Programación dinámica

mat-151

Programación dinámica

Programación dinámica

- La programación dinámica, igual que los métodos divide-and-conquer resuelve problemas combinando las soluciones de subproblemas.

Programación dinámica

- La programación dinámica, igual que los métodos divide-and-conquer resuelve problemas combinando las soluciones de subproblemas.
- Creada por Richard Bellman en 1953.

Programación dinámica

- La programación dinámica, igual que los métodos divide-and-conquer resuelve problemas combinando las soluciones de subproblemas.
- Creada por Richard Bellman en 1953.
- Los métodos divide-and-conquer: (1) dividen el problema en subproblemas independientes, (2) resuelven en problema de manera recursiva y (3) combinan sus soluciones.

Programación dinámica

- La programación dinámica, igual que los métodos divide-and-conquer resuelve problemas combinando las soluciones de subproblemas.
- Creada por Richard Bellman en 1953.
- Los métodos divide-and-conquer: (1) dividen el problema en subproblemas independientes, (2) resuelven en problema de manera recursiva y (3) combinan sus soluciones.
- La programación dinámica se aplica cuando los subproblemas no son independientes: cuando los subproblemas comparten subsubproblemas (donde hay traslape).

Programación dinámica

- La programación dinámica, igual que los métodos divide-and-conquer resuelve problemas combinando las soluciones de subproblemas.
- Creada por Richard Bellman en 1953.
- Los métodos divide-and-conquer: (1) dividen el problema en subproblemas independientes, (2) resuelven en problema de manera recursiva y (3) combinan sus soluciones.
- La programación dinámica se aplica cuando los subproblemas no son independientes: cuando los subproblemas comparten subsubproblemas (donde hay traslape).
- Los métodos divide-and-conquer en este caso hacen más trabajo del necesario, un algoritmo de programación dinámica resuelve cada subproblema una sola vez y guarda la solución en una tabla.

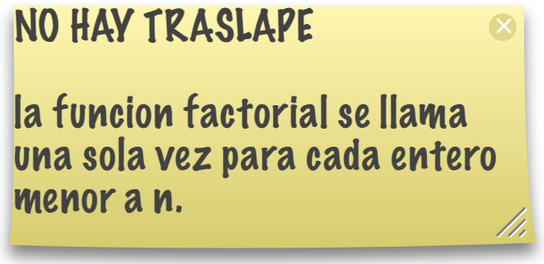
Programación dinámica: **subproblemas que traslapan**

NO HAY TRASLAPE

la funcion factorial se llama una sola vez para cada entero menor a n.

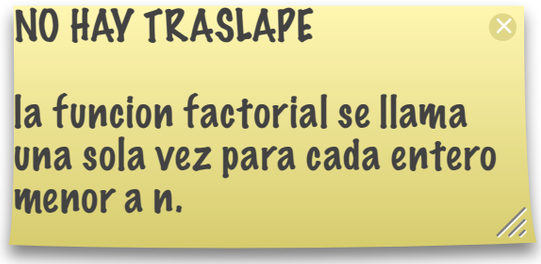
Programación dinámica: **subproblemas que traslapan**

- La programación dinámica se aplica típicamente a problemas de optimización, donde puede haber varias soluciones.



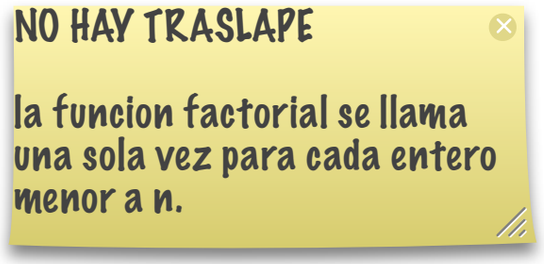
Programación dinámica: **subproblemas que traslapan**

- La programación dinámica se aplica típicamente a problemas de optimización, donde puede haber varias soluciones.
- Estos problemas deben tener:



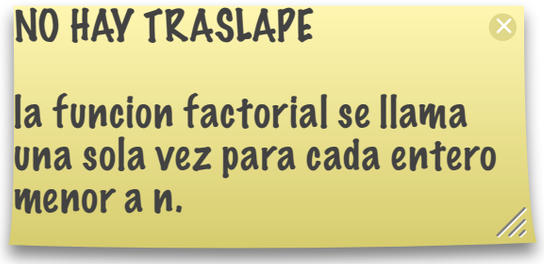
Programación dinámica: **subproblemas que traslapan**

- La programación dinámica se aplica típicamente a problemas de optimización, donde puede haber varias soluciones.
- Estos problemas deben tener:
  - subproblemas que traslapan



Programación dinámica: **subproblemas que traslapan**

- La programación dinámica se aplica típicamente a problemas de optimización, donde puede haber varias soluciones.
- Estos problemas deben tener:
  - subproblemas que traslapan
    - el problema se puede dividir en subproblemas cuya solución puede ser utilizada varias veces.



Programación dinámica: **subproblemas que traslapan**

- La programación dinámica se aplica típicamente a problemas de optimización, donde puede haber varias soluciones.
- Estos problemas deben tener:
  - subproblemas que traslapan
    - el problema se puede dividir en subproblemas cuya solución puede ser utilizada varias veces.
    - esto se relaciona estrechamente con la recursión.

NO HAY TRASLAPE

la funcion factorial se llama una sola vez para cada entero menor a n.

Programación dinámica: **subproblemas que traslapan**

- La programación dinámica se aplica típicamente a problemas de optimización, donde puede haber varias soluciones.
- Estos problemas deben tener:
  - subproblemas que traslapan
    - el problema se puede dividir en subproblemas cuya solución puede ser utilizada varias veces.
    - esto se relaciona estrechamente con la recursión.

NO HAY TRASLAPE

la funcion factorial se llama una sola vez para cada entero menor a n.

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ \prod_{k=1}^{n} k & \text{if } n > 0 \end{cases}$$

Programación dinámica: **subproblemas que traslapan**

- La programación dinámica se aplica típicamente a problemas de optimización, donde puede haber varias soluciones.
- Estos problemas deben tener:
  - subproblemas que traslapan
    - el problema se puede dividir en subproblemas cuya solución puede ser utilizada varias veces.
    - esto se relaciona estrechamente con la recursión.

NO HAY TRASLAPE

la funcion factorial se llama una sola vez para cada entero menor a n.

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ \prod_{k=1}^{n} k & \text{if } n > 0 \end{cases}$$

```
unsigned int factorial(unsigned int n){
    if(n==0)
        return 1;
    return n*factorial(n-1);
}
```

Programación dinámica: **números de Fibonacci**

$$F(n) = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

Programación dinámica: **números de Fibonacci**

$$F(n) = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

Programación dinámica: **números de Fibonacci**

$$F(n) = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

```
unsigned int fibonacci(unsigned int n){
    if (n==0)
        return 0;
    else if (n==1)
        return 1;
    return fibonacci(n-1)+fibonacci(n-2);
}
```

Programación dinámica: **números de Fibonacci**

$$F(n) = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

```
unsigned int fibonacci(unsigned int n){
    if (n==0)
        return 0;
    else if (n==1)
        return 1;
    return fibonacci(n-1)+fibonacci(n-2);
}
```

$O(c^n)$

Programación dinámica: **números de Fibonacci**

$$F(n) = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

```
unsigned int fibonacci(unsigned int n){
    if (n==0)
        return 0;
    else if (n==1)
        return 1;
    return fibonacci(n-1)+fibonacci(n-2);
}
```

$O(c^n)$

$$T(n) = \begin{cases} 1, & si\ n \le 1 \\ 2 \cdot T(n-1) + 1, & si\ n > 1 \end{cases}$$

$T(n) = 2*T(n-1) + 1 =$

$= 2*(2*T(n-2) + 1) + 1 = 2^2*T(n-2) + (2^2-1) =$

$\ldots$

$= 2^k*T(n-k) + (2^k-1)$

Para $k = n-1$, $T(n) = 2^{n-1}*T(1) + 2^{n-1}-1$, y por tanto $T(n)$ es $O(2^n)$.

Programación dinámica: **números de Fibonacci**

```
F[0]=0;
F[1]=1;
for( int i=2; i<=n; i++)
    F[i] = F[i-1] + F[i-2];
```

Programación dinámica: **números de Fibonacci**

- Programación dinámica bottom-up: resuelve todas las instancias a partir de las más chicas hasta la que se desea calcular.

```
F[0]=0;
F[1]=1;
for( int i=2; i<=n; i++)
    F[i] = F[i-1] + F[i-2];
```

Programación dinámica: **números de Fibonacci**

- Programación dinámica bottom-up: resuelve todas las instancias a partir de las más chicas hasta la que se desea calcular.

```
F[0]=0;
F[1]=1;
for( int i=2; i<=n; i++)
    F[i] = F[i-1] + F[i-2];
```

O(n)

Programación dinámica: **números de Fibonacci**

- Programación dinámica top-down: esquema recursivo que expresa el problema como subproblemas y memoriza los resultados para reutilizarlos más tarde.

Programación dinámica: **números de Fibonacci**

- Programación dinámica top-down: esquema recursivo que expresa el problema como subproblemas y memoriza los resultados para reutilizarlos más tarde.

```
int F(int i){
    static int knownF[maxN];
    if(knownF[i] != 0 return knownF[i];
    int t=i;
    if (i<0) return 0;
    if (i>1) t = F(i-1)+F(i-2);
    return knownF[i] = t;
}
```

Programación dinámica: **números de Fibonacci**

- Programación dinámica top-down: esquema recursivo que expresa el problema como subproblemas y memoriza los resultados para reutilizarlos más tarde.

```
int F(int i){
    static int knownF[maxN];
    if(knownF[i] != 0 return knownF[i];
    int t=i;
    if (i<0) return 0;
    if (i>1) t = F(i-1)+F(i-2);
    return knownF[i] = t;
}
```

O(n)

Programación dinámica: **números de Fibonacci**

- Programación dinámica top-down: esquema recursivo que expresa el problema como subproblemas y memoriza los resultados para reutilizarlos más tarde.

```
int F(int i){
    static int knownF[maxN];
    if(knownF[i] != 0 return knownF[i];
    int t=i;
    if (i<0) return 0;
    if (i>1) t = F(i-1)+F(i-2);
    return knownF[i] = t;
}
```

O(n)

- No requiere tanto espacio, F[45] es el máximo representable en 32 bits.

