

## 6. PROCESSAMENTO EM UMA ÚNICA MÁQUINA

### Hipótese

$$r_i = 0 \quad , \quad i=1,2,\dots,n$$

### Exemplos de Aplicação

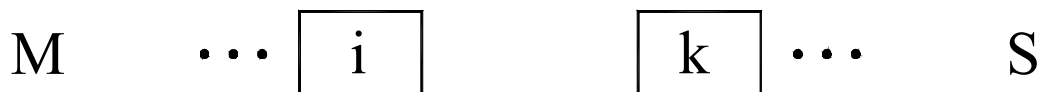
- fábrica de tintas; setor de pintura em fábricas de geladeira, fogões, automóveis, etc.
- máquina gargalo (bottleneck) em ambiente job shop.
- decomposição de problemas com várias máquinas em problemas de uma máquina.

## PROGRAMAS (SCHEDULES) DE PERMUTAÇÃO

**Teorema 6.1 :** Para um problema  $n/1//B$ , onde  $B$  é uma medida regular de desempenho, existe um programa ótimo sem tempo ocioso de máquina, isto é, a máquina inicia o processamento em  $t = 0$  e continua sem parar até  $t = C_{\max}$ .

### Demonstração

Seja  $S$  um programa ótimo com intervalo ocioso  $(\tau_1, \tau_2)$ .



Seja  $S'$  o programa obtido a partir de  $S$  ao se antecipar de  $\tau_2 - \tau_1$  todas as operações que se iniciam após  $\tau_1$ .



Seja  $C_i$  o instante de término de  $i$  em  $S$  e  $C'_i$  o instante de término de  $S'$ . Então

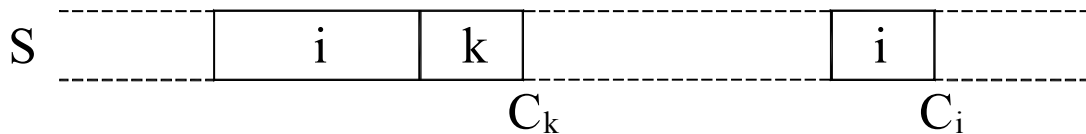
$$C'_i \leq C_i \quad i=1,2,\dots,n$$

o que implica que  $S'$  também é ótimo. Basta repetir o processo de retirada de intervalos ociosos para se obter o enunciado do teorema.

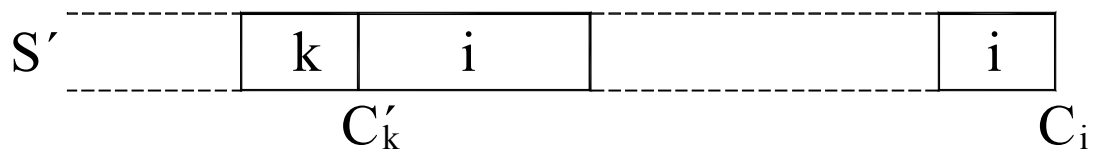
**Teorema 6.2 :** Em um problema  $n/1//B$ , onde  $B$  é uma medida regular de desempenho, não existe nenhuma melhora no programa ótimo a se permitir interrupção de operações.

**Demonstração (esboço) :**

Seja  $S$  um programa ótimo no qual a tarefa  $i$  é interrompida para processamento total da tarefa  $k$



Seja  $S'$  o programa obtido ao se trocar as posições de  $k$  e da 1ª parte de  $i$ .



Como  $C'_k < C_k$ , então  $S'$  também é ótimo. Repetindo este processo para a 1ª parte da tarefa  $i$ , obtém-se um programa ótimo no qual  $i$  é processada sem interrupção.

Os teoremas 3.1 e 3.2 implicam que para problemas  $n/1//B$  ( $B$  regular) só é necessário considerar programas de permutação, isto é, basta achar uma permutação das tarefas  $1, 2, \dots, n$  que forneça uma sequência que minimize  $B$ .

### Notação

$i(k)$  : tarefa programada na  $k$ -ésima posição na sequência de processamento,  $k = 1, 2, \dots, n$ .

