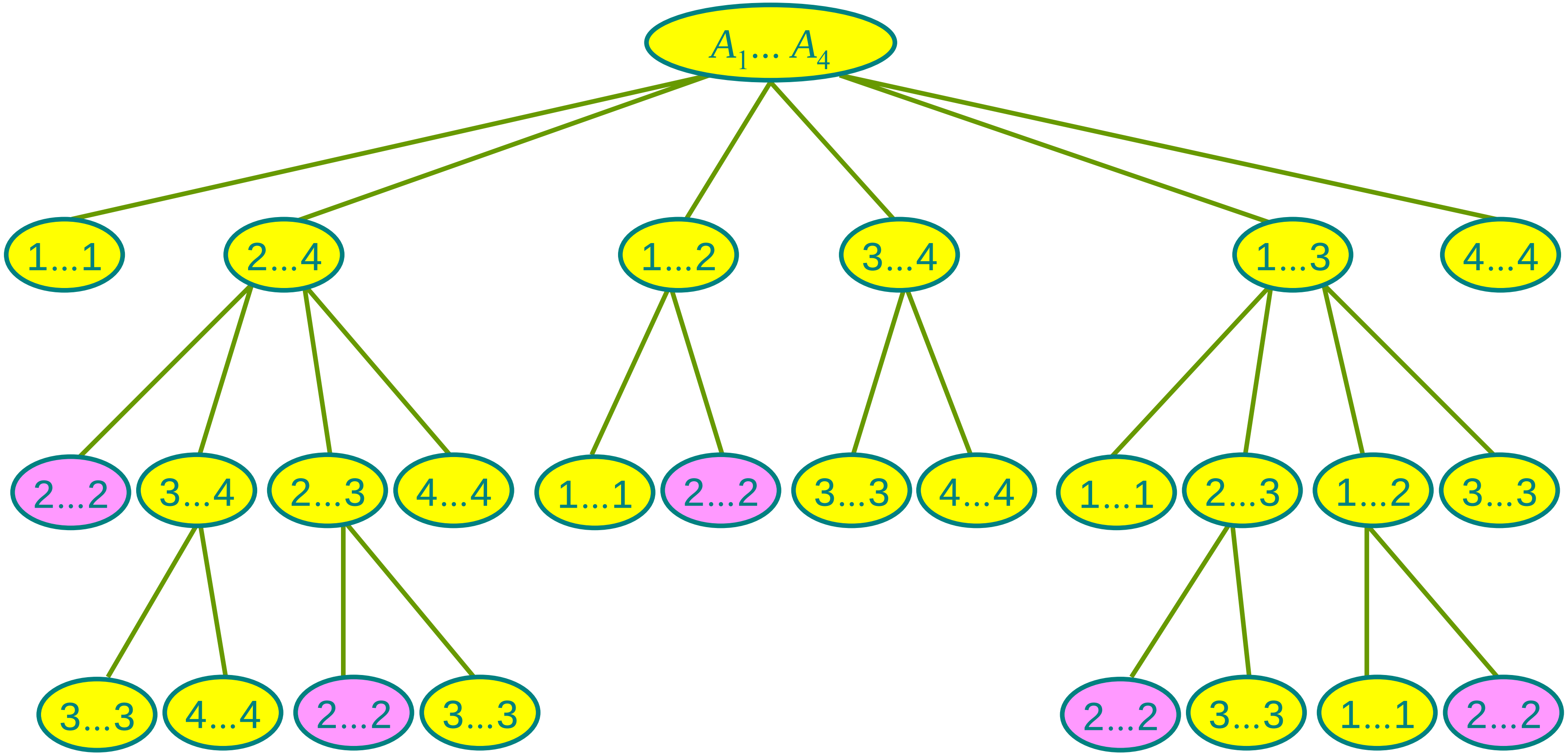


Recursion tree



Overlapping subproblems

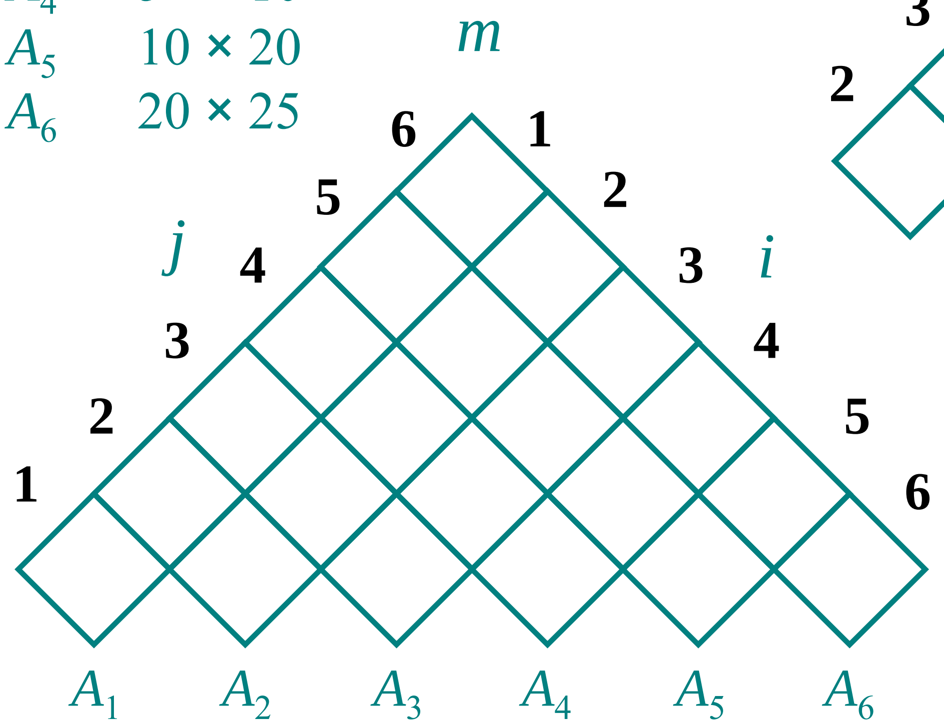
Recursion tree

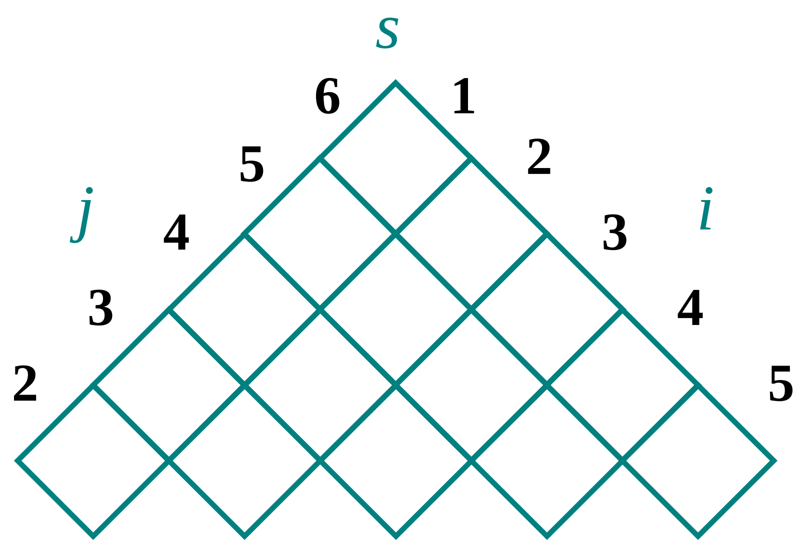


Overlapping subproblems

Computing the optimal costs

