Part IV:

Advanced Design and Analysis Techniques

songyou@buaa.edu.cn

Advanced Design and Analysis Techniques

- Enumeration（穷举法）
- Search （搜索）
- Backtrack （回溯）
- Divide-and-conquer（分治）
- Randomization （随机化）
- ……

- Dynamic programming（动态规划）
  - Used for optimization problems （通常用于求解最优化问题）
  - Not a specific algorithm, but a technique （不是一个具体的算法，而是一种具有普适性的方法）
  - “programming” means “tabular method”, not computer programming
    programming 表示列表方法，而不是指编程
- Greedy algorithms （贪心算法、贪婪算法）
  - To make each choice in a locally optimal manner. (coin-changing)
    以局部最优方式作每一次选择

15 Dynamic Programming

Four steps:

1. Characterize the structure of an optimal solution. (最优解的结构)
2. Recursively define the value of an optimal solution (最优解的递归函数)
   (to find Transition Function). (转移函数，传递函数)
3. Computing (Algorithms): Compute the value of an optimal solution in a bottom-up fashion (动态规划算法：按自底向上的方式计算最优值)
   (to solve Transition Function).
4. [Construct an optimal solution from computed information.] (计算一个最优解)

15 Dynamic Programming

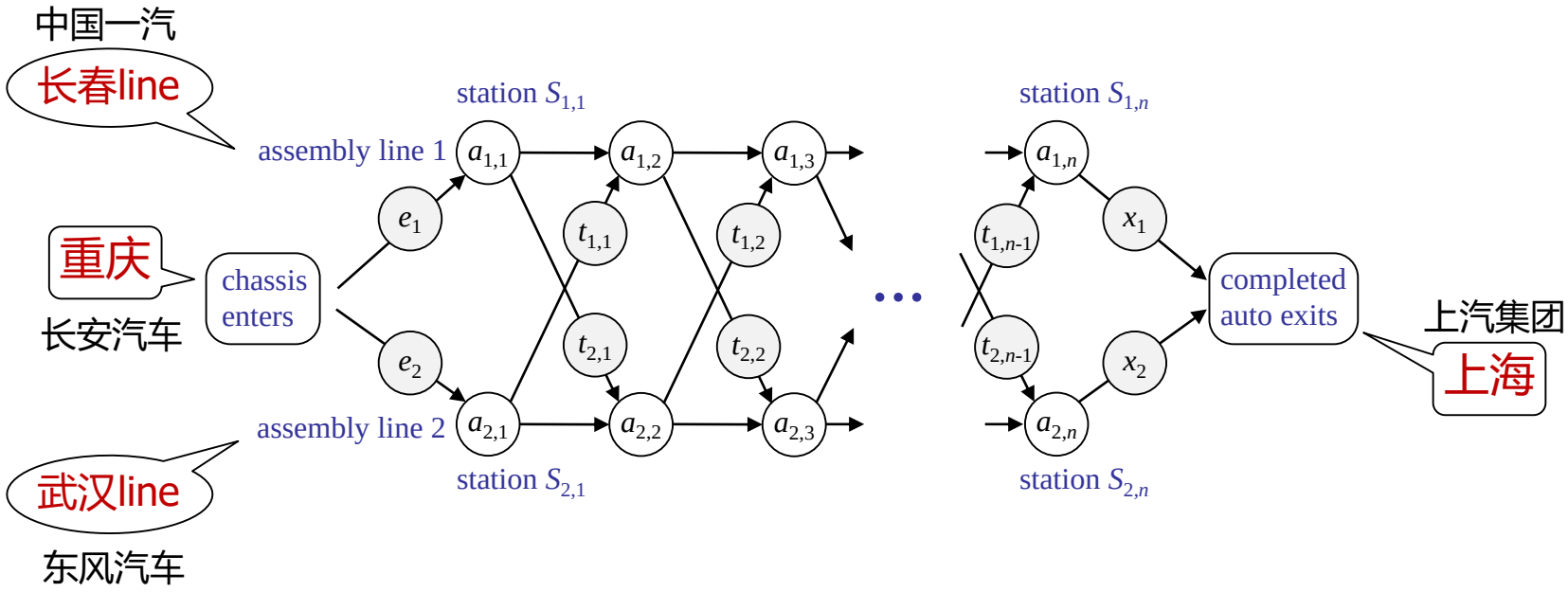
- Assembly lines scheduling (ALS)

  （流水线、装配线调度）
- Steel rod cutting（钢条、钢管切割）
- Matrix-chain multiplication（矩阵链相乘，矩阵连乘）
- Characteristics of dynamic programming

  （动态规划法的特征）
- Longest common subsequence （最长相同子序列）
- Optimal binary search trees （最优二叉搜索树）

15.0 Assembly-line scheduling

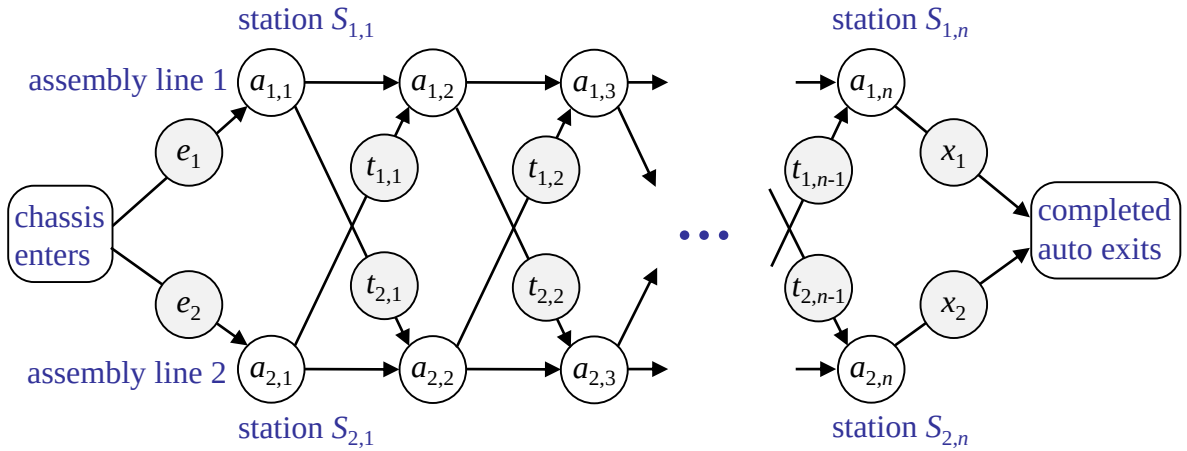
- Automobile factory with two assembly lines: A manufacturing problem to find the fastest way through a factory.（汽车制造有两条流水线，求汽车制造的流水线调度的最快方法）

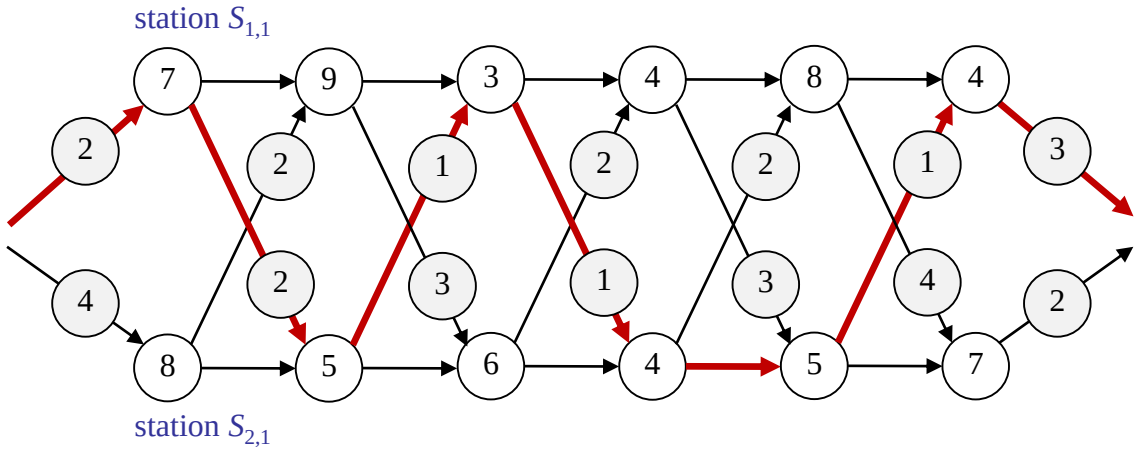


- Problem: determine which stations to choose from line 1 and which to choose from line 2 in order to minimize the total time through the factory for one auto.
（应选line1中哪一个装配站，line2中哪一个装配站，使得装配一个汽车的开销最小）