## MINIMIZAÇÃO DO TEMPO MÉDIO DE FLUXO

**Problema :**  $n/1//\bar{F}$

Para uma dada sequência de processamento

$$\begin{aligned}\bar{F} &= \frac{1}{n} \sum_{k=1}^n F_{i(k)} \\ &= \frac{1}{n} \sum_{k=1}^n (W_{i(k)} + p_{i(k)}) \\ &= \frac{1}{n} \sum_{k=1}^n W_{i(k)} + \frac{1}{n} \sum_{k=1}^n p_{i(k)}\end{aligned}$$

Como  $\sum_{k=1}^n p_{i(k)} = \sum_{i=1}^n p_i$  (constante), então minimizar  $\bar{F}$  é equivalente a minimizar  $\sum_{k=1}^n W_{i(k)}$ . Para isto basta minimizar cada  $W_{i(k)}$ .

$$W_{i(1)} = 0$$

$W_{i(2)} = p_{i(1)}$  : escolha  $i_1$  como a tarefa de menor tempo de processamento

$W_{i(3)} = p_{i(1)} + p_{i(2)}$  : escolha  $i_1$  como a tarefa de menor tempo de processamento e  $i_2$  como a tarefa de segundo menor tempo de processamento

Intuitivamente, o programa (ou sequência) shortest processing time (SPT) minimiza  $\bar{F}$ .

**Teorema 6.3 :** Para o problema  $n/1//\bar{F}$ , o tempo médio de fluxo é minimizado pela sequência SPT, isto é, a sequência tal que

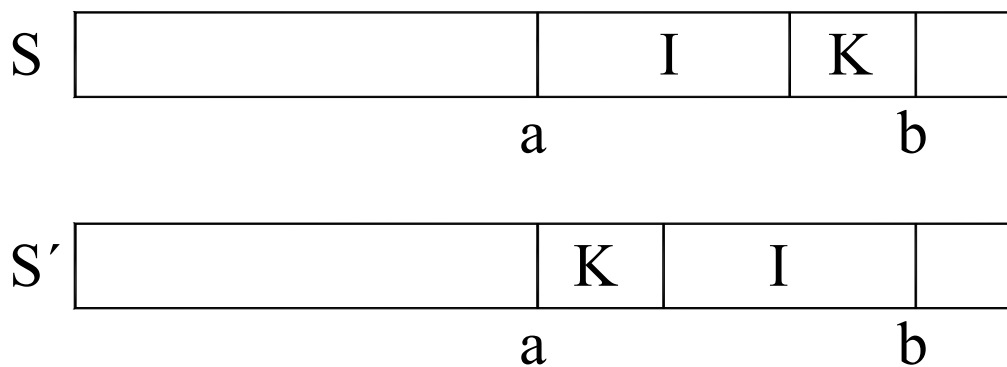
$$p_{i(1)} \leq p_{i(2)} \leq \dots \leq p_{i(n)}.$$

### Demonstração

Seja  $S$  uma sequência não-SPT. Então para algum  $k$

$$p_{i(k)} > p_{i(k+1)}$$

Seja  $S'$  uma sequência obtida pela troca de  $i(k) = I$  e  $i(k+1) = K$  em  $S$ .



A diferença do tempo de fluxo médio entre  $S$  e  $S'$  depende somente dos tempos de fluxo de  $I$  e  $K$ .

Seja

$$a = \sum_{\ell=1}^{k-1} p_{i(\ell)} = \begin{cases} W_I & \text{em } S \\ W'_K & \text{em } S' \end{cases}$$

$$\text{Em } S : F_I = a + p_I \quad \text{e} \quad F_K = a + p_I + p_K$$

$$\text{Em } S' : F'_I = a + p_k + p_I \quad \text{e} \quad F'_K = a + p_K$$

$$\begin{aligned} \frac{1}{n}(F_I + F_K) &= \frac{1}{n}(2a + p_I + p_K + p_I) \\ &> \frac{1}{n}(2a + p_I + p_K + p_K) \\ &= \frac{1}{n}(F'_I + F'_K) \end{aligned}$$

Portanto, a sequência SPT minimiza  $\bar{F}$ .