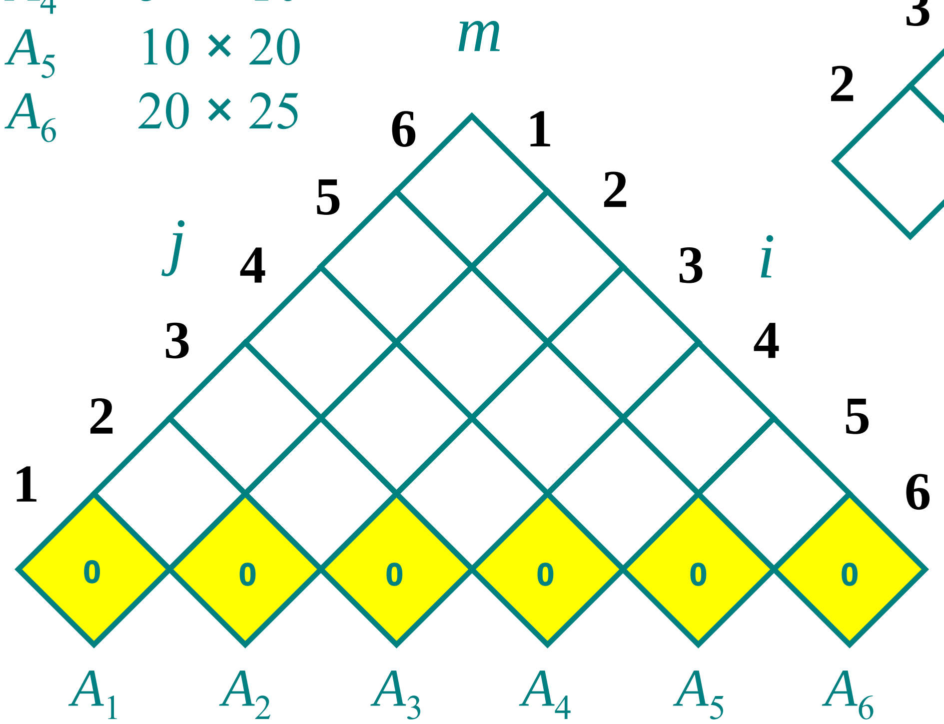
Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

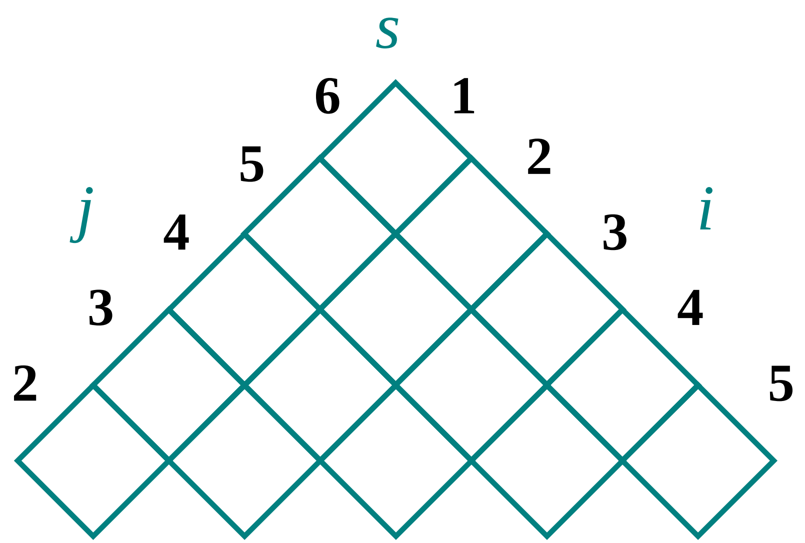




Computing the optimal costs

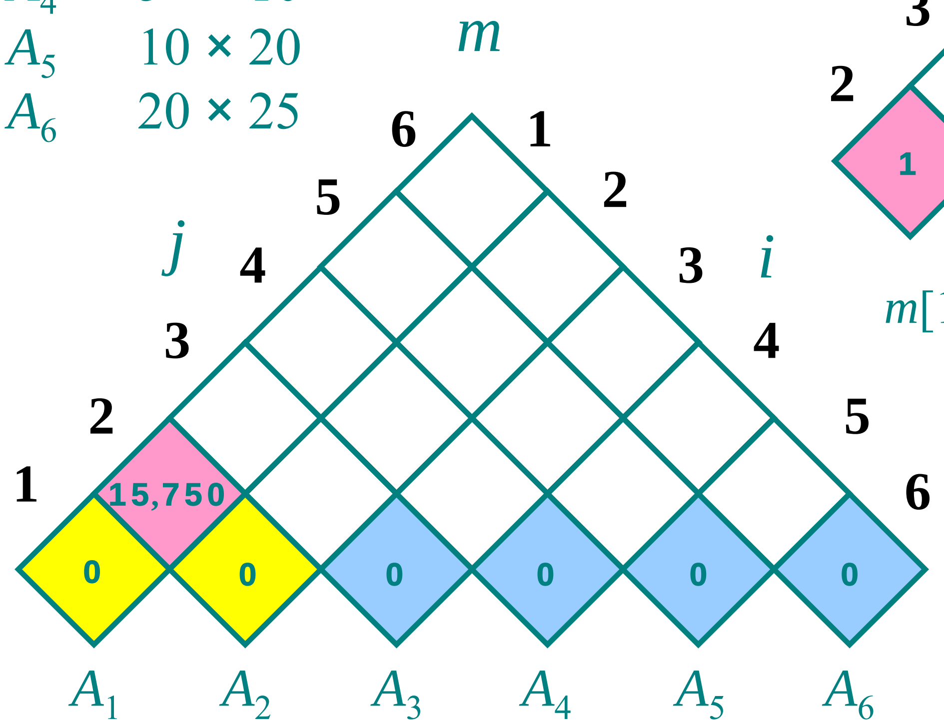
Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

