

# Arquivos - Manutenção

Prof.: Leonardo Tórtoro Pereira  
leonardop@usp.br

\*Material baseado em aulas dos professores: Elaine Parros Machado de Souza, Gustavo Batista, Robson Cordeiro, Moacir Ponti Jr., Maria Cristina Oliveira e Cristina Ciferri.

# Manutenção de Arquivos

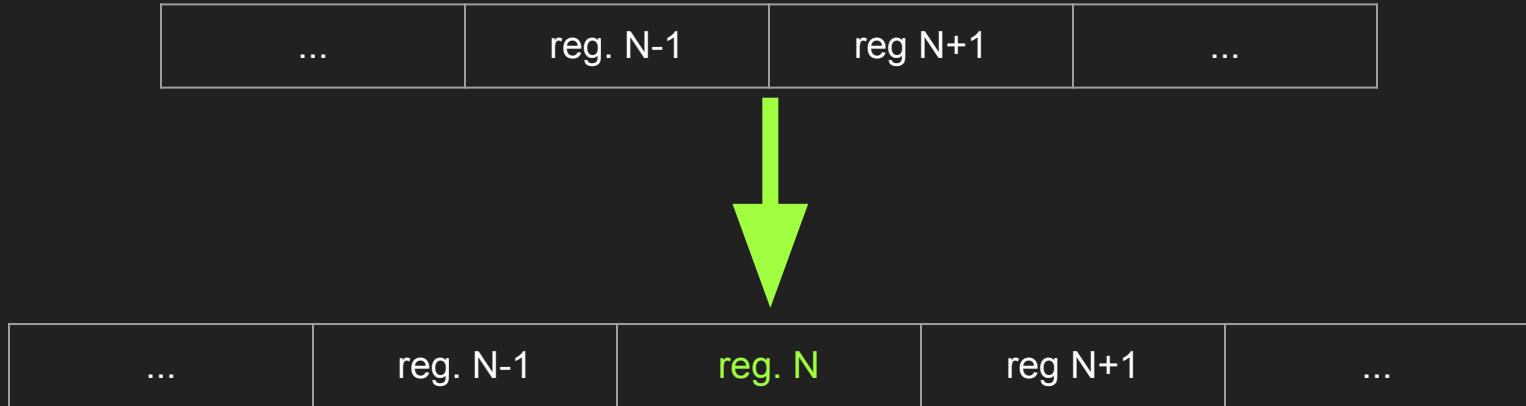
- A organização de um arquivo pode deteriorar durante o seu uso.
- Exemplos:
  - ◆ O que deve ser feito com o espaço liberado por registros removidos?
  - ◆ E quando registros são atualizados em arquivos com registros de tamanho variável?

# Manutenção de Arquivos

- Projetista deve considerar modificações no arquivo
  - ◆ Adição, atualização e remoção de registros
- Problema é simples quando:
  - ◆ Registros são de tamanho fixo
  - ◆ Apenas adições e atualizações ocorrem
- Porém, em outras circunstâncias...

# Manutenção de Arquivos

→ Ex: atualizar um registro variável que aumente de tamanho:



# Manutenção de Arquivos

- O que fazer com os dados adicionais?
  - ◆ Anexar ao final do arquivo e ligar as duas partes por “ponteiros” ?
    - Processamento de cada registro (ao longo do arq. todo) fica muito mais complexo
  - ◆ Apagar o registro original e reescrevê-lo todo no final do arquivo ?
    - Ok, é necessário reutilizar o espaço desocupado

# Manutenção de Arquivos

- Ex: remover um registro (tamanho fixo ou variável):
  - ◆ O que fazer com o espaço remanescente?
    - Necessário reutilizar o espaço vago
- Foco de manutenção de arquivos: reaproveitamento de espaços vagos
- OBS: uma atualização sempre pode ser vista como:
  - ◆ Atualização = Remoção + Adição

# Compactação e Reuso

# Compactação e Reuso

## → Compactação

- ◆ Busca por regiões do arquivo que não contêm dados
- ◆ Recupera os espaços perdidos