Programación dinámica: **números de Fibonacci**

- Programación dinámica top-down: esquema recursivo que expresa el problema como subproblemas y memoriza los resultados para reutilizarlos más tarde.

```
int F(int i){
    static int knownF[maxN];
    if(knownF[i] != 0 return knownF[i];
    int t=i;
    if (i<0) return 0;
    if (i>1) t = F(i-1)+F(i-2);
    return knownF[i] = t;
}
```

O(n)

- No requiere tanto espacio, F[45] es el máximo representable en 32 bits.
- Aliviar la redundancia algorítmica usando recursos en la memoria: programación dinámica

Programación dinámica: **subestructura óptima**

Programación dinámica: **subestructura óptima**

- subestructura óptima

Programación dinámica: **subestructura óptima**

- subestructura óptima
    - se puede resolver el problema de manera óptima a partir de la resolución óptima de los subproblemas.

Programación dinámica: **subestructura óptima**

- subestructura óptima
    - se puede resolver el problema de manera óptima a partir de la resolución óptima de los subproblemas.
- Los problemas con subestructuras óptimos se pueden resolver en general siguiendo los tres pasos:

Programación dinámica: **subestructura óptima**

- subestructura óptima
    - se puede resolver el problema de manera óptima a partir de la resolución óptima de los subproblemas.
- Los problemas con subestructuras óptimos se pueden resolver en general siguiendo los tres pasos:
    - dividir el problema en subproblemas más pequeños.

Programación dinámica: **subestructura óptima**

- subestructura óptima
    - se puede resolver el problema de manera óptima a partir de la resolución óptima de los subproblemas.
- Los problemas con subestructuras óptimos se pueden resolver en general siguiendo los tres pasos:
    - dividir el problema en subproblemas más pequeños.
    - resolver estos problemas de manera óptima usando este proceso de tres pasos recursivamente.

Programación dinámica: **subestructura óptima**

- subestructura óptima
    - se puede resolver el problema de manera óptima a partir de la resolución óptima de los subproblemas.
- Los problemas con subestructuras óptimos se pueden resolver en general siguiendo los tres pasos:
    - dividir el problema en subproblemas más pequeños.
    - resolver estos problemas de manera óptima usando este proceso de tres pasos recursivamente.
    - usar las soluciones óptimas para construir una solución óptima al problema original.

Programación dinámica

Programación dinámica

- Una resolución por programación dinámica se puede dividir en:

Programación dinámica

- Una resolución por programación dinámica se puede dividir en:
  - caracterización de la estructura de la solución óptima.

Programación dinámica

- Una resolución por programación dinámica se puede dividir en:
  - caracterización de la estructura de la solución óptima.
  - definición recursiva de la solución óptima.

Programación dinámica

- Una resolución por programación dinámica se puede dividir en:
  - caracterización de la estructura de la solución óptima.
  - definición recursiva de la solución óptima.
  - calculo bottom-up o top-down de las subsoluciones óptimas (implementación).

Programación dinámica

- Una resolución por programación dinámica se puede dividir en:
  - caracterización de la estructura de la solución óptima.
  - definición recursiva de la solución óptima.
  - calculo bottom-up o top-down de las subsoluciones óptimas (implementación).
  - construcción de la solución óptima a partir de la información calculada.

Programación dinámica: **líneas de producción**

Programación dinámica: **líneas de producción**

- Se tienen dos líneas de producción en una fábrica

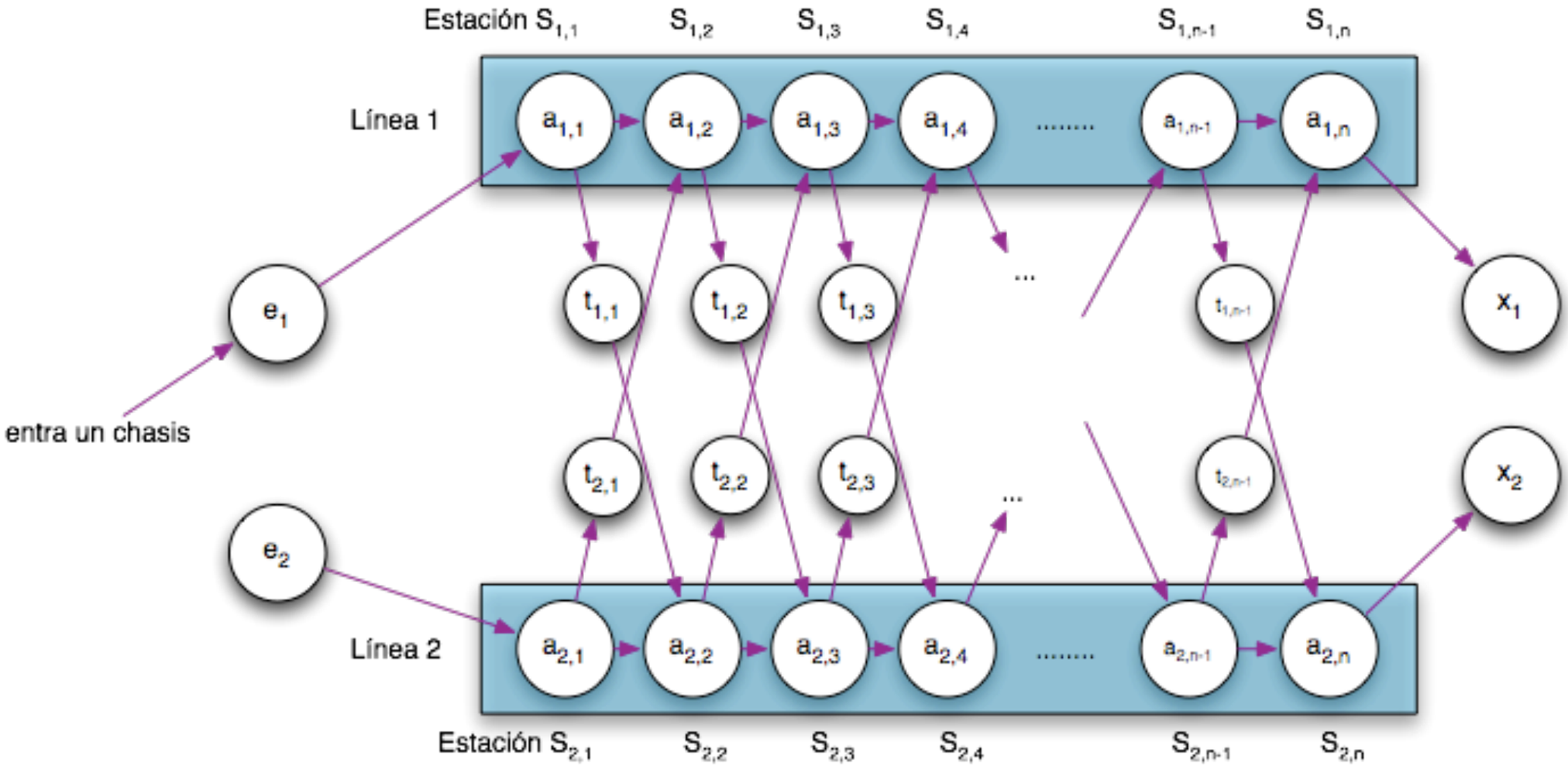
Programación dinámica: **líneas de producción**

- Se tienen dos líneas de producción en una fábrica

- ¿Cuáles estaciones elegir de cada línea para minimizar el tiempo total de ensamblado?

Programación dinámica: **líneas de producción**

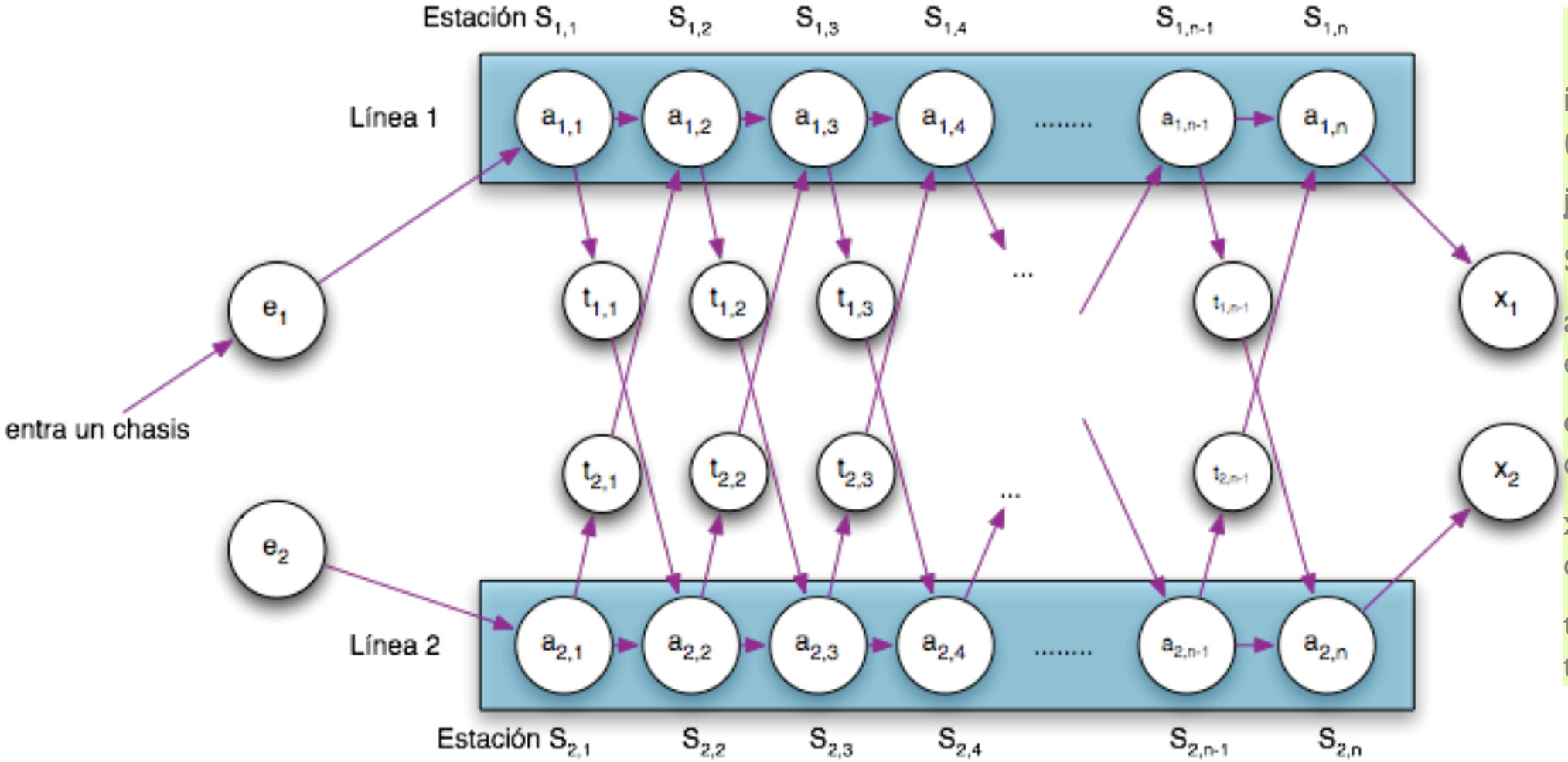
- Se tienen dos líneas de producción en una fábrica



- ¿Cuáles estaciones elegir de cada línea para minimizar el tiempo total de ensamblado?

