Análise de Algoritmos

Parte destes slides são adaptações de slides

do Prof. Paulo Feofiloff e do Prof. José Coelho de Pina.

Programação dinâmica

CLRS cap 15

= “recursão–com–tabela”
= transformação inteligente de recursão em iteração

Números de Fibonacci

$$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2}$$

n	0	1	2	3	4	5	6	7	8	9
F_n	0	1	1	2	3	5	8	13	21	34

Números de Fibonacci

$$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2}$$

n	0	1	2	3	4	5	6	7	8	9
F_n	0	1	1	2	3	5	8	13	21	34

Algoritmo recursivo para F_n:

```
FIBO-REC (n)
1    se n ≤ 1
2          então devolva n
3          senão a ← FIBO-REC (n − 1)
4                b ← FIBO-REC (n − 2)
5                devolva a + b
```

Consumo de tempo

```
FIBO-REC (n)
1    se n ≤ 1
2          então devolva n
3          senão a ← FIBO-REC (n − 1)
4                b ← FIBO-REC (n − 2)
5                devolva a + b
```

Tempo em segundos:

n	16	32	40	41	42	43	44	45	47
tempo	0.002	0.06	2.91	4.71	7.62	12.37	19.94	32.37	84.50

$F_{47} = 2971215073$

Consumo de tempo

```
FIBO-REC (n)
1    se n ≤ 1
2          então devolva n
3          senão a ← FIBO-REC (n − 1)
4                b ← FIBO-REC (n − 2)
5                devolva a+b
```

$T(n) :=$ número de somas feitas por FIBO-REC (n)

linha	número de somas
1-2	$= 0$
3	$= T(n-1)$
4	$= T(n-2)$
5	$= 1$
$T(n)$	$= T(n-1) + T(n-2) + 1$

Recorrência

$$T(0) = 0$$

$$T(1) = 0$$

$$T(n) = T(n-1) + T(n-2) + 1 \text{ para } n = 2, 3, \ldots$$

A que classe Ω pertence $T(n)$?
A que classe O pertence $T(n)$?

Recorrência

$$T(0) = 0$$

$$T(1) = 0$$

$$T(n) = T(n-1) + T(n-2) + 1 \text{ para } n = 2, 3, \ldots$$

A que classe Ω pertence $T(n)$?
A que classe O pertence $T(n)$?

Solução: $T(n) > (3/2)^n$ para $n \geq 6$.

n	0	1	2	3	4	5	6	7	8	9
T_n	0	0	1	2	4	7	12	20	33	54
$(3/2)^n$	1	1.5	2.25	3.38	5.06	7.59	11.39	17.09	25.63	38.44

Recorrência

Prova: $T(6) = 12 > 11.40 > (3/2)^6$ e $T(7) = 20 > 18 > (3/2)^7$.
Se $n \geq 8$, então

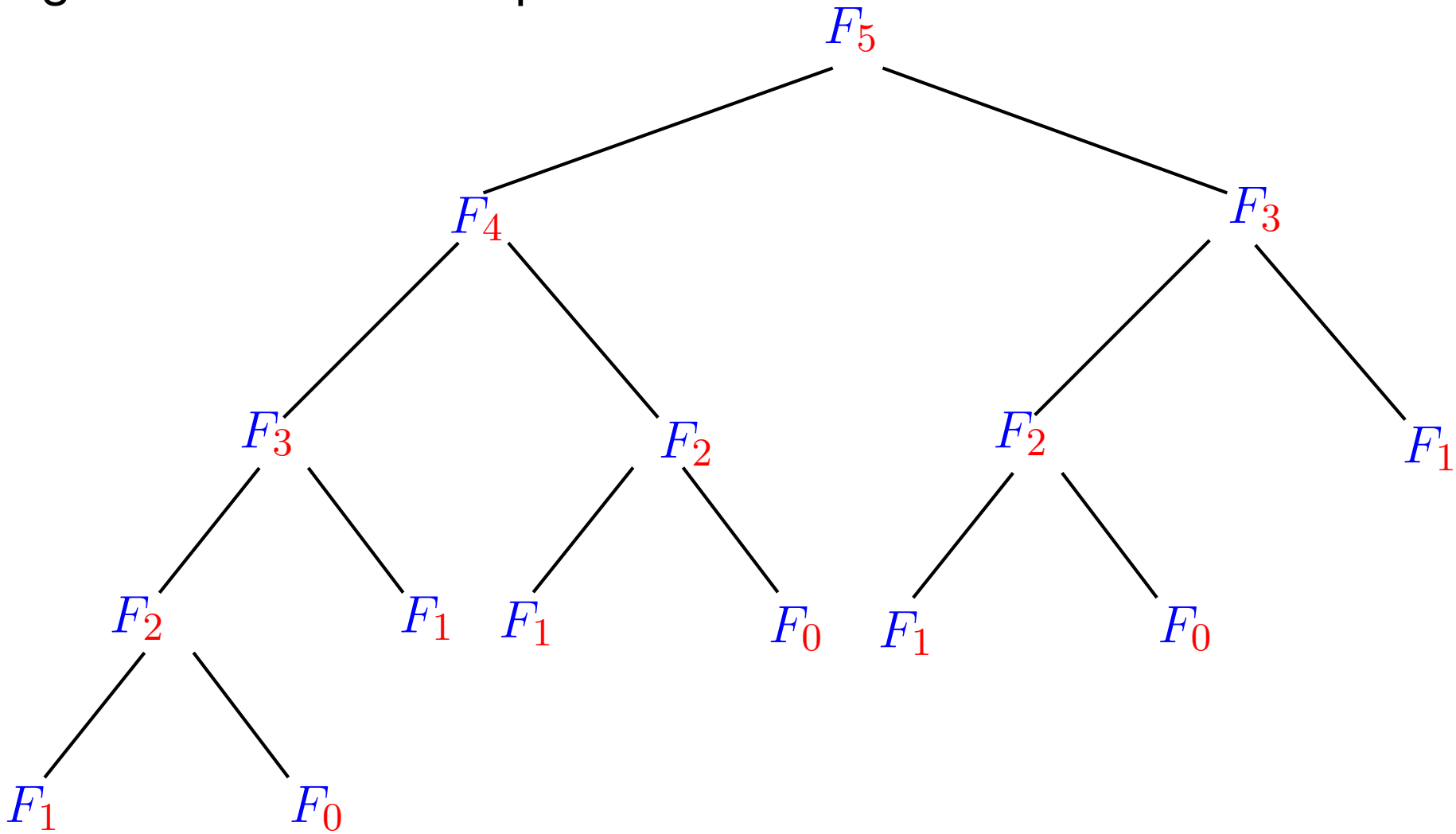
$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&\overset{\text{hi}}{>} (3/2)^{n-1} + (3/2)^{n-2} + 1 \\
&= (3/2+1)\,(3/2)^{n-2} + 1 \\
&> (5/2)\,(3/2)^{n-2} \\
&> (9/4)\,(3/2)^{n-2} \\
&= (3/2)^2 (3/2)^{n-2} \\
&= (3/2)^n\,.
\end{aligned}
$$

Logo, $T(n)$ é $\Omega((3/2)^n)$. Verifique que $T(n)$ é $\mathrm{O}(2^n)$.

Consumo de tempo

Consumo de tempo é exponencial.

Algoritmo resolve subproblemas muitas vezes.



Resolve subproblemas muitas vezes

```
FIBO-REC(5)
  FIBO-REC(4)
    FIBO-REC(3)
      FIBO-REC(2)
        FIBO-REC(1)
        FIBO-REC(0)
      FIBO-REC(1)
    FIBO-REC(2)
      FIBO-REC(1)
      FIBO-REC(0)
  FIBO-REC(3)
    FIBO-REC(2)
      FIBO-REC(1)
      FIBO-REC(0)
    FIBO-REC(1)
```

FIBO-REC(5) = 5

Resolve subproblemas muitas vezes

```
FIBO-REC(8)
  FIBO-REC(7)
    FIBO-REC(6)
      FIBO-REC(5)
        FIBO-REC(4)
          FIBO-REC(3)
            FIBO-REC(2)
              FIBO-REC(1)
              FIBO-REC(0)
            FIBO-REC(1)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
      FIBO-REC(4)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
    FIBO-REC(5)
      FIBO-REC(4)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
      FIBO-REC(3)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
        FIBO-REC(1)
  FIBO-REC(6)
    FIBO-REC(5)
      FIBO-REC(4)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
      FIBO-REC(3)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
        FIBO-REC(1)
    FIBO-REC(4)
      FIBO-REC(3)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
        FIBO-REC(1)
      FIBO-REC(2)
        FIBO-REC(1)
        FIBO-REC(0)
```

Programação dinâmica

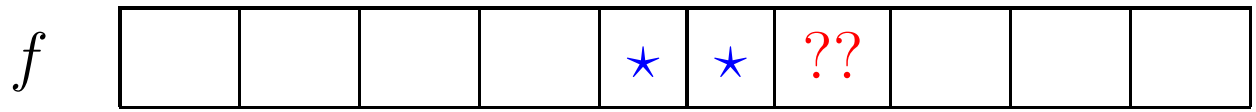
"Dynamic programming is a fancy name for divide-and-conquer with a table. Instead of solving subproblems recursively, solve them sequentially and store their solutions in a table. The trick is to solve them in the right order so that whenever the solution to a subproblem is needed, it is already available in the table. Dynamic programming is particularly useful on problems for which divide-and-conquer appears to yield an exponential number of subproblems, but there are really only a small number of subproblems repeated exponentially often. In this case, it makes sense to compute each solution the first time and store it away in a table for later use, instead of recomputing it recursively every time it is needed."

I. Parberry, *Problems on Algorithms*, Prentice Hall, 1995.

Algoritmo de programação dinâmica

```
FIBO (n)
1   f[0] ← 0
2   f[1] ← 1
3   para i ← 2 até n faça
4         f[i] ← f[i − 1] + f[i − 2]
5   devolva f[n]
```

Note a tabela $f[0\,..\,n-1]$.



Consumo de tempo (e de espaço) é $\Theta(n)$.

Algoritmo de programação dinâmica

Versão com economia de espaço.

```
FIBO (n)
0    se n = 0 então devolva 0
1    f_ant ← 0
2    f_atual ← 1
3    para i ← 2 até n faça
4         f_prox ← f_atual + f_ant
5         f_ant ← f_atual
6         f_atual ← f_prox
7    devolva f_atual
```

Algoritmo de programação dinâmica

Versão com economia de espaço.

```
FIBO (n)
0    se n = 0 então devolva 0
1    f_ant ← 0
2    f_atual ← 1
3    para i ← 2 até n faça
4         f_prox ← f_atual + f_ant
5         f_ant ← f_atual
6         f_atual ← f_prox
7    devolva f_atual
```

Consumo de tempo é $\Theta(n)$.

Consumo de espaço é $\Theta(1)$.

Versão recursiva eficiente

```
MEMOIZED-FIBO (f, n)
1   para i ← 0 até n faça
2         f[i] ← −1
3   devolva LOOKUP-FIBO (f, n)

LOOKUP-FIBO (f, n)
1   se f[n] ≥ 0
2         então devolva f[n]
3   se n ≤ 1
4         então f[n] ← n
5         senão f[n] ← LOOKUP-FIBO(f, n − 1)
                     + LOOKUP-FIBO(f, n − 2)
6   devolva f[n]
```

Não recalcula valores de f.

Linhas de produção

Duas linhas de produção para o mesmo produto, cada uma com n máquinas.

Linhas de produção

Duas linhas de produção para o mesmo produto, cada uma com n máquinas.

i-ésima máquina faz a mesma tarefa nas duas linhas, possivelmente com velocidades diferentes.

Linhas de produção

Duas linhas de produção para o mesmo produto, cada uma com n máquinas.

i-ésima máquina faz a mesma tarefa nas duas linhas, possivelmente com velocidades diferentes.

Tempo de set-up se um item muda duma linha para outra.

Linhas de produção

Duas linhas de produção para o mesmo produto,
cada uma com n máquinas.

i-ésima máquina faz a mesma tarefa nas duas linhas,
possivelmente com velocidades diferentes.

Tempo de set-up se um item muda duma linha para outra.

Para momentos de urgência,
qual é o caminho mais rápido de produção?