## → Reuso

- ◆ Insere dados nos espaços perdidos

## → Duas abordagens

- ◆ Estática
- ◆ Dinâmica



# Abordagem Estática

# Abordagem Estática

- Arquivo off-line e sujeito a modificações esporádicas (ex: lista de mala direta)
  - ◆ Espaços podem ser recuperados em batch

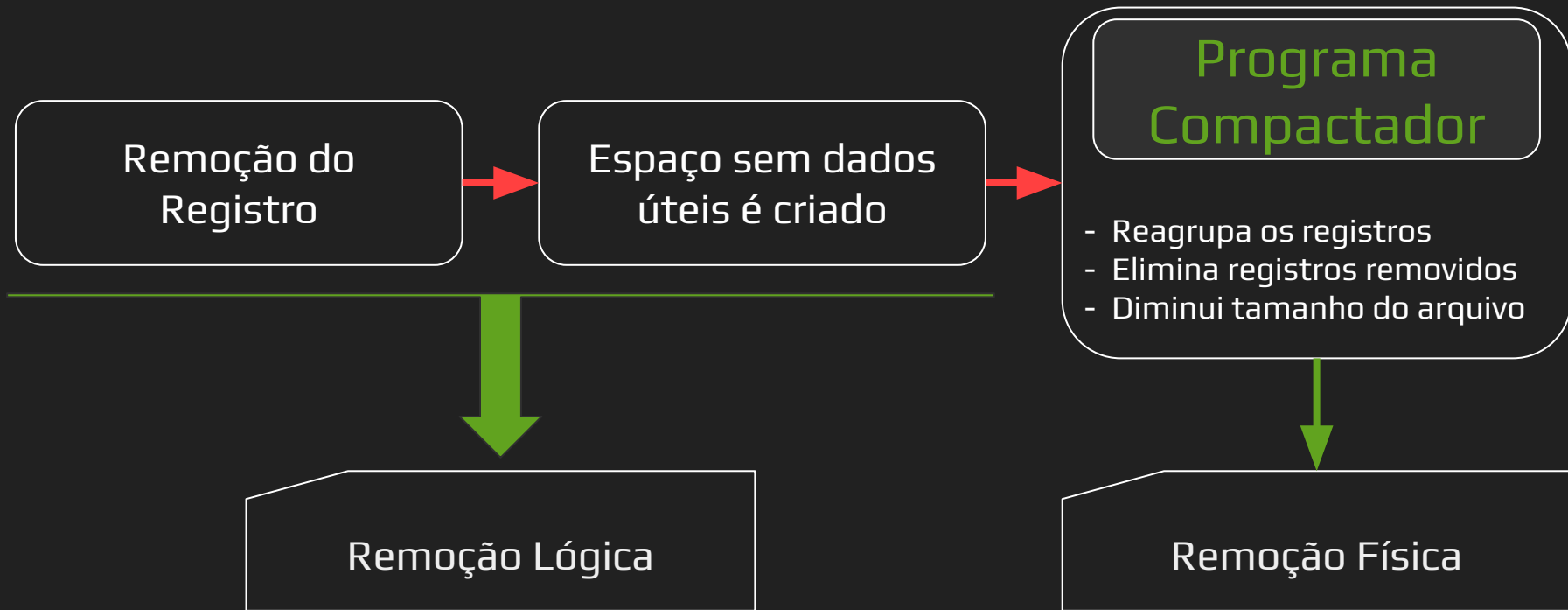
# Abordagem Estática

- Apenas “marcar” os registros no momento da eliminação, e periodicamente eliminá-los todos de uma vez
- ◆ Demanda um mecanismo que permita reconhecer quando uma área do arquivo corresponde a um registro que foi eliminada
- ◆ Usual: marcador especial no lugar do registro apagado (ex: "\*" nos primeiros 2 bytes do registro)

# Abordagem Estática

- Após um certo número de remoções:
  - ◆ Aciona-se um procedimento de compactação

# Abordagem Estática



# Compactação

# Compactação

- Procedimento de compactação: o espaço de todos os registros marcados é recuperado de uma só vez
- ◆ Esporádico
  - Espaço não fica disponível imediatamente após remoção

# Compactação

- Maneira mais simples de compactar, se existe espaço suficiente
  - ◆ Cópia sequencial:
    - Novo arquivo é gerado copiando o original
    - Substitui bytes dos registros eliminados
- Alternativa: compactação in-place
  - ◆ Mais complexa e computacionalmente custosa



## Compactação - Exemplo

→ Ex: Registros de Tamanho fixo e campo Variável

Ana Silva|22|8,0|.....