Programación dinámica: **líneas de producción**

- Se tienen dos líneas de producción en una fábrica



i: linea de produccion (1,2).

j: estación (1,2, ..., n).

$\mathbf{S_{i,j}}$: estación j en la línea i.

$\mathbf{a_{i,j}}$: tiempo de ensamblado en la estación $\mathbf{S_{i,j}}$.

$\mathbf{e_i}$: tiempo de entrada del chassis a la línea i.

$\mathbf{x_i}$: tiempo de salida del chasis de la línea i.

$\mathbf{t_{i,j}}$: tiempo de transferencia de línea.

- ¿Cuáles estaciones elegir de cada línea para minimizar el tiempo total de ensamblado?

Programación dinámica: **líneas de producción**

Programación dinámica: **líneas de producción**

- Una solución de fuerza bruta (probar todas las combinaciones) toma un tiempo de cálculo prohibitivo cuando hay muchas estaciones.

Programación dinámica: **líneas de producción**

- Una solución de fuerza bruta (probar todas las combinaciones) toma un tiempo de cálculo prohibitivo cuando hay muchas estaciones.
    - ¿Cuántas combinaciones?

Programación dinámica: **líneas de producción**

- Una solución de fuerza bruta (probar todas las combinaciones) toma un tiempo de cálculo prohibitivo cuando hay muchas estaciones.
  - ¿Cuántas combinaciones?
  - Calcular todas las combinaciones requeriría un tiempo de cálculo $\Omega(2^n)$

Programación dinámica: **líneas de producción**

- Una solución de fuerza bruta (probar todas las combinaciones) toma un tiempo de cálculo prohibitivo cuando hay muchas estaciones.
  - ¿Cuántas combinaciones?
  - Calcular todas las combinaciones requeriría un tiempo de cálculo $\Omega(2^n)$
- Si se nos da una secuencia de estaciones de las líneas 1 y 2 es fácil calcular el tiempo en $\Theta(n)$.

Programación dinámica: **líneas de producción**

- Una solución de fuerza bruta (probar todas las combinaciones) toma un tiempo de cálculo prohibitivo cuando hay muchas estaciones.
  - ¿Cuántas combinaciones? 2^n
  - Calcular todas las combinaciones requeriría un tiempo de cálculo $\Omega(2^n)$
- Si se nos da una secuencia de estaciones de las líneas 1 y 2 es fácil calcular el tiempo en $\Theta(n)$.

Programación dinámica: **líneas de producción**

Programación dinámica: **líneas de producción**

- Paso 1: la estructura de la solución óptima (tiempo de ensamblado más corto)

Programación dinámica: **líneas de producción**

- Paso 1: la estructura de la solución óptima (tiempo de ensamblado más corto)
  - Consideramos el chasis en la estación $\mathbf{s_{1,j}}$.

Programación dinámica: **líneas de producción**

- Paso 1: la estructura de la solución óptima (tiempo de ensamblado más corto)
  - Consideramos el chasis en la estación $s_{1,j}$.
  - Para **j=1** se eligió inicialmente la linea **1**.

Programación dinámica: **líneas de producción**

- Paso 1: la estructura de la solución óptima (tiempo de ensamblado más corto)
  - Consideramos el chasis en la estación $S_{1,j}$.
  - Para **j=1** se eligió inicialmente la linea **1**.
  - Para **j>1** hay dos opciones:

Programación dinámica: **líneas de producción**

- Paso 1: la estructura de la solución óptima (tiempo de ensamblado más corto)
  - Consideramos el chasis en la estación $S_{1,j}$.
  - Para **j=1** se eligió inicialmente la linea **1**.
  - Para **j>1** hay dos opciones:
    - el chasis viene de la estación $S_{1,j-1}$ y va directamente a la estación $S_{1,j}$ sin costo de transferencia.

Programación dinámica: **líneas de producción**

- Paso 1: la estructura de la solución óptima (tiempo de ensamblado más corto)
  - Consideramos el chasis en la estación $S_{1,j}$.
  - Para **j=1** se eligió inicialmente la linea **1**.
  - Para **j>1** hay dos opciones:
    - el chasis viene de la estación $S_{1,j-1}$ y va directamente a la estación $S_{1,j}$ sin costo de transferencia.
    - el chasis viene de la estación $S_{2,j-1}$ y va a la estación $S_{1,j}$ con un costo de transferencia $t_{2,j-1}$.

Programación dinámica: **líneas de producción**

Programación dinámica: **líneas de producción**

- En el primer caso la secuencia hasta $S_{1,j}$ pasando por $S_{1,j-1}$ tiene que ser óptima. (si hubiera una manera más rápida podríamos sustituirla para tener una manera más rápida: contradicción)

Programación dinámica: **líneas de producción**

- En el primer caso la secuencia hasta $S_{1,j}$ pasando por $S_{1,j-1}$ tiene que ser óptima. (si hubiera una manera más rápida podríamos sustituirla para tener una manera más rápida: contradicción)
- En el segundo caso tenemos lo mismo, la secuencia óptima hasta $S_{1,j}$ es pasando por la estación $S_{2,j-1}$.

Programación dinámica: **líneas de producción**

- En el primer caso la secuencia hasta $S_{1,j}$ pasando por $S_{1,j-1}$ tiene que ser óptima. (si hubiera una manera más rápida podríamos sustituirla para tener una manera más rápida: contradicción)
- En el segundo caso tenemos lo mismo, la secuencia óptima hasta $S_{1,j}$ es pasando por la estación $S_{2,j-1}$.
- Observamos que el chasis tuvo que haber tomado hasta ahora la ruta óptima desde el punto de partida.

Programación dinámica: **líneas de producción**

- En el primer caso la secuencia hasta $S_{1,j}$ pasando por $S_{1,j-1}$ tiene que ser óptima. (si hubiera una manera más rápida podríamos sustituirla para tener una manera más rápida: contradicción)
- En el segundo caso tenemos lo mismo, la secuencia óptima hasta $S_{1,j}$ es pasando por la estación $S_{2,j-1}$.

- Observamos que el chasis tuvo que haber tomado hasta ahora la ruta óptima desde el punto de partida.
- En general podemos decir que una solución óptima al problema (encontrar la ruta más rápida hasta la estación $S_{1,j}$) contiene la solución óptima a los subproblemas (encontrar la ruta más rápida hasta ($S_{1,j-1}$ o $S_{2,j-1}$) : subestructura óptima.

Programación dinámica: **líneas de producción**

Programación dinámica: **líneas de producción**

- La estructura de la solución es:

Programación dinámica: **líneas de producción**

- La estructura de la solución es:
  - para llegar más rápido a la estación $\mathbf{s_{1,j}}$ podemos:

Programación dinámica: **líneas de producción**

- La estructura de la solución es:
  - para llegar más rápido a la estación $S_{1,j}$ podemos:
    - llegar óptimamente a la estación $S_{1,j-1}$ y seguir la línea 1.

Programación dinámica: **líneas de producción**

- La estructura de la solución es:
  - para llegar más rápido a la estación $S_{1,j}$ podemos:
    - llegar óptimamente a la estación $S_{1,j-1}$ y seguir la línea 1.
    - llegar óptimamente a la estación $S_{2,j-1}$ y cambiar de línea.

Programación dinámica: **líneas de producción**

- La estructura de la solución es:
  - para llegar más rápido a la estación $S_{1,j}$ podemos:
    - llegar óptimamente a la estación $S_{1,j-1}$ y seguir la línea 1.
    - llegar óptimamente a la estación $S_{2,j-1}$ y cambiar de línea.
  - para llegar más rápido a la estación $S_{2,j}$ podemos:

Programación dinámica: **líneas de producción**

- La estructura de la solución es:
  - para llegar más rápido a la estación $S_{1,j}$ podemos:
    - llegar óptimamente a la estación $S_{1,j-1}$ y seguir la línea **1**.
    - llegar óptimamente a la estación $S_{2,j-1}$ y cambiar de línea.
  - para llegar más rápido a la estación $S_{2,j}$ podemos:
    - llegar óptimamente a la estación $S_{2,j-1}$ y seguir la línea **2**.

Programación dinámica: **líneas de producción**

- La estructura de la solución es:
  - para llegar más rápido a la estación $S_{1,j}$ podemos:
    - llegar óptimamente a la estación $S_{1,j-1}$ y seguir la línea **1**.
    - llegar óptimamente a la estación $S_{2,j-1}$ y cambiar de línea.
  - para llegar más rápido a la estación $S_{2,j}$ podemos:
    - llegar óptimamente a la estación $S_{2,j-1}$ y seguir la línea **2**.
    - llegar óptimamente a la estación $S_{1,j-1}$ y cambiar de línea.

Programación dinámica: **líneas de producción**

- La estructura de la solución es:
  - para llegar más rápido a la estación $S_{1,j}$ podemos:
    - llegar óptimamente a la estación $S_{1,j-1}$ y seguir la línea **1**.
    - llegar óptimamente a la estación $S_{2,j-1}$ y cambiar de línea.
  - para llegar más rápido a la estación $S_{2,j}$ podemos:
    - llegar óptimamente a la estación $S_{2,j-1}$ y seguir la línea **2**.
    - llegar óptimamente a la estación $S_{1,j-1}$ y cambiar de línea.
- nos queda entonces solo resolver el problema de llegar óptimamente a $S_{1,j-1}$ y $S_{2,j-1}$.