## Exemplo

Seja o problema  $7/1//\bar{F}$

Tarefa	1	2	3	4	5	6	7
Tempo de Processamento	6	4	8	3	2	7	1

Sequência SPT : (7, 5, 4, 2, 1, 6, 3)

$$F_{i(1)} = 1$$

$$F_{i(2)} = 1 + 2$$

$$F_{i(3)} = 1 + 2 + 3$$

$$F_{i(4)} = 1 + 2 + 3 + 4$$

$$F_{i(5)} = 1 + 2 + 3 + 4 + 6$$

$$F_{i(6)} = 1 + 2 + 3 + 4 + 6 + 7$$

$$F_{i(7)} = 1 + 2 + 3 + 4 + 6 + 7 + 8$$

$$\bar{F} = \frac{1}{7} \sum_{k=1}^7 F_{(k)} = \frac{90}{7}$$

## MINIMIZAÇÃO DO LATENESS MÁXIMO

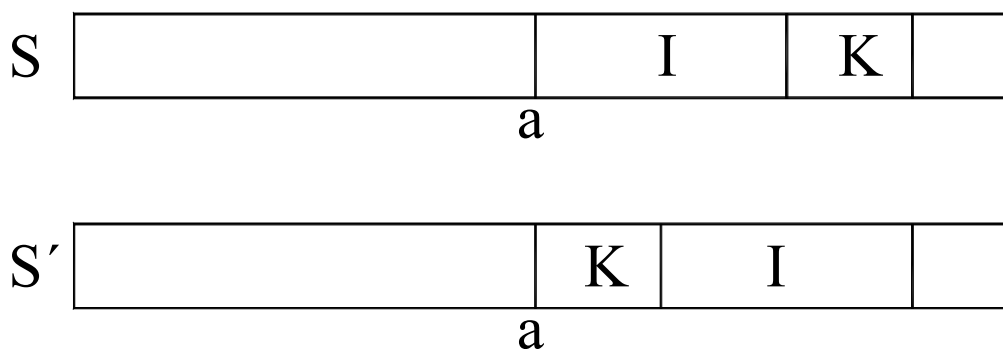
**Teorema 6.4 :** Para o problema  $n/1//L_{\max}$ , o máximo “lateness” é minimizado pela sequência earliest due date (EDD), isto é, a sequência tal que

$$d_{i(1)} \leq d_{i(2)} \leq \dots d_{i(n)}$$

onde  $d_{i(k)}$  é a data de entrega da tarefa que é processada em  $k$ -ésimo lugar.

### Demonstração

Seja  $S$  uma sequência tal que para algum  $k$ ,  $d_{i(k)} > d_{i(k+1)}$ . Seja  $i(k) = I$  e  $i(k+1) = K$ . Seja  $S'$  a sequência obtida ao se trocar as tarefas  $I$  e  $K$  em  $S$



$L$  = máximo lateness das  $n-2$  tarefas restantes em  $S$

$L'$  = máximo lateness das  $n-2$  tarefas restantes em  $S'$

Então,  $L = L'$

$L_I, L_K$  = lateness de  $I$  e  $K$  em  $S$

$L'_I, L'_K$  = lateness de  $I$  e  $K$  em  $S'$

Em  $S$ ,

$$L_{\max} = \max(L, L_I, L_K)$$

Em  $S'$

$$L'_{\max} = \max(L, L'_I, L'_K)$$

Em S,

$$L_I = a + p_I - d_I$$

$$L_K = a + p_I + p_K - d_K$$

Em S'

$$L'_K = a + p_K - d_K$$

$$L'_I = a + p_K + p_I - d_I$$

Logo

$$L_K > L'_K \quad (\text{pois } p_I > 0)$$

e

$$L_K > L'_I \quad (\text{pois } d_I > d_k)$$

Portanto,

$$L_K > \max(L'_I, L'_K)$$

$$\Rightarrow \max(L, L_I, L_K) \geq \max(L, L_K)$$

$$\geq \max(L, L'_I, L'_K)$$

$$\Rightarrow L_{\max} \geq L'_{\max}$$

A sequência EDD minimiza o máximo lateness e pelo Teorema 6.3 também minimiza o máximo atraso, isto é, também resolve o problema  $n/1//T_{\max}$ .