Jose Dantas|25|7,0|.....

Luiza Aires|19|5,0|.....

## Compactação - Exemplo

→ Após remoção do 2o registro

Ana Silva|22|8,0|.....

\*|se Dantas|25|7,0|.....

Luiza Aires|19|5,0|.....

## Compactação - Exemplo

→ Inserção de um novo registro

Ana Silva|22|8,0|.....

\*|se Dantas|25|7,0|.....

Luiza Aires|19|5,0|.....

Marcos Santos|19|5,0|.....

# Compactação - Exemplo

## → Compactação

Ana Silva|22|8,0|.....

Luiza Aires|19|5,0|.....

Marcos Santos|19|5,0|.....

# Características da Abordagem Estática

# Características da Abordagem Estática

- Técnica pode ser aplicada a
  - ◆ Registros de tamanho fixo
  - ◆ Registros de tamanho variável
- Frequência para se aplicar a técnica
  - ◆ Depende da aplicação
  - ◆ Depende da porcentagem de registros marcados como removidos

# Abordagem Dinâmica

# Abordagem Dinâmica

- Em aplicações on-line, que acessam arquivos altamente voláteis, pode ser necessário um processo dinâmico de recuperação de espaço
  - ◆ Marcar registros eliminados (remoção lógica)
  - ◆ Localizar os espaços desocupados quando necessário
    - **Sem buscas exaustivas !**



# Abordagem Dinâmica

- Ao adicionar um novo registro:
  - ◆ Saber imediatamente se há slots
    - Slot = espaço disponível de um registro removido
  - ◆ Acessar diretamente um slot, se existir
    - Diretamente = sem buscas exaustivas !

# Registros de Tamanho Fixo

# Registros de Tamanho Fixo

- RRN (relative record number)
  - ◆ Fornece a posição relativa de cada registro dentro do arquivo
  - ◆ Permite calcular o byte offset no qual cada registro começa

$$\text{byte offset} = \text{RRN} \times \text{tamanho do registro}$$

# Registros de Tamanho Fixo

## → Exemplo - RRN

- ◆ Registros de tamanho fixo
  - Tamanho de 46 bytes
- ◆ Campos de tamanho fixo
  - Nome: string de 12 caracteres (12 bytes)
  - Rua: string de 10 caracteres (10 bytes)
  - Número: inteiro (4 bytes)
  - Cidade: string de 20 caracteres (20 bytes)

# Registros de Tamanho Fixo

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
M	A	R	I	A	b	b	b	b	b	b	b	R	U	A	b	1	b	b	b	b	b	123			S	A	O	b	
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
C	A	R	L	O	S	b	b	b	b	b	b	b	b	b	b	J	O	A	O	b	b	b	b	b	b	b	b	R	U
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
A	b	A	b	b	b	b	b	255			R	I	O	b	C	L	A	R	O	b	b	b	b	b	b	b	b	b	b
90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
b	b	P	E	D	R	O	b	b	b	b	b	b	b	R	U	A	b	X	V	b	b	b	b	56			S	A	
120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149
O	b	C	A	R	L	O	S	b	b	b	b	b	b	b	b	b	b												

# Registros de Tamanho Fixo

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

M	A	R	I	A	b	b	b	b	b	b	b	R	U	A	b	1	b	b	b	b	b	123			S	A	O	b
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----	--	--	---	---	---	---

30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59

C	A	R	L	O	S	b	b	b	b	b	b	b	b	b	b	J	O	A	O	b	b	b	b	b	b	b	b	R	U
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

60 61 62 63 64 65 66

A	b	A	b	b	b	b
---	---	---	---	---	---	---

90 91 92 93 94 95 96

b	b	P	E	D	R	O
---	---	---	---	---	---	---

Registro	RRN	Byte Offset
primeiro (MARIA)	0	0 x 46 = 0
segundo (JOAO)	1	1 x 46 = 46
terceiro (PEDRO)	2	2 x 46 = 92

