

Testes

Baseado em Arndt von Staa

Especificação

- Objetivo dessa aula
 - Apresentar o processo de criação de casos de teste
 - Apresentar os critérios de teste caixa aberta: cobertura de instruções, de arestas, de decisões e de caminhos
- Referência básica:
 - Capítulo 15

Sumário

- O que são critérios de seleção de casos de teste
- Processo de seleção de casos de teste
- Critérios de cobertura de instruções, de arestas e de decisões
- Teste de repetições, arrasto
- Exemplo de cobertura de decisões
- Critério de cobertura de caminhos
- Transformação de casos de teste abstratos em casos de teste úteis

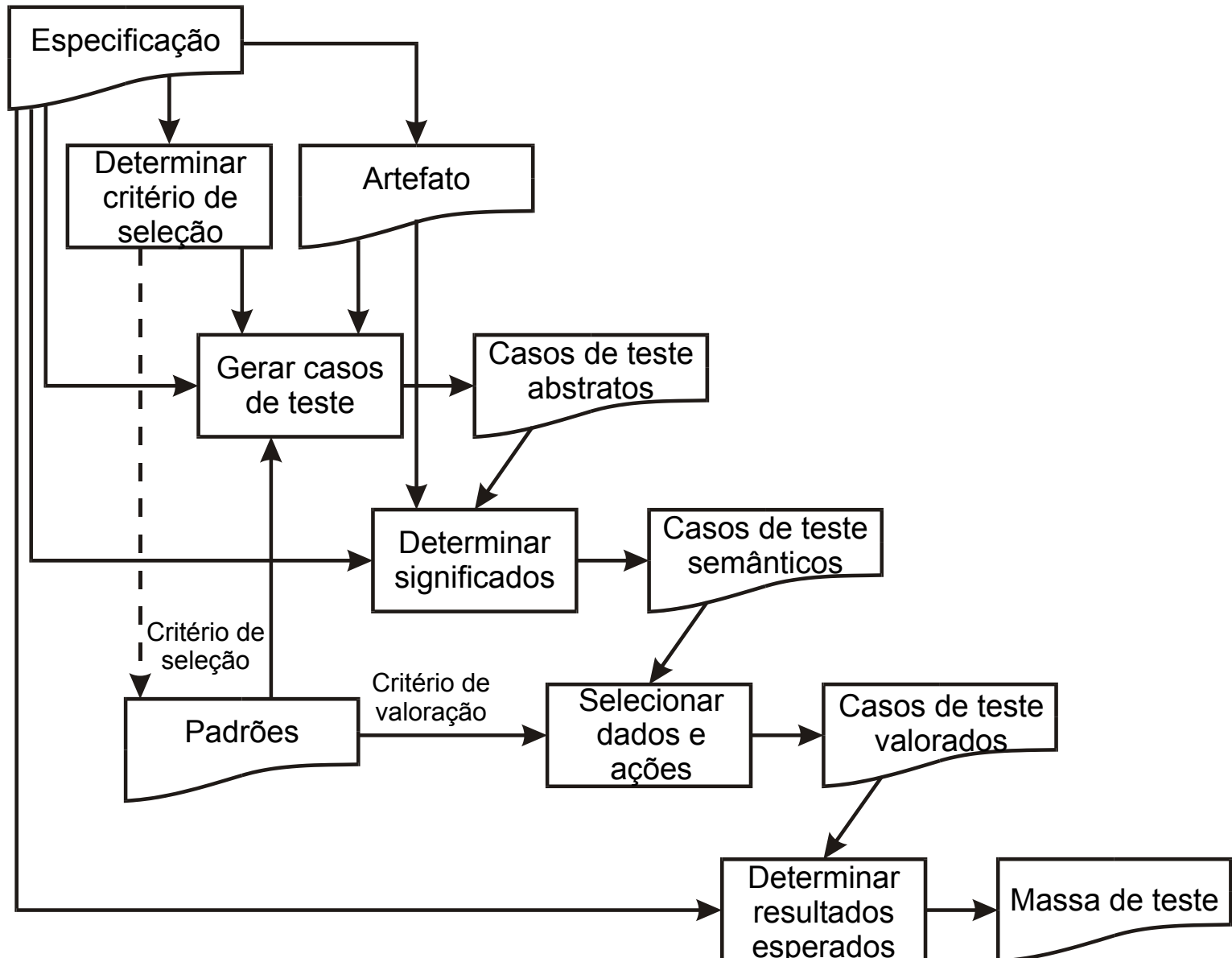
Critérios de seleção de casos de teste

- Critérios de **seleção de casos de teste** são utilizados para gerar os casos de teste que compõem a **massa de teste**
 - a geração pode ser manual, ou parcial ou totalmente automatizada
- **Categorias de critérios** de seleção de casos de teste
 - teste **caixa fechada**
 - gera os casos utilizando somente a especificação
 - a massa pode ser desenvolvida antes ou junto com o código
 - teste **caixa aberta** (teste estrutural)
 - gera os casos de teste utilizando o código completo e a especificação
 - teste de **estruturas de dados**
 - gera os casos de teste utilizando modelos e/ou o código de declaração da estrutura de dados e a especificação