Programación dinámica: **líneas de producción**

Programación dinámica: **líneas de producción**

- Paso 2: encontrar una solución recursiva

Programación dinámica: **líneas de producción**

- Paso 2: encontrar una solución recursiva
  - definir el valor de la solución óptima recursivamente en términos de las soluciones óptimas a los subproblemas:

Programación dinámica: **líneas de producción**

- Paso 2: encontrar una solución recursiva
  - definir el valor de la solución óptima recursivamente en términos de las soluciones óptimas a los subproblemas:
    - encontrar el camino más rápido a la estación **j** de ambas líneas para **j=1,2,...,n**.

Programación dinámica: **líneas de producción**

- Paso 2: encontrar una solución recursiva
  - definir el valor de la solución óptima recursivamente en términos de las soluciones óptimas a los subproblemas:
    - encontrar el camino más rápido a la estación **j** de ambas líneas para **j=1,2,...,n**.
  - sea $\mathbf{f_i[j]}$ el tiempo óptimo para llevar un chasis del punto inicial a la estación $\mathbf{S_{i,j}}$.

Programación dinámica: **líneas de producción**

- Paso 2: encontrar una solución recursiva
  - definir el valor de la solución óptima recursivamente en términos de las soluciones óptimas a los subproblemas:
    - encontrar el camino más rápido a la estación **j** de ambas líneas para **j=1,2,...,n**.
  - sea $\mathbf{f_i[j]}$ el tiempo óptimo para llevar un chasis del punto inicial a la estación $\mathbf{S_{i,j}}$.
  - determinar el tiempo óptimo global para llevar el chasis a la salida de la fábrica, que llamaremos $\mathbf{f^*}$.

Programación dinámica: **líneas de producción**

- Paso 2: encontrar una solución recursiva
  - definir el valor de la solución óptima recursivamente en términos de las soluciones óptimas a los subproblemas:
    - encontrar el camino más rápido a la estación **j** de ambas líneas para **j=1,2,...,n**.
    - sea $\mathbf{f_i[j]}$ el tiempo óptimo para llevar un chasis del punto inicial a la estación $\mathbf{S_{i,j}}$.
    - determinar el tiempo óptimo global para llevar el chasis a la salida de la fábrica, que llamaremos $\mathbf{f^*}$.
- $$\mathbf{f^* = \min(f_1[n] + x_1, f_2[n]+x_2)}.$$

Programación dinámica: **líneas de producción**

Programación dinámica: **líneas de producción**

- Inicialización: $j=1$,

Programación dinámica: **líneas de producción**

- Inicialización: **j=1**,

- Terminación: **f* para las dos líneas,**

Programación dinámica: **líneas de producción**

- Inicialización: **j=1**,

- Terminación: **f* para las dos líneas,**

- **para j>1,**

Programación dinámica: **líneas de producción**

- Inicialización: **j=1**,

$$\begin{cases} f_1[1] &= e_1 + a_{1,1}, \\ f_2[1] &= e_2 + a_{2,1}. \end{cases}$$

- Terminación: **f* para las dos líneas,**

- **para j>1,**

Programación dinámica: líneas de producción

- Inicialización: **j=1**,

$$\begin{cases} f_1[1] &= e_1 + a_{1,1}, \\ f_2[1] &= e_2 + a_{2,1}. \end{cases}$$

- Terminación: **f* para las dos líneas,**

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

- **para j>1,**

Programación dinámica: líneas de producción

- Inicialización: **j=1**,

$$\begin{cases} f_1[1] &= e_1 + a_{1,1}, \\ f_2[1] &= e_2 + a_{2,1}. \end{cases}$$

- Terminación: **f* para las dos líneas,**

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

- **para j>1,**

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

Programación dinámica: líneas de producción

- Inicialización: **j=1**,

$$\begin{cases} f_1[1] &= e_1 + a_{1,1}, \\ f_2[1] &= e_2 + a_{2,1}. \end{cases}$$

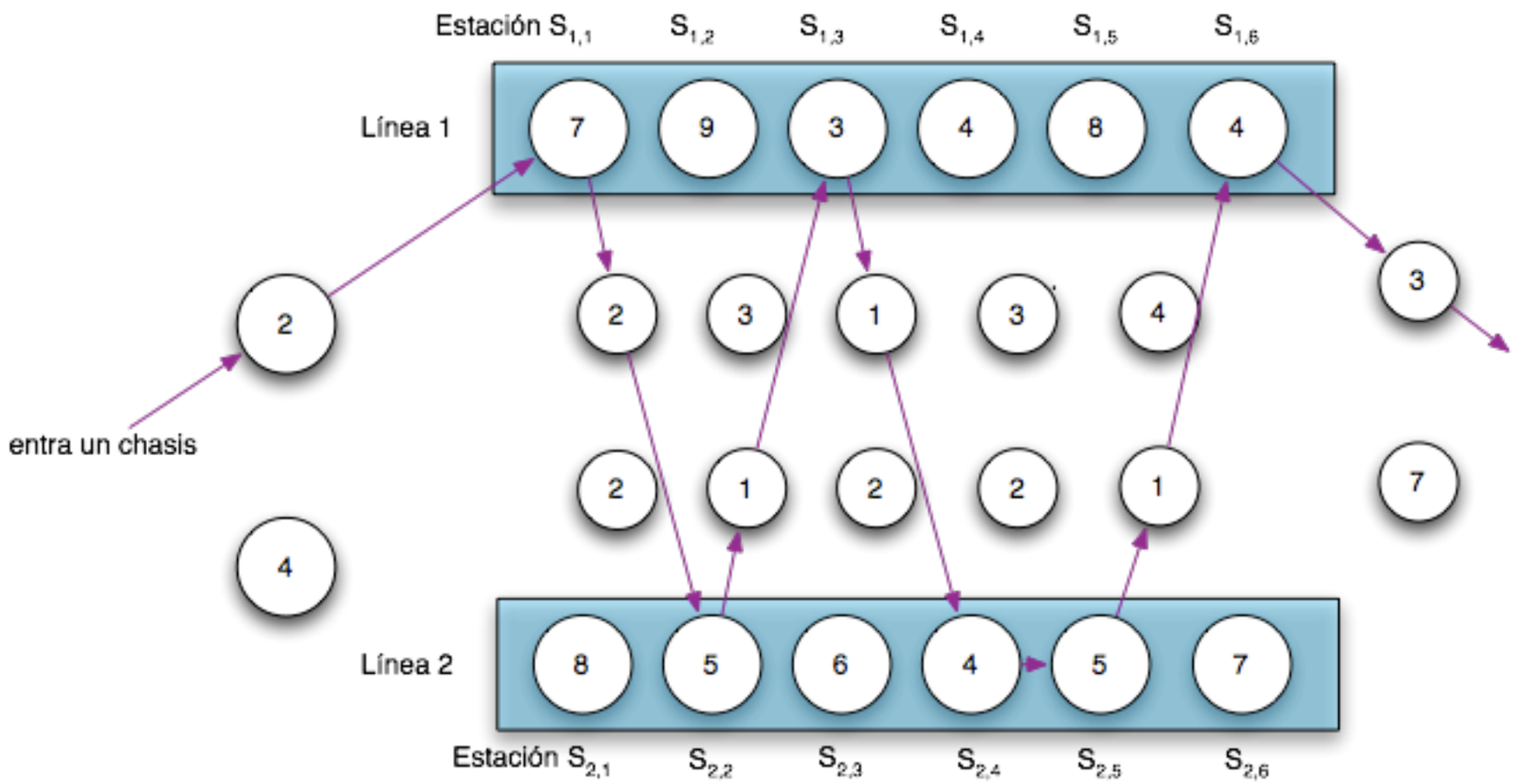
- Terminación: **f* para las dos líneas,**