120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149

O	b	C	A	R	L	O	S	b	b	b	b	b	b	b	b	b												
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--	--	--	--	--

## Registros de Tamanho Fixo

- Lista encadeada de registros eliminados disponíveis
- Cada elemento da lista armazena o RRN do próximo registro vago
- Cabeça da lista está no header do arquivo:
  - ◆ Cabeçalho (que é um registro) armazena RRN do 1º slot (registro vago)



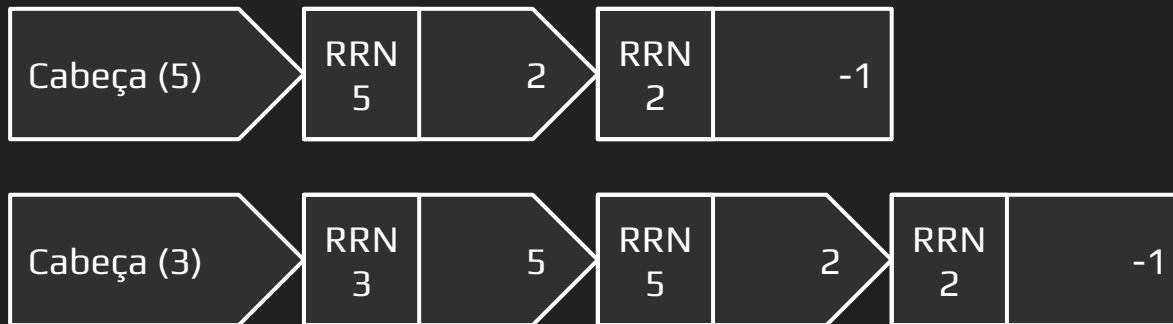
## Registros de Tamanho Fixo

- Inserção e remoção ocorrem sempre no início da lista
  - ◆ Lista encadeada operada como PILHA !
  - ◆ Pilha pode ser mantida no próprio arquivo !



# Registros de Tamanho Fixo

- Exemplo:
  - ◆ Inserção do registro com RRN 3
- inserção na pilha: registro eliminado do arquivo
- remoção da pilha: registro adicionado ao arquivo



# Registros de Tamanho Fixo - Exemplo

→ Arquivo Original

head = -1

Ana Silva|22|8,0|.....

Jose Dantas|25|7,0|.....

Luiza Aires|19|5,0|.....

# Registros de Tamanho Fixo - Exemplo

→ Após remoção do 3º registro

head = 2

Ana Silva|22|8,0|.....

Jose Dantas|25|7,0|.....

\*|-1Aires|19|5,0|.....

# Registros de Tamanho Fixo - Exemplo

→ Após remoção do 1º registro

head = 0

\*|2|va|22|8,0|.....

Jose Dantas|25|7,0|.....

\*|-1|Aires|19|5,0|.....

Após uma inserção?

# Registros de Tamanho Fixo - Exemplo

→ Após remoção do 1º registro

head = 2

Marcos Santos|19|5,0|.....

Jose Dantas|25|7,0|.....

\*|-1Aires|19|5,0|.....

## Registros de Tamanho Fixo

- Implementação: o cabeçalho pode ser implementado como uma struct em C:
  - ◆ Um dos campos armazena o RRN do 1º reg. Vago
    - ex: `int head.first_avail`
  - ◆ Demais campos podem armazenar outras infos
- O arquivo de dados em si começa após os bytes do cabeçalho

# Registros de Tamanho Variável



# Registros de Tamanho Variável

- Registros de tamanho variável: um problema adicional...
- ◆ Registros não são acessíveis por RRN...
  - Não se recupera os byte offsets pelos RRNs
- Solução:
- ◆ Armazenar os byte offsets na lista encadeada
  - ao invés dos RRNs
- ◆ Utilizar registros com indicador de tamanho
- ◆ Permite saber o tamanho do slot a partir do byte offset