15.0 Assembly-line scheduling

- The fastest way through the factory



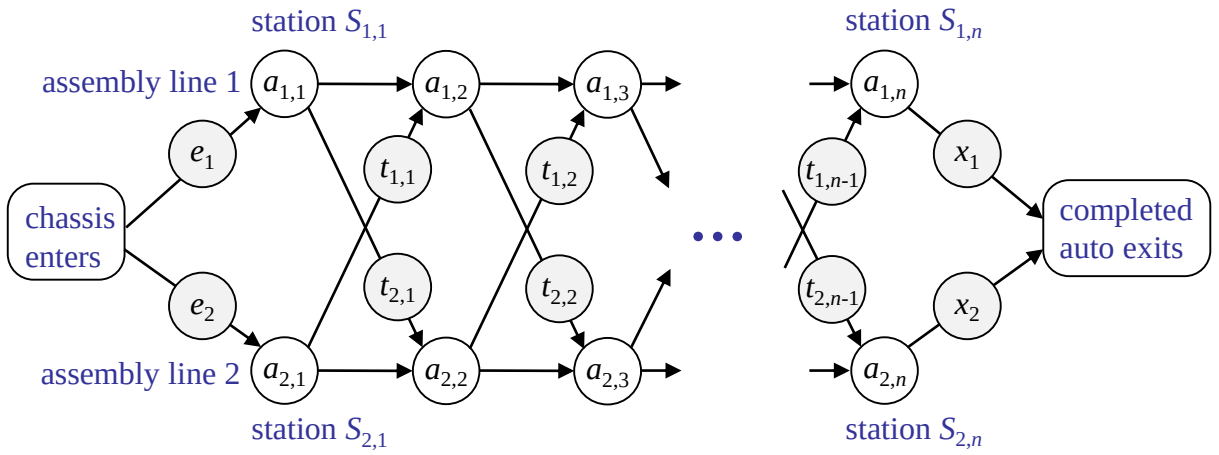


原问题 ⇒ 一种最快的流水线调度方案

- How to find the fastest way? 如何求最快的流水线调度方案?

15.0 Assembly-line scheduling

- 暴力穷举（枚举所有情况）
  “brute force” (enumerate all the way)



- 有 2^n 种可能的调度方案，计算时间 $\Omega(2^n)$, 当 n 较大时实际不可行 (infeasible)

- 如果已知 line1 和 line2 中分别有哪些 station 被使用，需要 $\Theta(n)$ 时间去计算该装配（调度）过程

applications in software engineering

2048（暑假小学期的2048游戏）：3个程序员，每个人就是一条line，分别按接口要求写函数实现相应功能，有40个函数（每个函数是一个station），测试每个函数的效率，选择合适的，集成为一个软件作品。最优的调度方案？brute force: 3^{40} 个可选项中挑选最优的。

```
void gotoxy(int x,int y)
void color(int a)
void clear(int x,int y)
void pp(int x,int y,int w)
void clearall()
void init()
void initb()
void getrand()
bool moveup()
bool moveleft()
bool movedown()
bool moveright()
...
bool check()
void printscreen()
void gameover()
bool checkwin()
void gamestart()
void Corner()
void Swing()
void Swirl()
void Random()
void AIplay()
```

line 1

```
void gotoxy2(int x,int y)
void color2(int a)
void clear2(int x,int y)
void pp2(int x,int y,int w)
void clearall2()
void init2()
void initb2()
void getrand2()
bool moveup2()
bool moveleft2()
bool movedown2()
bool moveright2()
...
bool check2()
void printscreen2()
void gameover2()
bool checkwin2()
void gamestart2()
void Corner2()
void Swing2()
void Swirl2()
void Random2()
void AIplay2()
```

line 2

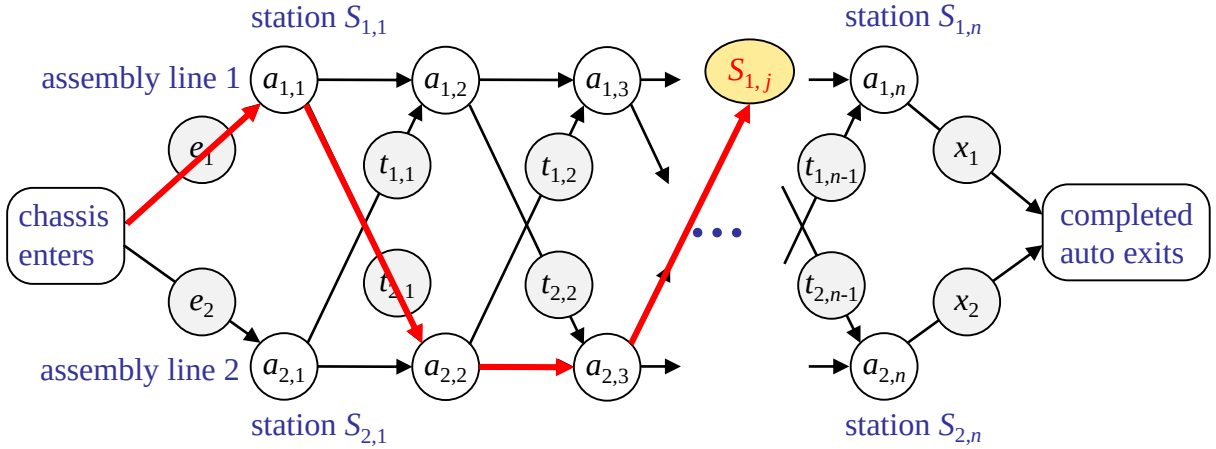
```
void gotoxy3(int x,int y)
void color3(int a)
void clear3(int x,int y)
void pp3(int x,int y,int w)
void clearall3()
void init3()
void initb3()
void getrand3()
bool moveup3()
bool moveleft3()
bool movedown3()
bool moveright3()
...
bool check3()
void printscreen3()
void gameover3()
bool checkwin3()
void gamestart3()
void Corner3()
void Swing3()
void Swirl3()
void Random3()
void AIplay3()
```

line 3

Step 1: The structure of the fastest way through the factory（最优调度方案的结构）

- Characterize the structure of an optimal solution.
  特征化一个最优解的结构

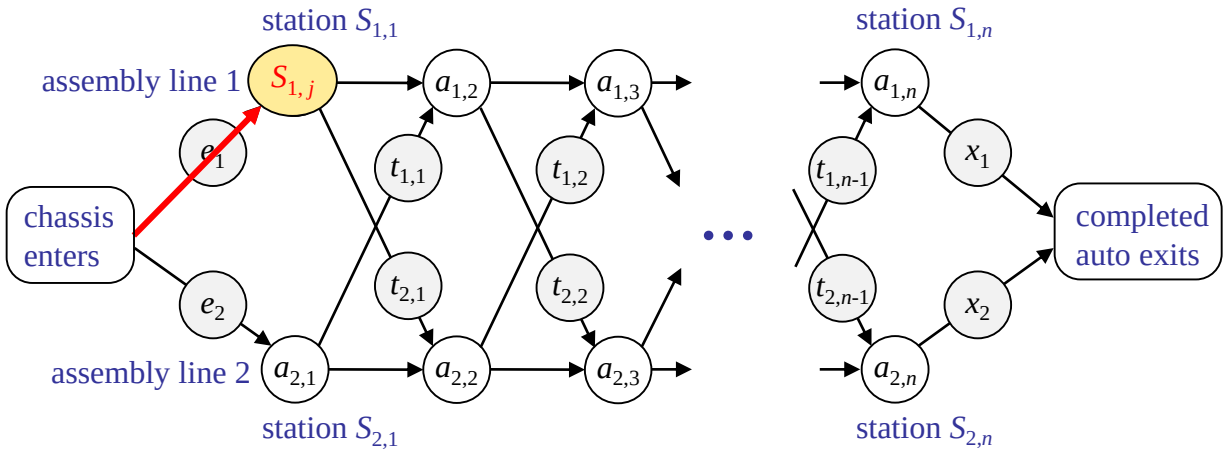
- Consider the fastest possible way for a chassis to get from the starting point through station $S_{1,j}$.
  子问题：通过 station $S_{1,j}$ 的最快的路径（最优调度方案）是什么？



Step 1: The structure of the fastest way through the factory（最优调度方案的结构）

- If $j = 1$, there is only one way that the chassis could have gone. It is easy to determine how long it takes to get through station $S_{1,j}$.