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

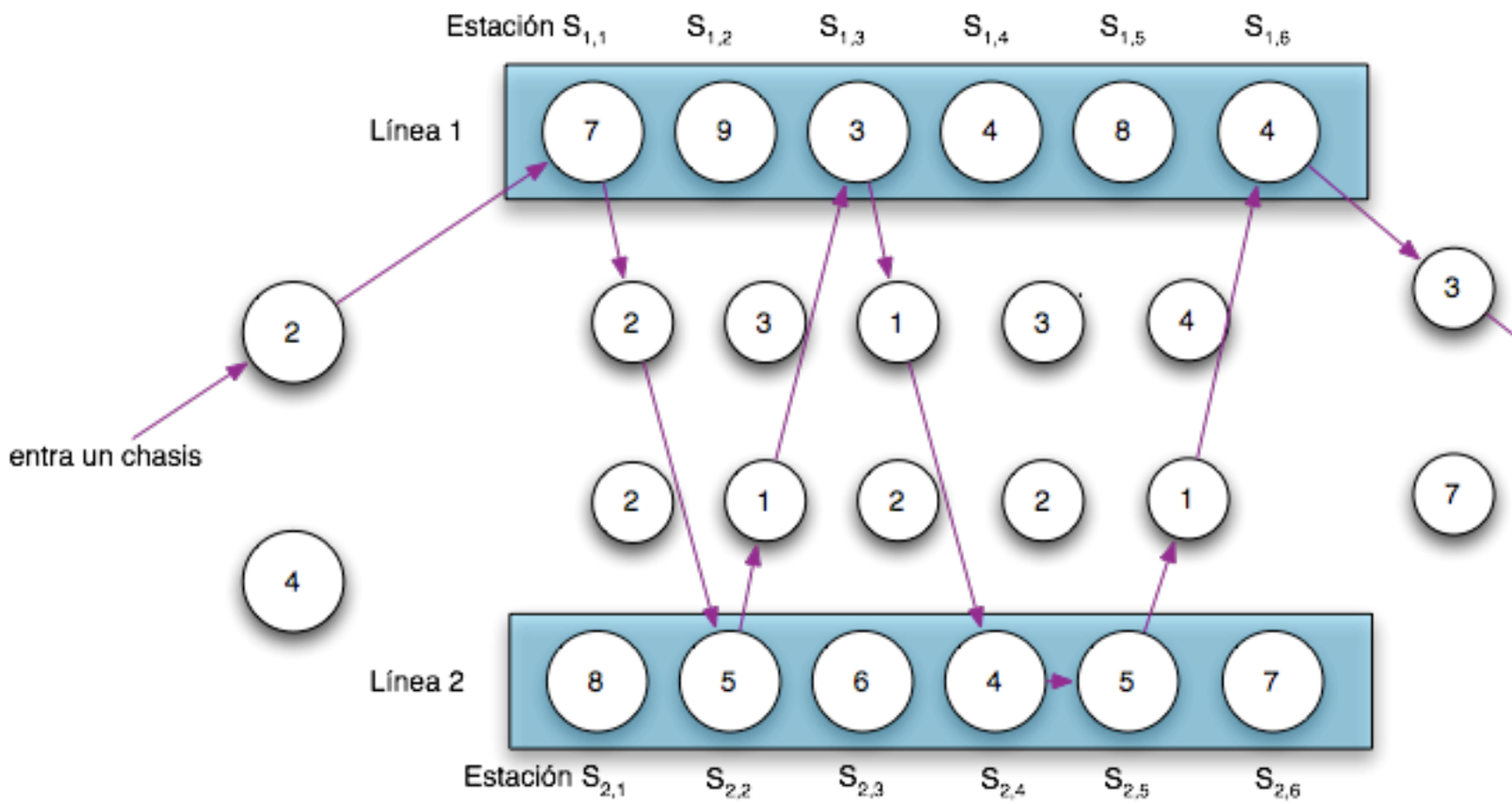
- **para j>1,**

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

Programación dinámica: **líneas de producción**

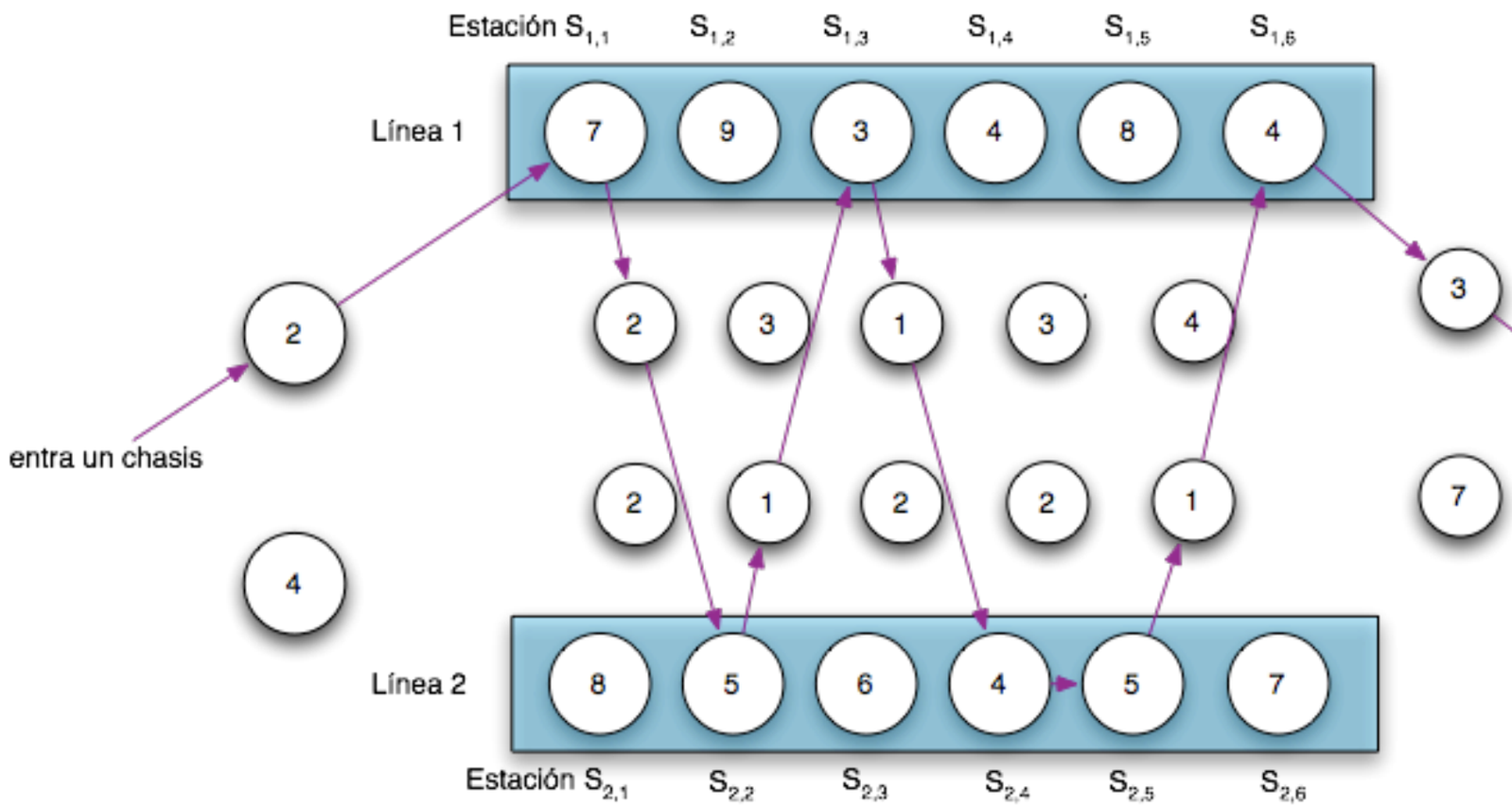


Programación dinámica: **líneas de producción**



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

Programación dinámica: **líneas de producción**

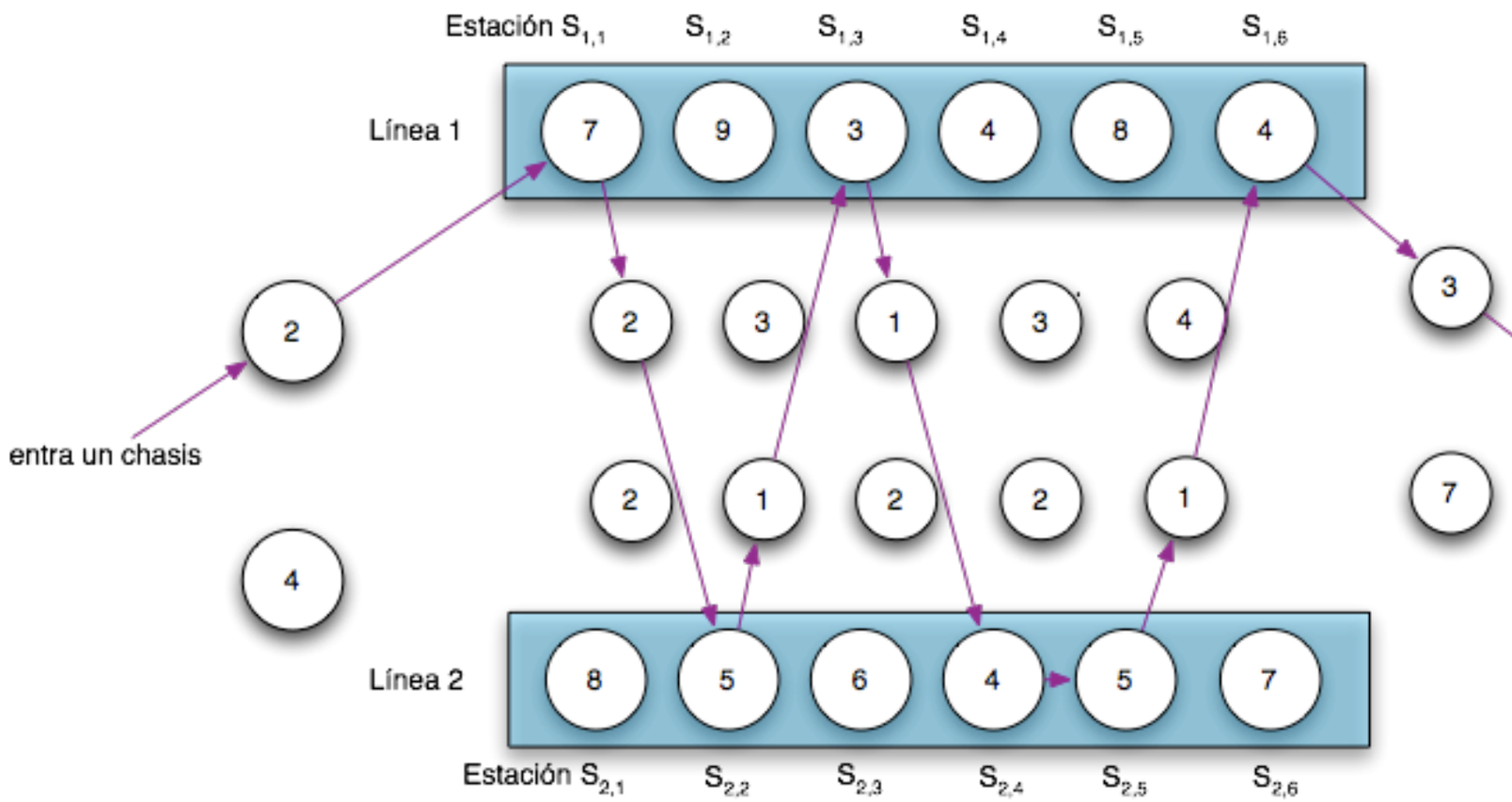


j	1	2	3	4	5	6
$f_1[n]$	9	18	20	24	32	35
$f_2[n]$	12	16	22	25	30	37

min(18,23)

min(17,16)