# Registros de Tamanho Fixo - Exemplo

→ Arquivo Original

head = -1

15Anal|Sao Carlos|14Jose|Campinas|16Luiza|Sao Paulo|



Indicador de Tamanho

## Registros de Tamanho Fixo - Exemplo

→ Após Remoção do 2º registro

head = 18

15Anal|Sao Carlos|14\*|-1.....|16Luiza|Sao Paulo|

# Registros de Tamanho Variável

- Mas o problema ainda não está resolvido...
- ◆ Registros são de tamanho variável => não é qualquer slot da lista que serve para acomodar um novo registro a ser adicionado
  - É preciso encontrar um slot grande o suficiente
    - Se não for encontrado, adiciona-se ao final do arquivo
  - E para isso, é preciso percorrer sequencialmente a lista

# Registros de Tamanho Variável

→ Adicionar registro de 55 bytes



47? Pequeno

38? Pequeno

72? Suficiente!

# Registros de Tamanho Variável

→ Adicionar registro de 55 bytes



# Fragmentação Interna

## Fragmentação Interna

- No Exemplo: usamos todos os 72 bytes de um slot para adicionar um registro de apenas 55 bytes
- ◆ Os 17 bytes extras dentro do registro ficaram inutilizados

Fragmentação Interna!



## Fragmentação Interna

- Tratamento de fragmentação interna
  - ◆ Os bytes não utilizados no registro é mantido como um slot menor na lista de registros eliminados

# Registros de Tamanho Variável

→ No Exemplo (slot de 72 bytes para um registro de 55):



Por que 13? Não deveria ser 17?



# Registros de Tamanho Variável

→ No Exemplo (slot de 72 bytes para um registro de 55):



Por que 13? Não deveria ser 17?

R: Indicador de tamanho (4 bytes)



# Fragmentação Externa

# Fragmentação Externa

- Após a inserção de um registro de 8 bytes, restam:
  - ◆ 4 bytes para indicador de tamanho
  - ◆ 1 byte para dados
- No entanto... qual a probabilidade de encontrar um ou mais registros que ocupem o espaço (slot) restante?



Fragmentação Externa!

# Fragmentação Externa

- Estratégias para combater a fragmentação externa
  - ◆ Compactação (off-line)
    - Gerar novamente o arquivo de tempos em tempos
  - ◆ Coalescing
    - Buscar por slots disponíveis que sejam fisicamente adjacentes e uni-los em slots disponíveis maiores
    - Dificuldade?

# Fragmentação Externa

- Estratégias para combater a fragmentação externa
  - ◆ Estratégias de Alocação de slots
    - Tentar evitar a fragmentação antes que ela ocorra por meio de estratégias de alocação de slots para novos registros

# Estratégias de Alocação de Slots



# Estratégias de Alocação de Slots

→ First-Fit

◆ Primeiro da lista que seja grande o suficiente

→ Best-Fit

◆ Aquele com tamanho mais parecido ao do registro

→ Worst-Fit

◆ Aquele com o maior tamanho de todos

## First-Fit

- Estratégia mais simples de todas
  - ◆ Requer apenas percorrer a lista
  - ◆ Exatamente o que fizemos até agora
- Na verdade, não tenta prevenir fragmentação
  - ◆ Responsabilidade da compactação e/ou coalescing

## Best-Fit

- Estratégia mais intuitiva de todas
- Requer manter a lista ordenada
  - ◆ Ordenação ascendente com o tamanho dos slots
  - ◆ Demanda tempo computacional extra: não é mais possível sempre adicionar um slot ao início da lista
- Mas, na verdade, pode piorar fragmentação
  - ◆ Se slot não for perfeito, sobra é mínima!

## Worst-Fit

- Estratégia menos intuitiva de todas
- ◆ Requer manter a lista ordenada
  - Ordenação decrescente com o tamanho dos slots
  - Tempo extra é compensado: se 1º slot não acomodar o registro, nenhum outro slot irá!
- ◆ Minimiza fragmentação
  - Já que raramente slot é perfeito, sobra é máxima!