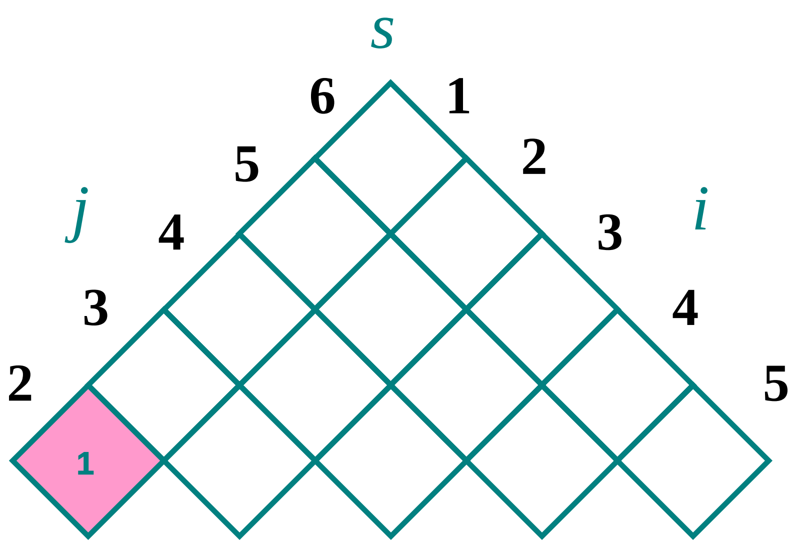




Computing the optimal costs

Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

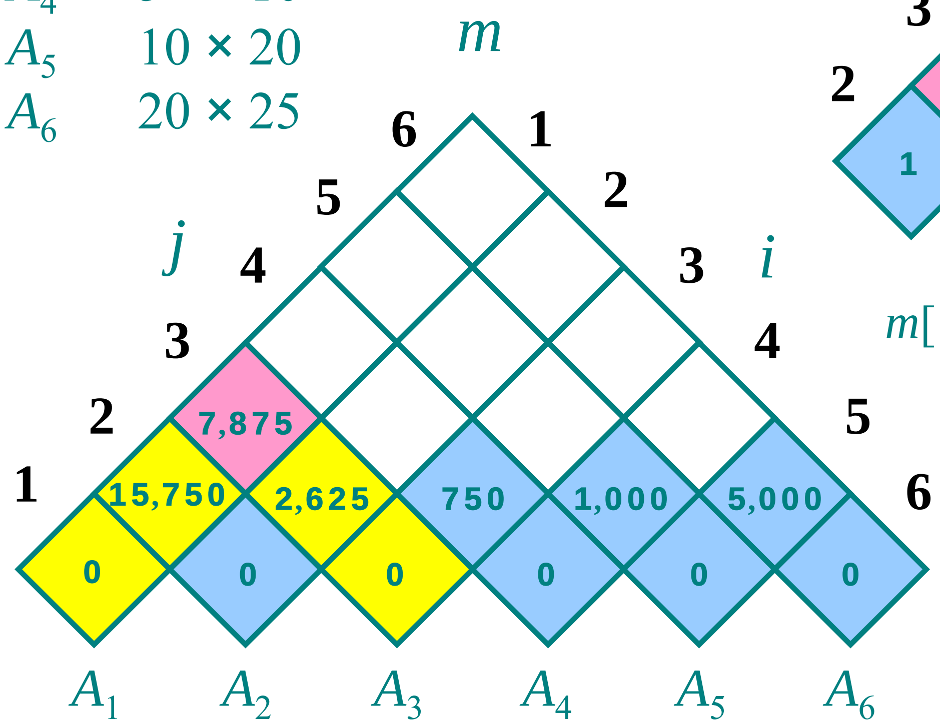


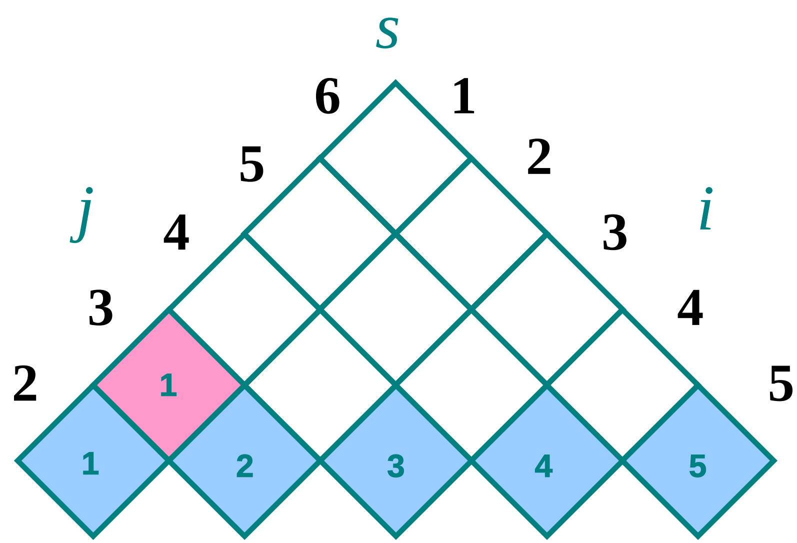


$$m[1,2] = m[1,1] + m[2,2] + p_0 \cdot p_1 \cdot p_2$$
$$= 0 + 0 + 30 \times 35 \times 15$$
$$= \mathbf{15{,}750}$$

Computing the optimal costs

Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25





$$m[1,3] = \min \begin{cases} m[1,1] + m[2,3] + p_0 \cdot p_1 \cdot p_3 \\ m[1,2] + m[3,3] + p_0 \cdot p_2 \cdot p_3 \end{cases}$$

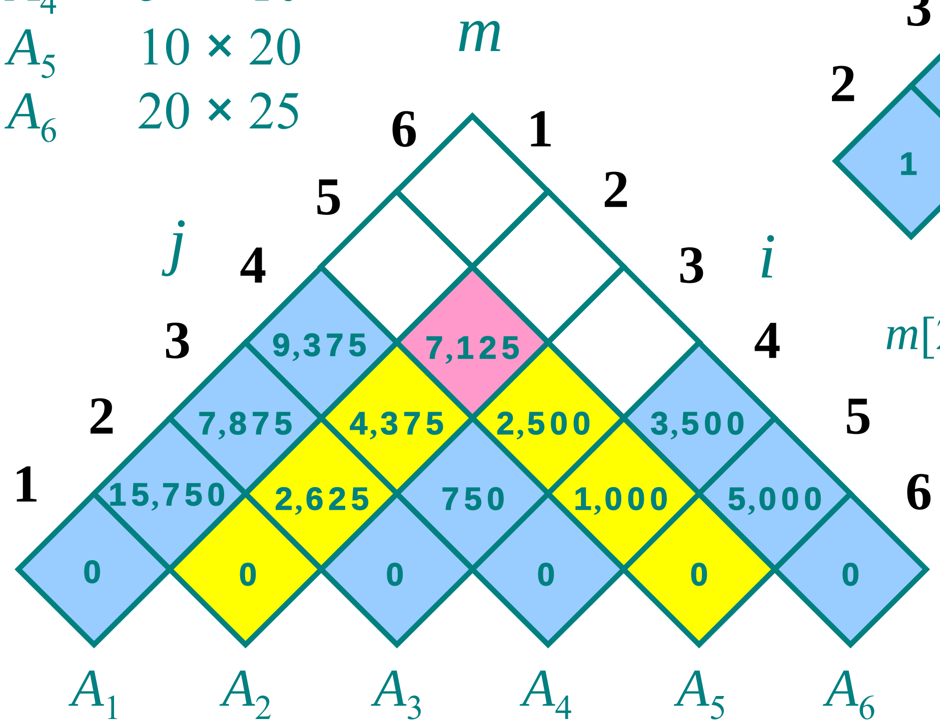
$$= \min \begin{cases} 0 + 2{,}625 + 30 \times 35 \times 5 \\ 15{,}750 + 0 + 30 \times 15 \times 5 \end{cases}$$