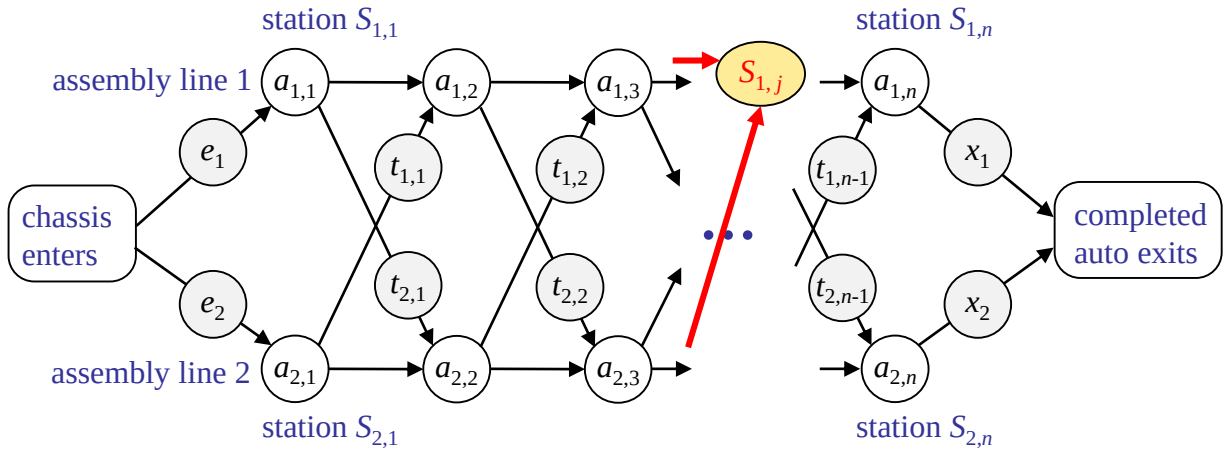
  $j = 1$ 时，只有一种方式通过$S_{1,1}$



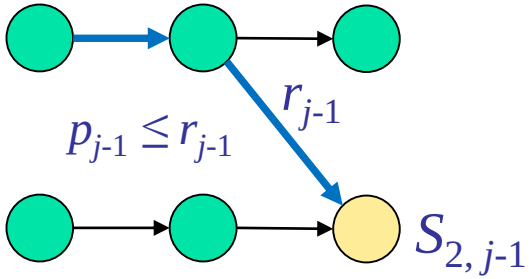
Step 1: The structure of the fastest way through the factory（最优调度方案的结构）

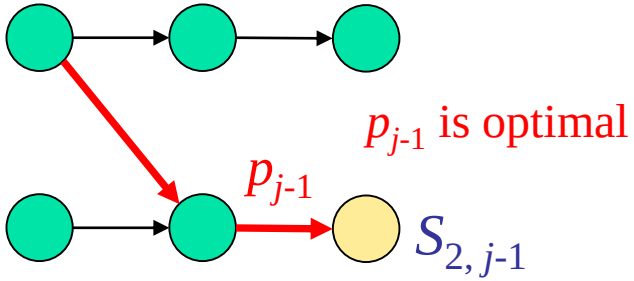
- For $j \geq 2$, two choices
  - through $S_{1,\,j\text{-}1}$, then directly to $S_{1,\,j}$, no transfer time;
  - through $S_{2,\,j\text{-}1}$, then transfer over to station $S_{1,\,j}$, transfer time $t_{2,\,j\text{-}1}$.
  - These two possibilities have much in common.

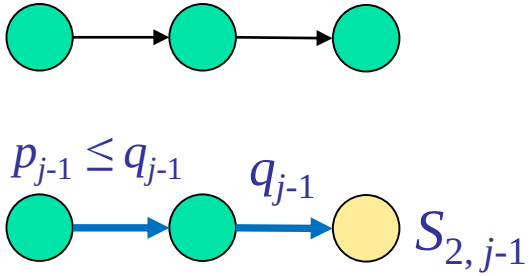
$j \geq 2$ 时，两种可能：同一个流水线上，直接到下一站，无切换时间；从一个流水线切换到另一个流水线的下一站，有切换时间。两种方案有很多共同点。

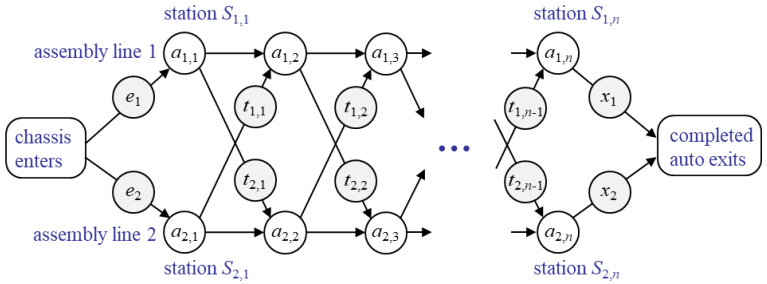


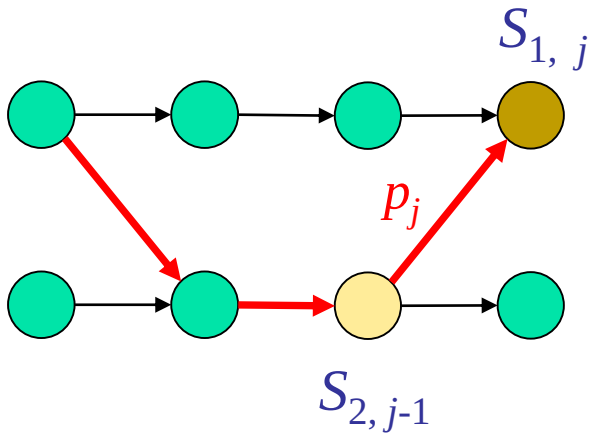
Step 1: The structure of the fastest way through the factory（最优调度方案的结构）









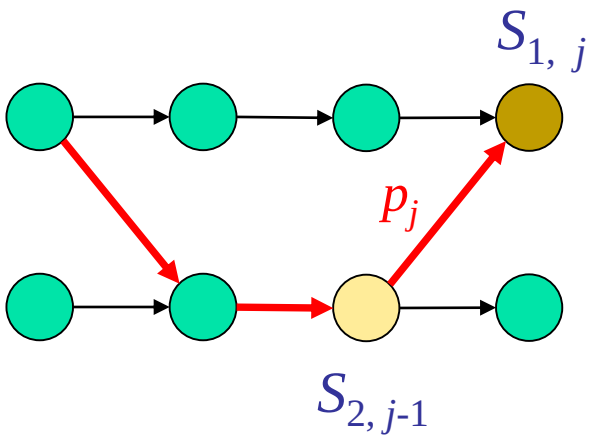


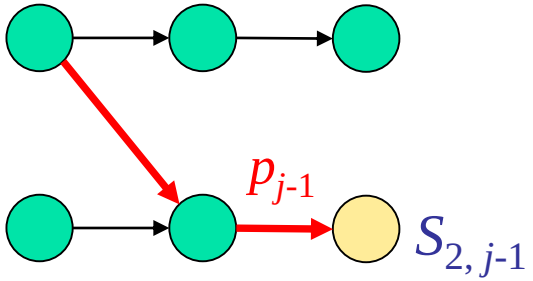
Optimal substructure（最优子结构）：

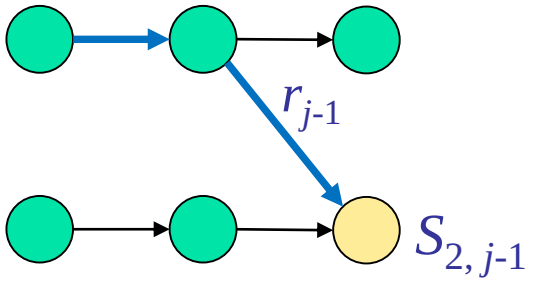
If p_j is optimal, $p_{j-1} \subset p_j$, then If p_{j-1} is optimal.

如果 p_j 是通过 $S_{1,\,j}$ 的最优路径，$p_{j-1} \subset p_j$，那么 p_{j-1} 是通过 $S_{2,\,j-1}$ 的最优路径.

Step 1: The structure of the fastest way through the factory（最优调度方案的结构）







Optimal substructure:

If p_j is optimal, $p_{j\text{-}1} \subset p_j$,

then If $p_{j\text{-}1}$ is optimal.

proof:

If $p_{j\text{-}1} > r_{j\text{-}1}$,

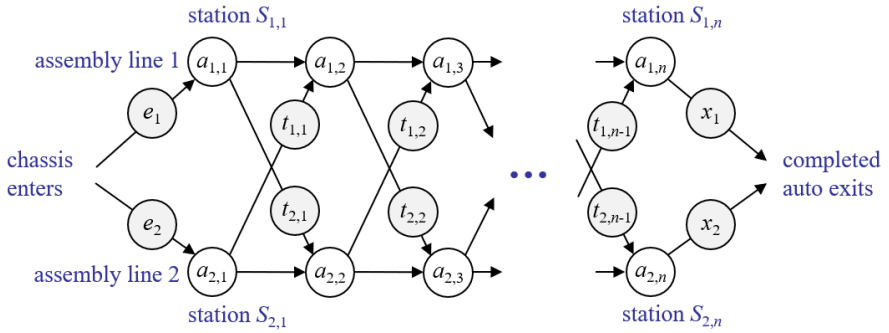
$p_j = p_{j\text{-}1} + t(S_{2,\ j\text{-}1}, S_{1,\ j})$

$> r_{j\text{-}1} + t(S_{2,\ j\text{-}1}, S_{1,\ j}) = p'_j$

contradiction to p_j is optimal.

