Part IV

Advanced Design and Analysis Techniques

Part I~II, 基本技术：模拟、分治、递归、概率、随机、…

Advanced Design and Analysis Techniques

- **Enumeration（穷举法）**
- **Search**
- **Backtrack （回溯）**
- **Divide-and-conquer: the solution of recurrences**
- **Randomization**
- **……**

- **Dynamic programming**
  - **Used for optimization problems (assembly-line scheduling).**
  - **Not a specific algorithm, but a technique (like divide-and-conquer).**
  - **"programming" means "tabular method", not computer programming.（programming表示列表方法，而不是指编程）**
- **Greedy algorithms**
  - **To make each choice in a locally optimal manner. (coin-changing) 以局部最优方式作每一次选择**

Four steps:

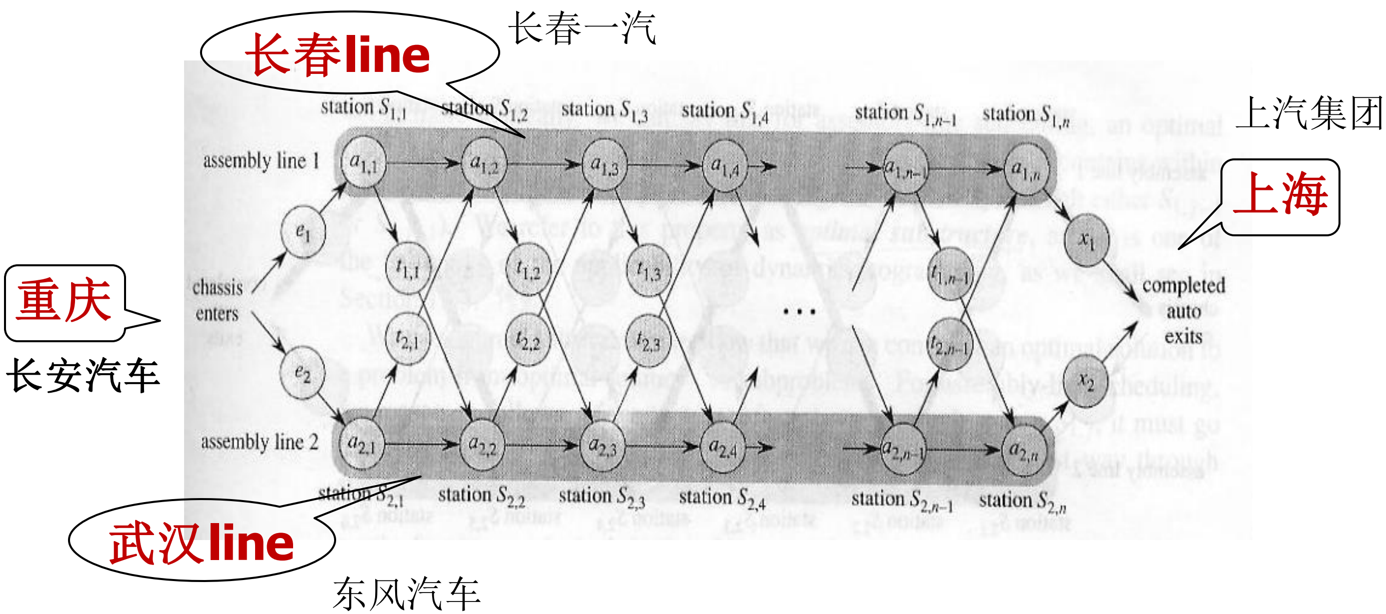
1. Theorem: **Characterize** the structure of an optimal solution.
2. Model: **Recursively** define the value of an optimal solution
   (to find **Transition Function**).
3. Computing (Algorithms): Compute the value of an optimal solution in a **bottom-up** fashion
   (to solve Transition Function).
4. [Construct an optimal solution from computed information.]