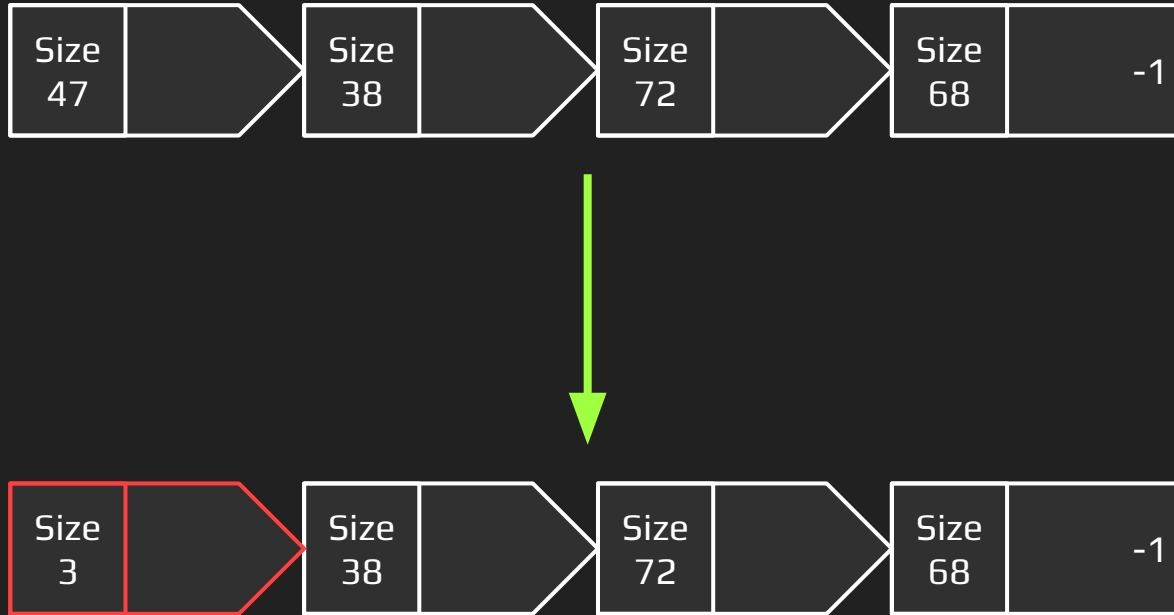
## Exemplo

→ Dado os dados do exemplo que já vimos:

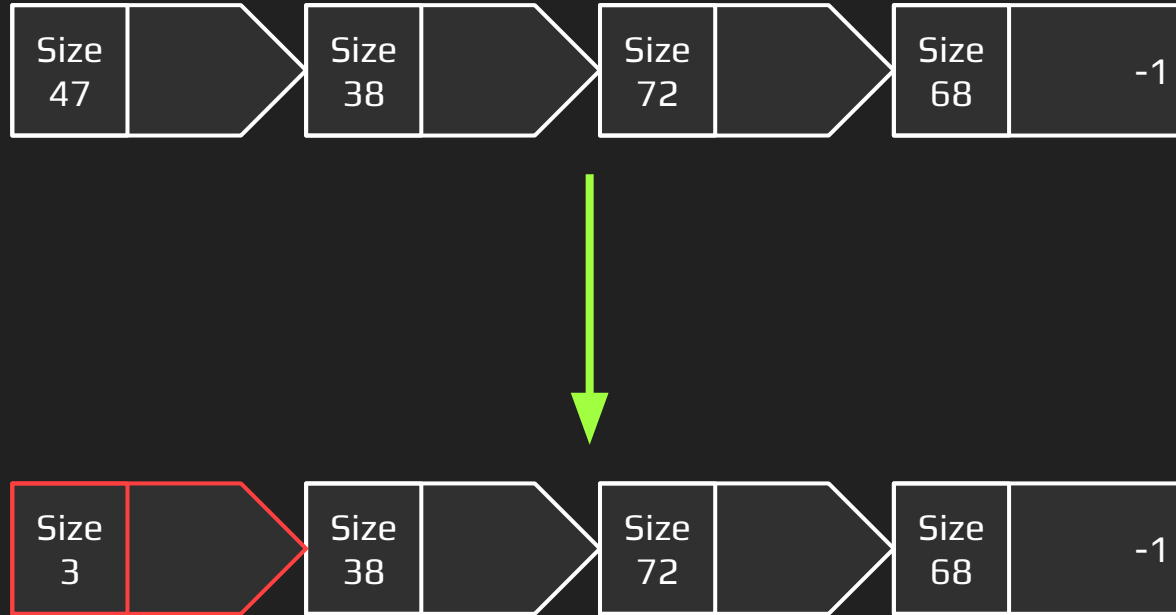


→ Ao inserirmos um registro de 40 bytes, onde ele seria inserido para cada algoritmo?