Programación dinámica: **líneas de producción**



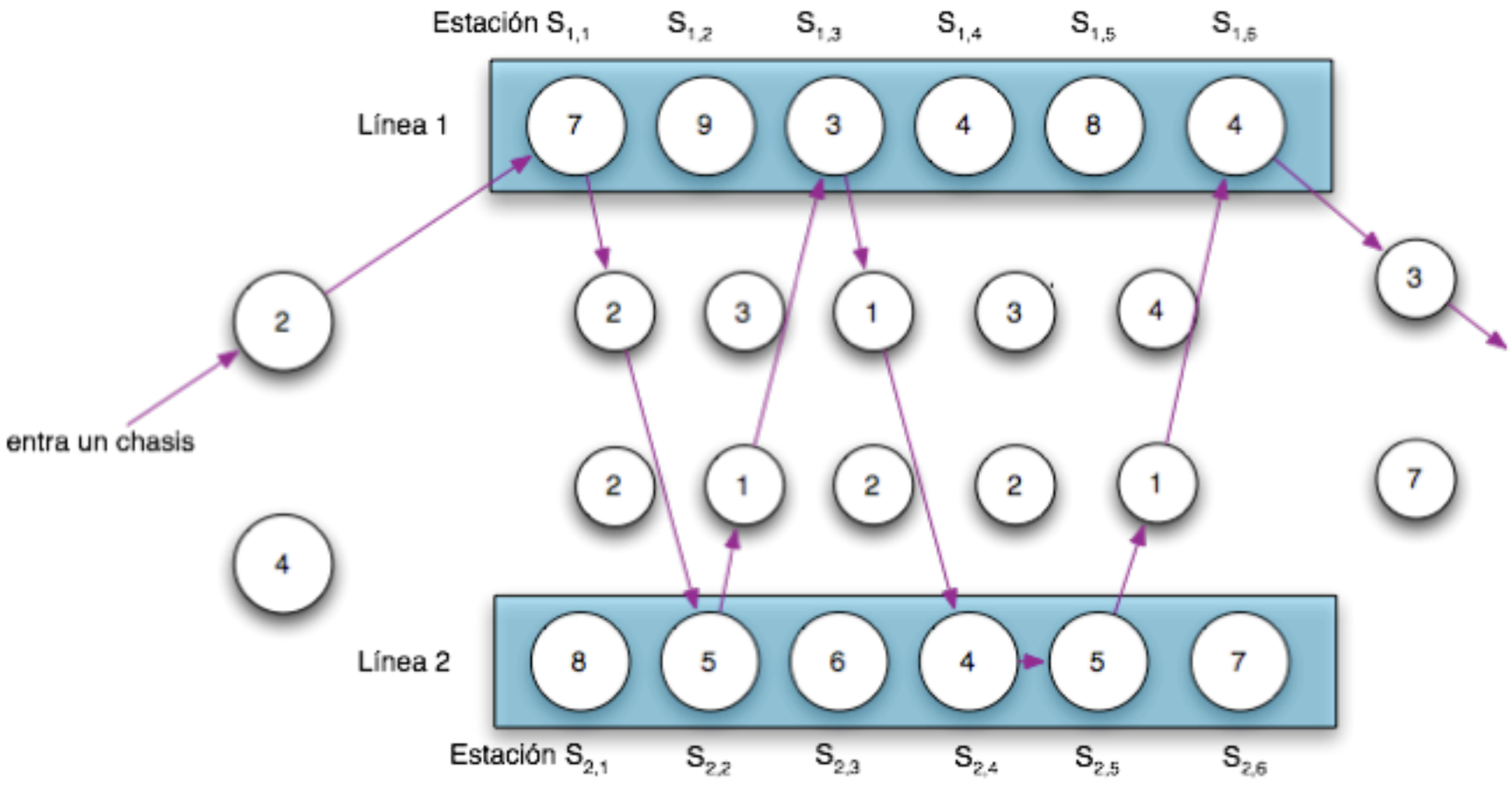
j	1	2	3	4	5	6
$f_1[n]$	9	18	20	24	32	35
$f_2[n]$	12	16	22	25	30	37

$f^* = 38$

min(18,23)

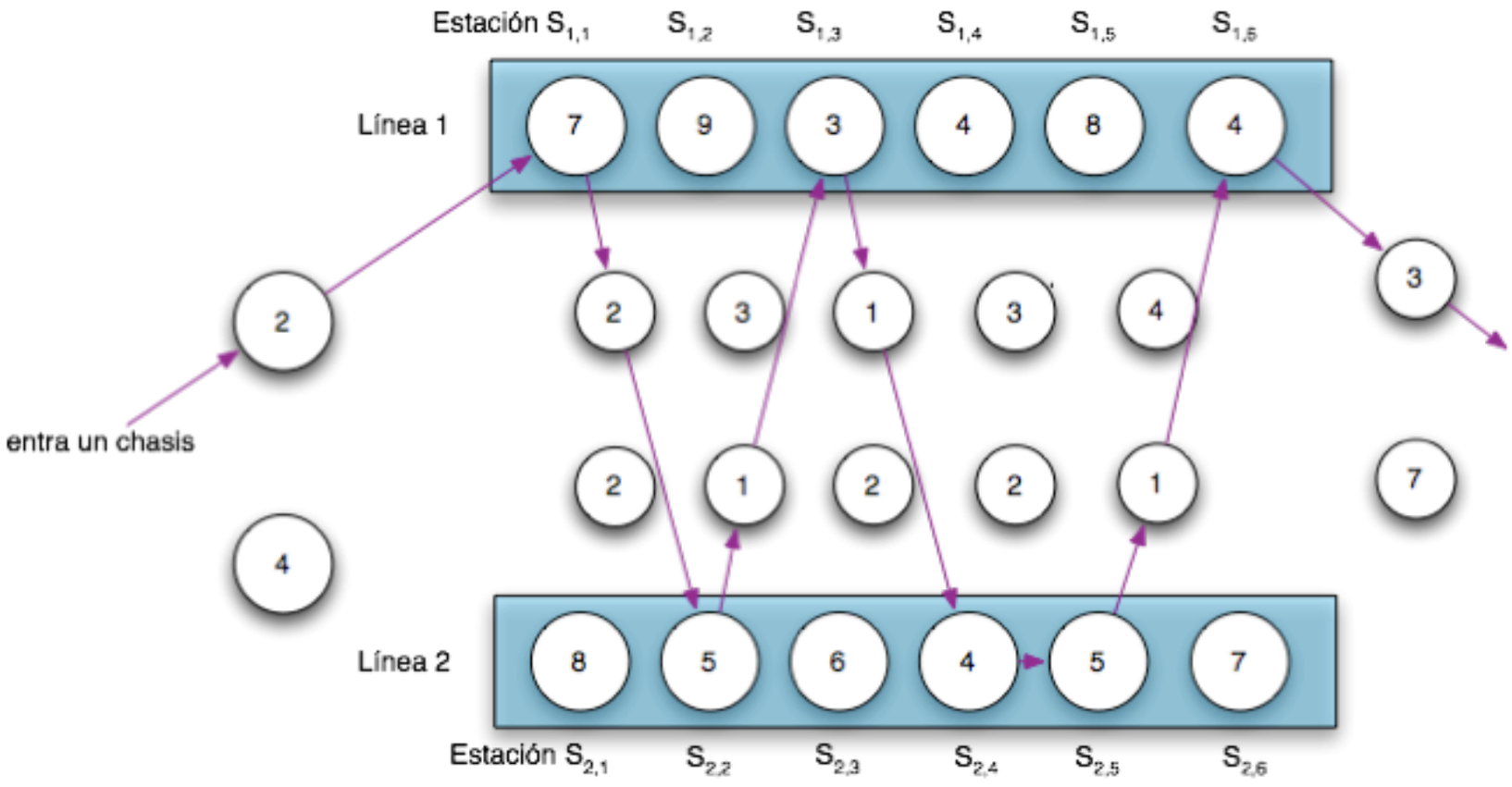
min(17,16)

Programación dinámica: **líneas de producción**



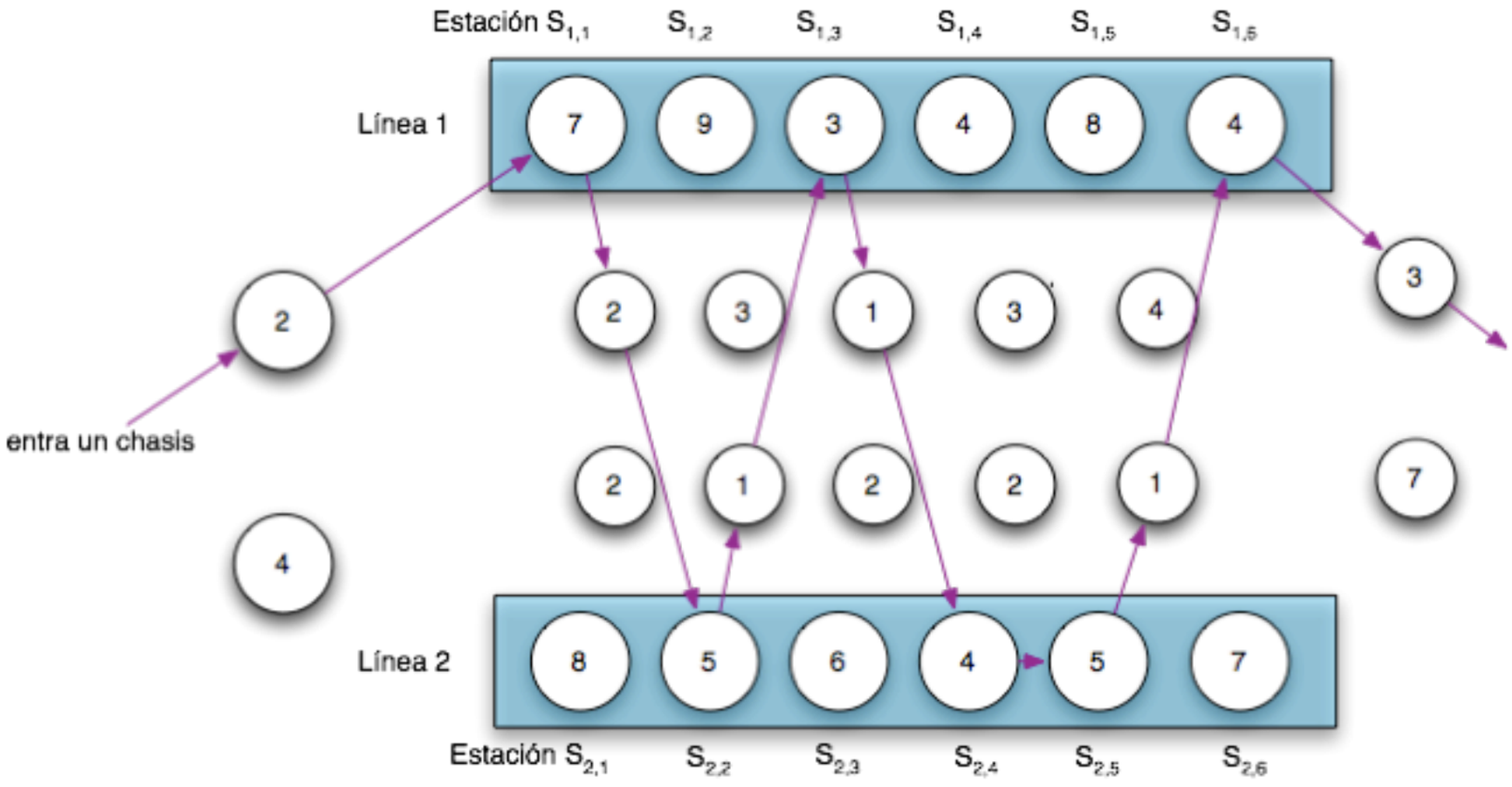
Programación dinámica: **líneas de producción**

- Para construir la solución optima definimos $\mathbf{l_i[j]}$ como la línea de ensamblado 1 o 2 más rápida para llegar a la estación $\mathbf{S_{i,j}}$ el arg min.