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）**

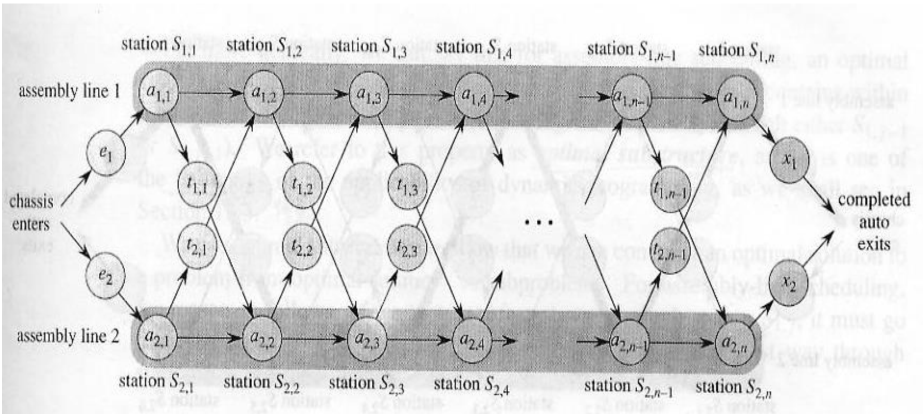
15.0 Assembly-line scheduling

- **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）**



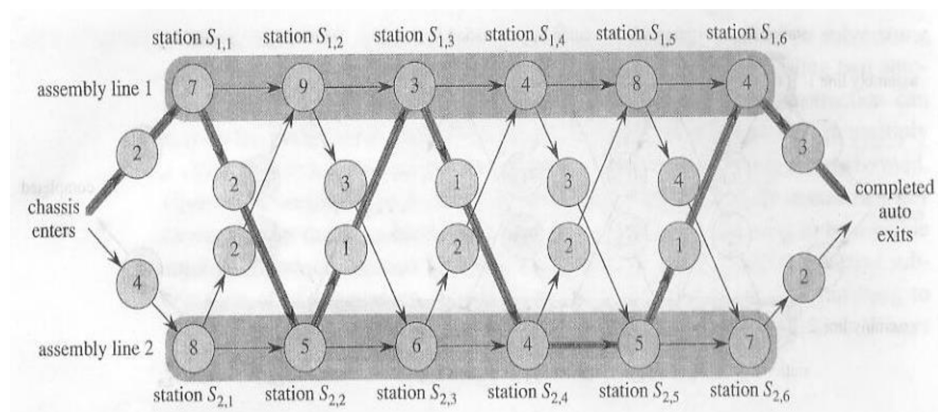
- **Problem: determine which stations to choose from line 1 and which to choose from line 2 in order to minimize the total time through the factory for one auto.** **（应选line1中哪一个装配站，line2中哪一个装配站，使得装配一个汽车的开销最小）**

- **The fastest way through the factory**



$f^* = 2+7+2+5+1+3$

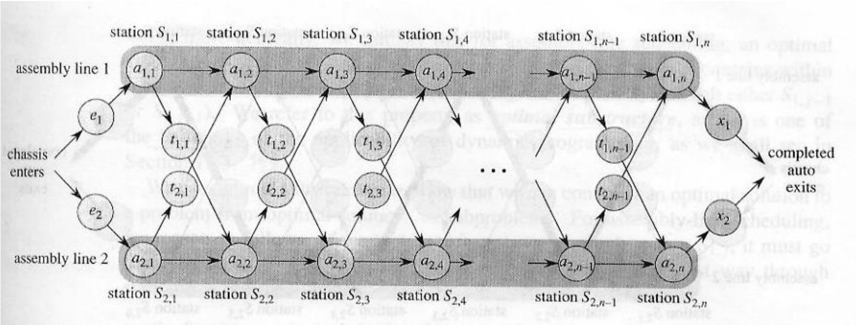
$+1+4+5+1+4+3=38$



- **How to find the fastest way?**

15.0 Assembly-line scheduling

- **"brute force" way (enumerate all the way) infeasible when there are many stations.**
  穷举法实际不可行



- **There are 2^n possible ways to choose stations. By enumerating all possible ways and computing how long each takes would require $\Omega(2^n)$ time, infeasible when n is large.**
  2^n种方法，$\Omega(2^n)$

- **If it is given a list of which stations to use in line 1 and which to use in line 2, it is easy to compute in $\Theta(n)$ time how long it takes a chassis to pass through the factory.**
  若已知 line1和 line2中哪些station被使用，需要$\Theta(n)$时间去计算该装配过程

application 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。

line 1

```
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 2

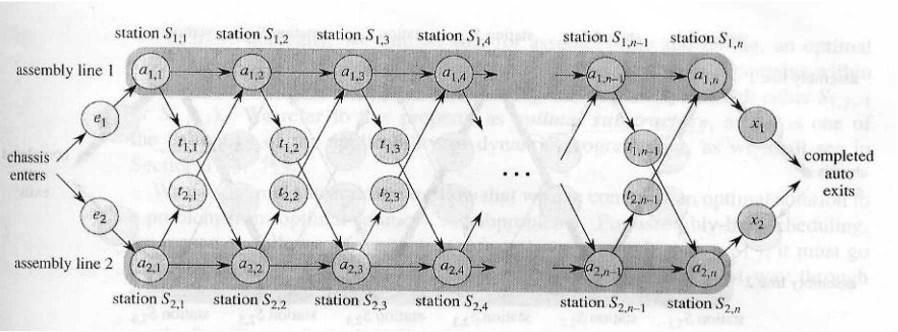
```
void gotoxy2(int x,int y)
void color2(int a)
void clear2(int x,int y)
void pp2(int x,int y,int w)
void clearall2()
void init2()
void initb2()
void getrand2()
bool moveup2()
bool moveleft2()
bool movedown2()
bool moveright2()
...
bool check2()
void printscreen2()
void gameover2()
bool checkwin2()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