Linhas de produção

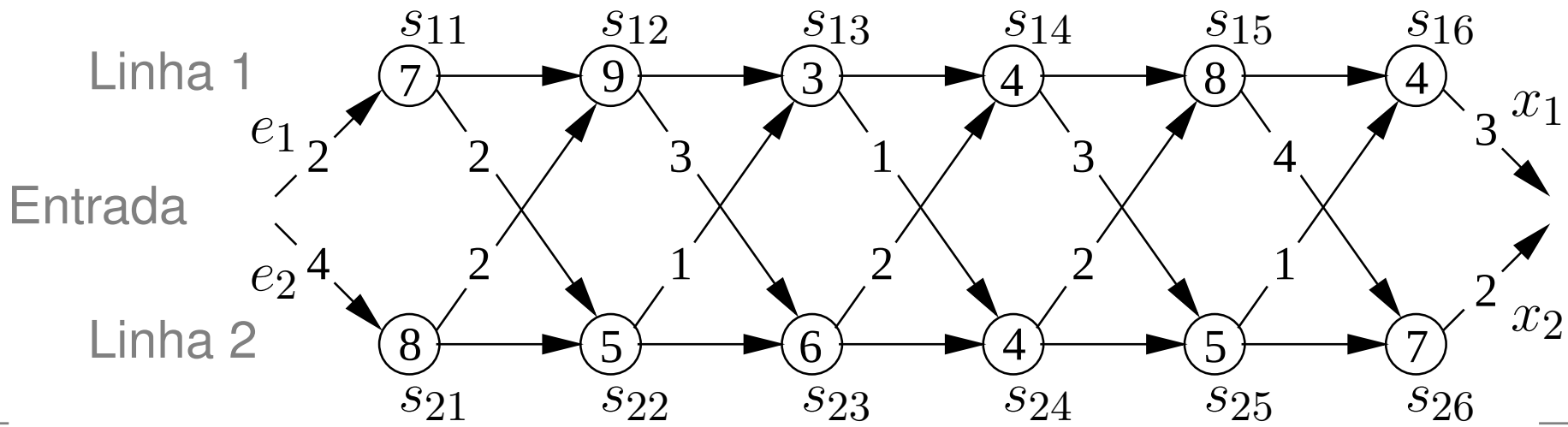
Duas linhas de produção para o mesmo produto, cada uma com n máquinas.

i-ésima máquina faz a mesma tarefa nas duas linhas, possivelmente com velocidades diferentes.

Tempo de set-up se um item muda duma linha para outra.

Para momentos de urgência,

qual é o caminho mais rápido de produção?



Linhas de produção

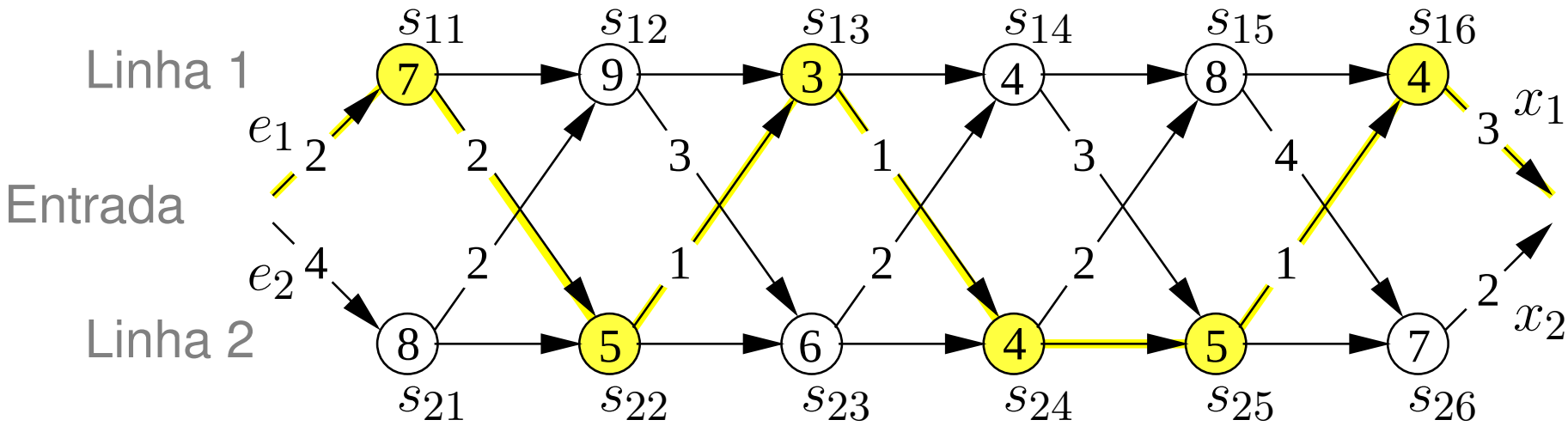
Duas linhas de produção, cada uma com n máquinas.

i-ésima máquina faz a mesma tarefa nas duas, possivelmente com velocidades diferentes.

Tempo de set-up se um item muda duma linha para outra.

Para momentos de urgência,

qual é o caminho mais rápido de produção?



Linhas de produção

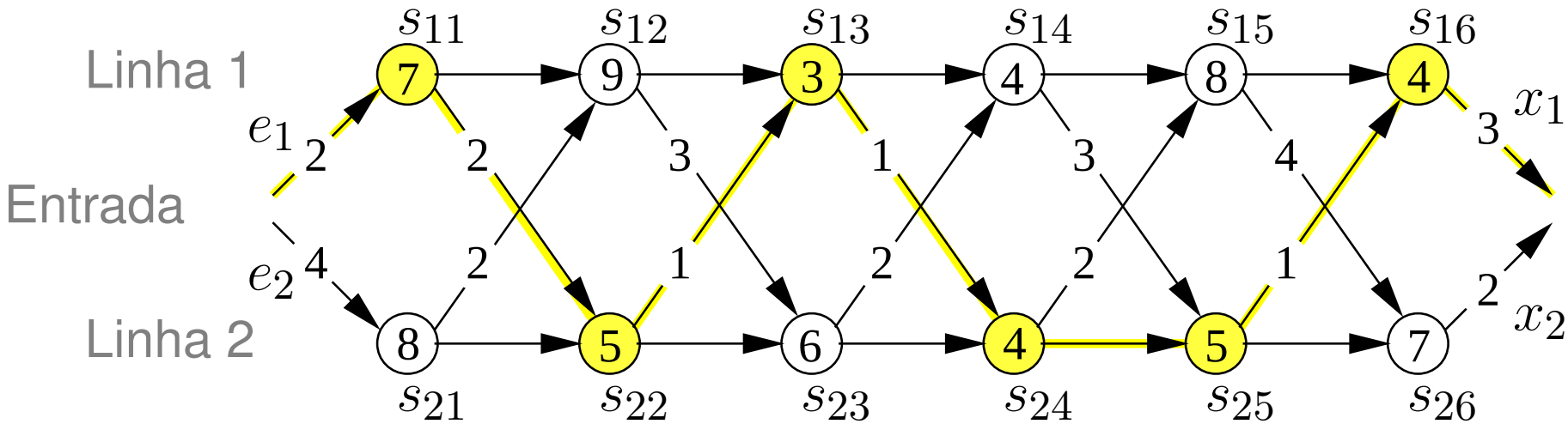
Duas linhas de produção, cada uma com n máquinas.

i-ésima máquina faz a mesma tarefa nas duas, possivelmente com velocidades diferentes.

Tempo de set-up se um item muda duma linha para outra.

Para momentos de urgência,

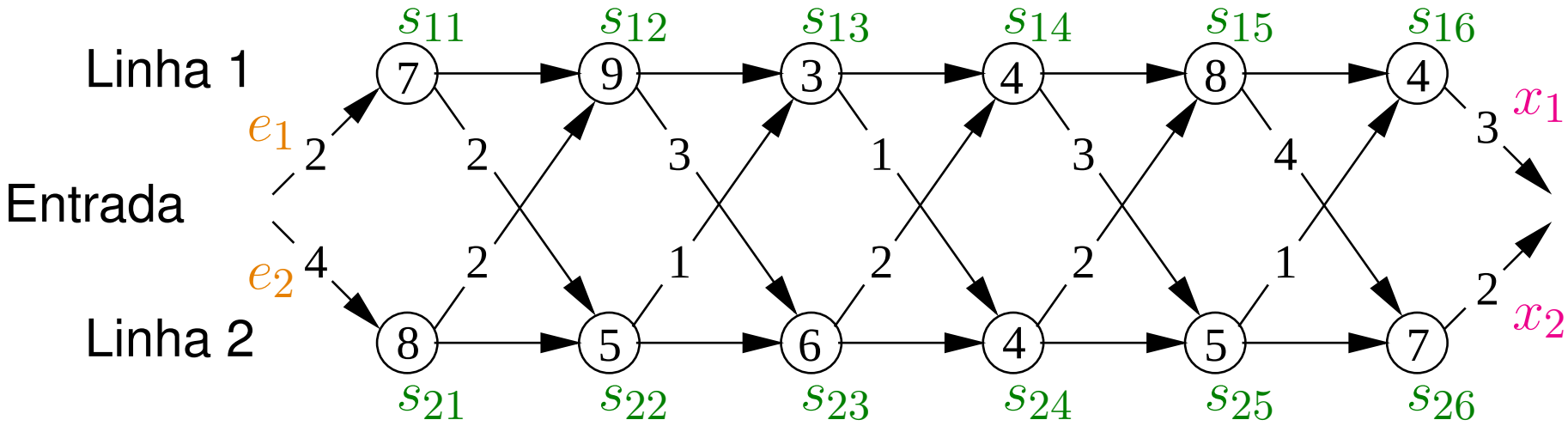
qual é o caminho mais rápido de produção?



Tempo de percurso do caminho amarelo:

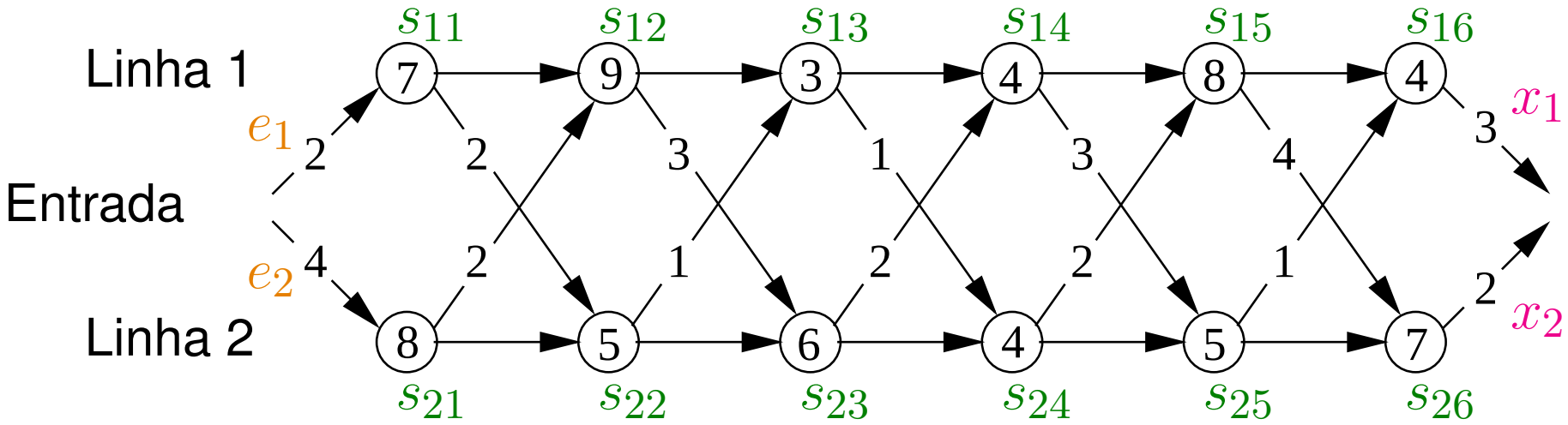
$$2 + 7 + 2 + 5 + 1 + 3 + 1 + 4 + 5 + 1 + 4 + 3 = 38.$$