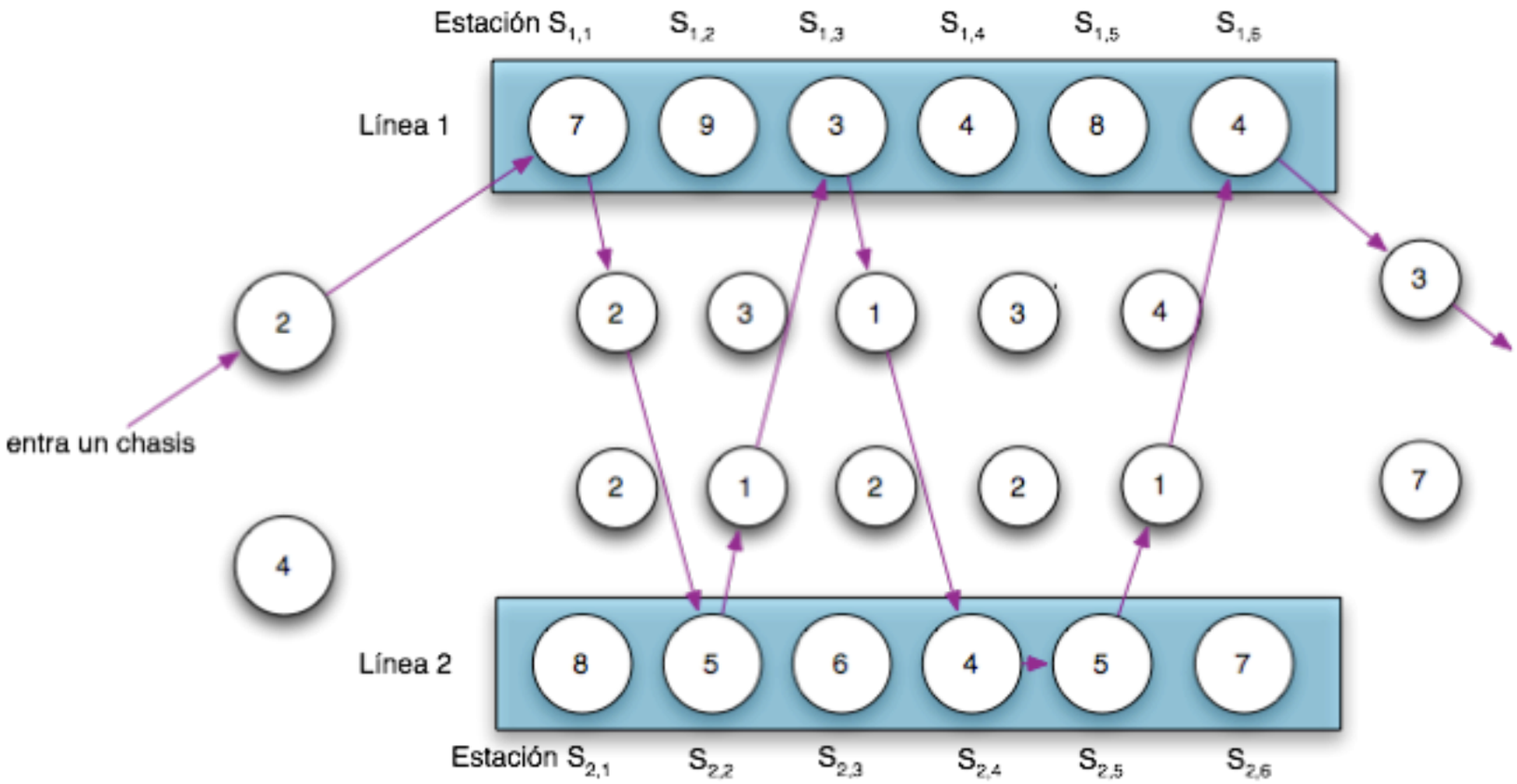
Programación dinámica: **líneas de producción**

- Para construir la solución optima definimos $\mathbf{l_i[j]}$ como la línea de ensamblado 1 o 2 más rápida para llegar a la estación $\mathbf{S_{i,j}}$ el arg min.
- Definimos de la misma forma $\mathbf{l^*}$ como la línea cuya estación *n* se usó como la más rápida para llegar a $\mathbf{S_{i,n}}$.



Programación dinámica: **líneas de producción**

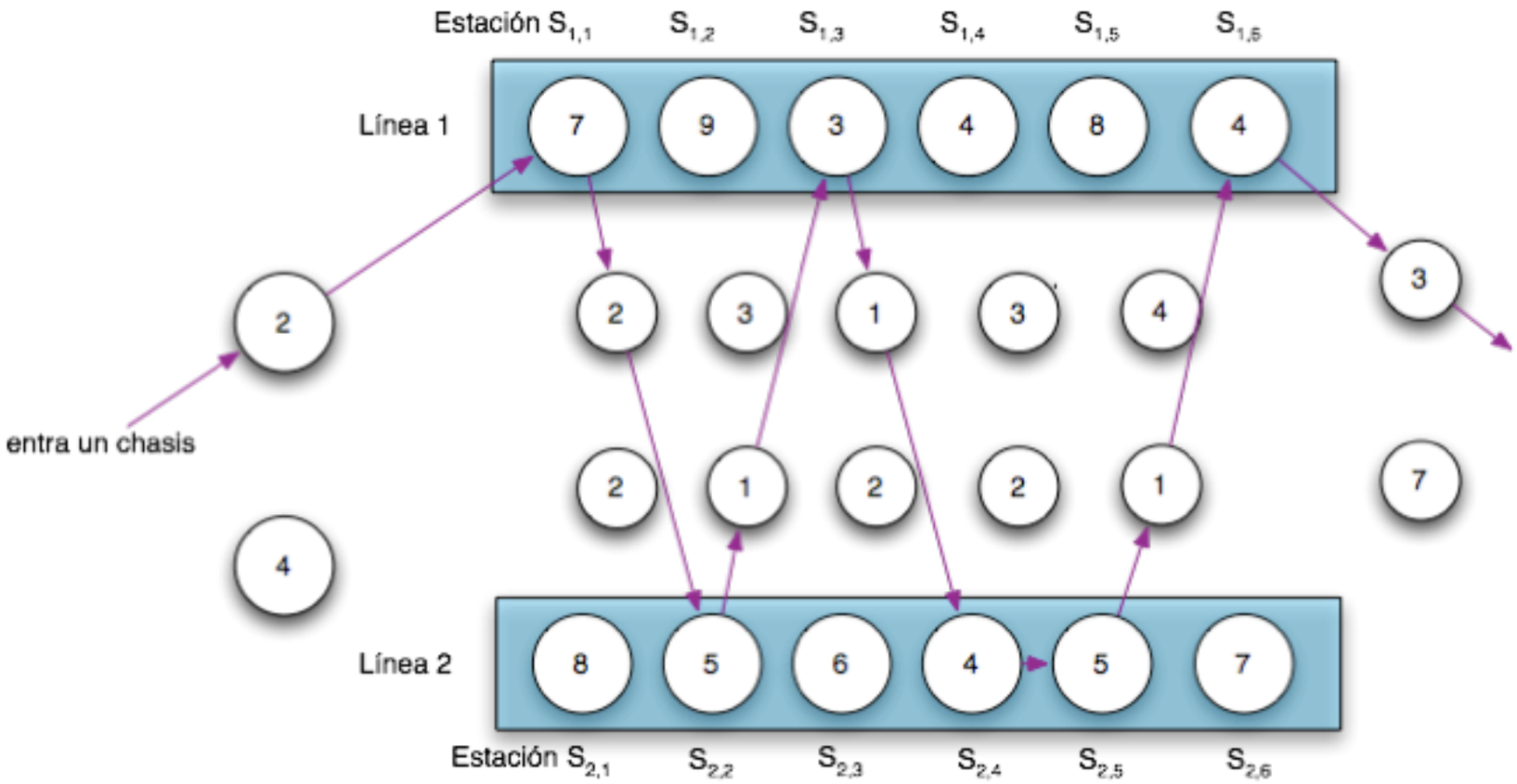
- Para construir la solución optima definimos $\mathbf{l_i[j]}$ como la línea de ensamblado 1 o 2 más rápida para llegar a la estación $\mathbf{S_{i,j}}$ el arg min.
- Definimos de la misma forma $\mathbf{l^*}$ como la línea cuya estación *n* se usó como la más rápida para llegar a $\mathbf{S_{i,n}}$.



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

Programación dinámica: **líneas de producción**

- Para construir la solución optima definimos $\mathbf{l_i[j]}$ como la línea de ensamblado 1 o 2 más rápida para llegar a la estación $\mathbf{S_{i,j}}$ el arg min.
- Definimos de la misma forma $\mathbf{l^*}$ como la línea cuya estación *n* se usó como la más rápida para llegar a $\mathbf{S_{i,n}}$.