line 3

```
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()
```

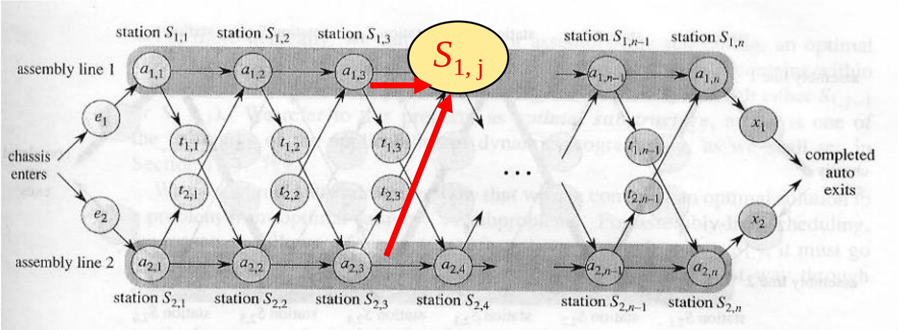
Step 1: The structure of the fastest way through the factory

- **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 $S_{1,j}$ 的以最快的路径是什么?
  - **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 $S_{1,j}$.** **（若 $j = 1$，则只有一种方式通过$S_{1,1}$）**

Step 1: The structure of the fastest way through the factory

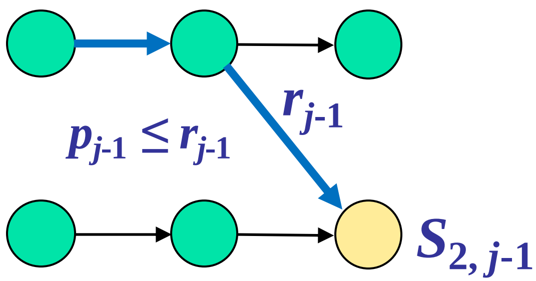


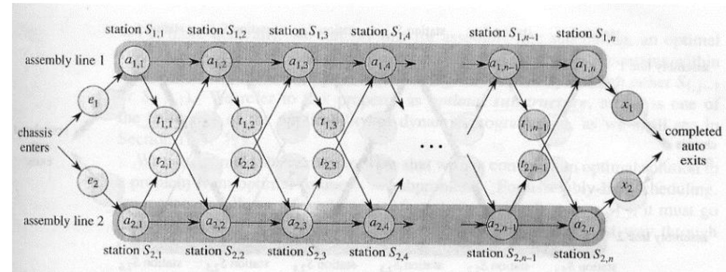
Considering the fastest possible way for a chassis to get from the starting point through station $S_{1,j}$

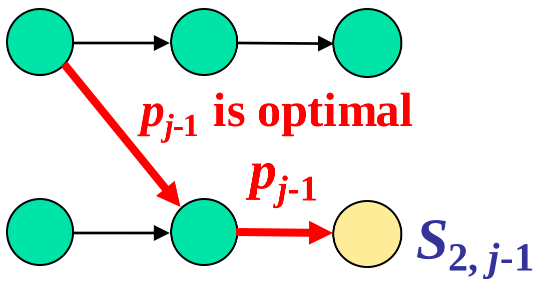
子问题：通过station $S_{1,j}$ 的以最快的路径是什么？

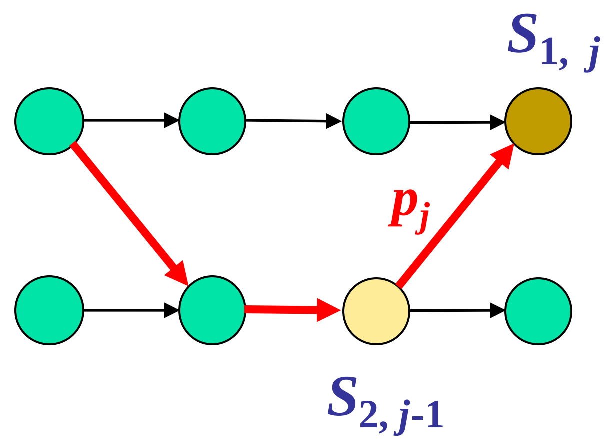
- If $j = 1$, only one way
- For $j \geq 2$, two choices
  - through $S_{1,j-1}$, then **directly** to $S_{1,j}$, no transfer time;
  - through $S_{2,j-1}$, then **transfer over** to station $S_{1,j}$, transfer time $t_{2,j-1}$.
  - These two possibilities have much in common.

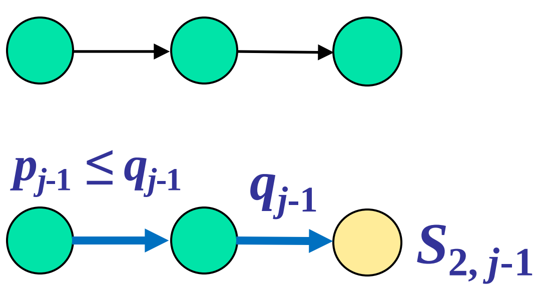
Step 1: The structure of the fastest way through the factory