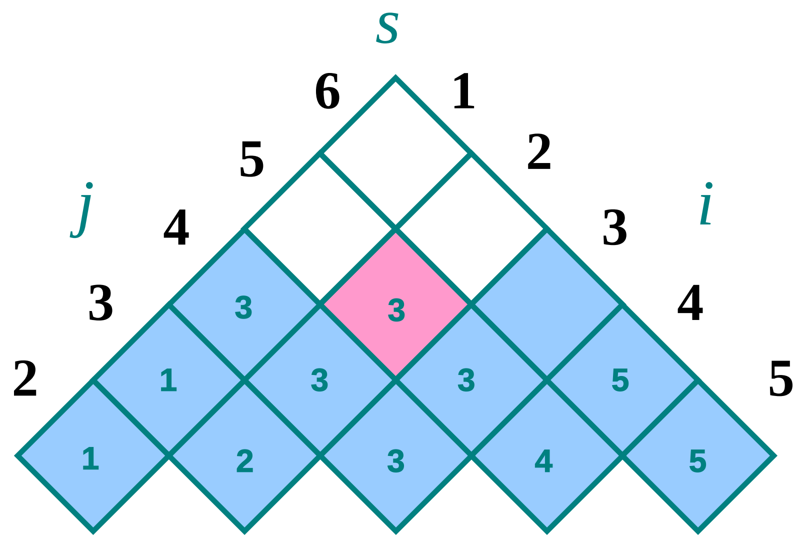
$$= \min \begin{cases} 7{,}875 \\ 18{,}000 \end{cases}$$

$$= \mathbf{7{,}875}$$

Computing the optimal costs

Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25





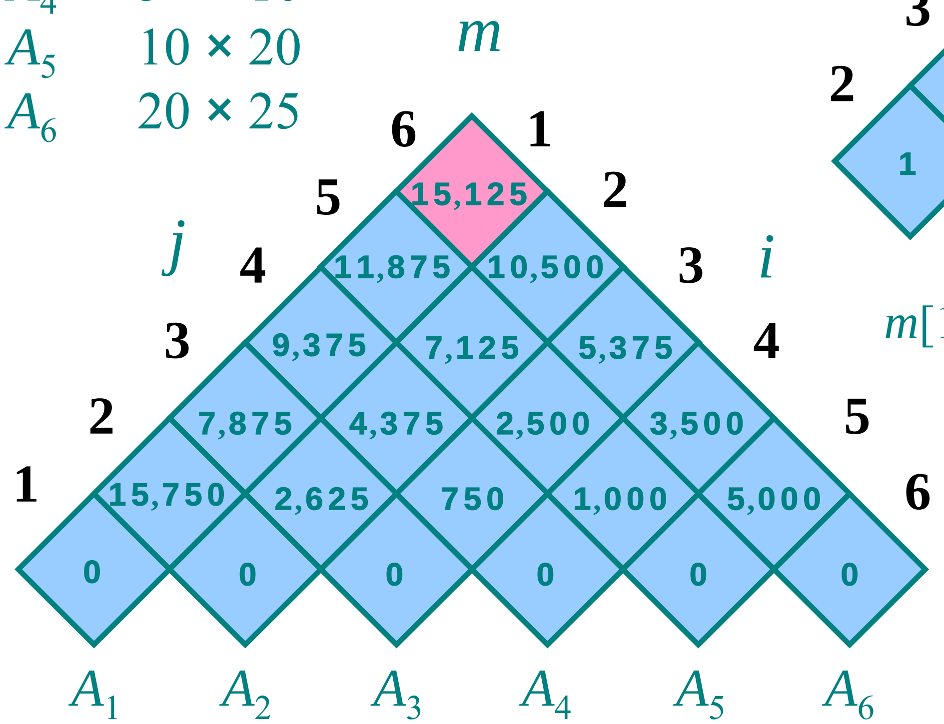
$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 \cdot p_2 \cdot p_5 \\ m[2,3] + m[4,5] + p_1 \cdot p_3 \cdot p_5 \\ m[2,4] + m[5,5] + p_1 \cdot p_4 \cdot p_5 \end{cases}$$

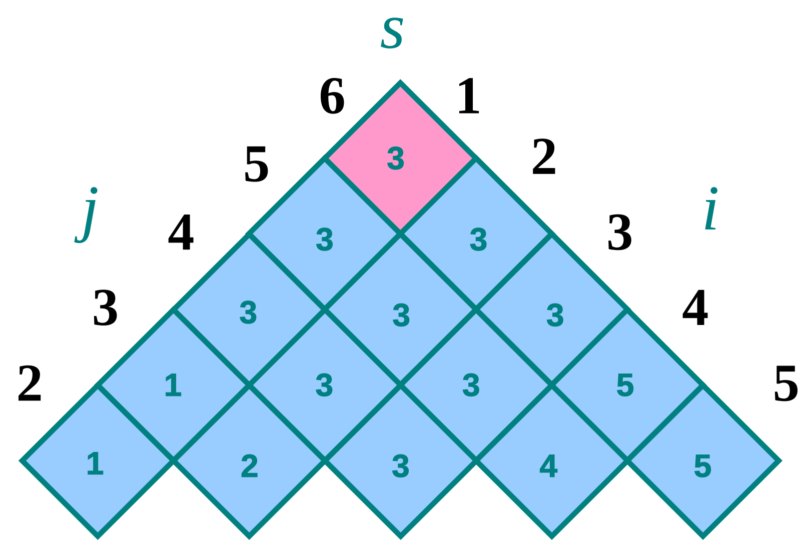
$$= \min \begin{cases} 0 + 2{,}500 + 30 \times 15 \times 20 \\ 2{,}625 + 1{,}000 + 30 \times 5 \times 20 \\ 4{,}375 + 0 + 35 \times 10 \times 20 \end{cases}$$

$$= \min \begin{cases} 13{,}000 \\ 7{,}125 \\ 11{,}375 \end{cases} = \mathbf{7{,}125}.$$