Notação



s_{ij}: tempo de execução da j-ésima máquina da linha i

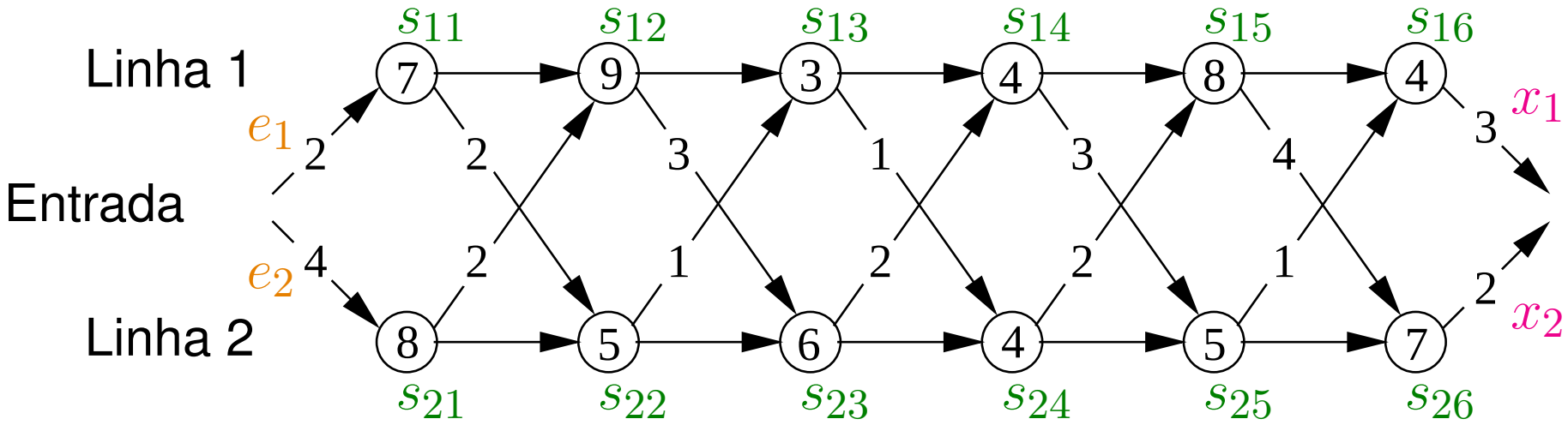
Notação



s_{ij}: tempo de execução da j-ésima máquina da linha i

e_i: tempo de set-up da primeira máquina da linha i

Notação

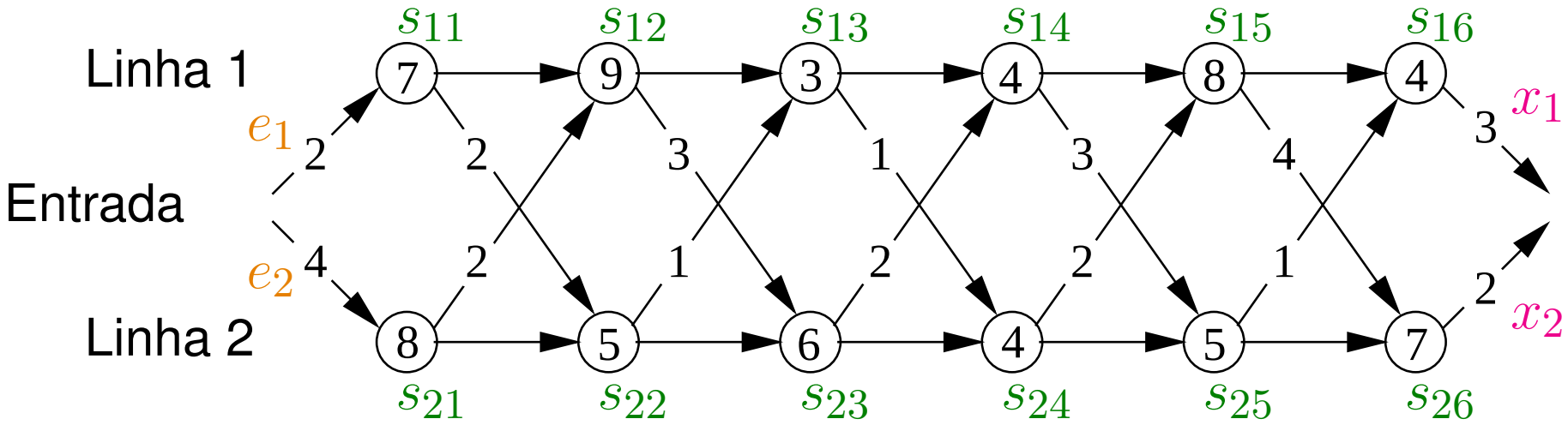


s_{ij}: tempo de execução da j-ésima máquina da linha i

e_i: tempo de set-up da primeira máquina da linha i

x_i: tempo de conclusão da última máquina da linha i

Notação



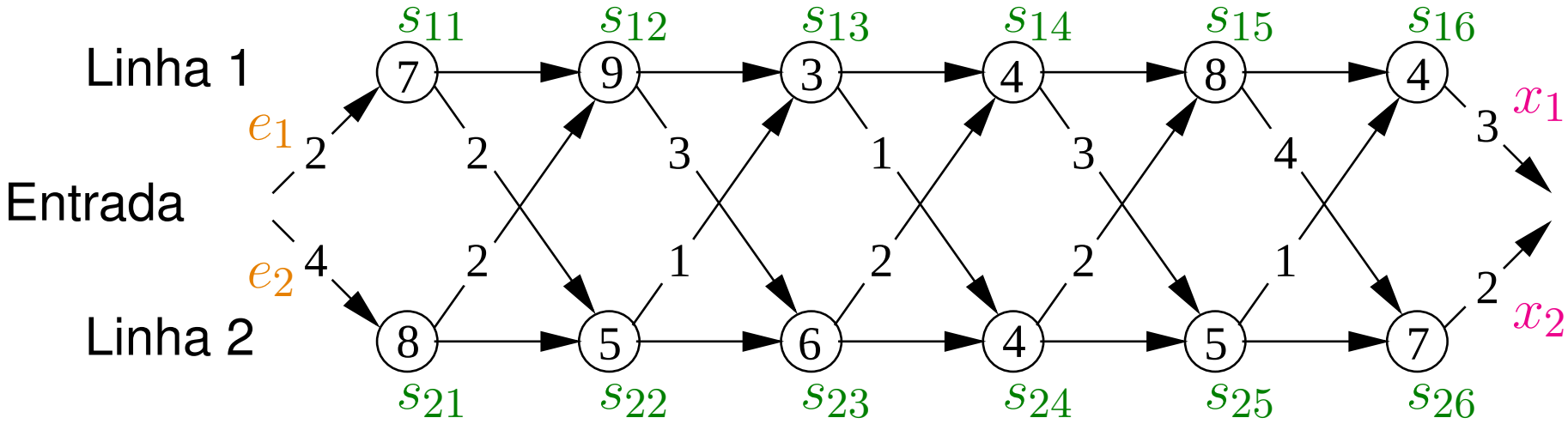
s_{ij}: tempo de execução da j-ésima máquina da linha i

e_i: tempo de set-up da primeira máquina da linha i

x_i: tempo de conclusão da última máquina da linha i

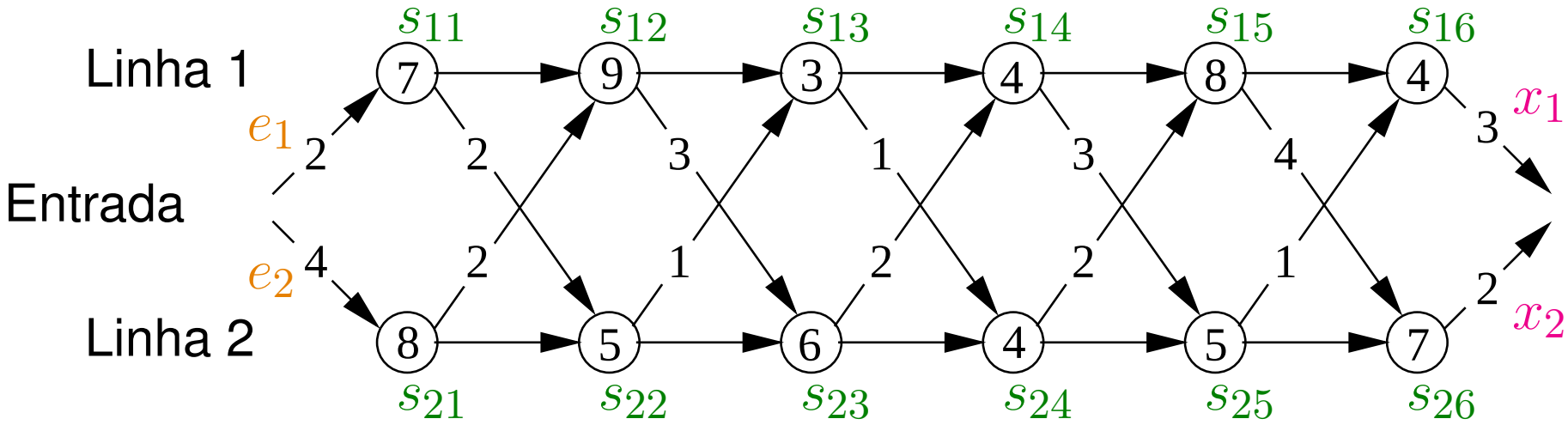
t_{ij}: tempo para mover um item da máquina j
da linha i para a máquina $j+1$ da linha $3-i$

Recorrência



Estrutura da solução: Se a solução passa pela máquina ij, então o caminho do início até ij é de custo mínimo.

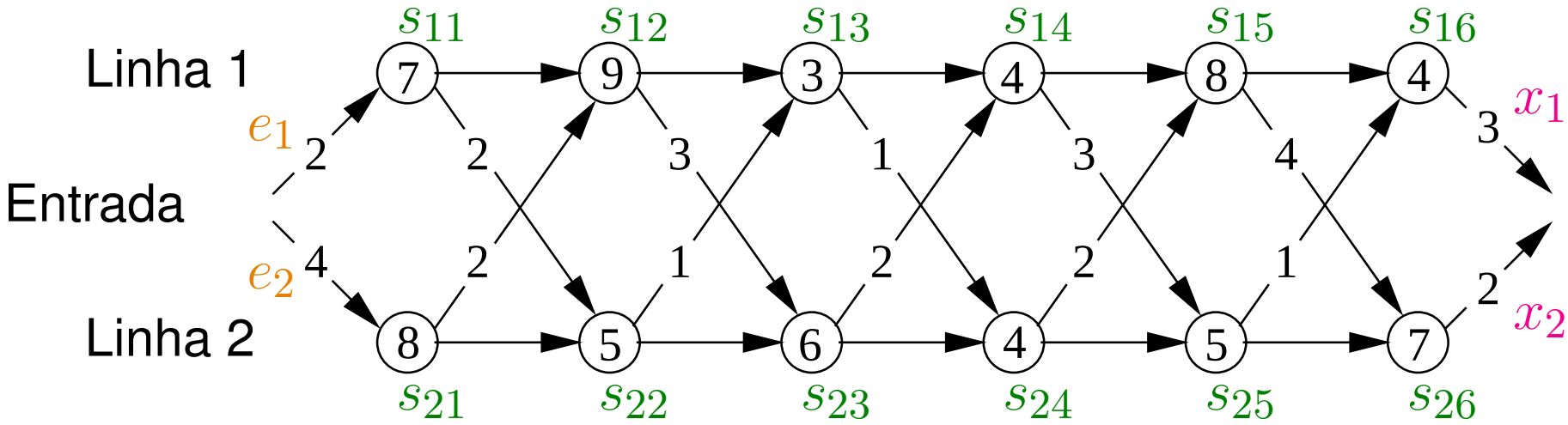
Recorrência



Estrutura da solução: Se a solução passa pela máquina ij, então o caminho do início até ij é de custo mínimo.

Ele pode vir da máquina $i(j-1)$ ou de $(3-i)(j-1)$.

Recorrência



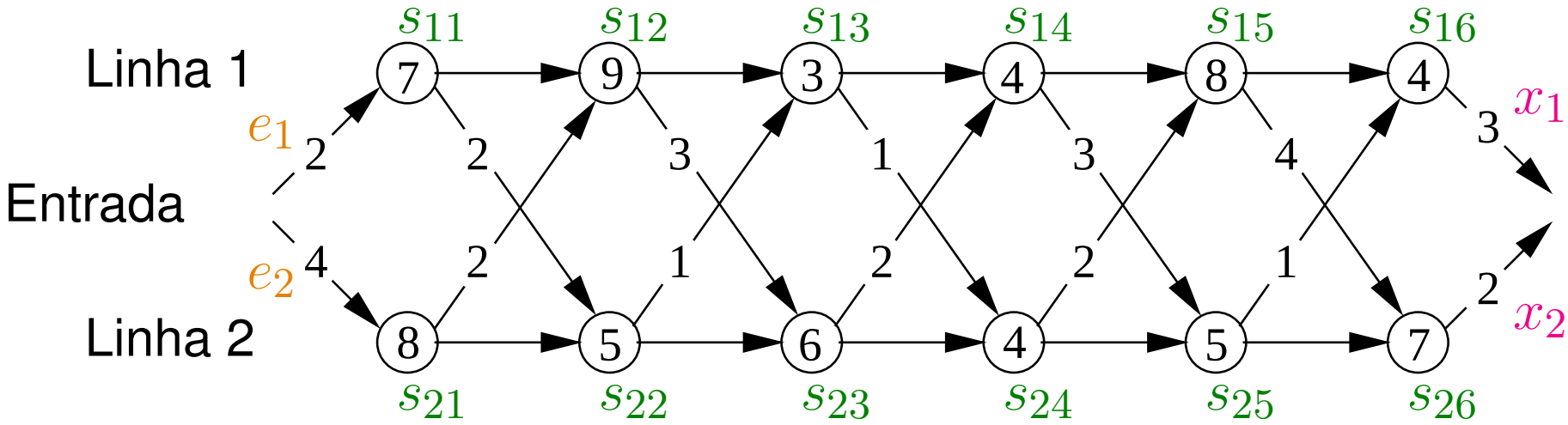
Estrutura da solução: Se a solução passa pela máquina ij, então o caminho do início até ij é de custo mínimo.