Step 1: The structure of the fastest way through the factory（最优调度方案的结构）

Optimal substructure（最优子结构）

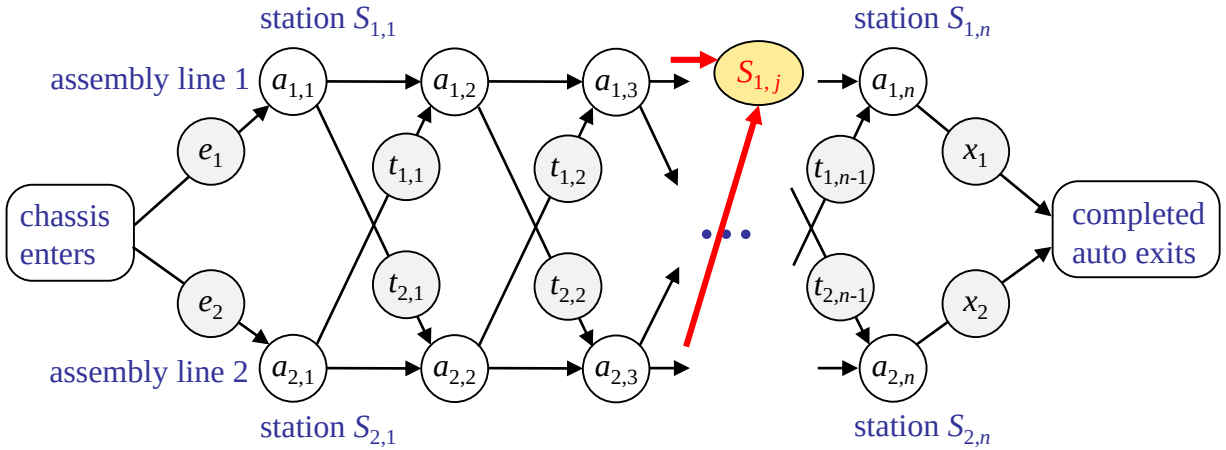


- An optimal solution to a problem (assembly-line scheduling, finding the fastest way through station $S_{i,j}$) contains within it an optimal solution to subproblems (finding the fastest way through either $S_{1,j-1}$ or $S_{2,j-1}$).
  问题的最优解包含其子问题的最优解，p_j contains p_{j-1}within it.

- Optimal substructure is one of the hallmarks of the applicability of dynamic programming.
  最佳子结构是动态规划法的重要特点之一。

Step 1: The structure of the fastest way through the factory（最优调度方案的结构）

- Use optimal substructure to construct an optimal solution to a problem from optimal solutions to subproblems.



使用最佳子结构可以从子问题的最优解来构造原问题的最优解

- First, fastest way through $S_{1,j}$ is either
  - fastest way through $S_{1,j-1}$ then directly through $S_{1,j}$, or
  - fastest way through $S_{2,j-1}$, transfer from line 2 to line 1, then through station $S_{1,j}$.

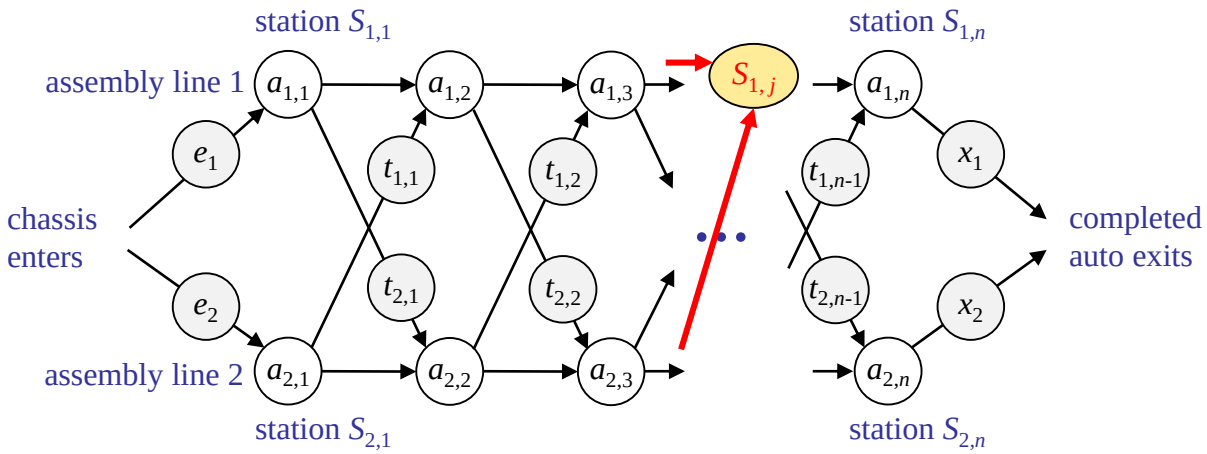
通过 $S_{1,j}$ 的最快方法有两种：一种是先通过 $S_{1,j-1}$ 然后直接通过 $S_{1,j}$；另一种先通过 $S_{2,j-1}$，然后从 line2 转移（切换）到 line1，再通过 $S_{1,j}$

- Symmetrically reasoning, through station $S_{2,j}$.

Step 1: The structure of the fastest way through the factory（最优调度方案的结构）

- Use optimal substructure to construct an optimal solution to a problem from optimal solutions to subproblems.

使用最佳子结构可以从子问题的最优解来构造原问题的最优解

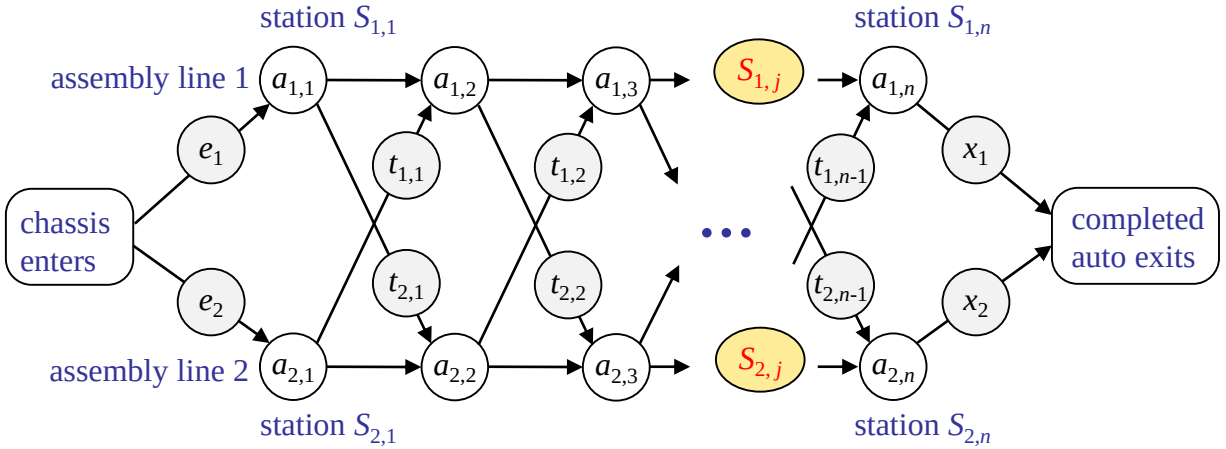


- Therefore, to solve problems of finding a fastest way through $S_{1,j}$ and $S_{2,j}$, solve subproblems of finding a fastest way through $S_{1,j-1}$ and $S_{2,j-1}$.

为了求解通过 $S_{1,j}$ 和 $S_{2,j}$ 的最快路径，可以通过先求解子问题 $S_{1,j-1}$ 和 $S_{2,j-1}$ 的最快路径

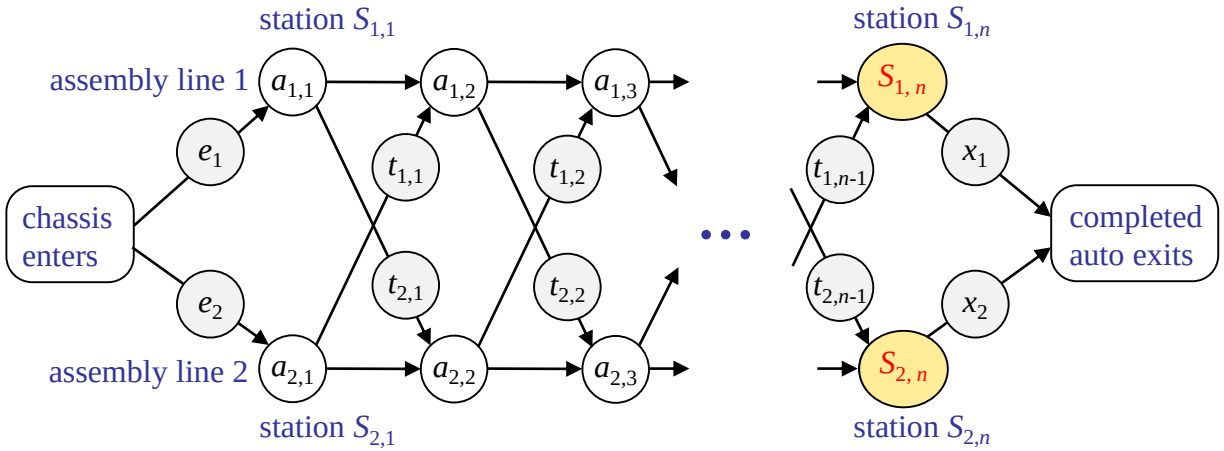
Step 2: A recursive solution（递归解）

- Define the value of an optimal solution recursively in terms of the optimal solutions to subproblems. 通过子问题的最优解来递归地定义原问题的最优解
- Subproblems: finding the fastest way through station j on both lines, for $j = 1, 2, ..., n$.
  子问题：寻找通过任意 station j (= 1, 2, ..., n)的最快路径
- Let $f_i[j]$ = fastest time to through $S_{i,j}$ from the starting point.
  从入口点开始，设以最快方式通过station $S_{i,j}$ 的时间为 $f_i[j]$



Step 2: A recursive solution（递归解）

- Let $f_i[j]$ = fastest time to through $S_{i,j}$ from the starting point.
（从入口点开始，设以最快方式通过station $S_{i,j}$ 的时间为 $f_i[j]$ ）



- Ultimate goal: fastest time to get a chassis all the way through the factory = f^*.