## Exemplo

Seja o problema  $6/1//T_{\max}$

Tarefa	1	2	3	4	5	6
Data de Entrega	7	3	8	12	9	3
Tempo de Processamento	1	1	2	4	1	3

Sequência EDD : (6, 2, 1, 3, 5, 4)

Tarefa	Instante de Término	Lateness	Atraso
$i(k)$	$C_{i(k)}$	$L_{i(k)} = C_{i(k)} - d_{i(k)}$	$T_{i(k)} = \max\{0, L_{i(k)}\}$
6	3	0	0
2	4	1	1
1	5	-2	0
3	7	-1	0
5	8	-1	0
4	12	0	0

## MINIMIZAÇÃO DO NÚMERO DE TAREFAS ATRASADAS

**Problema :**  $n/1//n_T$

Algoritmo de Moore e Hodgson

Passo 1. Sequencie as tarefas de acordo com a regra EDD

Passo 2. Identifique a primeira tarefa atrasada, por exemplo  $i(\ell)$ , na sequência corrente. Se não existe tarefa atrasada, vá para o passo 4.

Passo 3. Identifique a tarefa na subsequência  $(i(1), i(2), \dots, i(\ell))$  com o maior tempo de processamento e retire-a da sequência corrente. Retorne ao passo 2 com a sequência corrente.

Passo 4. Forme uma sequência ótima a partir da sequência corrente e adicione à mesma as tarefas retiradas no passo 3 que podem ser sequenciadas em qualquer ordem.

Nota : As tarefas atrasadas são aquelas retiradas no passo 3.

Seja o problema  $6/1//n_T$

Tarefa	1	2	3	4	5	6
Data de Entrega	15	6	9	23	20	30
Tempo de Processamento	10	3	4	8	10	6

Passos 1 e 2

Sequência Corrente	2	3	1	5	4	6
Data de Entrega	6	9	15	20	23	30
Tempo de Processamento	3	4	10	10	8	6
Instante de Término	3	7	17	27	35	41

quatro tarefas  
atrasadas

A tarefa 1 é a primeira tarefa atrasada e na subsequência (2, 3, 1) é a que possui maior tempo de processamento. Retire a tarefa 1 e vá para o passo 2.

Sequência Corrente	2	3	5	4	6
Data de Entrega	6	9	20	23	30
Tempo de Processamento	3	4	10	8	6
Instante de Término	3	7	17	25	31

Tarefas retiradas  
1

A tarefa 4 é a primeira tarefa atrasada na sequência e na subsequência (2, 3, 5, 4), a tarefa 5 tem o maior tempo de processamento. Retire a tarefa 5 e retorne ao passo 2.

Sequência Corrente	2	3	4	6
Data de Entrega	6	9	23	30
Tempo de Processamento	3	4	8	6
Instante de Término	3	7	15	21

Tarefas retiradas  
1, 5

Passo 4. Sequências ótimas: (2, 3, 4, 6, 1, 5) e (2, 3, 4, 6, 5, 1)

## RESTRICÇÕES DE PRECEDÊNCIA

### Exemplos

- 1) cliente preferencial: a tarefa associada tem prioridade sobre as outras tarefas.
- 2) agrupamento de tarefas com características semelhantes na preparação de máquina.
- 3) scheduling de programas em computador.



## SCHEDULING DE CADEIA DE TAREFAS

### Hipóteses

- 1)  $n$  tarefas particionadas em  $K$  cadeias de  $n_1, n_2, \dots, n_K$  tarefas.
- 2) ordem de processamento das tarefas dentro de uma cadeia é dada.
- 3) cada cadeia de tarefas deve ser processada inteiramente antes que se inicie outra cadeia.