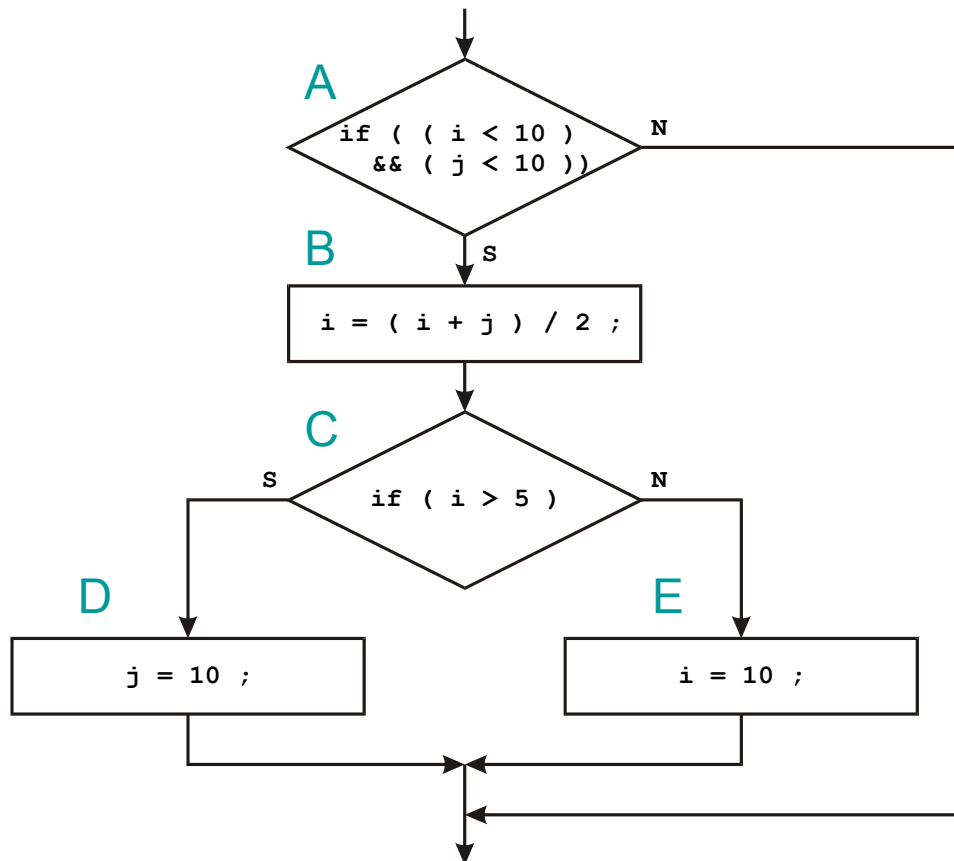
Critérios de seleção de casos de teste

- Um critério de seleção de casos de teste é
 - **válido:**
 - existindo defeitos no artefato testado, acusa falhas que permitam localizar esses defeitos
 - **confiável:**
 - é indiferente à escolha dos dados e ações usados na massa de teste
 - **completo:**
 - exercita todo o código segundo um padrão de completeza
 - **eficaz:**
 - quanto mais falhas encontrar, provocadas por diferentes defeitos
 - quanto mais defeitos for capaz de identificar
 - **eficiente:**
 - quanto menos recursos necessitar para executar os testes