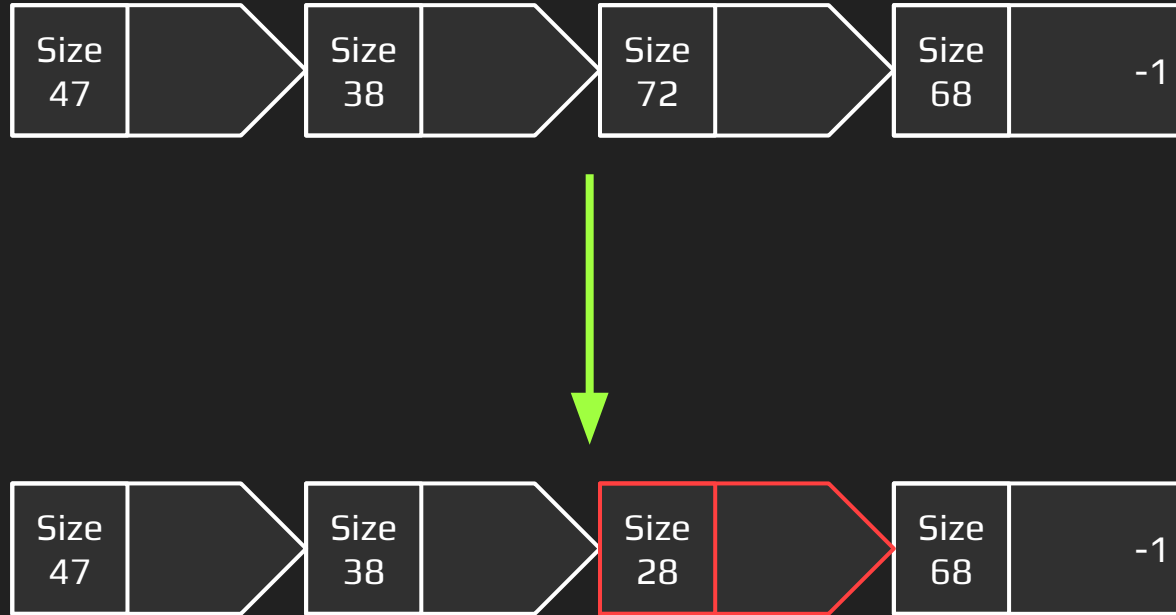
# First-Fit



# Best-Fit



# Worst-Fit





# Compressão

# Compressão

- Reduz arquivos!
  - ◆ Diminui armazenamento (e custo)
  - ◆ Transmissão mais rápida
    - Diminui tempo de acesso
    - Ou o mesmo tempo de acesso com largura de banda menor
  - ◆ Processamento mais rápido

# Compressão

- Envolve codificar informação em um arquivo para que ela ocupe menos espaço
- Existem várias técnicas
  - ◆ Algumas genéricas
  - ◆ Outras específicas para tipos de dado
    - Voz
    - Imagens
    - Texto
    - ...

# Compressão

- Normalmente, usar ou não uma técnica de compressão é situacional
  - ◆ Compressão requer processamento extra
  - ◆ Aumento da complexidade do código
  - ◆ E muitas vezes perda de legibilidade
- É preciso analisar cada caso, e cada algoritmo disponível
  - ◆ Somente usar caso o custo-benefício seja positivo