Computing the optimal costs

Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

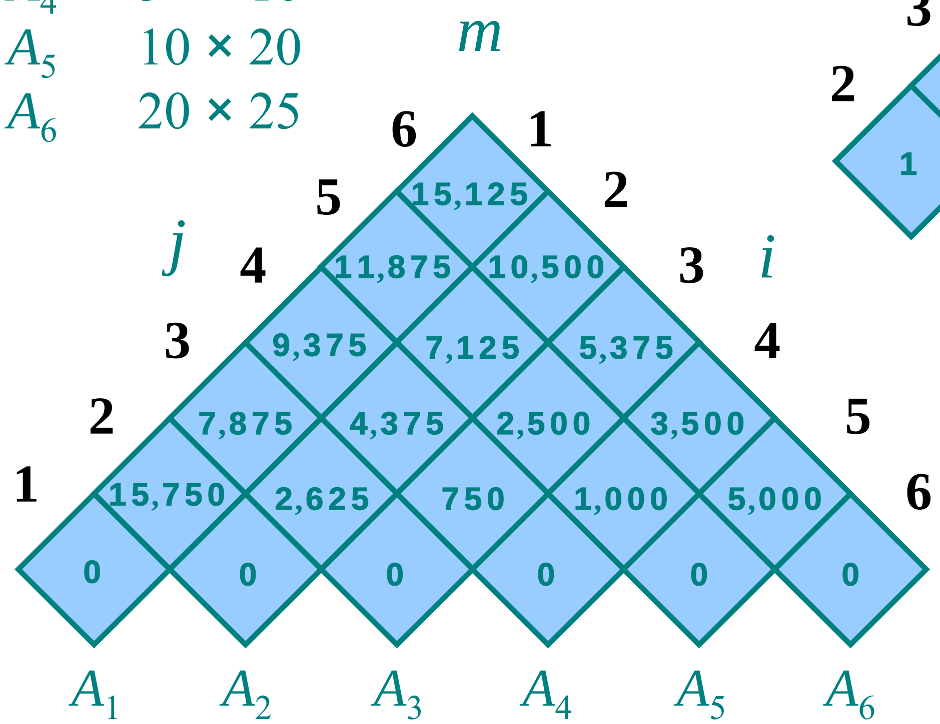


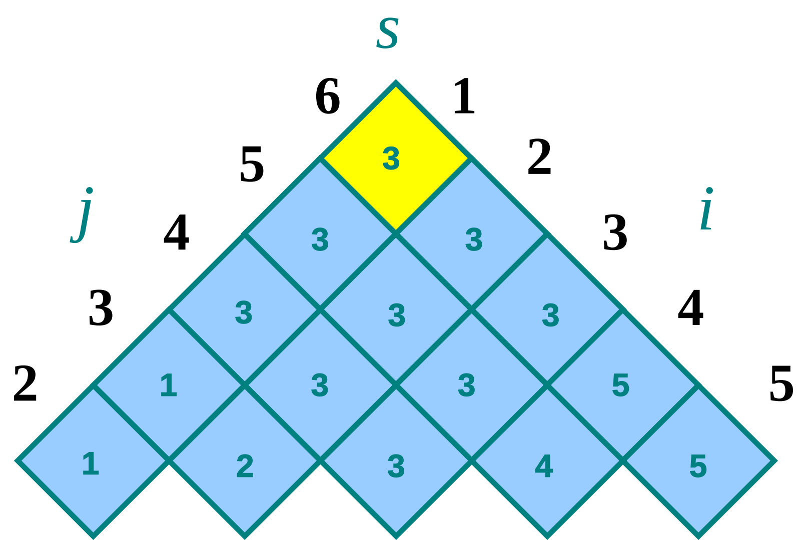


$m[1,6] =$ **15,125**

Constructing an optimal solution

Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25



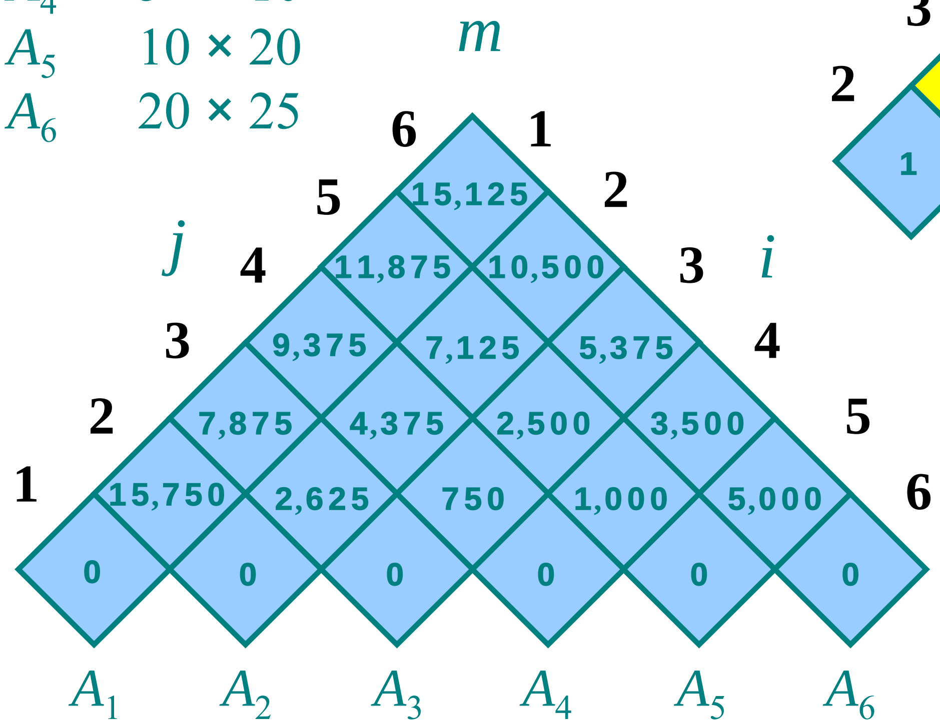


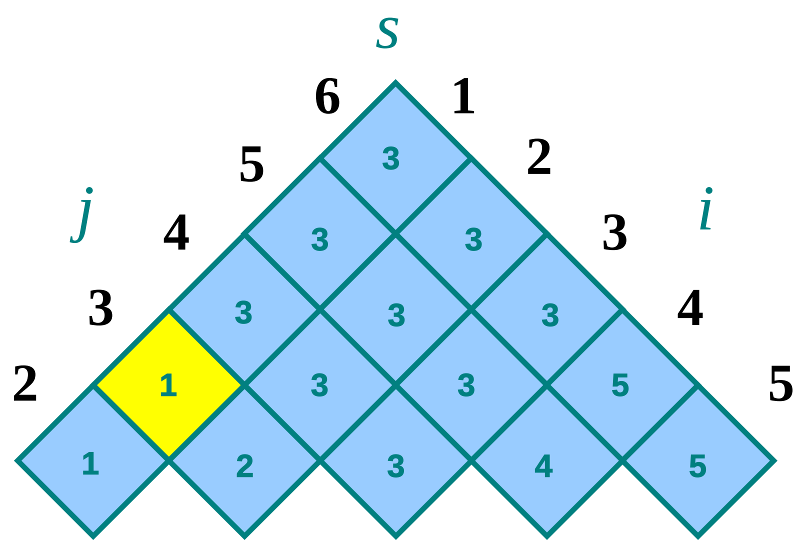
$m[1,6] = 15{,}125$

$(A_1 \ldots A_6) = ((A_1 \ldots A_3)(A_4 \ldots A_6))$

Constructing an optimal solution

Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25



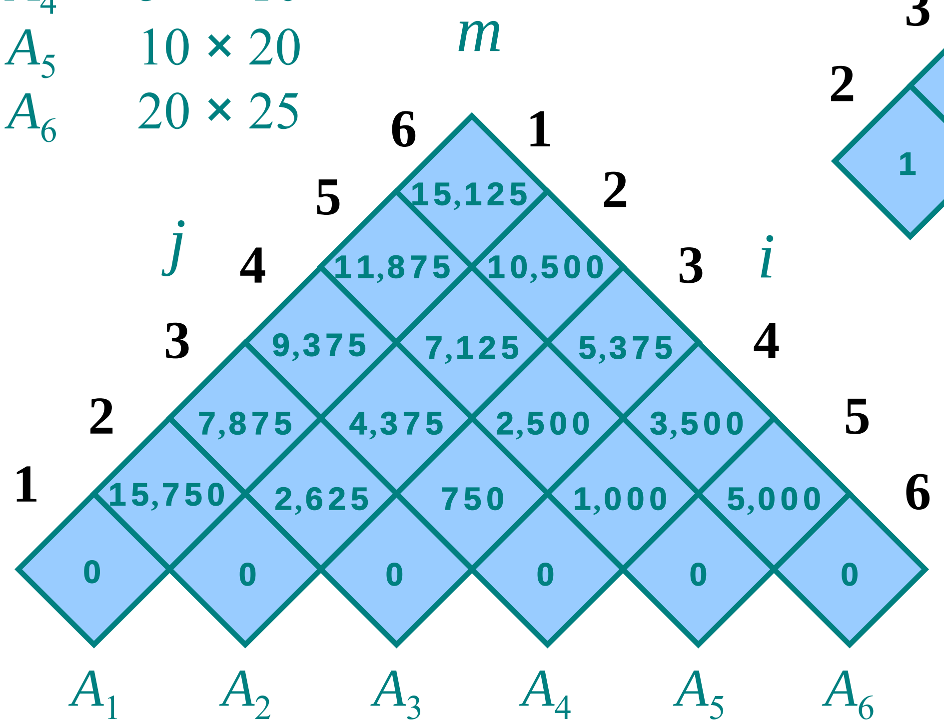


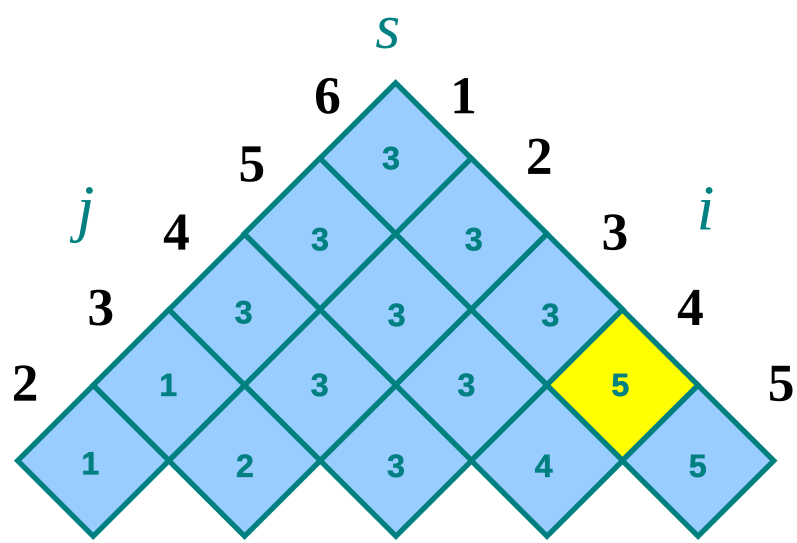
$m[1,6] = 15{,}125$

$$(A_1 \ldots A_6) = ((A_1 \ldots A_3)(A_4 \ldots A_6)) \\ = ((A_1(A_2\,A_3))(A_4 \ldots A_6))$$

Constructing an optimal solution

Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25





$m[1,6] = 15{,}125$

$$(A_1 \dots A_6) = ((A_1 \dots A_3)(A_4 \dots A_6))$$
$$= ((A_1(A_2 A_3))(A_4 \dots A_6))$$
$$= ((A_1(A_2 A_3))((A_4 A_5)A_6))$$

Elements of dynamic programming

Optimal substructure

- Dynamic programming builds an optimal solution to the problem from optimal solutions to subproblems.
- The solutions to the subproblems used within the optimal solution to the problem must themselves be optimal by using a "cut-and-paste" technique.