  The chassis has to get all the way through station n on either line 1 or line 2, and then to the factory exit.

  目标：以最快方式通过整个装配过程，设该时间为 f^* 。最后一步是在 line1 or line2 上通过station n，然后直接退出装配线

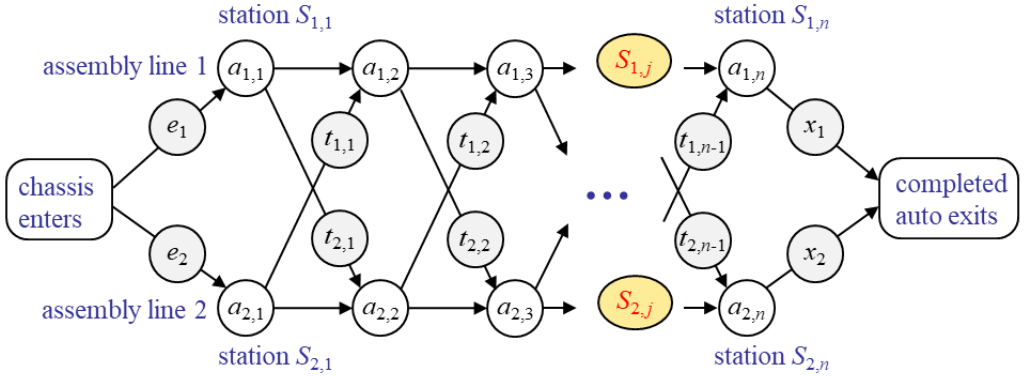
$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2) \qquad (15.1)$$

Step 2: A recursive solution（递归解）

- 从出发点可直接到 line1 或 line2 的 station1，直接计算 $f_1[1]$ 和 $f_2[1]$

$$f_1[1] = e_1 + a_{1,1} \qquad (15.2)$$

$$f_2[1] = e_2 + a_{2,1} \qquad (15.3)$$



- $j = 2, 3, \ldots, n$ $(i = 1, 2)$,
  - $f_1[j]$，通过 $S_{1,j}$ 的最快路线：

    最快路线通过 $S_{1,j-1}$，然后直接到 $S_{1,j}$，那么

    $f_1[j] = f_1[j-1] + a_{1,j}$；

    或，最快路线通过 $S_{2,j-1}$，然后切换到 $S_{1,j}$，那么

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

    因此

$$f_1[j] = \min(f_1[j-1] + a_{1,j},\ f_2[j-1] + t_{2,j-1} + a_{1,j}) \qquad (15.4)$$

  - 对称地，

$$f_2[j] = \min(f_2[j-1] + a_{2,j},\ f_1[j-1] + t_{1,j-1} + a_{2,j}) \qquad (15.5)$$

Step 2: A recursive solution（递归解）

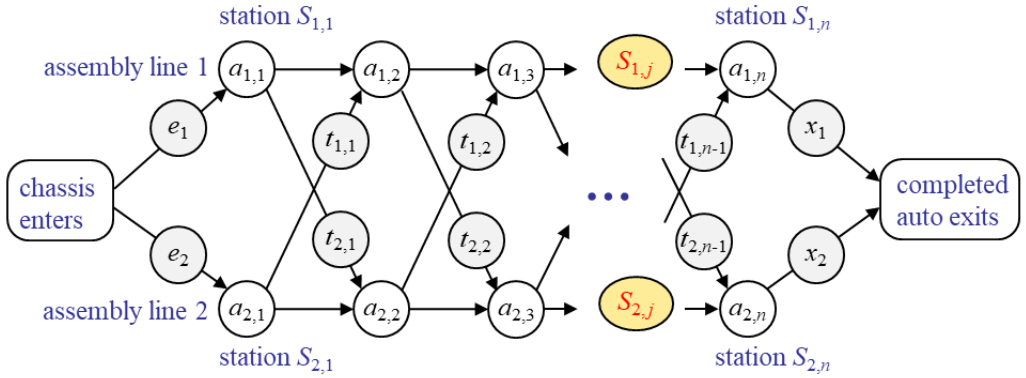
$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2) \qquad (15.1)$$

$$f_1[1] = e_1 + a_{1,1} \qquad (15.2)$$

$$f_2[1] = e_2 + a_{2,1} \qquad (15.3)$$

$$f_1[j] = \min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}) \qquad (15.4)$$

$$f_2[j] = \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}) \qquad (15.5)$$



通过方程 (15.2)–(15.5), 得到递归方程(recursive equations)

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & , \text{ if } j = 1 \\ \min(f_1[j-1] + a_{1,j} \ , \ f_2[j-1] + t_{2,j-1} + a_{1,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.6)$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & , \text{ if } j = 1 \\ \min(f_2[j-1] + a_{2,j} \ , \ f_1[j-1] + t_{1,j-1} + a_{2,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.7)$$

Transition Function（转移函数，传递函数）

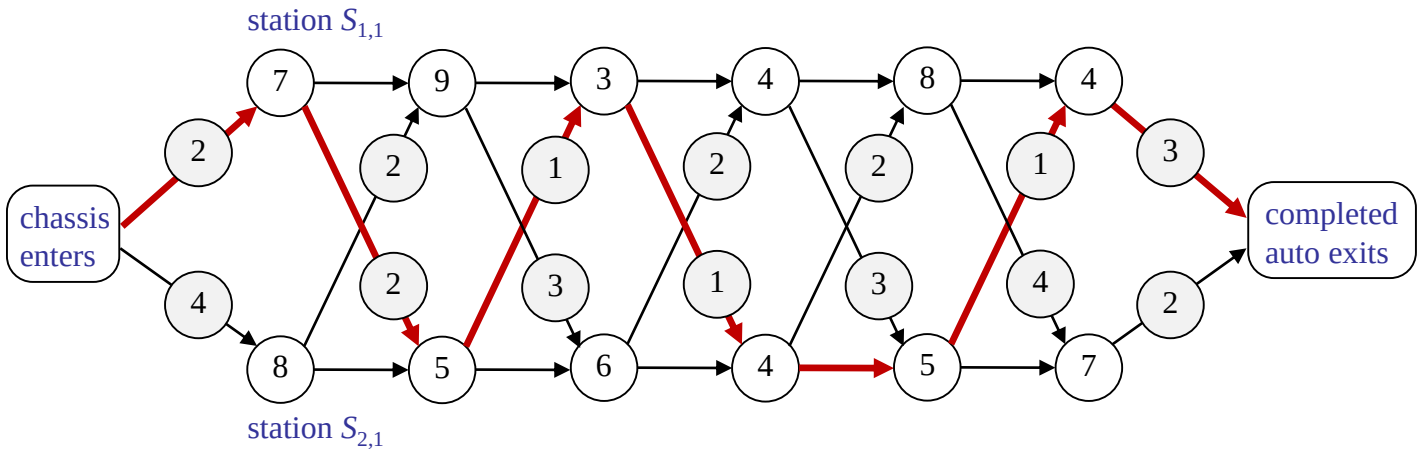
Step 2: A recursive solution（递归解）

- By equations (15.2)–(15.5), obtain the recursive equations

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & , \text{ if } j = 1 \\ \min(f_1[j-1] + a_{1,j} \ , \ f_2[j-1] + t_{2,j-1} + a_{1,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.6)$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & , \text{ if } j = 1 \\ \min(f_2[j-1] + a_{2,j} \ , \ f_1[j-1] + t_{1,j-1} + a_{2,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.7)$$

- 求出 $f_i[j]$值的一个实例



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

$f^* = 38$

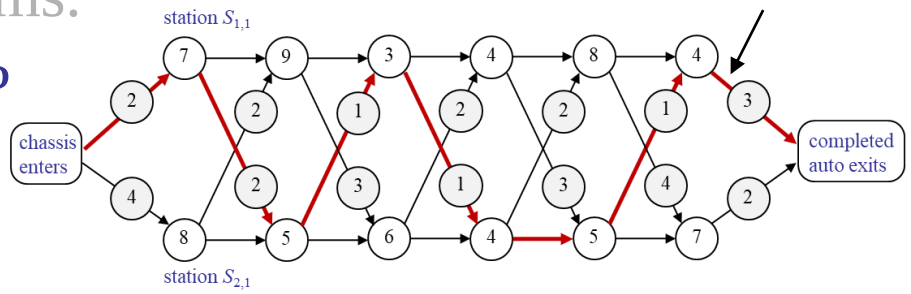
最优值 f^* 已经求出，如何求最优解（红色的路线）？

optimal value（最优值） vs optimal solutions（最优解）

- $f_i[j]$: the values of optimal solutions to subproblems.
- What if we want to construct an optimal solution?
  - Define $l_i[j]$ = line # (1 or 2), whose station j-1 is used in a fastest way through $S_{i,j}$.
  - Here $i = 1, 2$ and $2 \le j \le n$.