Processo de seleção de casos de teste

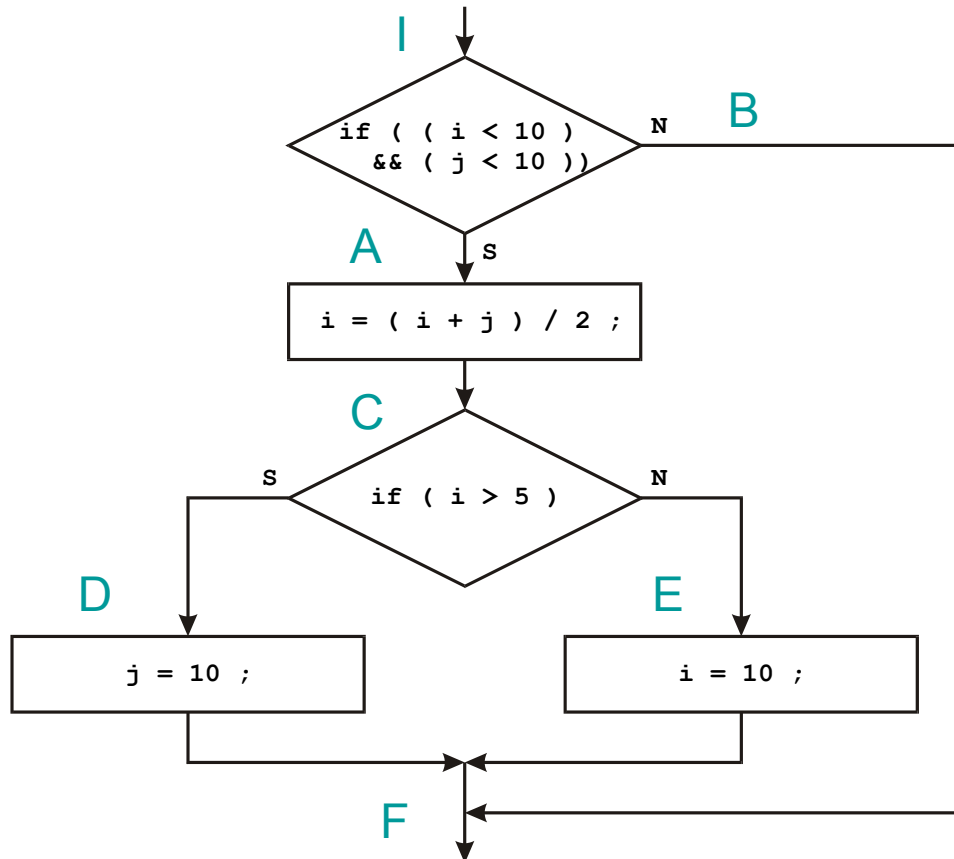


Critérios de cobertura: instruções



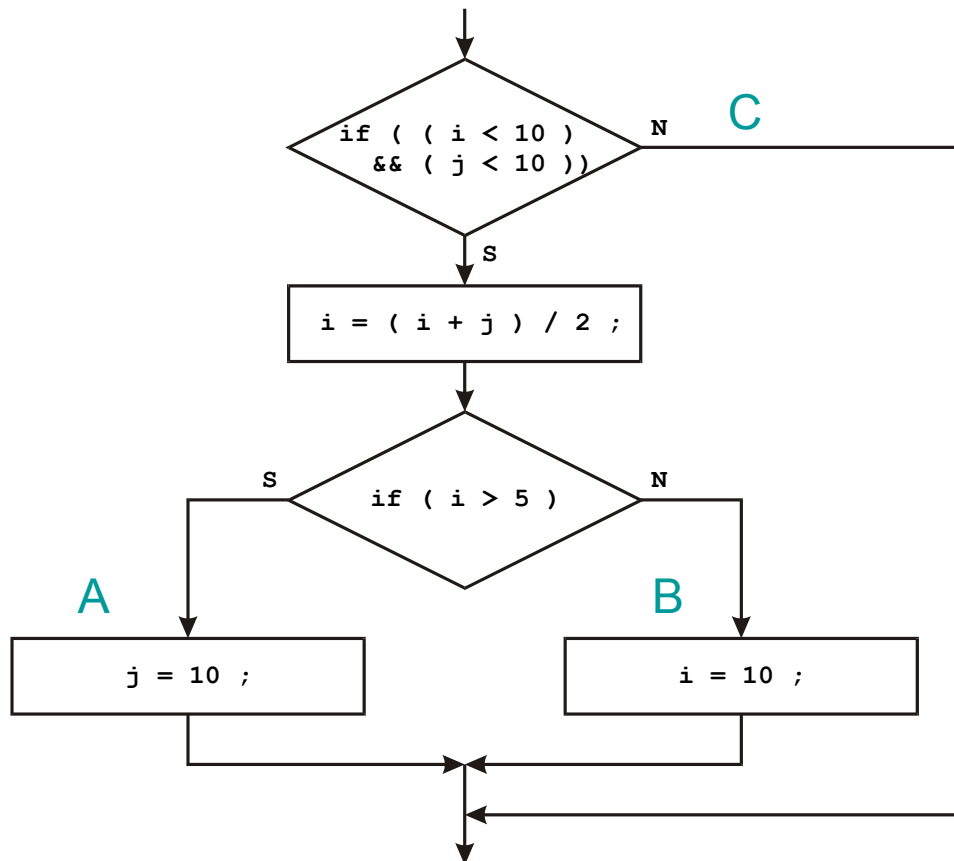
- Cobertura de **instruções**
 - Cada **instrução é executada pelo menos** uma vez no conjunto de todos os casos de teste
 - rotulam-se as instruções e criam-se os casos de teste
 - cada caso percorre pelo menos uma instrução ainda não percorrida
 - até que todas as instruções tenham sido percorridas
 - `i = 4 ; j = 8` → **A B C D**
 - `i = 4 ; j = 6` → **A B C E**

Critérios de cobertura: arestas



- Cobertura de **arestas**
 - Cada **aresta** é percorrida pelo **menos** uma vez no conjunto de todos os casos de teste
 - rotulam-se as arestas e criam-se os casos de teste
 - cada caso percorre pelo menos uma aresta ainda não percorrida
 - até que todas as arestas tenham sido percorridas
 - $i = 3 ; j = 9 \rightarrow I A C D F$
 - $i = 3 ; j = 7 \rightarrow I A C E F$
 - $i = 3 ; j = 10 \rightarrow I B F$

Critérios de cobertura: decisões



- Cobertura de **decisões**

- Cada forma de **avaliar expressões lógicas é exercitada pelo menos** uma vez no conjunto de todos os casos de teste

- `i = 9 ; j = 9` → A

- `i = 3 ; j = 9` → A

- `i = 3 ; j = 7` → B

- `i = 9 ; j = 10` → C

- `i = 10 ; j = 9` → C

- `i = 10 ; j = 10` → C

} critérios de
valoração

Critérios de cobertura: repetições

- Caso cobertura de instrução
 - executar a repetição para $n > 1$ iterações;
- Caso cobertura de arestas
 - executar a repetição para:
 - $n = 0$ iteração
 - $n = 1$ iteração
 - $n \geq 2$ iterações
- Caso cobertura de decisões
 - como na cobertura de arestas
 - em adição: cada uma das possibilidades de se decidir pelo término foi exercitada para os 3 casos acima
 - **break**, ou `return` no corpo da repetição
 - expressão de controle de término composta

Critérios de cobertura: repetições

- O custo do teste cresce com o número de iterações
 - portanto o número de iterações a testar no caso $n > 1$ deverá
 - ser pequeno e
 - ser suficientemente grande para que o teste possa ser assumido válido