Ele pode vir da máquina $i(j-1)$ ou de $(3-i)(j-1)$.

Seja c_{1j} o caminho mínimo do início à máquina $1j$:

$$c_{1j} = \begin{cases} e_1 + s_{1j} & \text{se } j = 1 \\ \min\{c_{1\,j-1} + s_{1j}, c_{2\,j-1} + t_{2\,j} + s_{1j}\} & \text{se } j > 1. \end{cases}$$

Recorrência



Estrutura da solução: Se a solução passa pela máquina ij, então o caminho do início até ij é de custo mínimo.

Ele pode vir da máquina $i(j-1)$ ou de $(3-i)(j-1)$.

Seja c_{2j} o caminho mínimo do início à máquina $2j$:

$$c_{2j} = \begin{cases} e_2 + s_{2j} & \text{se } j = 1 \\ \min\{c_{2\,j-1} + s_{2j}, c_{1\,j-1} + t_{1\,j} + s_{2j}\} & \text{se } j > 1. \end{cases}$$

Recorrência

Estrutura da solução: Se a solução passa pela máquina ij, então o caminho do início até ij é mínimo.

Ele pode vir de $i(j-1)$ ou de $(3-i)(j-1)$.

Seja c_{1j} o caminho mínimo do início à máquina $1j$:

$$c_{1j} = \begin{cases} e_1 + s_{1j} & \text{se } j = 1 \\ \min\{c_{1\,j-1} + s_{1j}, c_{2\,j-1} + t_{2\,j} + s_{1j}\} & \text{se } j > 1, \end{cases}$$

e seja c_{2j} o caminho mínimo do início à máquina $2j$:

$$c_{2j} = \begin{cases} e_2 + s_{2j} & \text{se } j = 1 \\ \min\{c_{2\,j-1} + s_{2j}, c_{1\,j-1} + t_{1\,j} + s_{2j}\} & \text{se } j > 1. \end{cases}$$

Recorrência

Estrutura da solução: Se a solução passa pela máquina ij, então o caminho do início até ij é mínimo.

Ele pode vir de $i(j-1)$ ou de $(3-i)(j-1)$.

Seja c_{1j} o caminho mínimo do início à máquina $1j$:

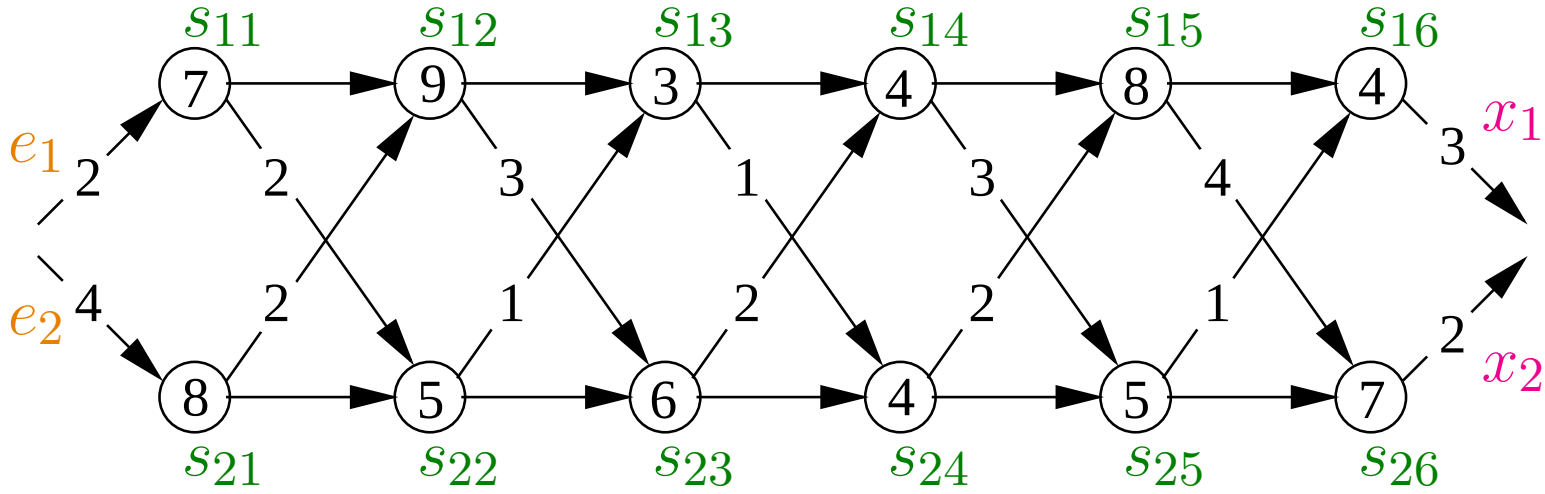
$$c_{1j} = \begin{cases} e_1 + s_{1j} & \text{se } j = 1 \\ \min\{c_{1\,j-1} + s_{1j}, c_{2\,j-1} + t_{2\,j} + s_{1j}\} & \text{se } j > 1, \end{cases}$$

e seja c_{2j} o caminho mínimo do início à máquina $2j$:

$$c_{2j} = \begin{cases} e_2 + s_{2j} & \text{se } j = 1 \\ \min\{c_{2\,j-1} + s_{2j}, c_{1\,j-1} + t_{1\,j} + s_{2j}\} & \text{se } j > 1. \end{cases}$$

O tempo final da produção é $c^* = \min\{c_{1\,n} + x_1, c_{2\,n} + x_2\}$.

Simulação



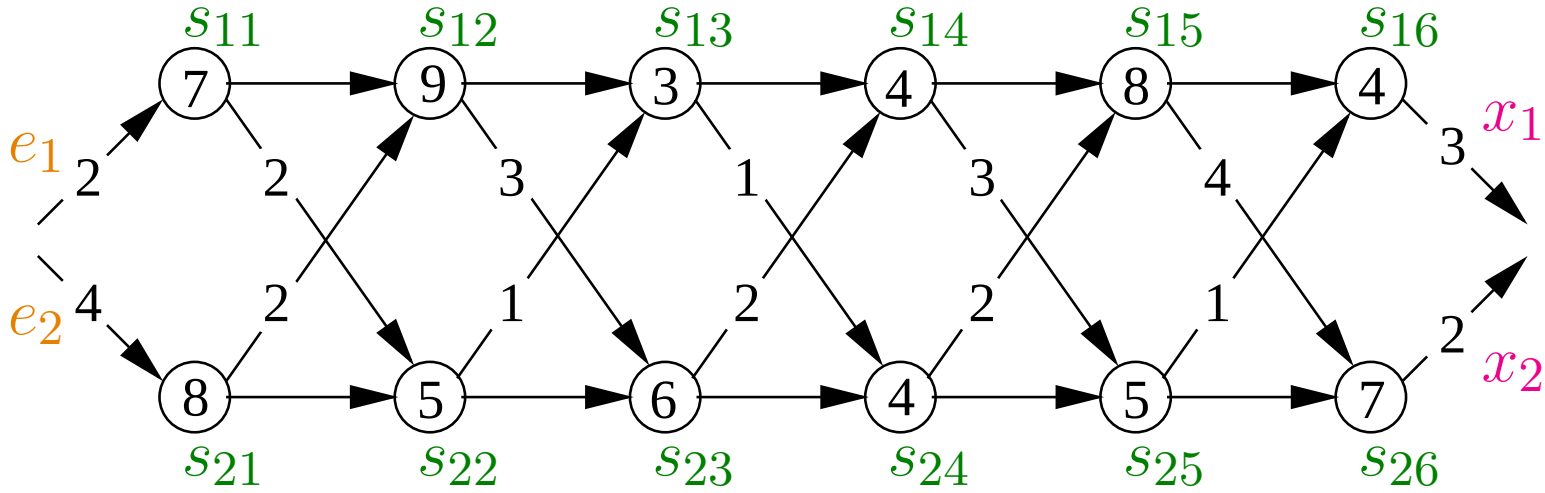
s	1	2	3	4	5	6
1	7	9	3	4	8	4
2	8	5	6	4	5	7