    (avoid defining $l_i[1]$ because no station precedes station 1 on either line.)
  - l^* = line # whose station n is used.

one of optimal solutions



optimal value

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

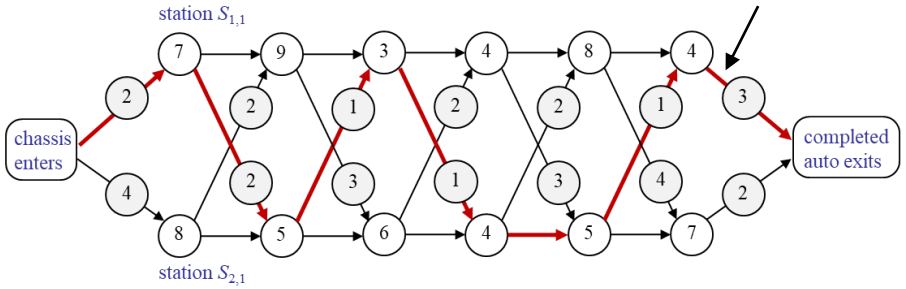
$l^* = 1$

以最快路径 p_j 通过 $S_{i,j}$ 时，令 $l_i[j]$ 表示 p_j 中 $S_{i,j}$ 的前一站(即station j-1) 的行号，其中 $2 \le j$。 注意：$j \ne 1$，因为，如果 $j=1$，则 $l_i[1]$ 表示通过 station 0 的行号，错（没有station 0）。 l^* 表示以最快方式通过 station n 的行号。

Step 2: A recursive solution（递归解）

- Define $l_i[j]$ = line # (1 or 2), whose station j-1 is used in a fastest way through $S_{i,j}$.
- l^* = line # whose station n is used.
- l^* and $l_i[j]$ can help us trace a fastest way through the factory.
  利用 l^* 和 $l_i[j]$ 可以追踪通过所有装配线的最快方法
- l^* = 1 (use station $S_{1,6}$);
  $l_1[6]$ = 2 (through $S_{2,5}$);
  $l_2[5]$ = 2 ($S_{2,4}$);
  $l_2[4]$ = 1 ($S_{1,3}$);
  $l_1[3]$ = 2 ($S_{2,2}$);
  $l_2[2]$ = 1 ($S_{1,1}$).

one of optimal solutions



optimal value

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

$l^* = 1$

Step 3: Computing the fastest times （算法设计）

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2) \qquad (15.1)$$

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & , \text{ if } j = 1 \\ \min(f_1[j-1] + a_{1,j} \ , \ f_2[j-1] + t_{2,j-1} + a_{1,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.6)$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & , \text{ if } j = 1 \\ \min(f_2[j-1] + a_{2,j} \ , \ f_1[j-1] + t_{1,j-1} + a_{2,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.7)$$

- It is simple to write a recursive algorithm based on equation (15.1), (15.6) and (15.7), to compute the fastest way through the factory.

  根据递归方程，很容易写出递归算法

- But the algorithm's running time is exponential in n. (2^n)

  递归算法的时间复杂度为 n 的指数次方

Step 3: Computing the fastest times （算法设计）

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2) \qquad (15.1)$$

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & , \text{ if } j = 1 \\ \min(f_1[j-1] + a_{1,j} \ , \ f_2[j-1] + t_{2,j-1} + a_{1,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.6)$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & , \text{ if } j = 1 \\ \min(f_2[j-1] + a_{2,j} \ , \ f_1[j-1] + t_{1,j-1} + a_{2,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.7)$$

- Let $r_i(j)$ = # of references made to $f_i[j]$ in a recursive algorithm.

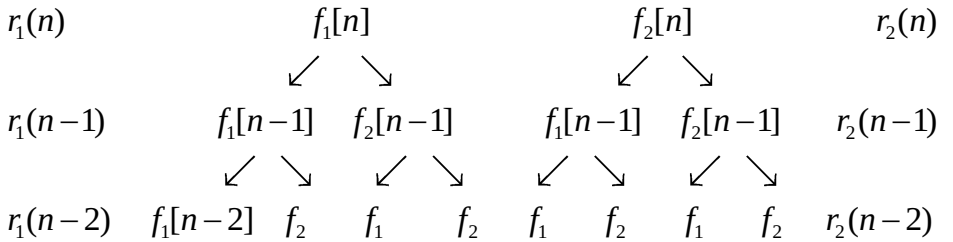
  $r_i(j)$ 表示递归算法中 $f_i(j)$ 被调用的次数

  From equation (15.1),

  $r_1(n) = r_2(n) = 1$

  From (15.6) and (15.7),

  $r_1(j) = r_2(j) = r_1(j+1) + r_2(j+1)$



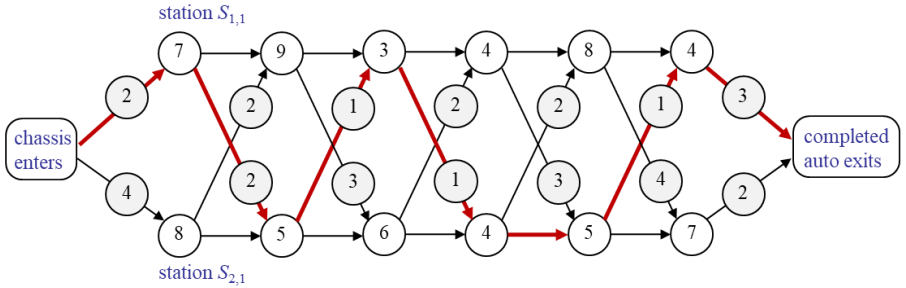
- For $j = 1, 2, \ldots, n\text{-}1$, $r_i(j) = 2^{n-j}$, $f_1[1]$ alone is referenced 2^{n-1} times! the total number of references to all $f_i[j]$ values is $\Theta(2^n)$.

  仅 $f_1[1]$ 就将被调用 2^{n-1} 次！ 所有 $f_i[j]$ 的被引用的次数为 $\Theta(2^n)$

Step 3: Computing the fastest times （算法设计）

- Compute the $f_i[j]$ values in a different order from the recursion. 采用与递归不一样的顺序来计算 $f_i[j]$

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & , \text{ if } j = 1 \\ \min(f_1[j-1] + a_{1,j} \ , \ f_2[j-1] + t_{2,j-1} + a_{1,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.6)$$



- Observation: $f_i[j]$ depends only on $f_1[j\text{-}1]$ and $f_2[j\text{-}1]$ (for $j \geq 2$). computing the $f_i[j]$ values in order of increasing station numbers j —— from left to right. This is **Dynamic Programming**.
以station数 j 增加的方式来计算 $f_i[j]$, in $\Theta(n)$ time.
填表（打表）即规划过程，动态体现在填表过程中要进行选择！

optimal value

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

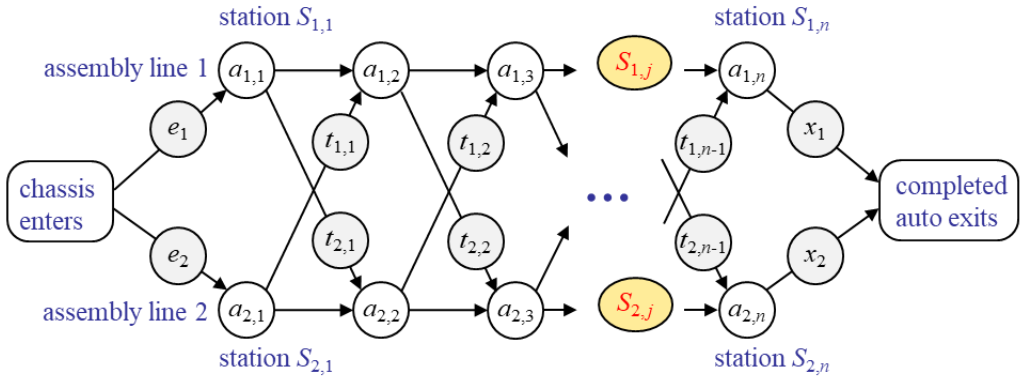
$f^* = 38$

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

$l^* = 1$

- 很多打表法不能称为动态规划算法，比如，求fibonacci数，$f(n\text{-}2) + f(n\text{-}1) \rightarrow f(n)$，这是打表算法，但其中没有选择，因此 $f(n\text{-}2) + f(n\text{-}1) \rightarrow f(n)$ 不是动态规划。

Step 3: Computing the fastest times （算法设计）



$$f^* = \min(f_1[n] + x_1,\ f_2[n] + x_2) \qquad (15.1)$$

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & , \text{ if } j = 1 \\ \min(f_1[j-1] + a_{1,j}\ ,\ f_2[j-1] + t_{2,j-1} + a_{1,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.6)$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & , \text{ if } j = 1 \\ \min(f_2[j-1] + a_{2,j}\ ,\ f_1[j-1] + t_{1,j-1} + a_{2,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.7)$$