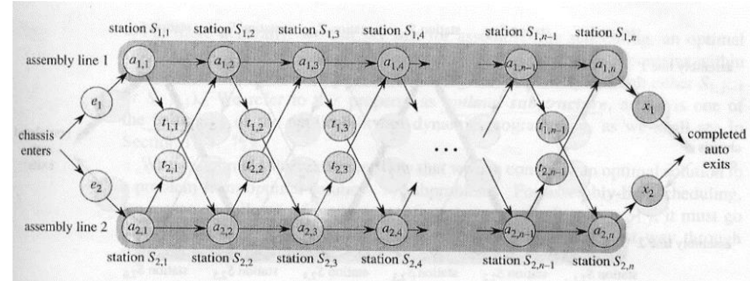


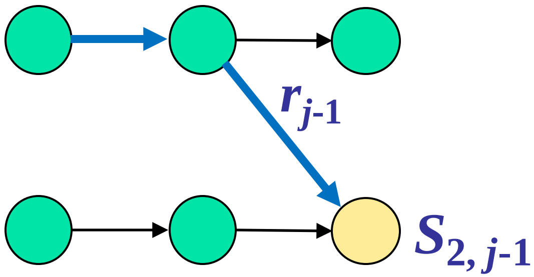
Optimal substructure:

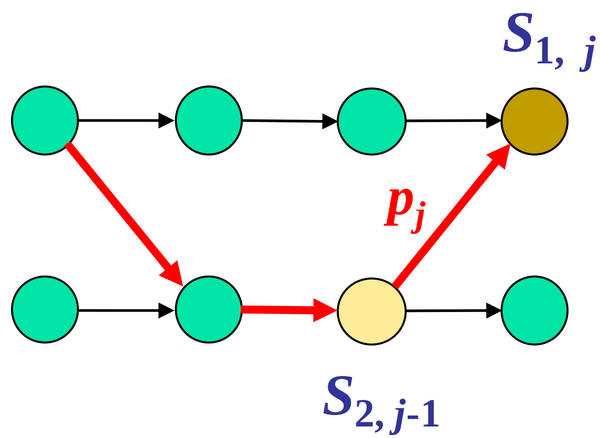
If p_j is optimal, $p_{j\text{-}1} \subset p_j$,

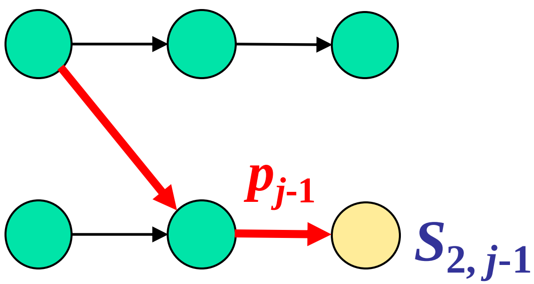
then If $p_{j\text{-}1}$ is optimal.

Proof of Optimal substructure









Optimal substructure:

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

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

proof:

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

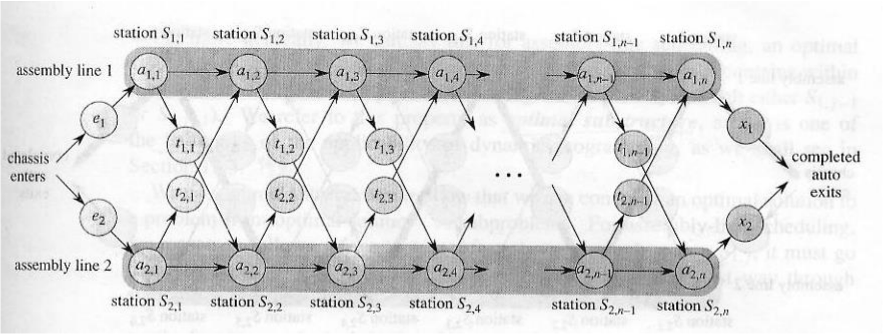
$p_j = p_{j\text{-}1} + t(S_{2,j\text{-}1}, S_{1,j})$

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

contradiction to p_j is optimal.