# Run-Length

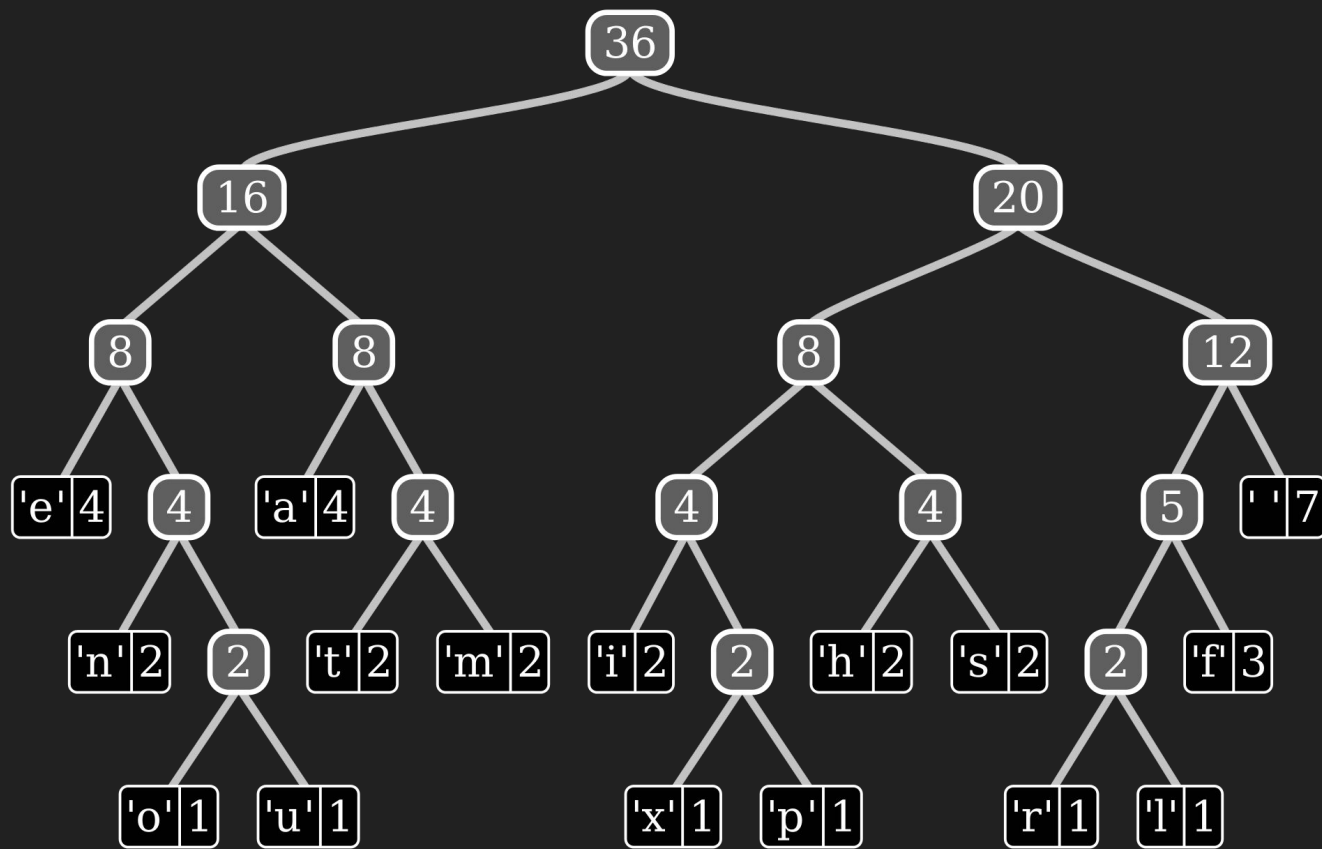


run-length encoding



Fonte: <https://programmersought.com/article/8100198818/>

# Codificação de Huffman



Char	Freq	Code
space	7	111
a	4	010
e	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
l	1	11001
o	1	00110
p	1	10011
r	1	11000
u	1	00111
x	1	10010

"this is an example of a huffman tree"

Fonte: [https://en.wikipedia.org/wiki/Huffman\\_coding](https://en.wikipedia.org/wiki/Huffman_coding)



## Referências

- M. J. Folk, B. Zoellick and G. Riccardi. File Structures: An object-oriented approach with C++, Addison Wesley, 1998.
- M. J. Folk and B. Zoellick, File Structures, Second Edition, Addison Wesley, 1992.