算法：FASTEST-WAY

输入：$a_{i,j}, t_{i,j}, e_i, x_i, n$

输出：$f_{i,j}, l_{i,j}, f^*, l^*$

optimal value

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

$l^* = 1$

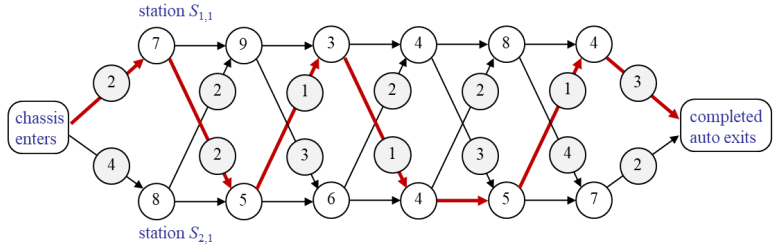
Step 3: Computing the fastest times （算法设计）

```
FASTEST-WAY(a, t, e, x, n)
1   f1[1] ← e1 + a1,1
2   f2[1] ← e2 + a2,1
3   for j ← 2 to n
4       if f1[j-1] + a1,j ≤ f2[j-1] + t2,j-1 + a1,j
5           f1[j] ← f1[j-1] + a1,j
6           l1[j] ← 1
7       else f1[j] ← f2[j-1] + t2,j-1 + a1,j
8           l1[j] ← 2
9       if f2[j-1] + a2,j ≤ f1[j-1] + t1,j-1 + a2,j
10          f2[j] ← f2[j-1] + a2,j
11          l2[j] ← 2
12      else f2[j] ← f1[j-1] + t1,j-1 + a2,j
13          l2[j] ← 1
14  if f1[n] + x1 ≤ f2[n] + x2
15      f* = f1[n] + x1
16      l* = 1
17  else f* = f2[n] + x2
18      l* = 2
```

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & , \text{ if } j = 1 \\ \min(f_1[j-1] + a_{1,j} \ , \ f_2[j-1] + t_{2,j-1} + a_{1,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.6)$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & , \text{ if } j = 1 \\ \min(f_2[j-1] + a_{2,j} \ , \ f_1[j-1] + t_{1,j-1} + a_{2,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.7)$$

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2) \qquad (15.1)$$



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

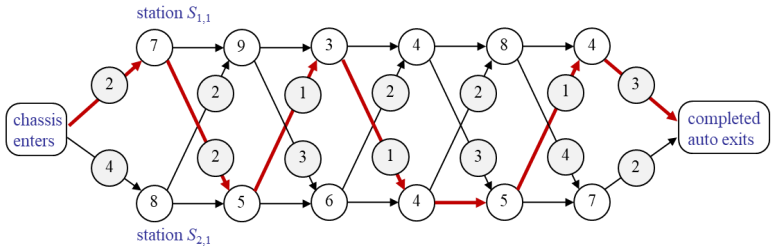
$f^* = 38$

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

$l^* = 1$

Step 3: Computing the fastest times （算法设计）

```
FASTEST-WAY(a, t, e, x, n)
1   f1[1] ← e1 + a1,1
2   f2[1] ← e2 + a2,1
3   for j ← 2 to n
4       if f1[j-1] + a1,j ≤ f2[j-1] + t2,j-1 + a1,j
5           f1[j] ← f1[j-1] + a1,j
6           l1[j] ← 1
7       else f1[j] ← f2[j-1] + t2,j-1 + a1,j
8           l1[j] ← 2
9       if f2[j-1] + a2,j ≤ f1[j-1] + t1,j-1 + a2,j
10          f2[j] ← f2[j-1] + a2,j
11          l2[j] ← 2
12      else f2[j] ← f1[j-1] + t1,j-1 + a2,j
13          l2[j] ← 1
14  if f1[n] + x1 ≤ f2[n] + x2
15      f* = f1[n] + x1
16      l* = 1
17  else f* = f2[n] + x2
18      l* = 2
```



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

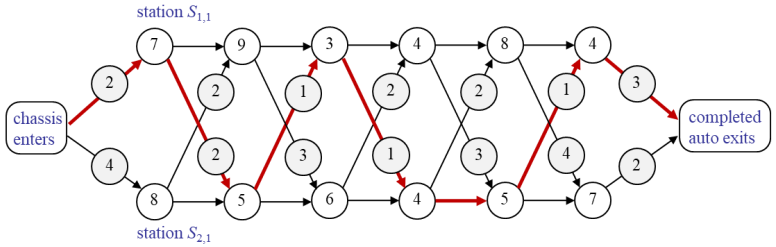
$l^* = 1$

Filling tables containing values $f_i[j]$ and $l_i[j]$ from left to right (and top to bottom within each column). To fill in $f_i[j]$, we need the values of $f_1[j\text{-}1]$ and $f_2[j\text{-}1]$, that we have already computed and stored .

动态规划算法：填表计算 $f_i[j]$ 和 $l_i[j]$，按列从左到右依序计算填表，同一列时从上到下计算，后面值的计算依赖于前面已经计算出来的值。

Step 4: Constructing the fastest way through the factory （求最优解）

- Use $f_i[j]$, f^*, $l_i[j]$, and l^*, to construct the sequence of stations used in the fastest way.
  使用 $f_{i,j}$, $l_{i,j}$, f^*, l^* 去构造最快路径
- PRINT-STATION procedure prints out the stations used, in decreasing order of station #.
  以 station 降序的方式输出 station #



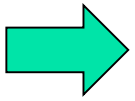
j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

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

$l^* = 1$

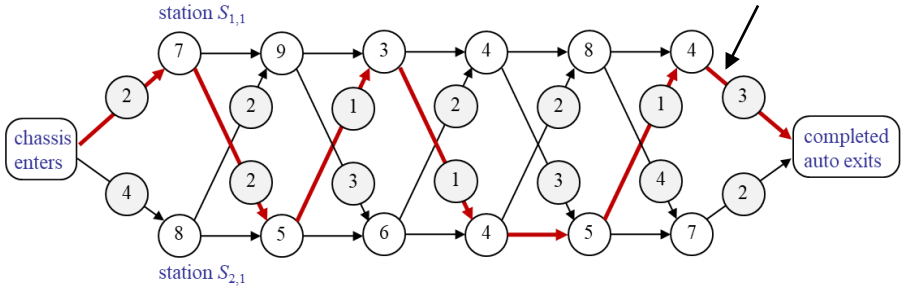
```
PRINT-STATIONS(l, n)
1   i ← l*
2   print " station " n ", line " i
3   for j ← n downto 2
4         i ← l_i[j]
5         print "station" j-1 ", line" i
```



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

optimal value（最优值） vs optimal solutions（最优解）

one of optimal solutions



optimal value

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

$f^* = 38$

- Optimization problems
  - There can be many possible solutions,
    多个可能（可行）解
  - Each solution has a value,
    每个解对一个值（函数值）
  - We wish to find a solution with the optimal (min or max) value.
    求一个解，使得目标函数最优（最小或最大）

- We call such a solution *an* optimal solution to the problem, as opposed to *the* optimal solution, since there may be several solutions that achieve the optimal value.
  称问题的某个解为一个最优解，而不是单纯地称为最优解，因为可能有多个解能得出问题的最优值