Step 1: The structure of the fastest way through the factory

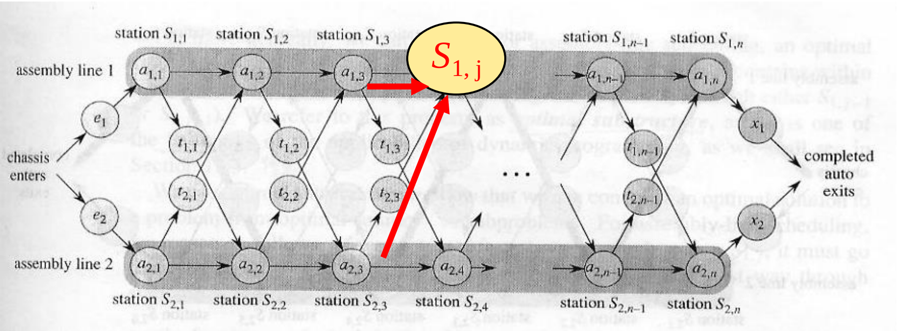
Optimal substructure



- 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



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

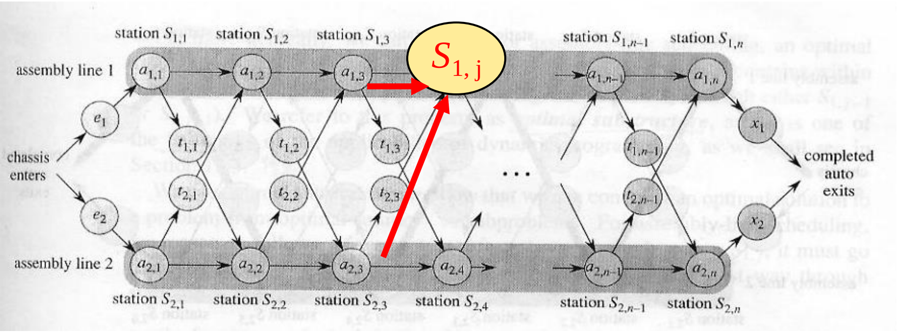
  使用最佳子结构可以从子问题的最优解来构造原问题的最优解

- **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



- 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

- **Define the value of an optimal solution recursively in terms of the optimal solutions to subproblems .**

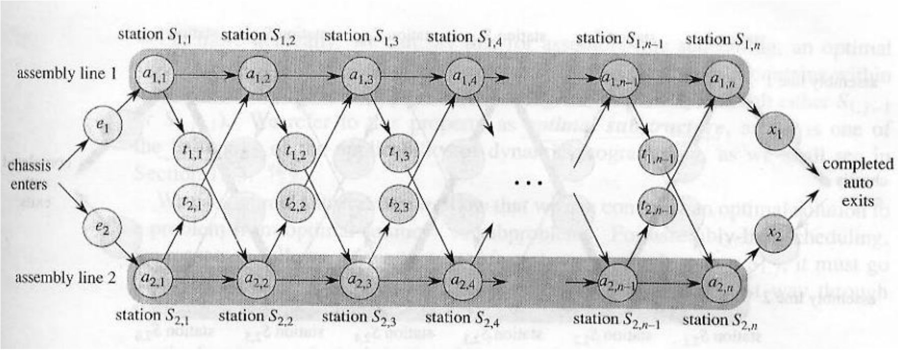
  通过子问题的最优解来递归地定义原问题的最优解

- **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

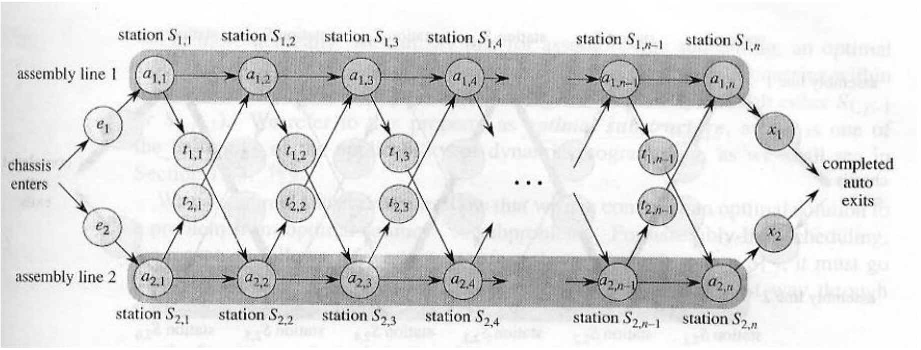
- **Let $f_i[j]$ = fastest time to through $S_{i,\,j}$ from the starting point.**

  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

- 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

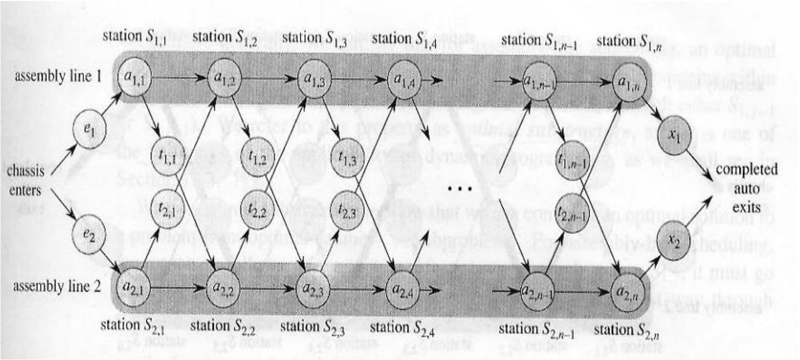
$$f^* = \min(f_1[n] + x_1,\ f_2[n] + x_2) \qquad (15.1)$$

Step 2: A recursive solution

- **Compute easily $f_1[1]$ and $f_2[1]$. Directly to station 1 on either line.**