- Subproblems are ***independent***.

Overlapping subproblems

- Recursive algorithm revisits the same problem over and over again.
- In contrast, a problem for which a divide-and-conquer approach is suitable usually generates brand-new problems at each step of the recursion.

Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix $= 2 \times 2$ matrix of $(n/2) \times (n/2)$ submatrices:

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$\boldsymbol{C} \quad = \quad \boldsymbol{A} \quad \cdot \quad \boldsymbol{B}$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\}$$

recursive

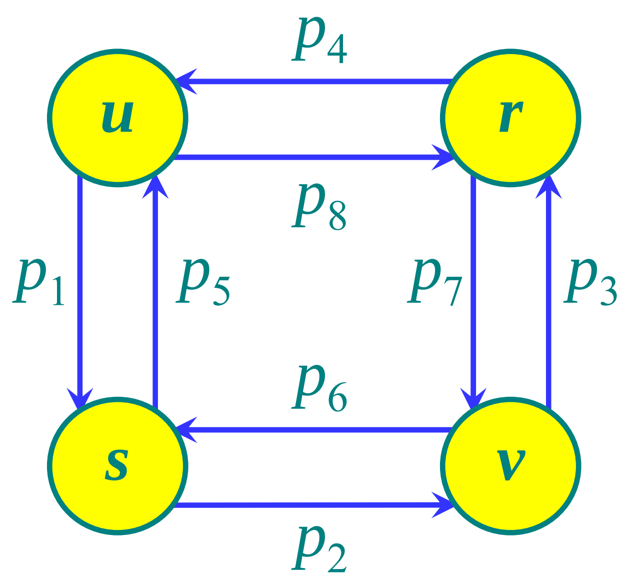
8 mults of $(n/2) \times (n/2)$ submatrices

4 adds of $(n/2) \times (n/2)$ submatrices

Subtleties

Given a directed graph $G = (V, E)$ and vertices $u, v \in V$.

- **Unweighted shortest path:** Find a path from u to v consisting the fewest edges.
- **Unweighted longest simple path:** Find a path from u to v consisting the most edges.



Subtleties

Given a directed graph $G = (V, E)$ and vertices $u, v \in V$.

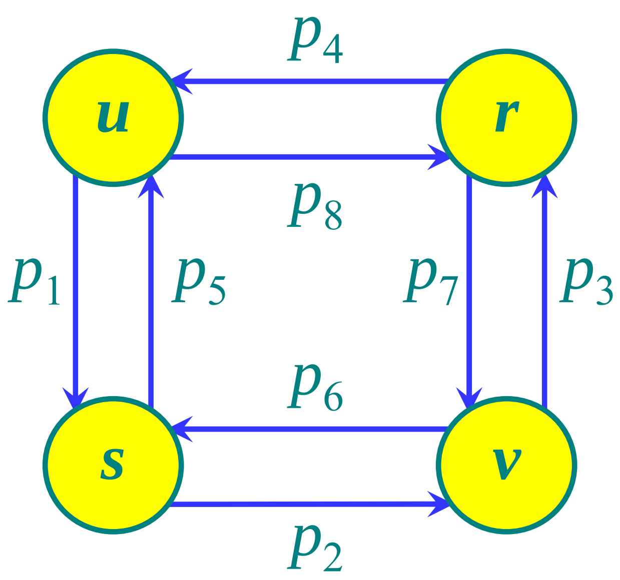
- **Unweighted shortest path:** Find a path from u to v consisting the fewest edges.
- **Unweighted longest simple path:** Find a path from u to v consisting the most edges.