Arrasto

- Arrasto
 - é o **maior dos menores** números de iterações necessárias para que todas as variáveis ou estados inicializados antes e modificados durante a repetição passem a **depende exclusivamente de valores computados** em iterações anteriores de uma mesma instância de execução dessa repetição
- Exemplos:
 - `A[0] = 0 ; A[1] = 0 ; A[2] = 0 ; A[3] = 0 ;`
 - `memset(A , 0 , sizeof(A)) ;`
 - `pElem = ProcurarSimbolo(pTabela , pSimbolo) ;`
 - todos têm `Arrasto == 0`
- Arrasto: força de resistência ao avanço de um objeto em um fluido, resultante da ação do meio

Arrasto: exemplos

- `for (i = 0 ; i < 10 ; i++) ...`
- `for (pElem = pOrg ; pElem != NULL ; pElem = pElem->pProx) ...`
- `fgets, fputs, fread, fwrite`
- `tpEstado Corrente ;`

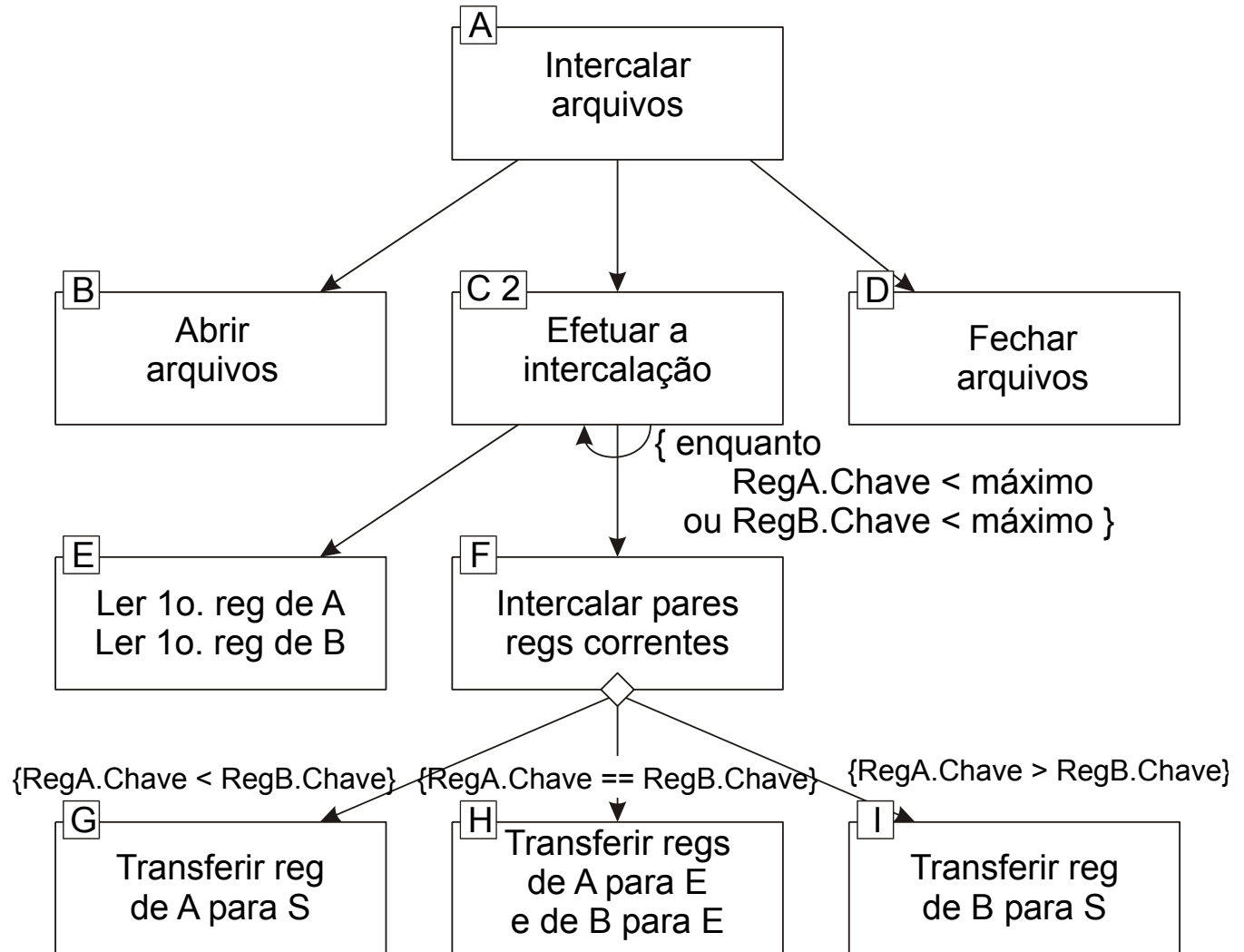
```
Corrente = DefinirPrimeiro( ValorProcurado ) ;
while ( !Terminou( Corrente ))
{
    if ( Comparar( ObterValor( Corrente ), ValorProcurado )
        == EH_IGUAL )
    {
        return Corrente ;
    } /* if */
    Corrente = DefinirProximo( Corrente , ValorProcurado ) ;
} /* while */
return ESTADO_NIL ;
```

- Todos têm `Arrasto == 1`

Arrasto: exemplos

- Série de Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
 - lei de formação $F_n = F_{n-2} + F_{n-1}$
 - tem Arrasto == 2