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

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



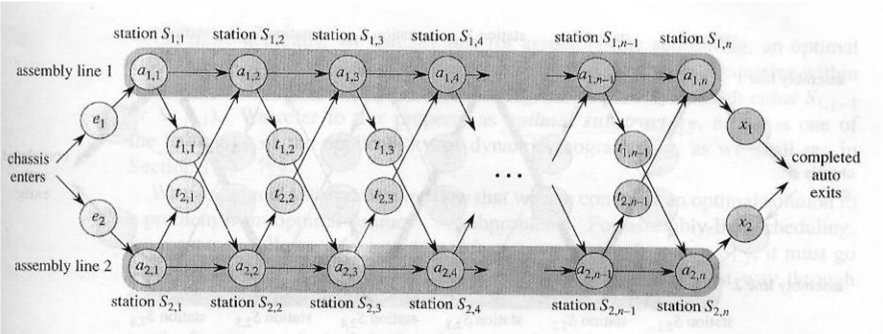
- **For j = 2, 3, ..., n (i = 1, 2),**
  - **First, for $f_1[j]$, the fastest way through $S_{1,j}$ is, either the fastest way through $S_{1,j-1}$, directly through $S_{1,j}$, then $f_1[j]=f_1[j-1]+a_{1,j}$; or the fastest way through $S_{2,j-1}$, transfer from line 2 to line 1, and through $S_{1,j}$, then $f_1[j]=f_2[j-1]+t_{2,j-1}+a_{1,j}$. Thus**

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

  - **Symmetrically,**

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

Step 2: A recursive solution



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

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

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

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

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

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

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

Transition Function

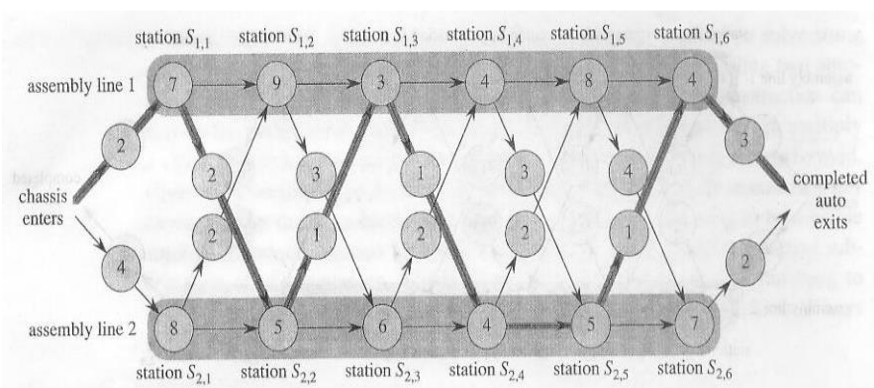
Step 2: A recursive solution

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

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

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

- ***$f_i[j]$* values for the example**



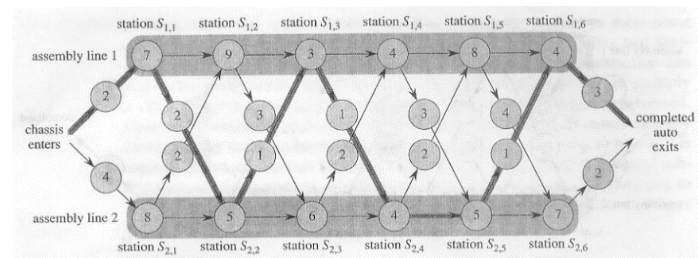
j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

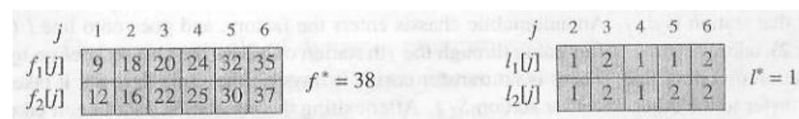
$f^* = 38$

How to get an optimal solution?

optimal value vs optimal solutions

- $f_i[j]$: 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.**





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

$f^* = 38$

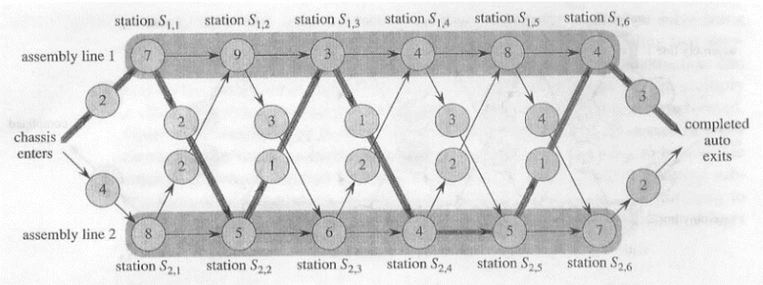
j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

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$2 \le j$。若j=1，则$l_i[1]$表示通过station 0 的行号，错。l^*表示以最快方式通过station n 的行号）。

Step 2: A recursive solution

- What if we want to construct an optimal solution?
  - Define $l_i[j]$ = line # (1or 2), whose station j-1 is used in a fastest way through $S_{i,j}$.
  - l^* = line # whose station n is used.



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。 l^* 表示以最快方式通过 station n 的行号）

- 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 ($S_{1,1}$).