Shortest path from u to v.

$$u \xrightarrow{p_1} s \xrightarrow{p_2} v.$$

For intermediate vertex s.

p_1 and p_2 must be shortest path.



Subtleties

Given a directed graph $G = (V, E)$ and vertices $u, v \in V$.

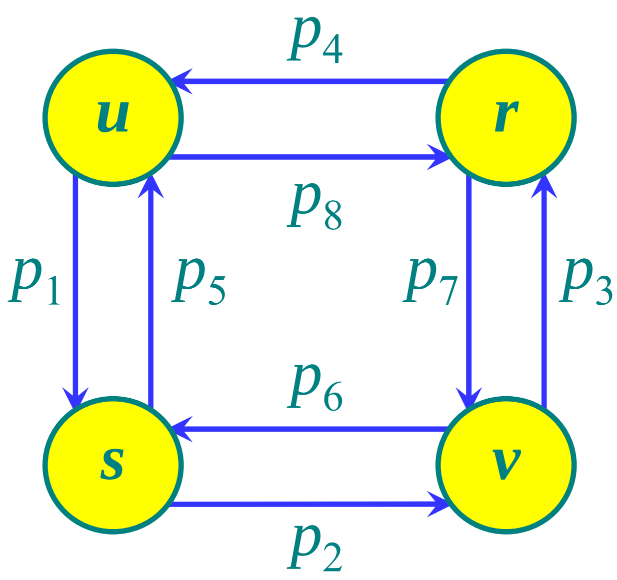
- **Unweighted shortest path:** Find a path from u to v consisting the fewest edges.
- **Unweighted longest simple path:** Find a path from u to v consisting the most edges.

Longest path from u to v.

$$u \xrightarrow{p_8} r \xrightarrow{p_7} v.$$

For intermediate vertex v.

Is p_8 longest simple path form u to r?



Subtleties

Given a directed graph $G = (V, E)$ and vertices $u, v \in V$.

- **Unweighted shortest path:** Find a path from u to v consisting the fewest edges.
- **Unweighted longest simple path:** Find a path from u to v consisting the most edges.

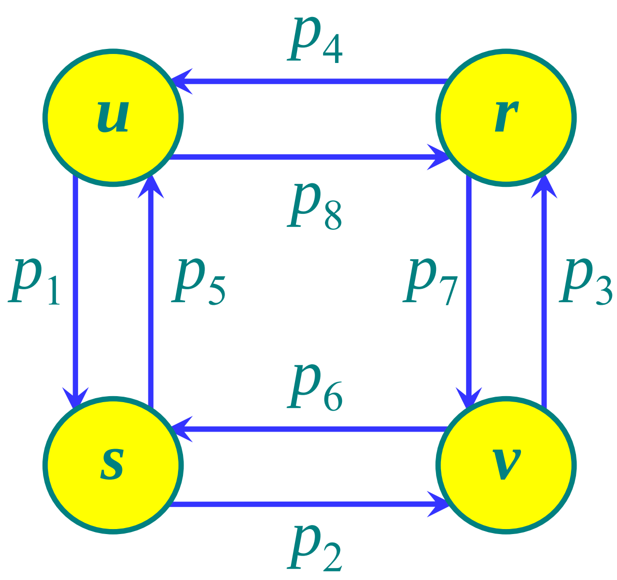
Longest path from u to v.

$$u \xrightarrow{p_8} r \xrightarrow{p_7} v.$$

For intermediate vertex v.

Is p_8 longest simple path form u to r ?

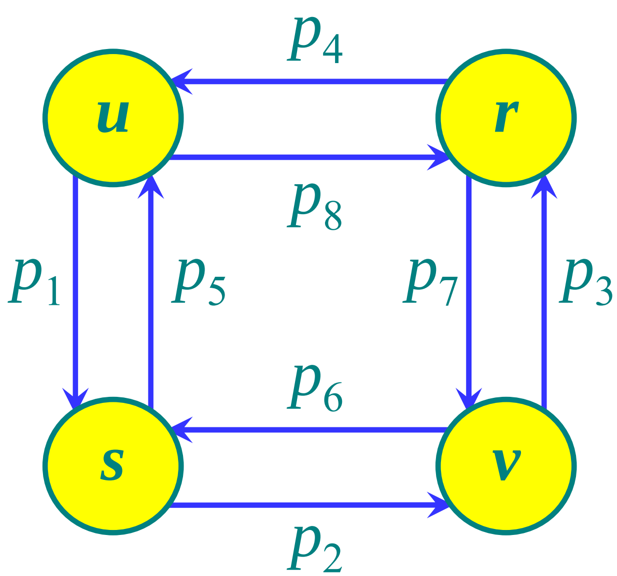
No. It is $u \xrightarrow{p_1} s \xrightarrow{p_2} v \xrightarrow{p_3} r$



Subtleties

Given a directed graph $G = (V, E)$ and vertices $u, v \in V$.

- **Unweighted shortest path:** Find a path from u to v consisting the fewest edges.
- **Unweighted longest simple path:** Find a path from u to v consisting the most edges.



Longest path from u to v.

$$u \xrightarrow{p_8} r \xrightarrow{p_7} v.$$

For intermediate vertex v.

Is p_7 longest simple path form r to v ?

Subtleties

Given a directed graph $G = (V, E)$ and vertices $u, v \in V$.

- **Unweighted shortest path:** Find a path from u to v consisting the fewest edges.
- **Unweighted longest simple path:** Find a path from u to v consisting the most edges.

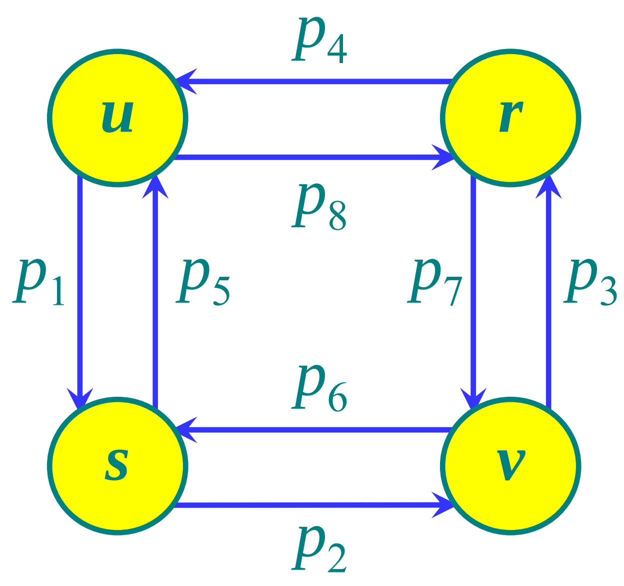
Longest path from u to v.

$$u \xrightarrow{p_8} r \xrightarrow{p_7} v.$$

For intermediate vertex v.

Is p_7 longest simple path form r to v ?

No. It is $r \xrightarrow{p_4} u \xrightarrow{p_1} s \xrightarrow{p_2} v$



Subtleties

Given a directed graph $G = (V, E)$ and vertices $u, v \in V$.