Arrasto: exemplos



- Tem Arrasto == 2

Critérios de cobertura: repetições

- O arrasto é o número mínimo de iterações a realizar para que todos os valores que possam ser modificados durante a repetição tenham sido de fato modificados
 - corresponde ao número mínimo de iterações para atingir o estado “genérico”
 - é função do projetista determinar o arrasto
- Os casos de teste a para as repetições são:
 - caso 0 iteração (caso especial)
 - caso 1 iteração (base da indução)
 - caso $n \geq \text{arrasto} + 1$ iterações (simula o passo de indução)
 - devem sempre ser considerados os casos de término:
 - **break** ou **return** no corpo da iteração
 - atingiu a condição de término

Critérios de cobertura: repetições

- A preocupação ao testar programas é obter uma informação confiável a respeito da qualidade do artefato
 - pode requerer o uso de dados pouco plausíveis do ponto de vista do uso produtivo do programa, exemplos:
 - intercalar dois arquivos vazios produzindo arquivos vazios
 - procurar um símbolo em uma tabela vazia
 - calcular o produto de matrizes de tamanho 0×0

Critério cobertura de decisões: exemplo

- Esquema do algoritmo para pesquisa em qualquer tabela

```
tpEstado Corrente ;  
Corrente = DefinirPrimeiro( ValorProcurado ) ;  
while ( !Terminou( Corrente ) )  
{  
    if ( Comparar( ObterValor( Corrente ),  
                  ValorProcurado ) == EH_IGUAL )  
    {  
        return Corrente ;  
    } /* if */  
    Corrente = DefinirProximo( Corrente , ValorProcurado ) ;  
} /* while */  
return ESTADO_NIL ;
```

- arrasto == 1

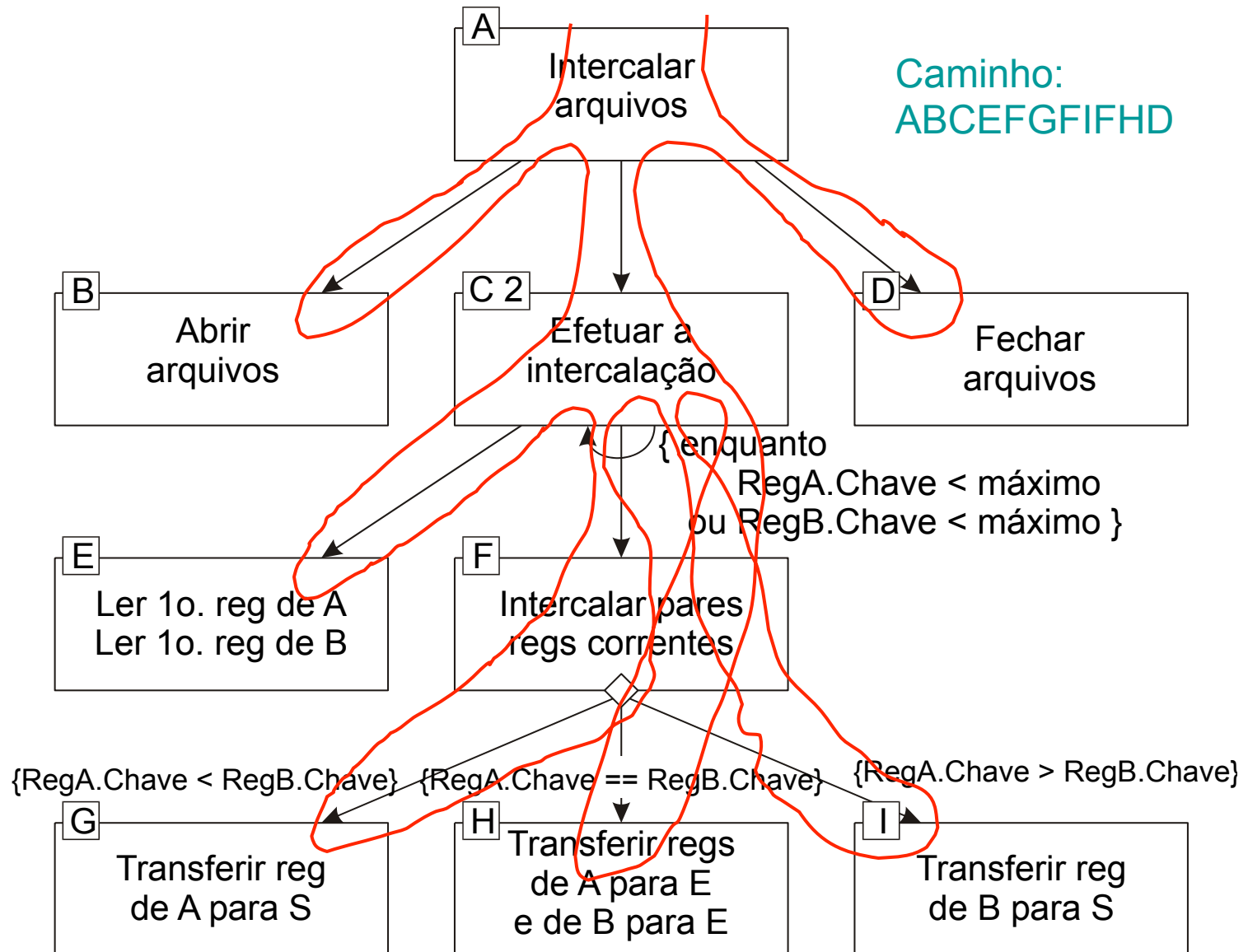
Critério cobertura de decisões: exemplo

- Caso 0 iterações:
 - tabela vazia
 - tabela com 1 ou mais elementos e acha o primeiro elemento
- Caso 1 iteração:
 - tabela com 1 elemento e não acha o elemento
 - tabela com 2 ou mais elementos e acha o segundo elemento
- Caso arrasto + 1 iterações:
 - tabela com 2 elementos e não acha o elemento
 - tabela com 3 ou mais elementos e acha o segundo elemento