Step 3: Computing the fastest times

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

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

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

- **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.**
- **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

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

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

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

- **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),

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

$$\begin{array}{cccccccccc}
r_1(n) & & f_1[n] & & & & f_2[n] & & & r_2(n) \\
 & & \swarrow\ \searrow & & & & \swarrow\ \searrow & & & \\
r_1(n-1) & f_1[n-1] & & f_2[n-1] & & f_1[n-1] & & f_2[n-1] & & r_2(n-1) \\
 & \swarrow\ \searrow & & \swarrow\ \searrow & & \swarrow\ \searrow & & \swarrow\ \searrow & & \\
r_1(n-2) & f_1[n-2] & f_2 & f_1 & f_2 & f_1 & f_2 & f_1 & f_2 & r_2(n-2)
\end{array}$$

- **For j=1, 2, ..., n-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 $\Theta(2^n)$.

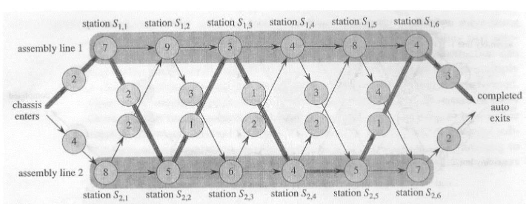
  仅 $f_1[1]$ 就将被调用2^{n-1}次！所有 $f_i[j]$ 的被引用的次数为$\Theta(2^n)$

Step 3: Computing the fastest times

- **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. **以station数 j 增加的方式来计算 $f_i[j]$**



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

$l^* = 1$

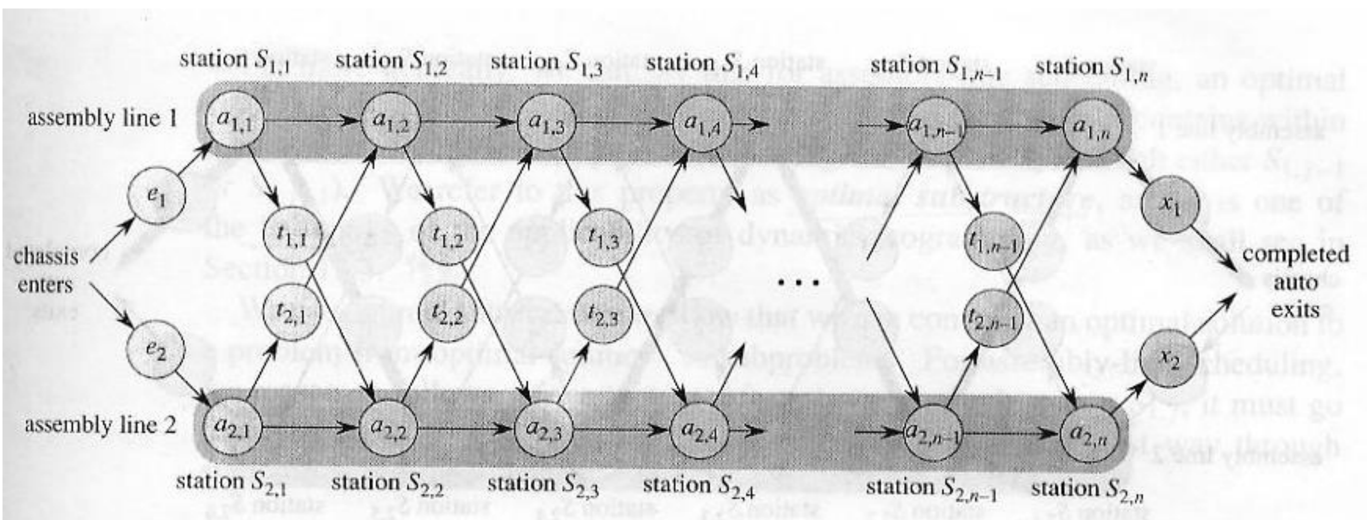
we can compute the fastest way through the factory, and the time it takes, in $\Theta(n)$ time.

仅需要$\Theta(n)$时间就可以计算出通过完整工序的最快方法

Step 3: Computing the fastest times

The FASTEST-WAY procedure takes as input the values $a_{i,j}$, $t_{i,j}$, e_i, and x_i, as well as n, the number of stations in each assembly line.

以 $a_{i,j}$, $t_{i,j}$, e_i, x_i, n 作为输入，可设计程序 FASTEST-WAY



Step 3: Computing the fastest times

```
FASTEST-WAY(a, t, e, x, n)
f1[1] ← e1 + a1,1
f2[1] ← e2 + a2,1
for j ← 2 to n
    if f1[j-1] + a1,j ≤ f2[j-1] + t2,j-1 + a1,j
        f1[j] ← f1[j-1] + a1,j
        l1[j] ← 1
    else f1[j] ← f2[j-1] + t2,j-1 + a1,j
        l1[j] ← 2
    if f2[j-1] + a2,j ≤ f1[j-1] + t1,j-1 + a2,j
        f2[j] ← f2[j-1] + a2,j
        l2[j] ← 2
    else f2[j] ← f1[j-1] + t1,j-1 + a2,j
        l2[j] ← 1
if f1[n] + x1 ≤ f2[n] + x2
    f* = f1[n] + x1
    l* = 1
else f* = f2[n] + x2
    l* = 2
```