- **Unweighted shortest path:** Find a path from u to v consisting the fewest edges.
- **Unweighted longest simple path:** Find a path from u to v consisting the most edges.

Longest path from u to v.

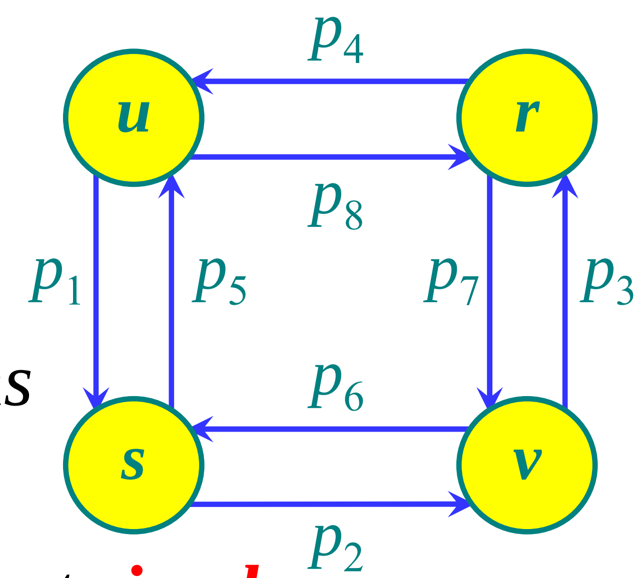
$$u \xrightarrow{p_8} r \xrightarrow{p_7} v.$$

For intermediate vertex v.

Combine the longest simple paths

$$u \xrightarrow{p_1} s \xrightarrow{p_2} v \xrightarrow{p_3} r \xrightarrow{p_4} u \xrightarrow{p_1} s \xrightarrow{p_2} v$$

*The path contains **cycles** and is not **simple**.*



Independent

- Subproblems in finding the longest simple path are not ***independent***, whereas for shortest paths they are.
- Subproblems being independent means that the solution to one subproblem does not affect the solution to another subproblem.
- For longest simple path problem, we choose the first path $u \to s \to v \to r$, and so we have also used the vertices s and t. We can no longer use these vertices in the second subproblem.
- Our use of ***resources*** in solving one subproblem has rendered them unavailable for the other subproblem.

Four steps of development

- Characterize the ***structure*** of an optimal solution.
- ***Recursively*** define the value of an optimal solution.
- Compute the value of an optimal solution in a ***bottom-up*** fashion.
- ***Construct*** an optimal solution from computed information.

Longest Common Subsequence

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

$$\left.\begin{array}{lccccccc} x: & A & B & C & B & D & A & B \\ \\ y: & B & D & C & A & B & A & \end{array}\right\} BCBA = LCS(x, y)$$

Brute-force LCS algorithm

Check every subsequence of $x[1 \ldots m]$ to see if it is also a subsequence of $y[1 \ldots n]$.

Analysis

- Checking = $O(n)$ time per subsequence.
- 2^m subsequences of x (each bit-vector of length m determines a distinct subsequence of x).
- Worst-case running time $= O(n2^m)$
  $=$ exponential time.

It is infeasible!

Optimal substructure of an LCS

Given a sequence $W = \langle w_1, w_2, \ldots, w_n \rangle$, define the ith ***prefix*** of W, for $i = 0, 1, \ldots, m$, as $W_i = \langle w_1, w_2, \ldots, w_i \rangle$

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

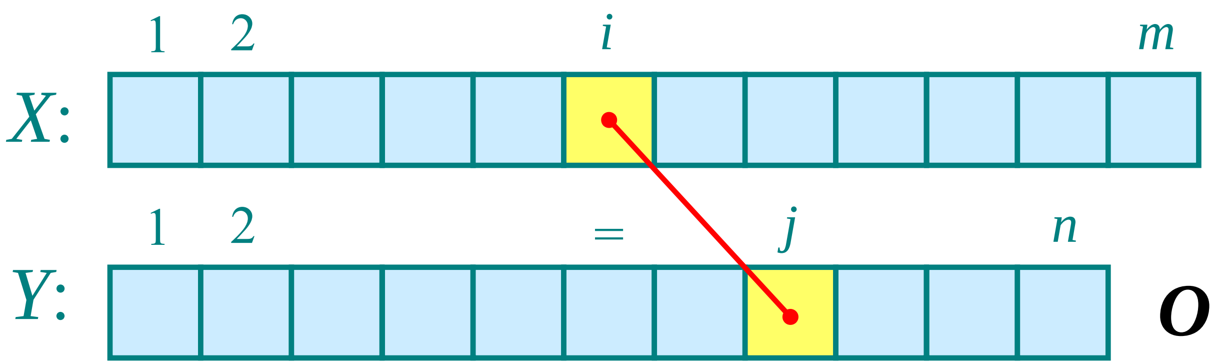
- If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
- If $x_m \neq y_n$, then $z_k \neq x_m$ and Z is an LCS of X_{m-1} and Y.
- If $x_m \neq y_n$, then $z_k \neq y_n$ and Z is an LCS of X and Y_{n-1}.

Recursive solution

Let us define $c[i, j]$ to be the length of an *LCS* of the sequences X_i and Y_j.

The optimal substructure of the *LCS* problem gives the recursive formula.

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



Our goal is $c[m, n]$

Computing LCS

c	j	0	1	2	3	4	5	6
i		y_j	***B***	***D***	***C***	***A***	***B***	***A***
0	x_i							
1	***A***							
2	***B***							
3	***C***							
4	***B***							
5	***D***							
6	***A***							
7	***B***							

Computing LCS

c	j	0	1	2	3	4	5	6
i		y_j	***B***	***D***	***C***	***A***	***B***	***A***
0	x_i	0	0	0	0	0	0	0
1	***A***	0						
2	***B***	0						
3	***C***	0						
4	***B***	0						
5	***D***	0						
6	***A***	0						
7	***B***	0						

Computing LCS

c	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

$x_1 \neq y_1$ and

$c[0, 1] \geq c[1, 0]$ then

$c[1, 1] = c[0, 1]$

Computing LCS

c	j	0	1	2	3	4	5	6
i		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		
2	***B***	0						
3	***C***	0						
4	***B***	0						
5	***D***	0						
6	***A***	0						
7	***B***	0						

$x_1 = y_4$ then

$c[1, 4] = c[0, 3] + 1$

Computing LCS

c / i	j	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	
2	***B***	0						
3	***C***	0						
4	***B***	0						
5	***D***	0						
6	***A***	0						
7	***B***	0						

$x_1 \neq y_5$ then

$c[0, 5] < c[1, 4]$ then

$c[1, 5] = c[1, 4]$

Computing LCS

c	j	0	1	2	3	4	5	6
i		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

$x_7 \neq y_6$ and

$c[6, 6] \geq c[7, 5]$ then

$c[7, 6] = c[6, 6]$

Constructing an LCS

c	j	0	1	2	3	4	5	6
i		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

$c[7, 6] = 4$ and

$LCS(X, Y) =$ ***BCBA***

Any question?

Xiaoqing Zheng
Fundan University