Dynamic programming

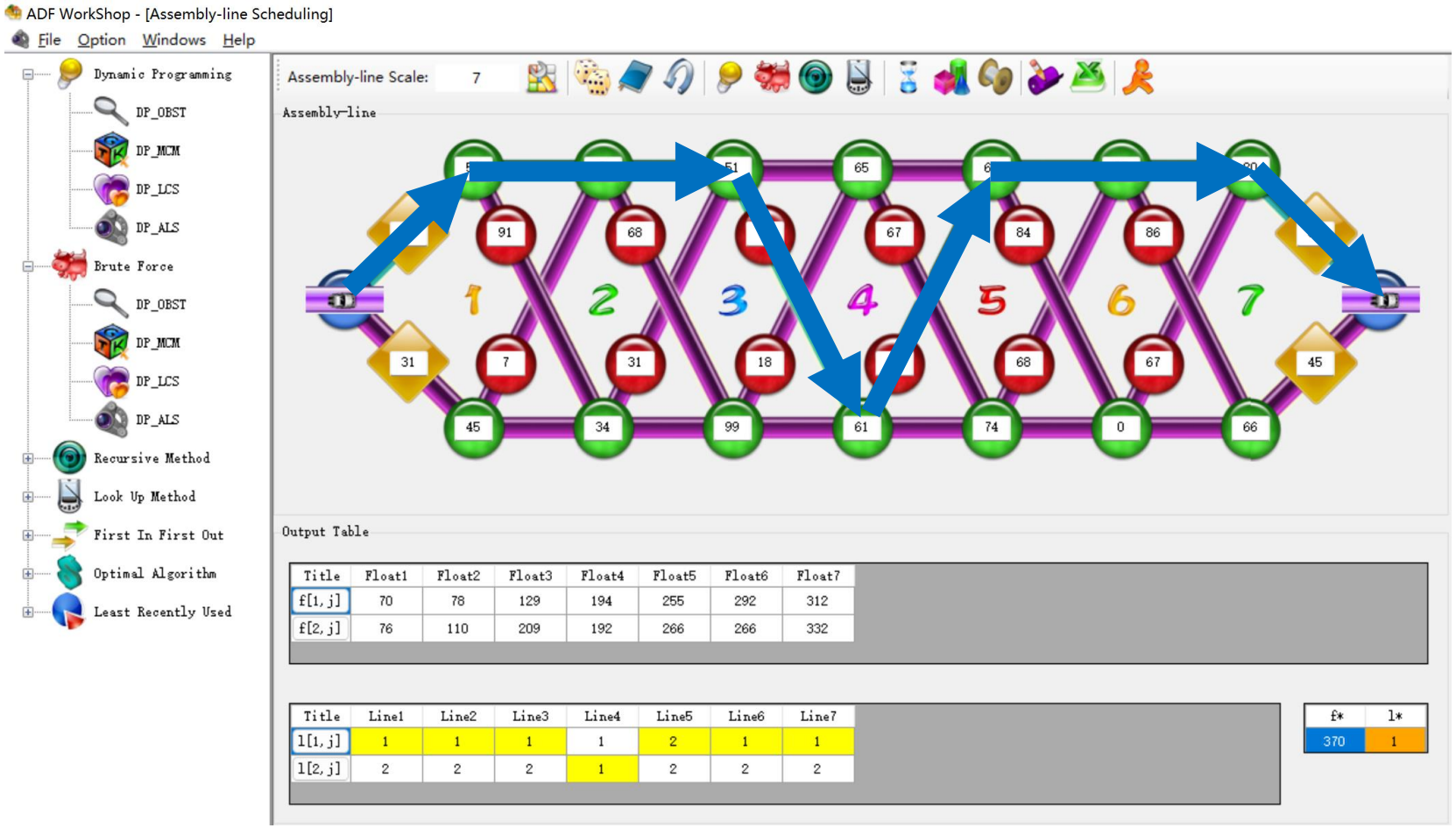
- The development of a dynamic-programming algorithm: four steps.

  1. Characterize the structure of an optimal solution. （分析最优解的结构特征）
  2. Recursively define the value of an optimal solution（求最优解的递归函数，转移函数）
  3. Compute the value of an optimal solution in a bottom-up fashion （按自底向上的方式计算最优值）
  4. [Construct an optimal solution from computed information.] （计算一个最优解）

- Step 4 can be omitted if only the value of an optimal solution is required. When we do perform step 4, we sometimes maintain additional information during the computation in step 3 to ease the construction of an optimal solution.
  当仅需要计算最优值时 step 4 通常可以省略。为了执行step 4〔更容易构造最优解〕，通常在执行step 3 时记录必要的信息。

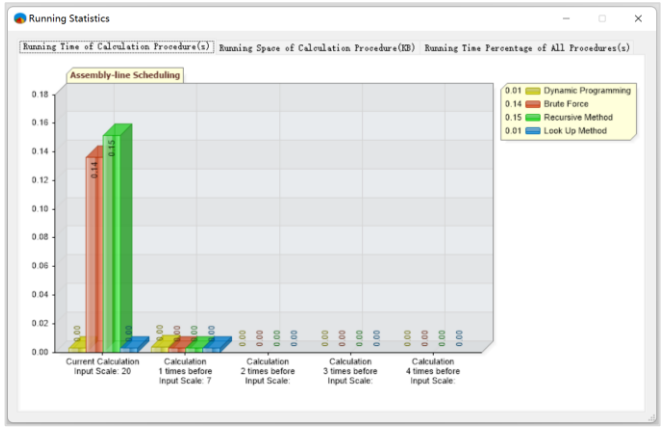
Exercise 1

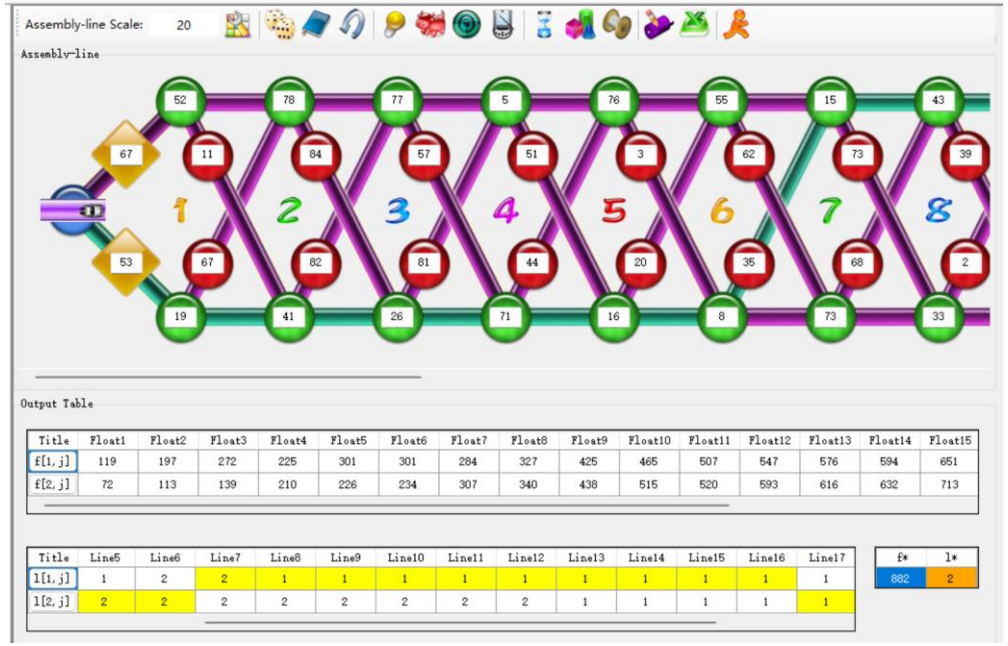
ADF_WorkShop

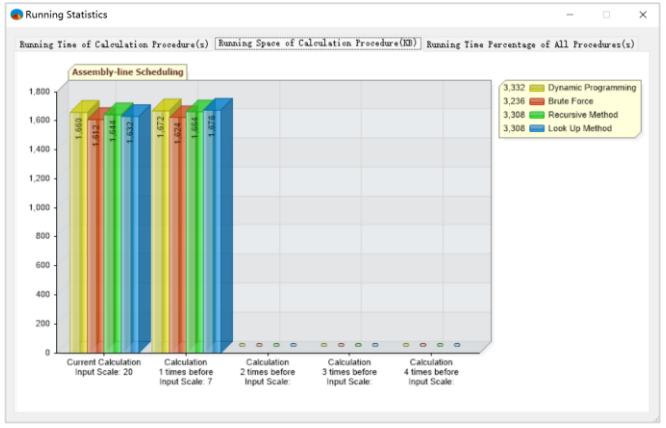


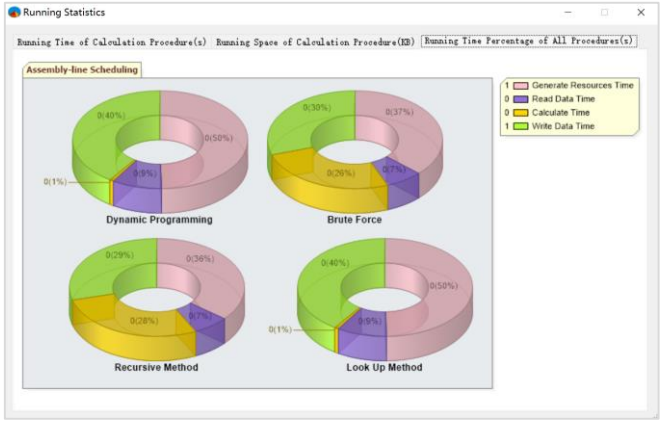
Exercise 1

ADF_WorkShop









Exercise 2

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2) \qquad (15.1)$$

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & , \text{ if } j = 1 \\ \min(f_1[j-1] + a_{1,j} \ , \ f_2[j-1] + t_{2,j-1} + a_{1,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.6)$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & , \text{ if } j = 1 \\ \min(f_2[j-1] + a_{2,j} \ , \ f_1[j-1] + t_{1,j-1} + a_{2,j}), & \text{if } j \geq 2 \end{cases} \qquad (15.7)$$

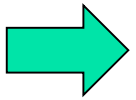
- Recursive algorithm (RA)? 递归算法如何写？
- Running time of RA? 递归算法的计算时间？

Exercise 3

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

$l^* = 1$

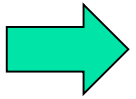
```
PRINT-STATIONS(l, n)
1   i ← l*
2   print " station " n ", line " i
3   for j ← n downto 2
4         i ← l_i[j]
5         print "station" j-1 ", line" i
```



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

为了以 station 增序的方式输出，应如何修改 PRINT-STATIONS ？

```
?
```



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