e	
1	2
2	4

x	
1	3
2	2

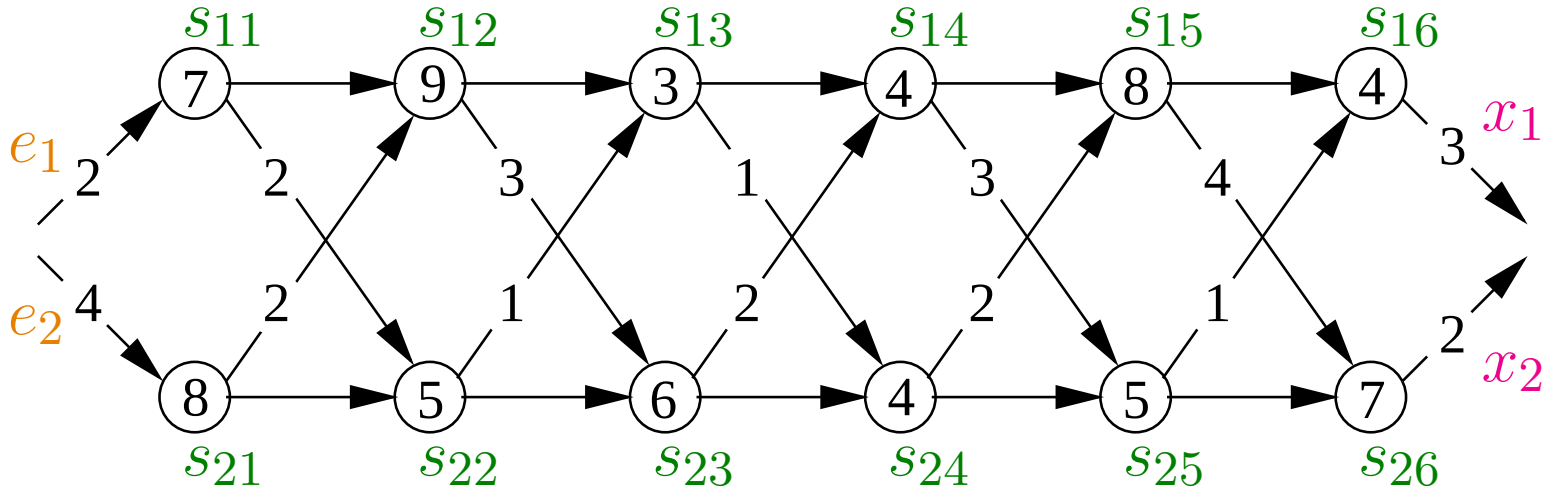
t	1	2	3	4	5
1	2	3	1	3	4
2	2	1	2	2	1

Simulação



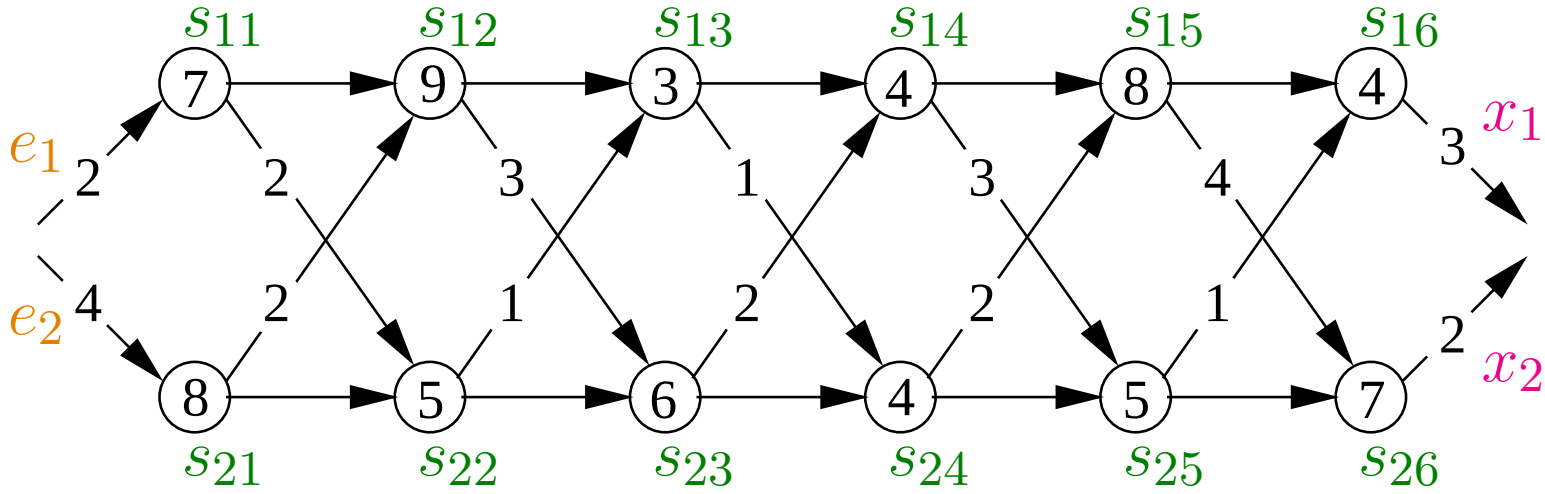
c	1	2	3	4	5	6	j
1	??						
2							

Simulação



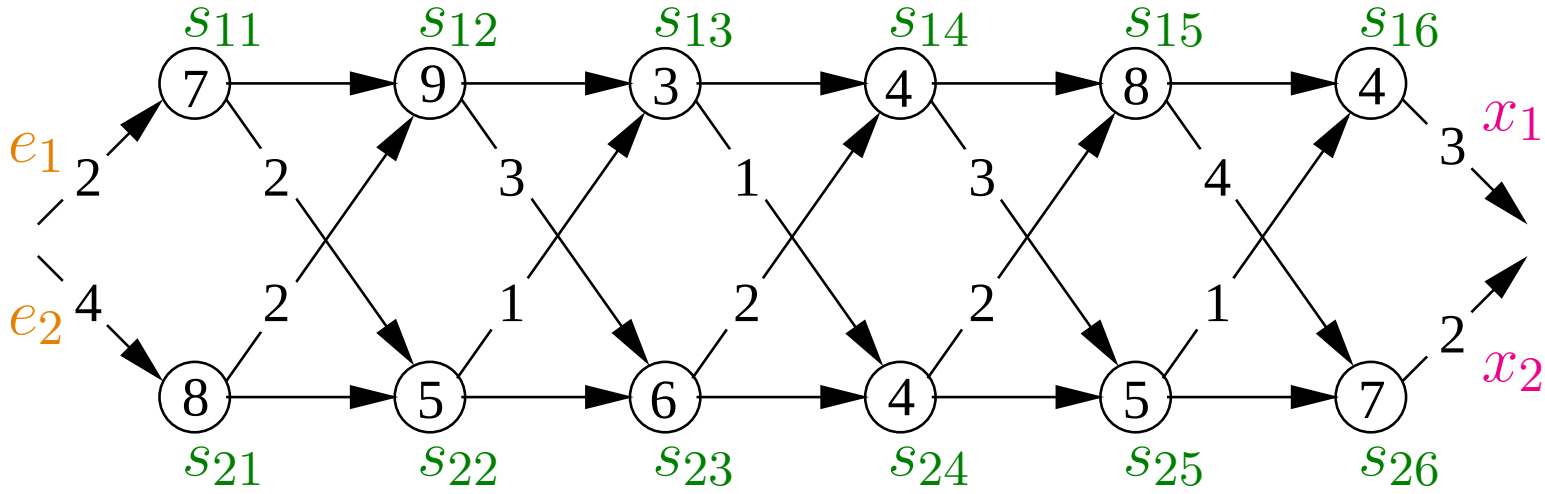
c	1	2	3	4	5	6	j
1	9						
2	??						

Simulação



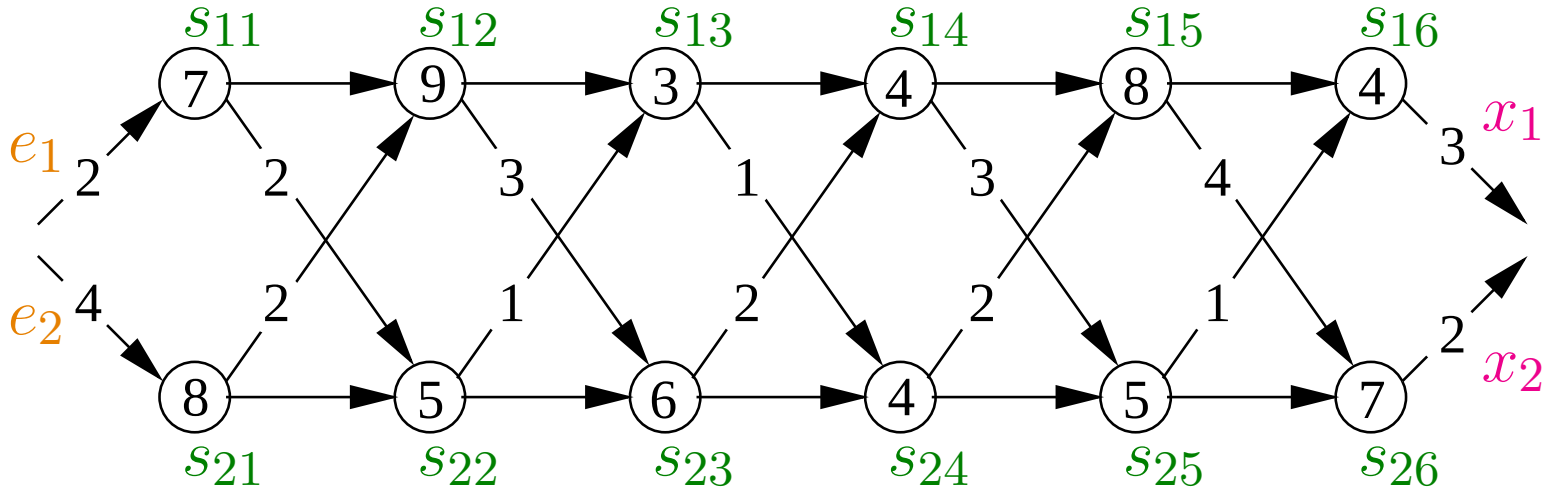
c	1	2	3	4	5	6	j
1	9	??					
2	12						

Simulação



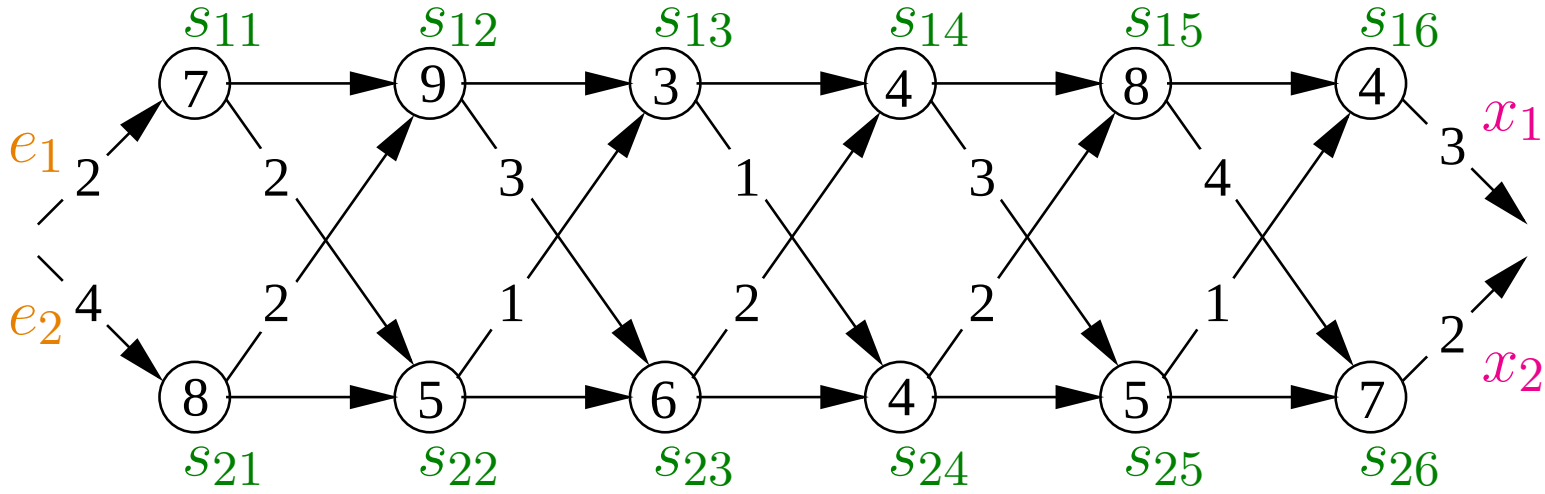
c	1	2	3	4	5	6	j
1	9	18					
2	12	??					

Simulação



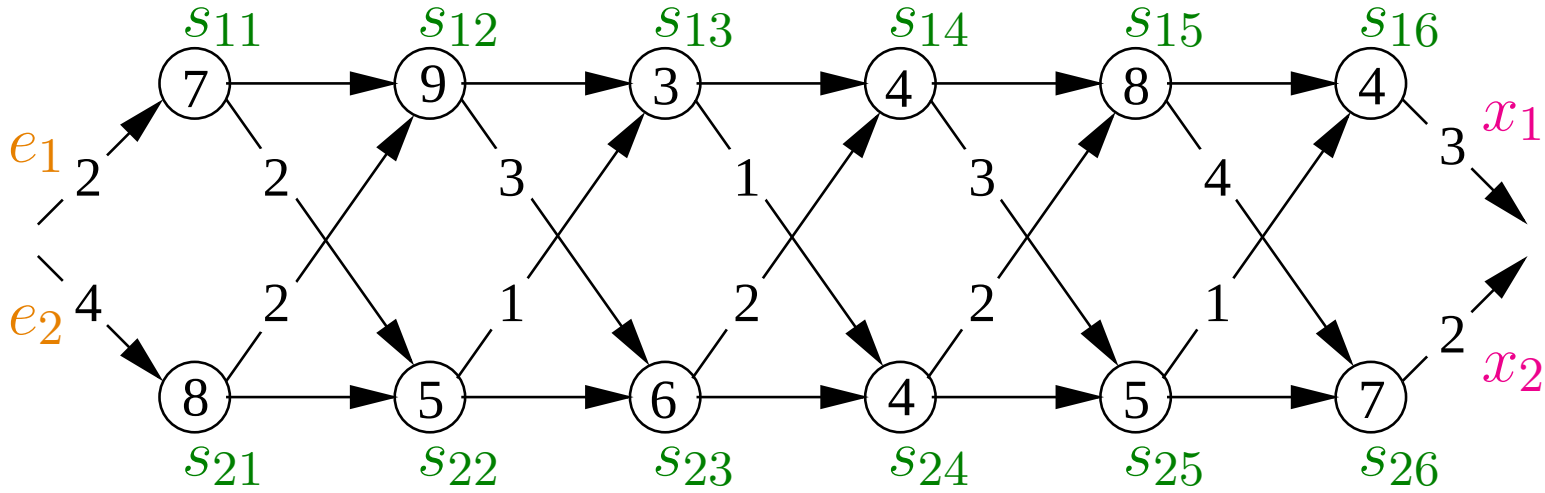
c	1	2	3	4	5	6	j
1	9	18	??				
2	12	16					

Simulação



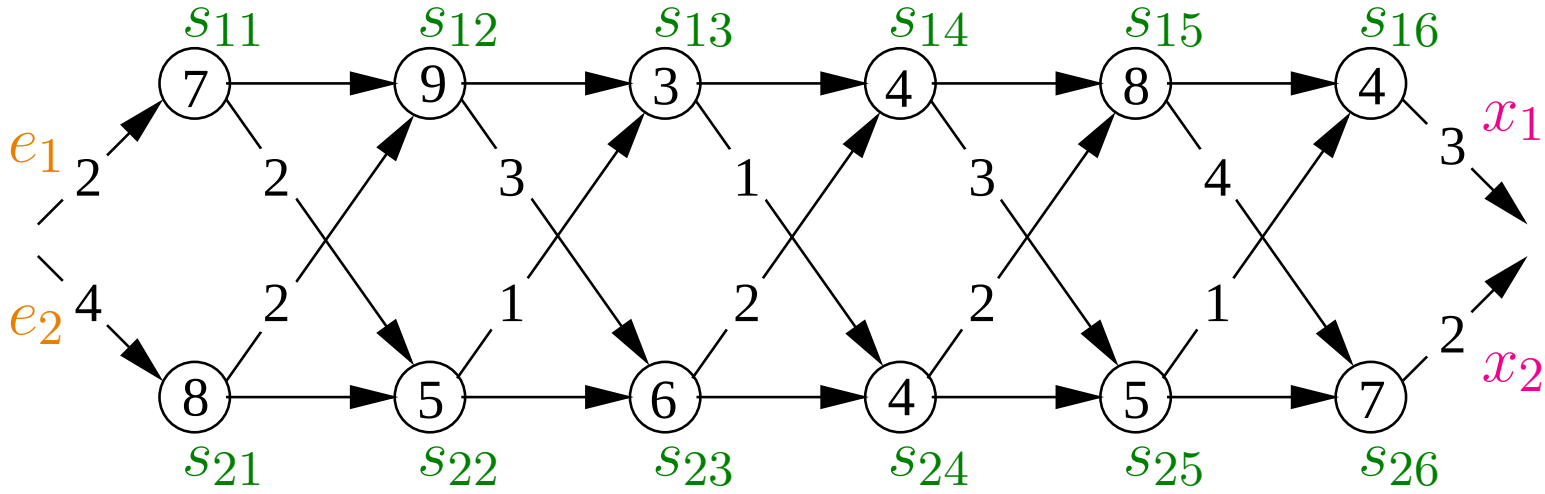
c	1	2	3	4	5	6	j
1	9	18	20				
2	12	16	??				