Cobertura de caminhos

- Um **caminho** (arco de execução) corresponde a uma seqüência de execução de comandos dentro de um programa
 - muitas vezes é restrito ao corpo de uma função
- Depende de
 - seleções realizadas (`if`, `switch`, `throw / catch`)
 - número de iterações de repetições e de chamadas recursivas
 - formas de terminar (`break`, `return`, `throw`)
- Um programa que contém repetições admite, de maneira geral, um número infinito de caminhos

Cobertura de caminhos: exemplos de caminhos



Cobertura de caminhos

- O critério cobertura de caminhos seleciona um conjunto de caminhos
 - cada caminho é um **caso de teste abstrato**
 - os dados devem ser escolhidos para que se execute exatamente o caminho escolhido
 - o conjunto de caminhos forma a massa de testes
- Para manter pequeno o custo do teste deseja-se
 - um conjunto pequeno de caminhos curtos
 - o conjunto de caminhos deve
 - ser completo \Rightarrow exercitar todo o código
 - simular a argumentação da corretude
 - auxiliar a argumentação da corretude
 - utilizar a estrutura do código

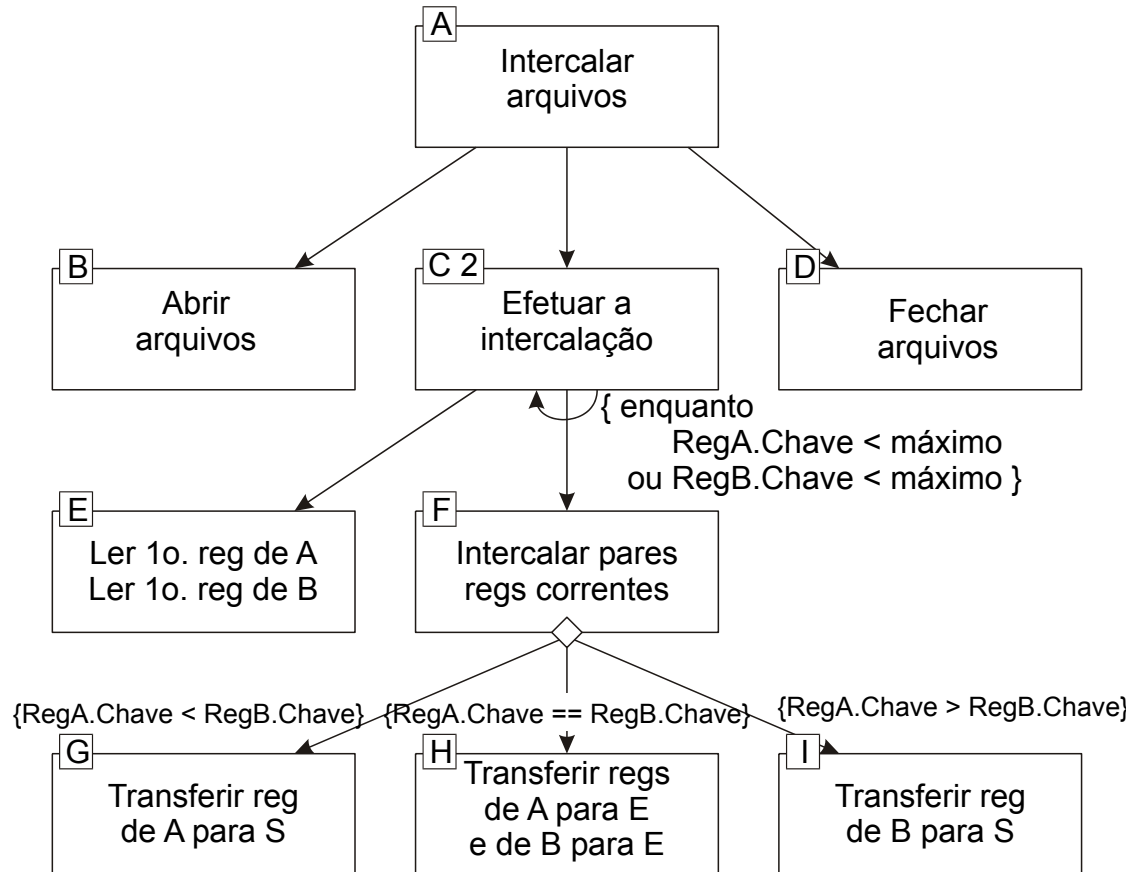
Cobertura de caminhos

- Calcula-se a expressão algébrica dos caminhos do procedimento.
 - em um programa estruturado será sempre uma *expressão regular*
 - caminha-se na estrutura do código do início para o fim (se projeto estruturado: em ordem prefixada pela esquerda)
 - externa-se o rótulo do bloco visitado
 - se o bloco for início de controle de repetição externa-se '['
 - calcula-se o arrasto e externa-se o valor *arrasto* + 1
 - ao terminar o controle de repetição e o bloco externa-se ']'
 - se for início de controle de seleção externa-se '('
 - ao atingir um **else** ou **else if** externa-se '|'
 - ao terminar o controle de repetição externa-se ')'