Seja

$p_{ij}$  = tempo de processamento da  $j$ -ésima tarefa na  $i$ -ésima cadeia

$p_i = \sum_{j=1}^{n_i} p_{ij}$  = tempo total de processamento da  $i$ -ésima cadeia

$F_{ij}$  = tempo de fluxo da  $j$ -ésima tarefa na  $i$ -ésima cadeia

Fluxo médio de uma tarefa é dada por

$$\bar{F} = \frac{1}{n} \sum_{i=1}^K \sum_{j=1}^{n_i} F_{ij}$$

**Problema :**  $n/1//\bar{F}$  para cadeias de seqüências

**Teorema 6.5 :** Para o problema  $n/1//\bar{F}$  sob as hipóteses 1, 2, 3 um programa ótimo é tal que

$$\frac{p'_{i(1)}}{n_{i(1)}} \leq \frac{p'_{i(2)}}{n_{i(2)}} \leq \dots \leq \frac{p'_{i(K)}}{n_{i(K)}} \quad (3.1)$$

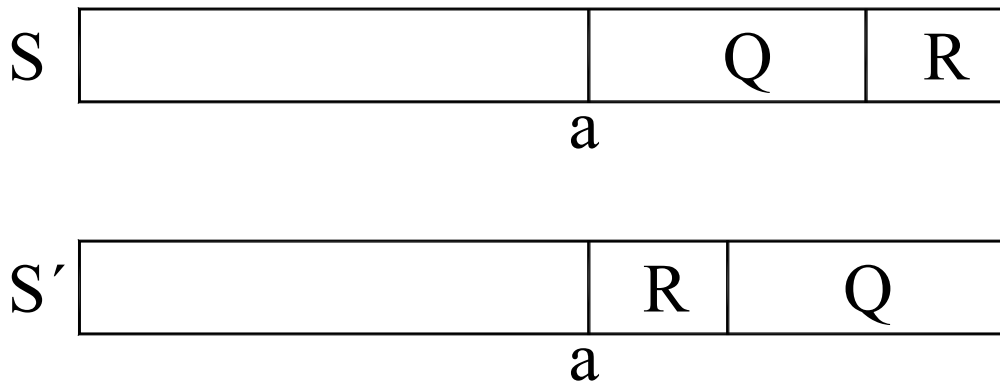
onde a  $i(1)$ -ésima cadeia é sequenciada em 1<sup>o</sup> lugar, a  $i(2)$ -ésima cadeia é sequenciada em 2<sup>o</sup> lugar, etc.

### **Demonstração**

Seja  $S$  um programa tal que (3.1) não é verdadeiro. Então existem duas cadeias  $Q$  e  $R$  adjacentes tais que

$$\frac{p'_Q}{n_Q} > \frac{p'_R}{n_R} \quad (3.2)$$

Seja  $S'$  o programa formado pela troca das cadeias Q e R



Alteração no fluxo total:

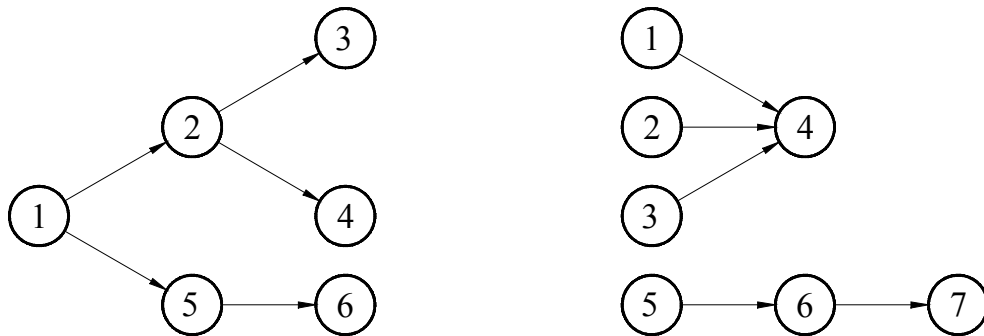
- i) contribuição da cadeia Q aumenta de  $n_Q p'_R$
- ii) contribuição da cadeia R diminui de  $n_R p'_Q$

De (3.2)

$$n_R p'_Q > n_Q p'_R$$

e portanto, a troca de Q e R traz uma redução no tempo de fluxo médio.