Simulação



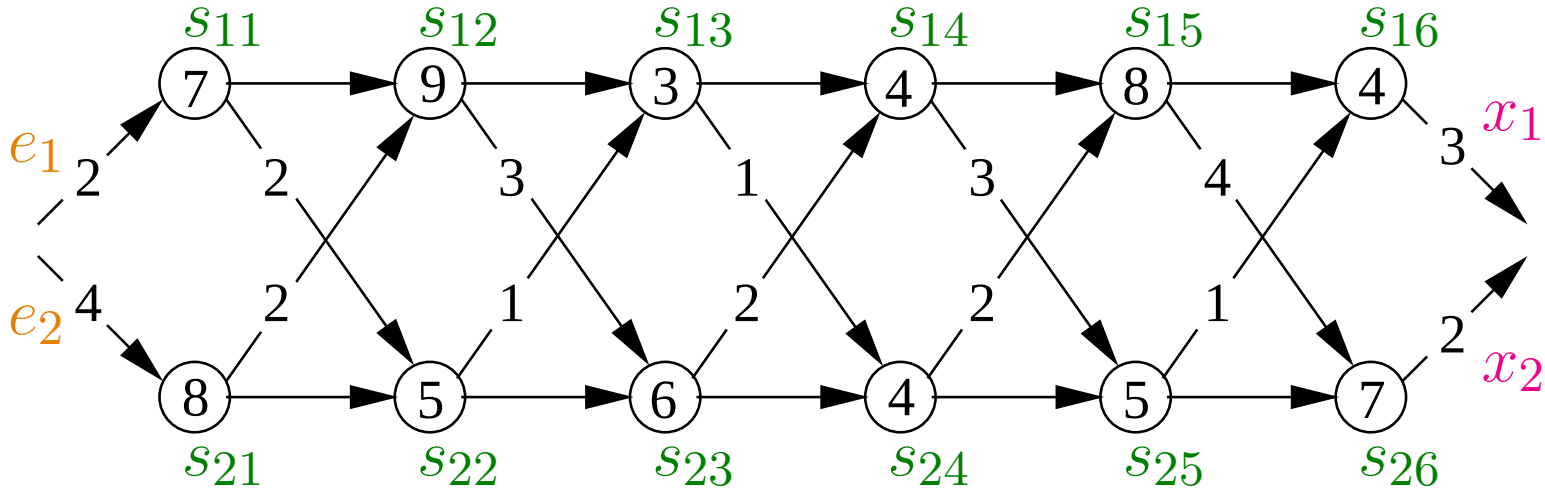
c	1	2	3	4	5	6	j
1	9	18	20	??			
2	12	16	22				

Simulação



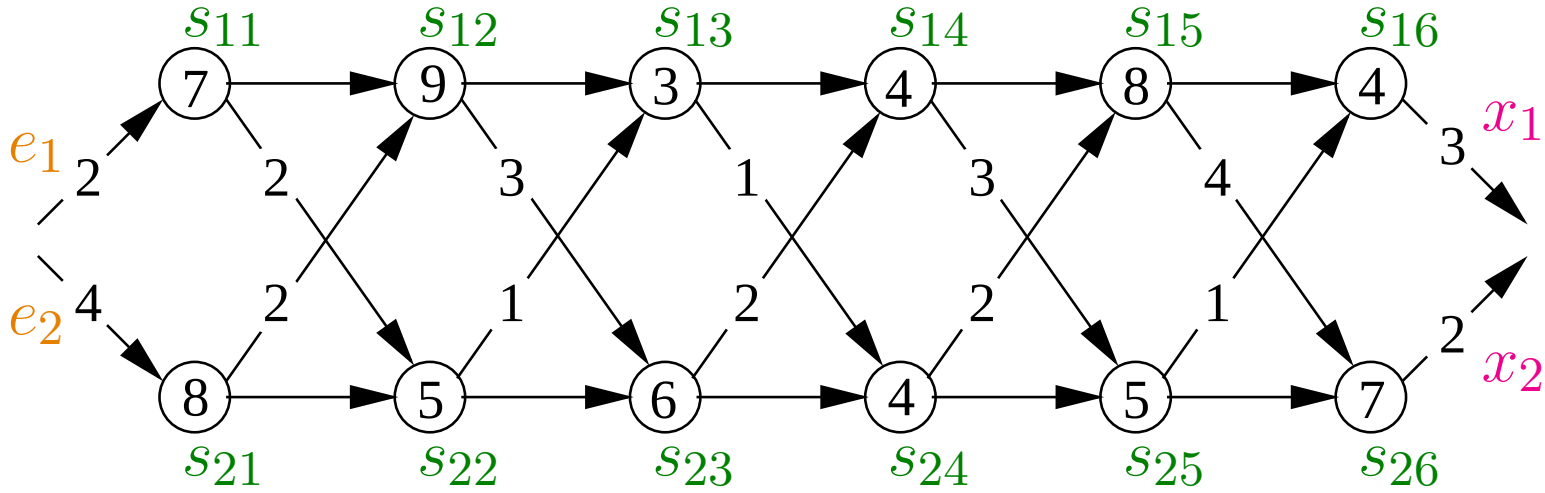
c	1	2	3	4	5	6	j
1	9	18	20	24			
2	12	16	22	??			

Simulação



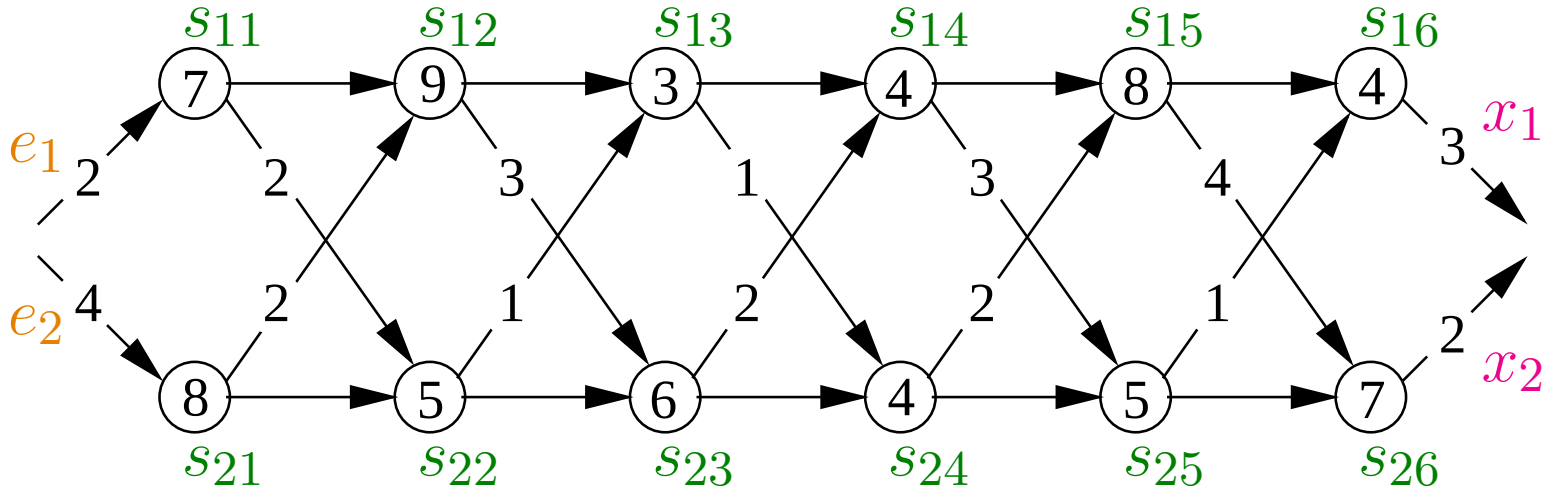
c	1	2	3	4	5	6	j
1	9	18	20	24	??		
2	12	16	22	25			

Simulação



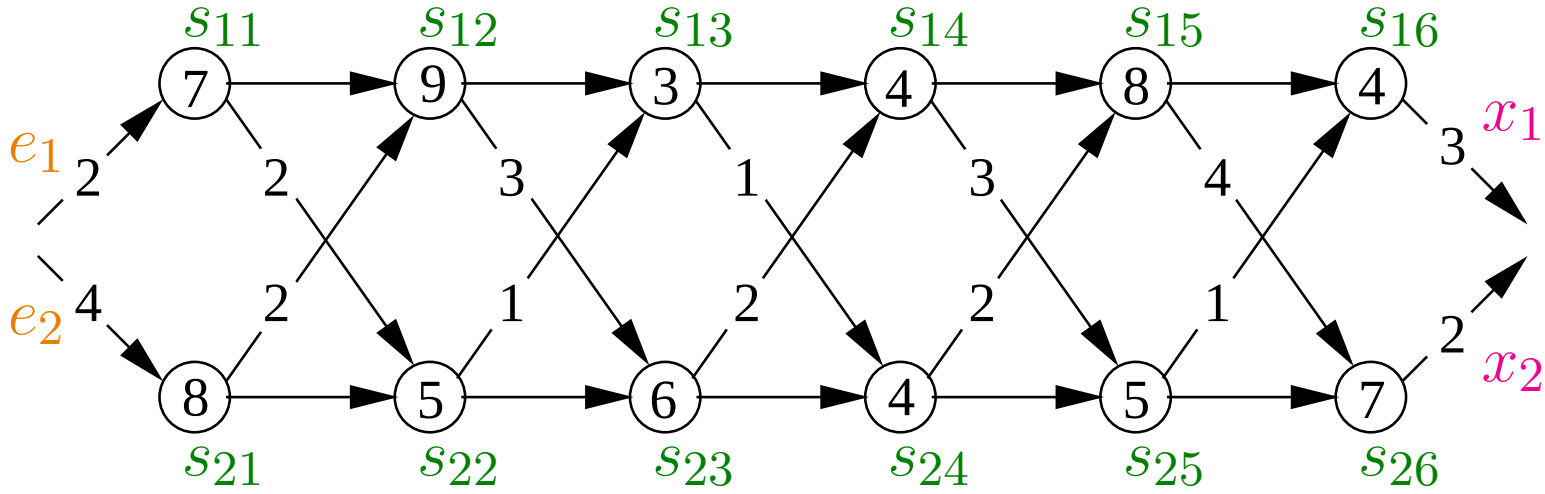
c	1	2	3	4	5	6	j
1	9	18	20	24	32		
2	12	16	22	25	??		