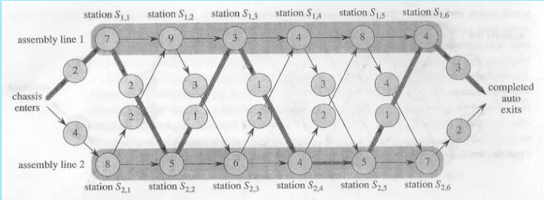
$$f_1[1] = e_1 + a_{1,1}\ , \qquad (15.2)$$

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

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

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

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



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

$f^* = 38$

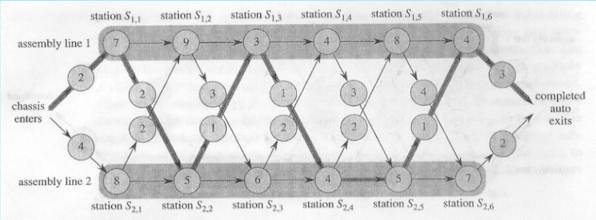
j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

Step 3: Computing the fastest times

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



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

$f^* = 38$

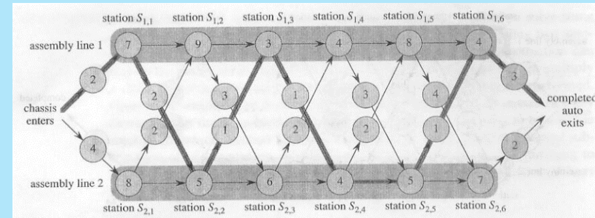
j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

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 $f_1[j-1]$ and $f_2[j-1]$, that we have already computed and **stored** .

Step 4: Constructing the fastest way through the factory

- **Use $f_i[j]$, f^*, $l_i[j]$, and l^*, to construct the sequence of stations used in the fastest way.**
  使用 f 和 l 去构造最快路径
- **PRINT-STATION procedure prints out the stations used, in decreasing order of 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)
    i ← l*
    print " station " n ", line " i
    for j ← n downto 2
            i ← l_i[j]
            print "station" j-1 ", line" i
```

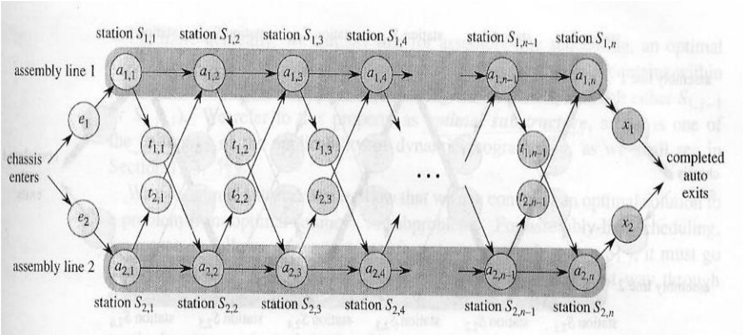
station 6, line 1
station 5, line 2
station 4, line 2
station 3, line 1
station 2, line 2
station 1, line 1

Comparison between Divide-and-conquer and Dynamic programming

<table>
<tr><th></th><th>Divide-and-conquer</th><th>Dynamic programming</th></tr>
<tr><td>similarity</td><td colspan="2">solves problems by combining the solutions to subproblems.</td></tr>
<tr><td rowspan="2">difference</td><td>A:
1) partition the problem into independent subproblems,
2) solve the subproblems recursively,
3) combine their solutions to solve the original problem.</td><td>B:
Divide, but the subproblems are not independent, that is, when subproblems share subsubproblems.
子问题不独立，不同子问题共享相同的子子问题）</td></tr>
<tr><td>C:
might do more work than necessary, repeatedly solving the common subsubproblems.</td><td>D:
solves every subsubproblem just once and then saves its answer in a table, avoiding recomputing the answer every time the subsubproblem is encountered.</td></tr>
</table>

归并排序：分治策略，步骤A
递归的ALS：分治策略，步骤B，C
动态的ALS：动态规划，步骤B，D

Comparison between Divide-and-conquer and Dynamic programming



- **Dynamic programming is typically applied to optimization problems.**
- **Optimization problems**
  - **There can be many possible solutions,**
  - **Each solution has a value,**
  - **We wish to find a solution with the optimal (min or max) value.**
- **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。

Comparison between Divide-and-conquer and Dynamic programming

- **The development of a dynamic-programming algorithm can be broken into 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 in a bottom-up fashion.**
  4. **Construct an optimal solution from computed information.**
- **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.** （当仅需求最优值时step 4 通常可以省略。为了执行step 4〔更容易第构造最优解〕，通常在执行step 3 时记录必要的信息）

Exercises

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

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

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

- **Recursive algorithm ?**
- **Running time ?**

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)
i ← l*
print " station " n ", line " i
for j ← n downto 2
        i ← l_i[j]
        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

Exercises

Show how to modify the PRINT-STATIONS procedure to print out the stations in increasing order of station number.