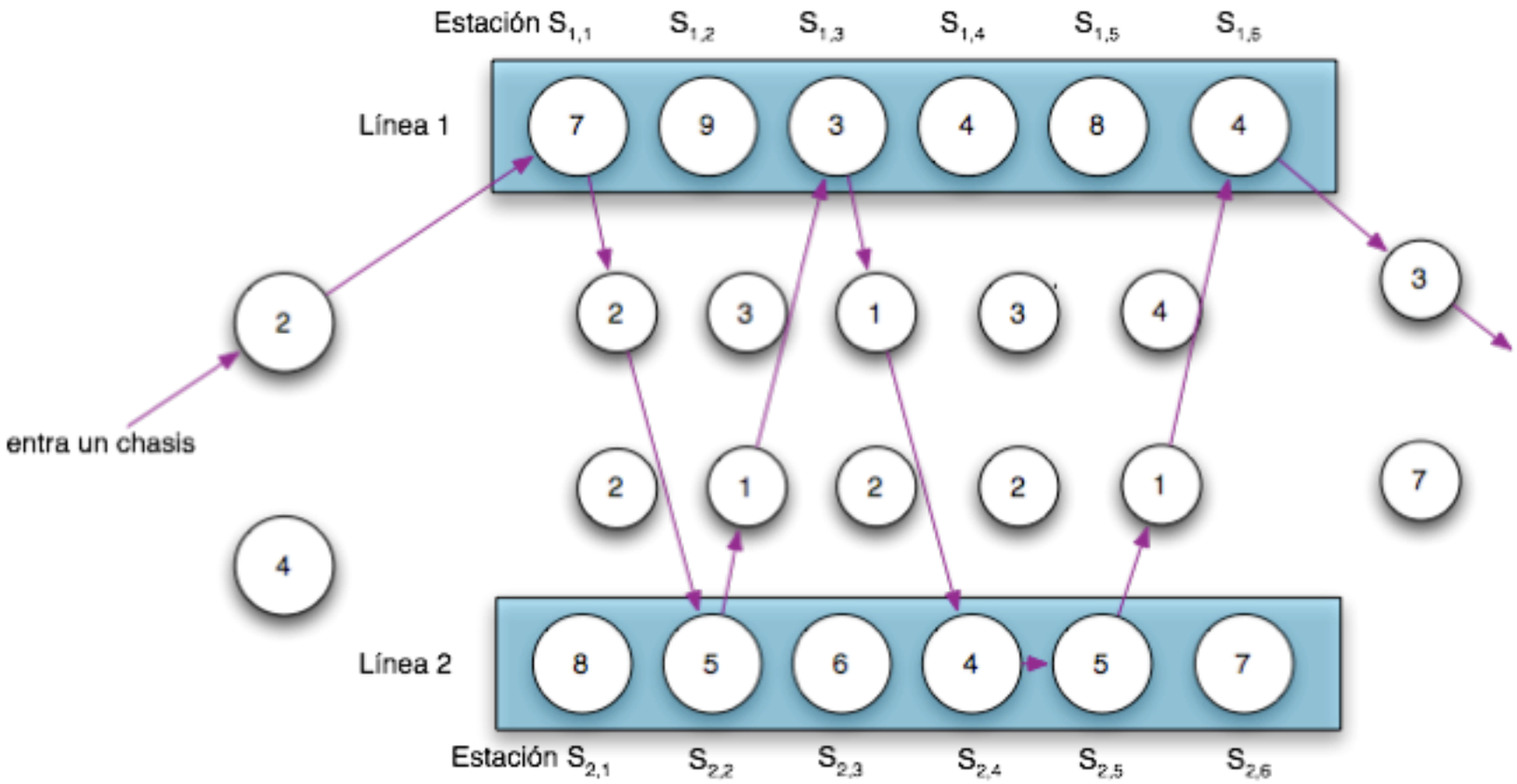


j	1	2	3	4	5	6
$f_1[n]$	9	18	20	24	32	35
$f_2[n]$	12	16	22	25	30	37

$f^* = 38$

Programación dinámica: **líneas de producción**

- Para construir la solución optima definimos $\mathbf{l_i[j]}$ como la línea de ensamblado 1 o 2 más rápida para llegar a la estación $\mathbf{S_{i,j}}$ el arg min.
- Definimos de la misma forma $\mathbf{l^*}$ como la línea cuya estación *n* se usó como la más rápida para llegar a $\mathbf{S_{i,n}}$.



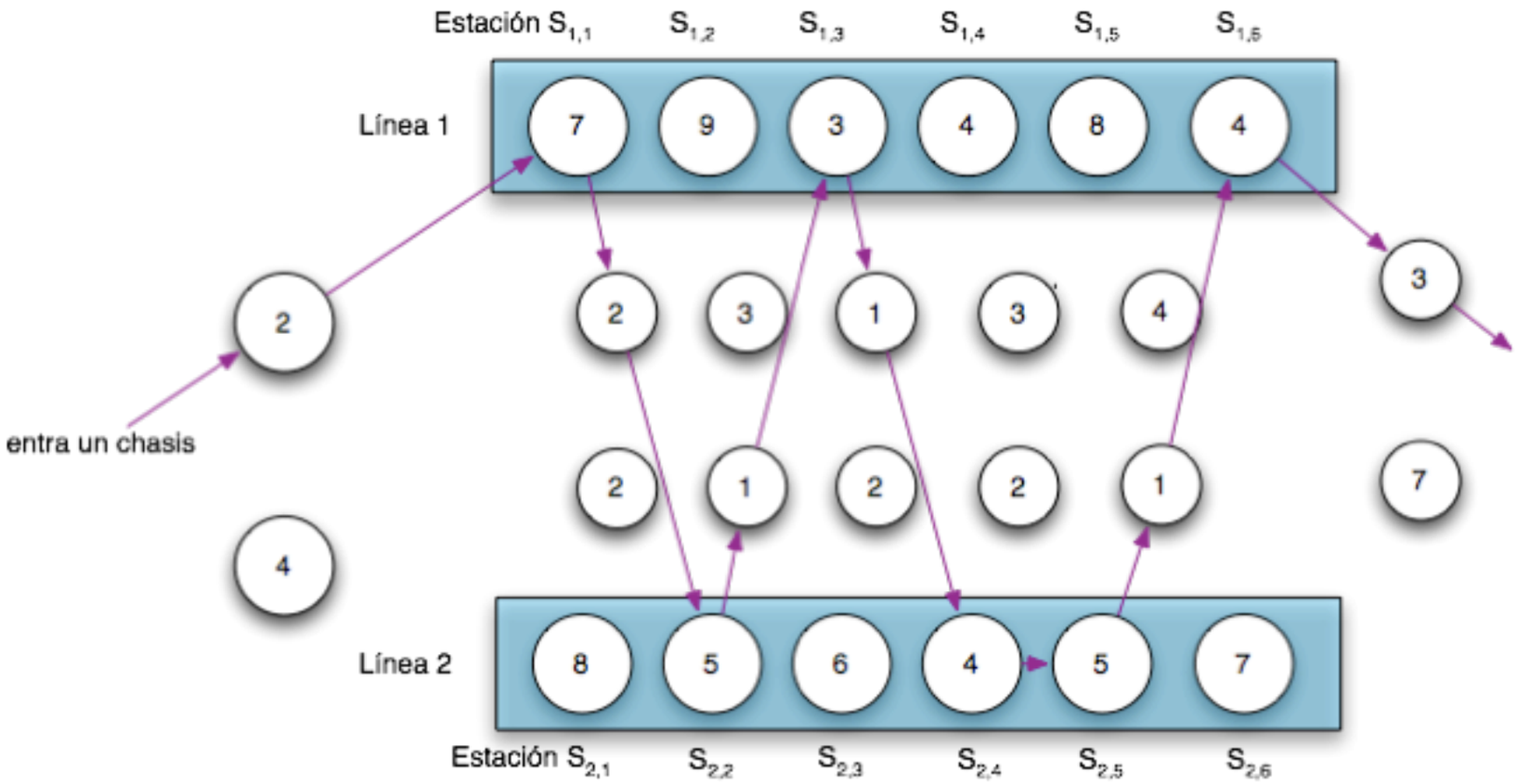
j	1	2	3	4	5	6
$f_1[n]$	9	18	20	24	32	35
$f_2[n]$	12	16	22	25	30	37

$f^* = 38$

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

Programación dinámica: **líneas de producción**

- Para construir la solución optima definimos $\mathbf{l_i[j]}$ como la línea de ensamblado 1 o 2 más rápida para llegar a la estación $\mathbf{S_{i,j}}$ el arg min.
- Definimos de la misma forma $\mathbf{l^*}$ como la línea cuya estación *n* se usó como la más rápida para llegar a $\mathbf{S_{i,n}}$.



j	1	2	3	4	5	6
$f_1[n]$	9	18	20	24	32	35
$f_2[n]$	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

$l^* = 1$

Ejemplo: línea de ensamblado

Ejemplo: línea de ensamblado

3. Cálculo de las líneas más rápidas (implementación)

Ejemplo: línea de ensamblado

3. Cálculo de las líneas más rápidas (implementación)

```
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.       do if f1[j-1]+a1,j ≤ f2[j-1] + t2,j-1 + a1,j
5.          then 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.         then 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
```

Ejemplo: línea de ensamblado

3. Cálculo de las líneas más rápidas (implementación)

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

1. $f_1[1] \leftarrow e_1 + a_{1,1}$
2. $f_2[1] \leftarrow e_2 + a_{2,1}$
3. for $j \leftarrow 2$ to n
4. do if $f_1[j-1]+a_{1,j} \leq f_2[j-1] + t_{2,j-1} + a_{1,j}$
5. then $f_1[j] \leftarrow f_1[j-1]+a_{1,j}$
6. $l_1[j] \leftarrow 1$
7. else $f_1[j] \leftarrow f_2[j-1] + t_{2,j-1} + a_{1,j}$
8. $l_1[j] \leftarrow 2$
9. if $f_2[j-1] + a_{2,j} \leq f_1[j-1]+t_{1,j-1}+a_{2,j}$
10. then $f_2[j] \leftarrow f_2[j-1]+a_{2,j}$
11. $l_2[j] \leftarrow 2$
12. else $f_2[j] \leftarrow f_1[j-1] + t_{1,j-1} + a_{2,j}$
13. $l_2[j] \leftarrow 1$
14. if $f_1[n] + x_1 \leq f_2[n] + x_2$
15. then $f^* \leftarrow f_1[n]+x_1$
16. $l^* \leftarrow 1$
17. else $f^* \leftarrow f_2[n]+x_2$
18. $l^* \leftarrow 2$

Ejemplo: línea de ensamblado

Ejemplo: línea de ensamblado

4. Construcción de la vía más rápida en la fábrica.

Ejemplo: línea de ensamblado

4. Construcción de la vía más rápida en la fábrica.

```
PRINT-STATIONS ( l, l*, n)

14. i ← l*
1. print “línea” i “,estación” n
2. for j ← n downto 2
3.     do i ← l_i[j]
4.           print “línea” i “,estación” j-1
```

Ejemplo: línea de ensamblado

4. Construcción de la vía más rápida en la fábrica.

```
PRINT-STATIONS ( l, l*, n)

14. i ← l*
1. print "línea" i ",estación" n
2. for j ← n downto 2
3.     do i ← l_i[j]
4.           print "línea" i ",estación" j-1
```

línea 1, estación 6
línea 2, estación 5
línea 2, estación 4
línea 1, estación 3
línea 2, estación 2
línea 1, estación 1