Cobertura de caminhos

	Expressão de caminhos
// (A) Intercalar arquivos	
// (B) Abrir arquivos	
// (E) Inicializar	
// Ler primeiro registro de arquivo A	A
// Ler primeiro registro de arquivo B	
// (C 2) Intercalar pares de registros	ABE
while (tem registro a processar)	ABEC[3
// (G) Transferir registro de A para S	ABEC[3(G
if (chave buffer A < chave buffer B)	ABEC[3(G H I
transferir de buffer A para S	ABEC[3(G H I)D
// (H) Transferir registro de A e B para D	
else if (chave buffer A ==	
chave buffer B)	
transferir registro de A para D	
transferir registro de B para D	
// (I) Transferir registro de B para S	
else	
transferir de buffer B para S	
// (D) Fechar arquivos	

Cobertura de caminhos



Expressão de caminhos

AB

ABCE

ABCE[3F

ABCE[3F(G


ABCE[3F(G|H|I)]D

Cobertura de caminhos

- $A B C E [{}_3 F (G | H | I)] D$
- Cria-se a **lista dos caminhos a testar**, usando a expressão como **gerador** → casos de teste abstratos
 - 0 ciclos (caminhos dos casos especiais)
 - **ABCED**
 - 1 ciclo (caminhos da base da indução)
 - **ABCEFGD**
 - **ABCEFHD**
 - **ABCEFID**
 - 3 (= arrasto + 1) ciclos (caminhos do passo de indução)
 - **ABCEFGFGFGD**
 - **ABCEFGFGFHD**
 - **ABCEFGFGFID**
 - **ABCEFGFHFHD**
 - **ABCEFGFHFID**
 - **...**

Cobertura de caminhos: problema

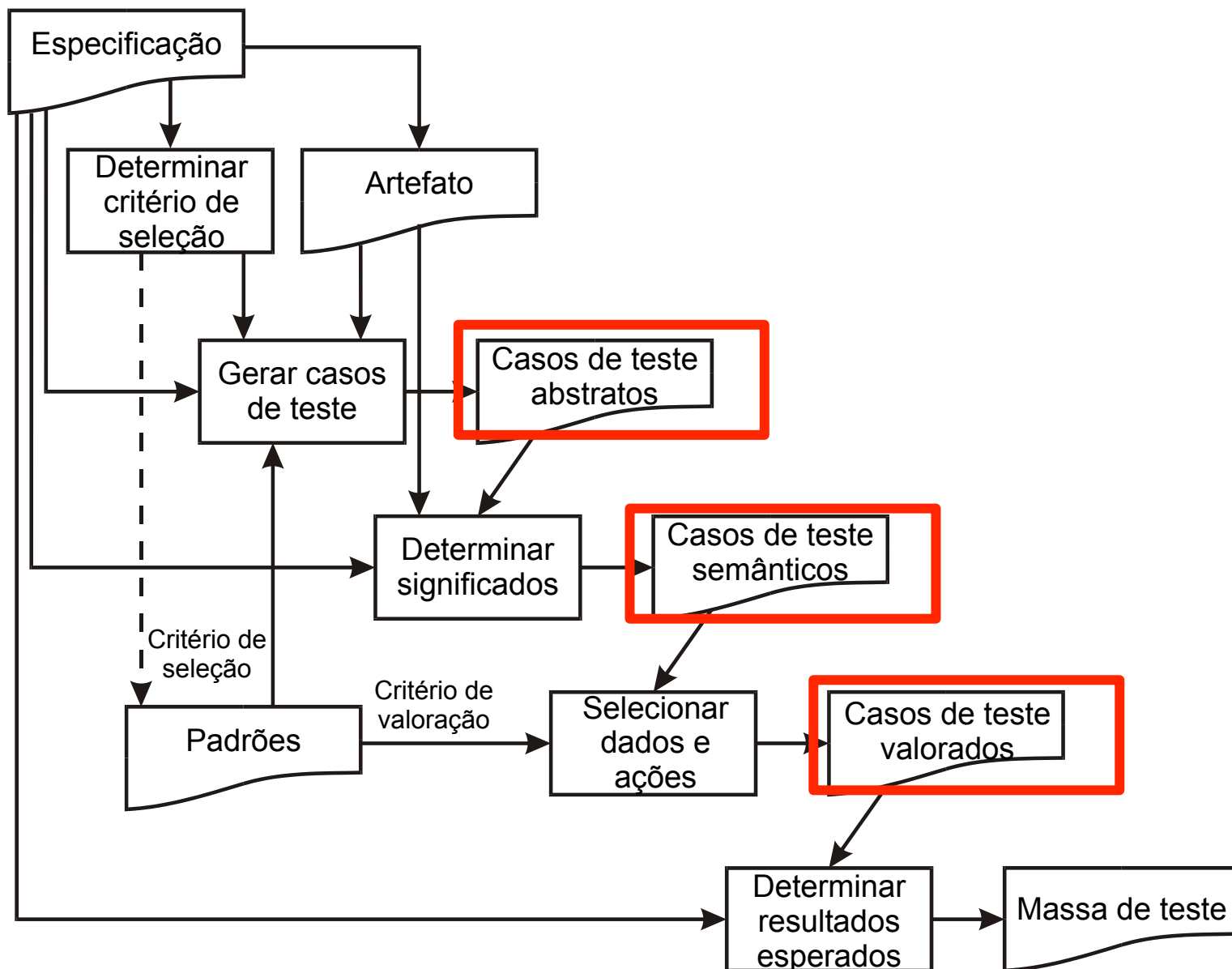
- O número de casos de teste gerados para um determinado algoritmo cresce **multiplicativamente** em função do número de alternativas em **controles** (**if, while**) **aninhados** $\rightarrow O(n^k)$