Restrições de precedência mais gerais: certas tarefas devem ser processadas antes de outras tarefas



Algoritmo de Lawler

Minimizar

$$\max_{i=1, \dots, n} \{\gamma_i(C_i)\}$$

onde  $\gamma_i(C_i)$  é o custo da tarefa  $i$  e  $\gamma_i$  é uma função não decrescente em  $C_i$  (medida de desempenho regular)

Casos particulares

$$\gamma_i(C_i) = C_i - d_i = L_i \quad \text{corresponde a minimizar } L_{\max}$$

$$\gamma_i(C_i) = \max\{C_i - d_i, 0\} \quad \text{corresponde a minimizar } T_{\max}$$

**Teorema 3.6 :** Seja o problema  $n/1//\max_{i=1}^n \{\gamma_i(C_i)\}$  com restrições de precedência. Seja  $V$  o subconjunto de tarefas que não precedem outras tarefas (isto é, tarefas processadas em último lugar). Note que a última tarefa de um programa é completada em  $\tau = \sum_{i=1}^n p_i$ . Seja  $k$  uma tarefa em  $V$  tal que

$$\gamma_k(\tau) = \min_{i \in V} \{\gamma_i(\tau)\} \quad (3.3)$$

isto é, de todas as tarefas que podem ser completadas em  $\tau$ ,  $k$  é a de mínimo custo. Então existe um programa ótimo no qual a tarefa  $k$  é processada em último lugar.

## Demonstração

Seja  $S$  um programa ótimo no qual a tarefa não é processada em último lugar. Então  $S$  tem a forma

$$S = (A, k, B, \ell)$$

onde  $\ell$  é a última tarefa em  $S$  e  $A$  e  $B$  são subsequências das  $(n - 2)$  tarefas restantes. Considere a nova sequência

$$S' = (A, B, \ell, k)$$

Como  $S$  obedece as relações de precedência e  $k$  pode ser a última tarefa então  $S'$  é factível

Todos os instantes de término, com exceção de  $k$ , decresceram de  $S$  para  $S'$ . Como

$$\gamma_k(\tau) = \min_{i \in V} \{\gamma_i(\tau)\} \leq \gamma_\ell(\tau)$$

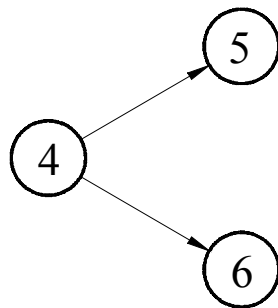
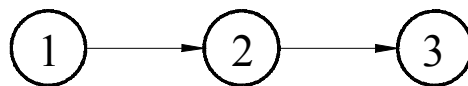
então o custo de  $S'$  não excede o de  $S$ .

## Esboço do algoritmo

- 1) Escolha a tarefa  $k$  através de (3.3).
- 2) Aplique o Teorema 3.6 às  $(n - 1)$  tarefas restantes.

## Exemplo

Seja o problema  $6/1//L_{\max}$



Tarefa	1	2	3	4	5	6
Tempo de Processamento	2	3	4	3	2	1
Data de entrega	3	6	9	7	11	7

### **Tarefa processada em 6º lugar**

$$\tau = 15$$

$$V = \{3, 5, 6\}$$

$$\text{Mínimo lateness em } V : \min\{(15-9), (15-11), (15-7)\} = 4$$

$\Rightarrow$  tarefa 5 é processada em 6º lugar

### **Tarefa processada em 5º lugar : retire tarefa 5 do problema**

$$\tau = 15 - 2 = 13$$

$$V = \{3, 6\} \quad \min\{(13-9), (13-7)\} = 4$$

$\Rightarrow$  tarefa 3 é processada em 5º lugar

**Tarefa processada em 4º lugar** : retire tarefa 3 do problema

$$\tau = 13 - 4 = 9$$

$$V = \{2, 6\} \quad \min\{(9-6), (9-7)\} = 2$$

$\Rightarrow$  tarefa 6 é processada em 4º lugar

**Tarefa processada em 3º lugar** : retire tarefa 6 do problema

$$\tau = 9 - 1 = 8$$

$$V = \{2, 4\} \quad \min\{(8-6), (8-7)\} = 1$$

$\Rightarrow$  tarefa 4 é processada em 3º lugar

O programa ótimo é

$(1, 2, 4, 6, 3, 5)$  com  $L_{\max} = 4$