Simulação



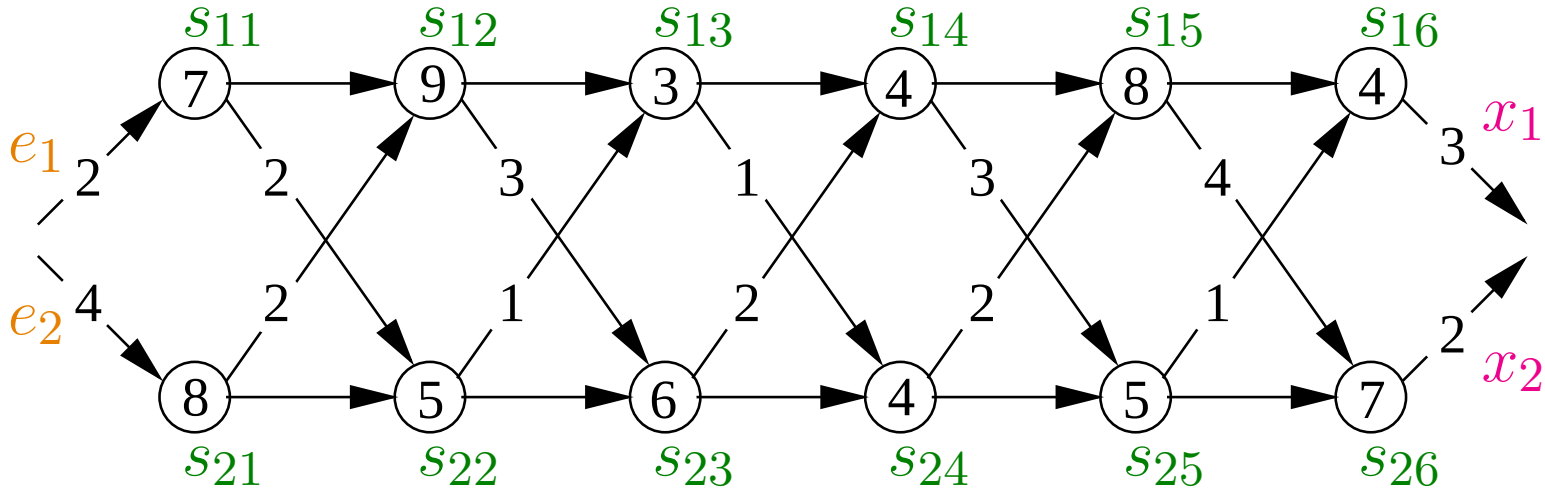
c	1	2	3	4	5	6 j
1	9	18	20	24	32	??
2	12	16	22	25	30	

Simulação



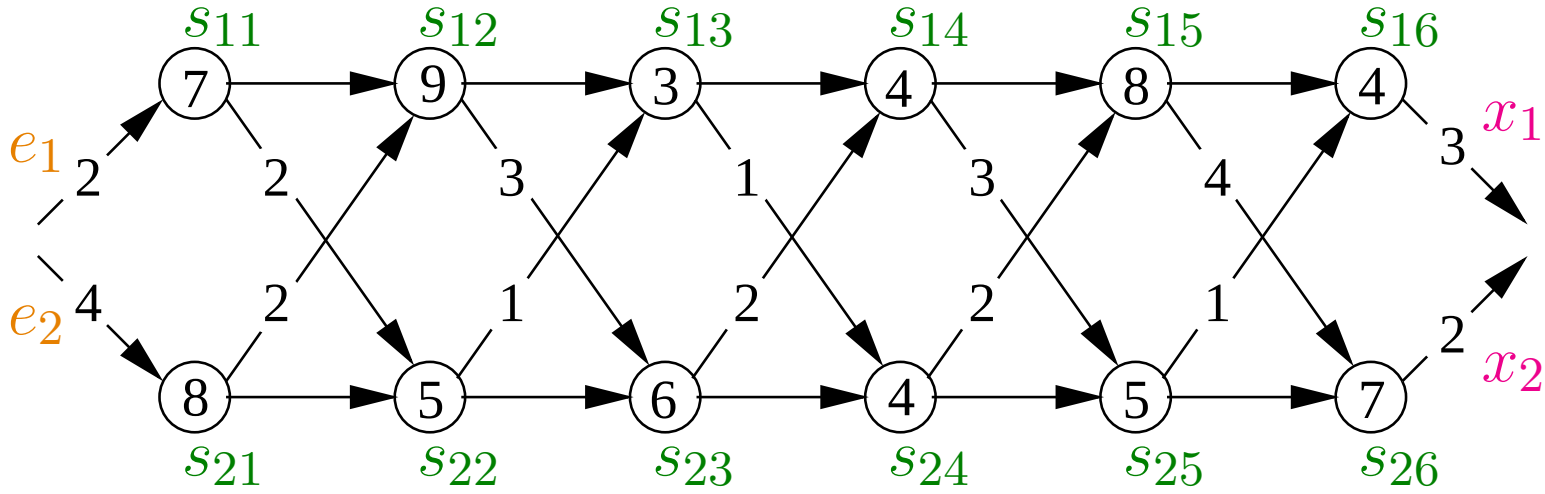
c	1	2	3	4	5	6 j
1	9	18	20	24	32	35
2	12	16	22	25	30	??

Simulação



c	1	2	3	4	5	6	j
1	9	18	20	24	32	35	
2	12	16	22	25	30	37	

Simulação



c	1	2	3	4	5	6	j
1	9	18	20	24	32	35	
2	12	16	22	25	30	37	

$$c^* = \min\{35+3, 37+2\} = \min\{38, 39\} = 38$$

Linhas de produção

LINHA-de-PRODUÇÃO (s, t, e, x, n)

```
 1   c[1,1] ← e[1] + s[1,1]
 2   c[2,1] ← e[2] + s[2,1]
 3   para j ← 2 até n faça
 4         se c[1,j−1] + s[1,j] ≤ c[2,j−1] + t[2,j−1] + s[1,j]
 5               então c[1,j] ← c[1,j−1] + s[1,j]
 6               senão c[1,j] ← c[2,j−1] + t[2,j−1] + s[1,j]
 7         se c[2,j−1] + s[2,j] ≤ c[1,j−1] + t[1,j−1] + s[2,j]
 8               então c[2,j] ← c[2,j−1] + s[2,j]
 9               senão c[2,j] ← c[1,j−1] + t[1,j−1] + s[2,j]
10   se c[1,n] + x[1] ≤ c[2,n] + x[2]
11         então c* ← c[1,n] + x[1]
12         senão c* ← c[2,n] + x[2]
13   devolva c*
```

Linhas de produção

LINHA-de-PRODUÇÃO (s, t, e, x, n)

```
 1   c[1,1] ← e[1] + s[1,1]
 2   c[2,1] ← e[2] + s[2,1]
 3   para j ← 2 até n faça
 4         se c[1,j−1] + s[1,j] ≤ c[2,j−1] + t[2,j−1] + s[1,j]
 5               então c[1,j] ← c[1,j−1] + s[1,j]
 6               senão c[1,j] ← c[2,j−1] + t[2,j−1] + s[1,j]
 7         se c[2,j−1] + s[2,j] ≤ c[1,j−1] + t[1,j−1] + s[2,j]
 8               então c[2,j] ← c[2,j−1] + s[2,j]
 9               senão c[2,j] ← c[1,j−1] + t[1,j−1] + s[2,j]
10   se c[1,n] + x[1] ≤ c[2,n] + x[2]
11         então c* ← c[1,n] + x[1]
12         senão c* ← c[2,n] + x[2]
13   devolva c*
```

O tempo desse algoritmo é $\Theta(n)$.

Linhas de produção

LINHA-de-PRODUÇÃO (s, t, e, x, n)

```
 1   c[1,1] ← e[1] + s[1,1]
 2   c[2,1] ← e[2] + s[2,1]
 3   para j ← 2 até n faça
 4         se c[1,j−1] + s[1,j] ≤ c[2,j−1] + t[2,j−1] + s[1,j]
 5               então c[1,j] ← c[1,j−1] + s[1,j]
 6               senão c[1,j] ← c[2,j−1] + t[2,j−1] + s[1,j]
 7         se c[2,j−1] + s[2,j] ≤ c[1,j−1] + t[1,j−1] + s[2,j]
 8               então c[2,j] ← c[2,j−1] + s[2,j]
 9               senão c[2,j] ← c[1,j−1] + t[1,j−1] + s[2,j]
10   se c[1,n] + x[1] ≤ c[2,n] + x[2]
11         então c* ← c[1,n] + x[1]
12         senão c* ← c[2,n] + x[2]
13   devolva c*
```

O tempo desse algoritmo é $\Theta(n)$. O espaço é $\Theta(n)$.

Linhas de produção

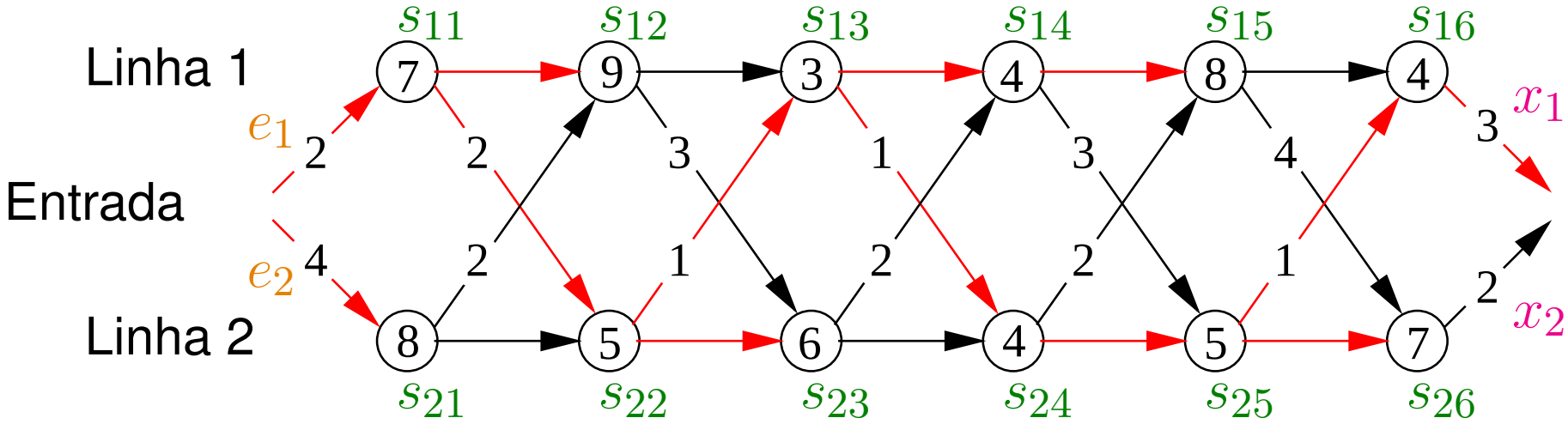
LINHA-de-PRODUÇÃO (s, t, e, x, n)

```
 1   c[1,1] ← e[1] + s[1,1]
 2   c[2,1] ← e[2] + s[2,1]
 3   para j ← 2 até n faça
 4       se c[1,j−1] + s[1,j] ≤ c[2,j−1] + t[2,j−1] + s[1,j]
 5           então c[1,j] ← c[1,j−1] + s[1,j]
 6           senão c[1,j] ← c[2,j−1] + t[2,j−1] + s[1,j]
 7       se c[2,j−1] + s[2,j] ≤ c[1,j−1] + t[1,j−1] + s[2,j]
 8           então c[2,j] ← c[2,j−1] + s[2,j]
 9           senão c[2,j] ← c[1,j−1] + t[1,j−1] + s[2,j]
10   se c[1,n] + x[1] ≤ c[2,n] + x[2]
11       então c* ← c[1,n] + x[1]
12       senão c* ← c[2,n] + x[2]
13   devolva c*
```

Consegue economizar espaço?

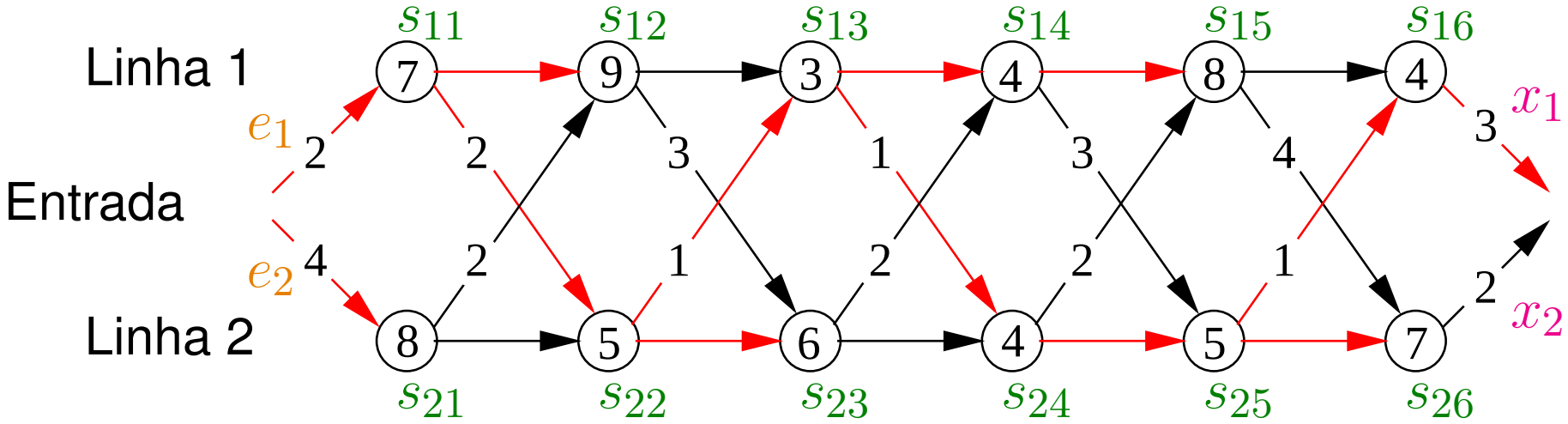
Determinação de caminho ótimo

Marque de onde vieram os valores...