da ordem de 

$$O \left(\prod_{i=1}^{numControles} numAlternativasControle[i] \right)$$

- No caso da intercalação:
 - 0 iterações \rightarrow 1 caso 1
 - 1 iteração \rightarrow 3 casos 3
 - 3 iterações \rightarrow 3 ** 3 casos 27 total 31 casos
- Solução que em geral dá certo
 - particionar os aninhamentos criando diversas funções

Transformação em caso de teste úteis



Transformação em caso de teste úteis

- Casos de teste abstratos estabelecem as condições a serem satisfeitas pelos casos de teste. Exemplo os caminhos:
 - ABCED
 - ABCEFGD
 - ABCEFHD
 - ABCEFID
 - ABCEFGFGFGD
 - ABCEFGFGFHD
 - ABCEFGFGFID
 - ABCEFGFHFHD
 - ABCEFGFHFID
 - . . .

Transformação em caso de teste úteis

- Casos de teste **semânticos** determinam o **significado das condições** a serem satisfeitas pelos casos de teste
 - **ABCED**
 - Arq A vazio, Arq B vazio
 - **ABCEFGD**
 - Arq A com 1, Arq B vazio
 - **ABCEFHD**
 - Arq A com 1, Arq B com 1 e chaves de ambos iguais
 - **ABCEFID**
 - Arq A vazio, Arq B com 1
 - **ABCEFGFGFGD**
 - Arq A com 3, Arq B vazio
 - **ABCEFGFGFHD**
 - Arq A com 3, Arq B com 1, chave em B igual à chave do 3º. em A
 - . . .

Transformação em caso de teste úteis

- Casos de teste **valorados** determinam **os valores e comandos a serem utilizados** como dados dos casos de teste
 - Criar arquivos $A0 = \{\}$, $B0 = \{\}$, $A1 = \{ c1 \}$, $B11 = \{ c1 \}$, $A3 = \{ c1, c2, c3 \}$, $B12 = \{ c3 \}$, . . .
 - Arq A vazio, Arq B vazio = $A0, B0$
 - Arq A com 1, Arq B vazio = $A1, B0$
 - Arq A com 1, Arq B com 1 e chaves de ambos iguais = $A1, B11$
 - Arq A vazio, Arq B com 1 = $A0, B11$
 - Arq A com 3, Arq B vazio = $A3, B0$
 - Arq A com 3, Arq B com 1, chave em B igual à chave do 3º. de A = $A3, B12$
 - . . .

Transformação em caso de teste úteis

- Casos de teste **úteis** estabelecem os **resultados esperados** em função dos dados
 - Criar arquivos $S0 = \{\}$, $E0 = \{\}$, $S1 = \{ c1 \}$,
 $E11 = \{ c1 , c1 \}$, $S2 = \{ c1 , c2 \}$, $S3 = \{ c1 , c2 , c3 \}$,
 $E12 = \{ c3 , c3 \}$, . . .
 - $A0 , B0 \quad \rightarrow S0 , E0$
 - $A1 , B0 \quad \rightarrow S1 , E0$
 - $A1 , B11 \quad \rightarrow S0 , E11$
 - $A0 , B11 \quad \rightarrow S1 , E0$
 - $A3 , B0 \quad \rightarrow S3 , E0$
 - $A3 , B12 \quad \rightarrow S2 , E12$
 - . . .

Transformação em caso de teste úteis

- Dica para automação
 - criar todos os arquivos requeridos
 - criar um programa de teste específico que
 - para cada caso de teste recebe linhas com os nomes dos 4 arquivos
 - executa a intercalação com os dois primeiros arquivos
 - compara os arquivos resultantes com os dois últimos
 - o programa de intercalação pode ser implementado como uma função que recebe quatro parâmetros
 - neste caso pode-se utilizar o arcabouço de apoio ao teste
 - o programa de intercalação pode ser implementado como um programa principal recebendo 4 parâmetros da linha de comando
 - neste caso pode-se implementar um batch (`.bat`) ou um programa LUA que controla a execução

Exemplo de roteiro de teste

- Um **roteiro de teste** estabelece um **cenário de teste** e fornece uma série de **instruções** a serem utilizadas durante os testes manuais ou automatizados:

== Intercalar A e B vazios

=intercalar A0 B0 S0 E0

== Intercalar A contendo 1 registro com B vazio

=intercalar A1 B0 S1 E0

== Intercalar A e B contendo 1 registro, chaves iguais

=intercalar A1 B11 S0 E11

== Intercalar A vazio com B contendo 1 registro

=intercalar B0 A1 S1 E0

. . .

FIM