Determinação de caminho ótimo

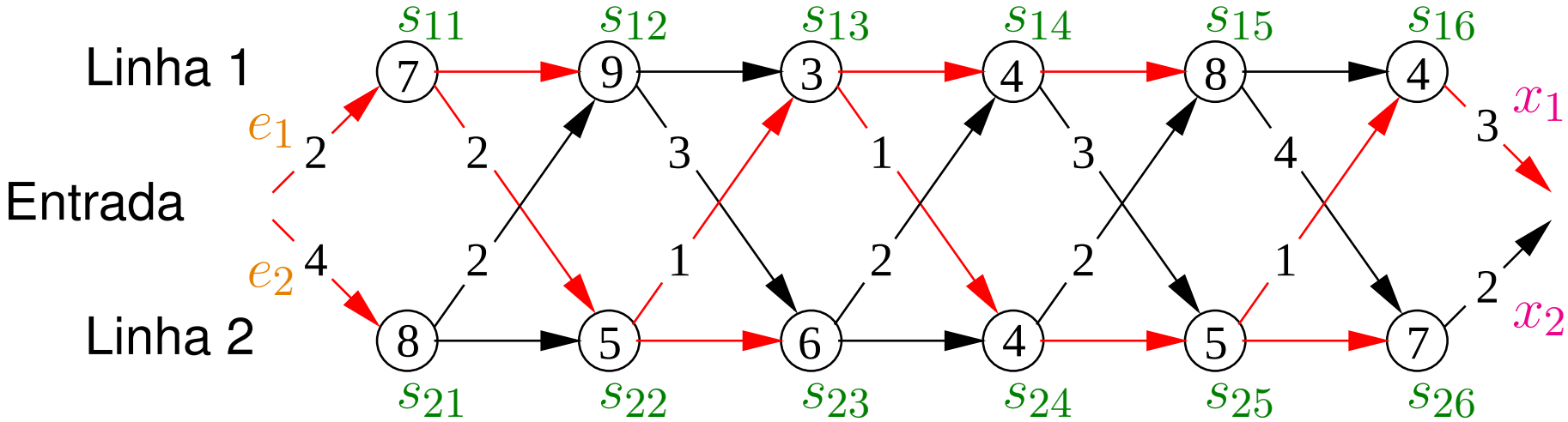
Marque de onde vieram os valores...



Outra matriz que guarda de que nó veio o seu valor.

Determinação de caminho ótimo

Marque de onde vieram os valores...



Outra matriz que guarda de que nó veio o seu valor.

Com essa informação, pode-se
rastrear de volta o caminho, de trás para frente.

Linhas de produção

LINHA-de-PRODUÇÃO (s, t, e, x, n)

```
 1   c[1,1] ← e[1] + s[1,1]
 2   c[2,1] ← e[2] + s[2,1]
 3   para j ← 2 até n faça
 4       se c[1,j−1] + s[1,j] ≤ c[2,j−1] + t[2,j−1] + s[1,j]
 5           então c[1,j] ← c[1,j−1] + s[1,j]
 6                 l[1,j] ← 1
 7           senão c[1,j] ← c[2,j−1] + t[2,j−1] + s[1,j]
 8                 l[1,j] ← 2
 9       se c[2,j−1] + s[2,j] ≤ c[1,j−1] + t[1,j−1] + s[2,j]
10           então c[2,j] ← c[2,j−1] + s[2,j]
11                 l[2,j] ← 2
12           senão c[2,j] ← c[1,j−1] + t[1,j−1] + s[2,j]
13                 l[2,j] ← 1
...
```

Linha de produção

LINHA-de-PRODUÇÃO (s, t, e, x, n)

```
...
14    se c[1, n] + x[1] ≤ c[2, n] + x[2]
15          então c* ← c[1, n] + x[1]
16                l* ← 1
17          senão c* ← c[2, n] + x[2]
18                l* ← 2
19    devolva c*, l*, l
```

Linha de produção

LINHA-de-PRODUÇÃO (s, t, e, x, n)

```
...
14    se c[1,n] + x[1] ≤ c[2,n] + x[2]
15          então c* ← c[1,n] + x[1]
16                l* ← 1
17          senão c* ← c[2,n] + x[2]
18                l* ← 2
19    devolva c*, l*, l
```

O valor l^* e a matrix l podem ser usados para determinarmos um caminho ótimo.

Linha de produção

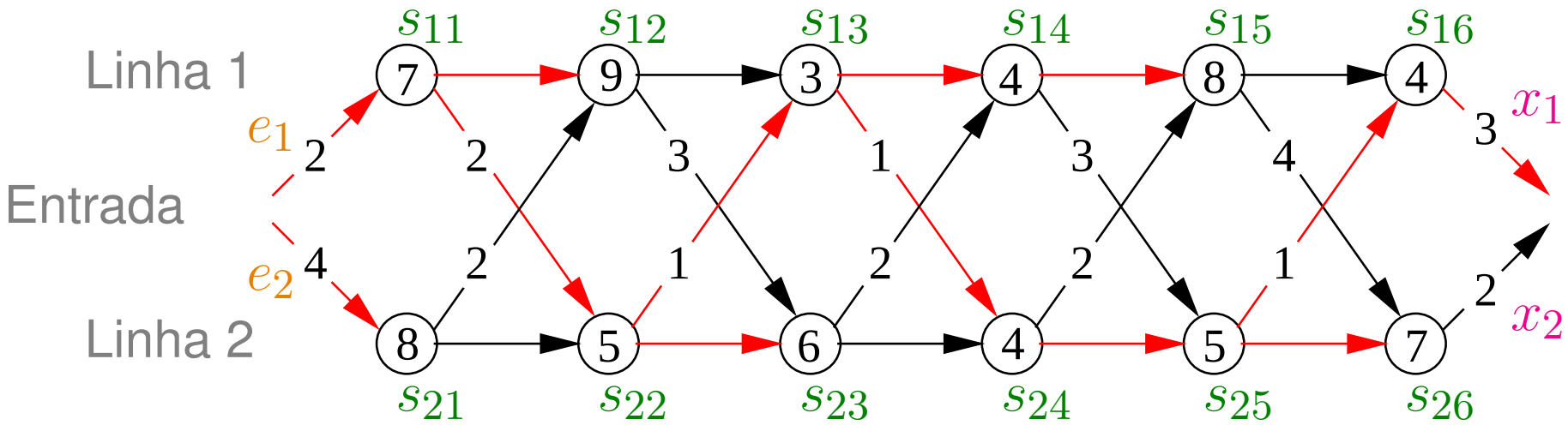
LINHA-de-PRODUÇÃO (s, t, e, x, n)

```
...
14    se c[1, n] + x[1] ≤ c[2, n] + x[2]
15          então c* ← c[1, n] + x[1]
16                l* ← 1
17          senão c* ← c[2, n] + x[2]
18                l* ← 2
19    devolva c*, l*, l
```

O valor l^* e a matrix l podem ser usados para determinarmos um caminho ótimo.

O consumo de tempo e espaço desse algoritmo é $\Theta(n)$.

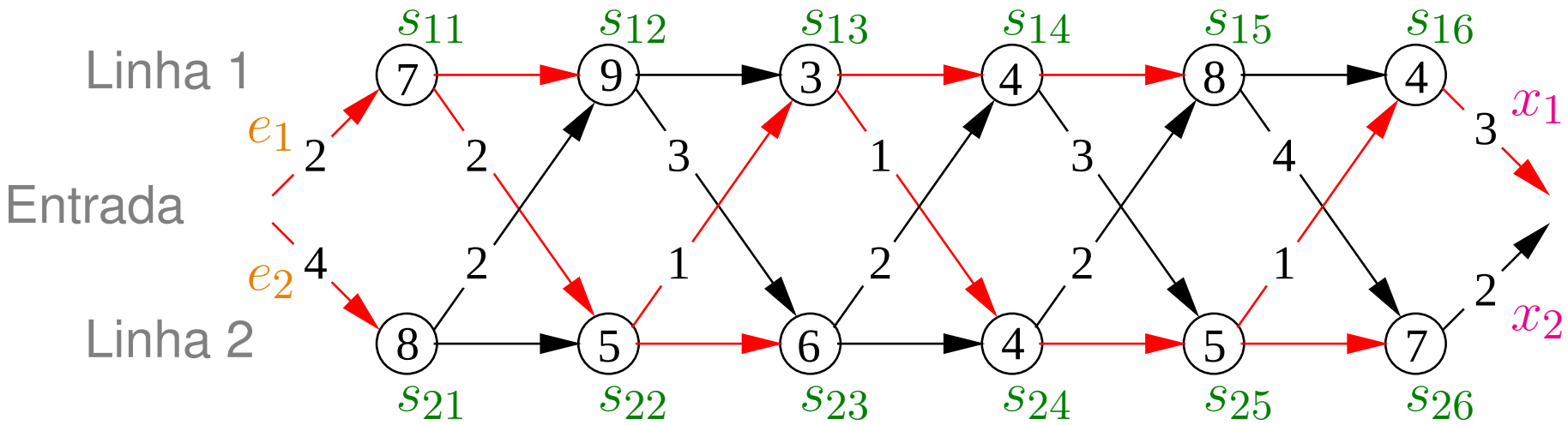
Linha de produção



IMPRIME-MÁQUINAS (l, l^*, n)

```
1   i ← l*
2   imprima "linha i, máquina n"
3   para j ← n até 2 faça
4       i ← l[i, j]
5       imprima "linha i, máquina j − 1"
```

Linha de produção

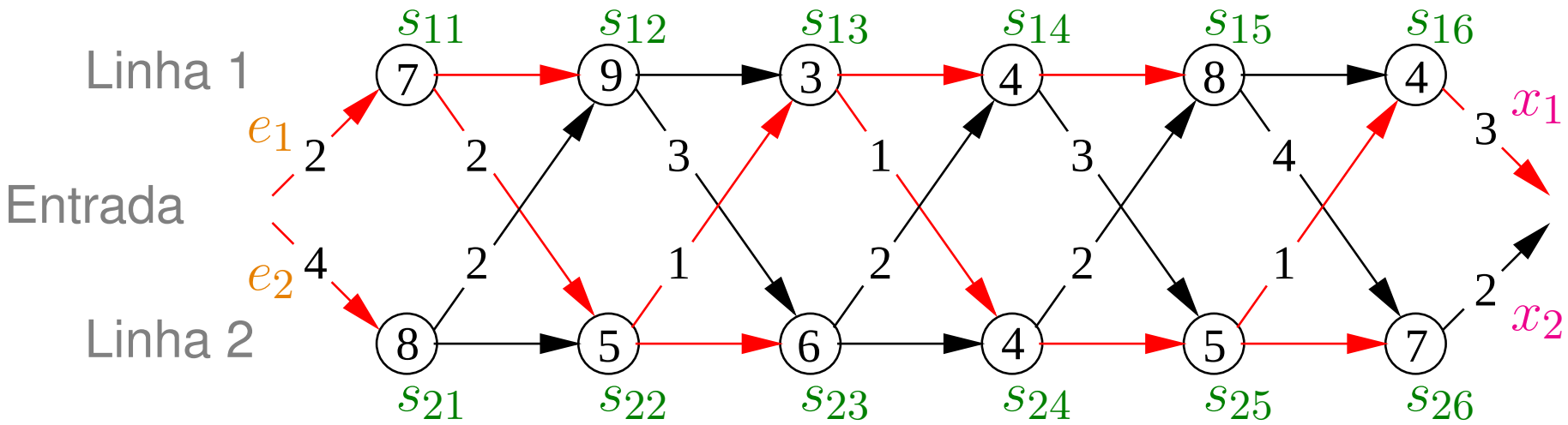


IMPRIME-MÁQUINAS (l, l^*, n)

```
1   i ← l*
2   imprima "linha i, máquina n"
3   para j ← n até 2 faça
4       i ← l[i, j]
5       imprima "linha i, máquina j − 1"
```

O tempo e espaço desse algoritmo é $\Theta(n)$.

Linha de produção



IMPRIME-MÁQUINAS (l, l^*, n)

```
1   i ← l*
2   imprima "linha i, máquina n"
3   para j ← n até 2 faça
4       i ← l[i, j]
5       imprima "linha i, máquina j − 1"
```

O tempo e espaço desse algoritmo é $\Theta(n)$.

Consegue economizar espaço?