



## Teste de Software

Arineiza Cristina Pinheiro

SCC0202 – Algoritmos e Estruturas de Dados I

## Início

Pergunta:



Resposta: Grace Murray Hopper



## Início

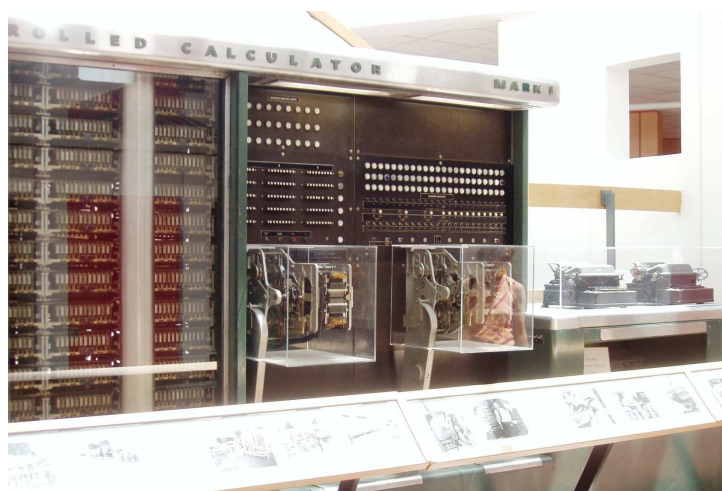
- ▶ **Grace Murray Hopper (10/12/1906 – 1/1/1992)**
  - ▶ Cientista da Computação
  - ▶ Contra-Almirante da Marinha Americana
    - ▶ Entre as décadas de 40 e 60
  - ▶ **E...?**
    - ▶ Criou o primeiro compilador
      - ▶ A-0, A-1, A-2 e A-3 (MATH-MATIC)
    - ▶ Criou o conceito de **bibliotecas de rotinas**
    - ▶ Linguagem baseada em inglês
      - ▶ B-0 (FLOW-MATIC)
    - ▶ COBOL (COmmon Business Oriented Language).



▶ 3

## Início


- ▶ **Harvard Mark I**



▶ 4

## Início

### ▶ Diário de Bordo

0800 Antan started  
 1000 stopped - antan ✓  
 1300 (033) MP-MC ~~1.52177000~~ { 1.2700 9.037 847 025 }  
 (033) PRO 2 2.130476415 ~~2.130476415~~ (033) 4.615925059(-2) 9.037 846 985 connect  
 connect 2.130676415  
 Relays 6-2 in 033 failed special speed test  
 in relay .. 11,000 test. Relay  
3145  
Relay 3377  
 Relays changed  
 1100 Started Cosine Tape (Sine check)  
 1525 Started Multi-Adder Test  
 1545  Relay #70 Panel F  
 (moth) in relay.  
 First actual case of bug being found.  
 1630 Antan started.  
 1700 closed down.

“O primeiro caso real de inseto (Bug) a ser encontrado”

▶ 5

## Testar por quê?

- ▶ 1º Errar é humano!
  - ▶ Ou seja, erros acontecem
- ▶ 2º Situações imprevisíveis
  - ▶ Usários são enorme fonte de comportamentos inesperados
- ▶ 3º Software simples?
  - ▶ Esqueça!
  - ▶ Novos domínios e alta complexidade
- ▶ 4º Falta de métodos durante o desenvolvimento
  - ▶ Códigos “write-only”
- ▶ ...

▶ 6

## *Houston, we have a problem!*

---

- ▶ Softwares espaciais são campeões de bugs
  - ▶ Software complexo e crítico
  - ▶ Falhas levam a perdas gigantescas!
  
- ▶ Exemplos:
  - ▶ Phobos II (Marte) – 1989 (Russo)
  - ▶ Ariane 5 – 1996 (Europeu)
  - ▶ Pathfinder (Marte) – 1997 (Americano)

---

▶ 7

## Phobos II

---

- ▶ Lançado 12 de julho.
- ▶ Sumiu 2 de setembro.
  - ▶ Foi emitido um comando de terra que desabilitou o sistema referencial na nave.
  - ▶ Com isso ela perdeu o sol e não pôde recarregar as baterias solares.
- ▶ Por que alguém iria emitir tal comando?
  - ▶ Falha humana.
- ▶ Por que uma nave deveria ter uma funcionalidade para desabilitar o sistema de orientação?
  - ▶ Não deveria.
  - ▶ Útil para algumas rotinas de teste.
  - ▶ Software em ROM.

---

▶ 8

## Lições

---

- ▶ 1º - Teste é teste, produção é produção.
- ▶ 2º - Mantenha o software simples, remova o desnecessário.
- ▶ 3º - Se alguma coisa pode dar errado, vai dar.

---

▶ 9

## Ariane 5

---

- ▶ Após 40 segundos do lançamento, o foguete perdeu completamente sua orientação, tombou e se autodestruiu.
- ▶ Foi determinado que o sistema de orientação deixou de funcionar por causa de um conversão de tipos.
- ▶ Ao tentar converter um valor real em inteiro, houve uma exceção que resetou o computador que cuidava da orientação.
- ▶ Ou seja, quando consultado para verificar a posição do foguete estava em processo de inicialização e devolveu valor expúrio.

---

▶ 10

## Ariane 5

---

- ▶ 1° - Software foi reaproveitado do Ariane 4.
  - ▶ Existia um sistema de ajuste do foguete na plataforma, antes de ser lançado.
  - ▶ Sistema continuava operante por 40 segundos depois era desligado.
- ▶ 2° - Esse sistema não era usado no Ariane 5, mas estava presente.
  - ▶ Com a trajetória do Ariane 5, ocorria um erro de conversão que era tratado (ou não) resetando-se o computador.
  
- ▶ Portanto, sistema não foi testado com a nova trajetória

---

▶ 11

## Lições

---

- ▶ 4° - Reúso de software é ótimo
  - ▶ Mas a necessidade de teste é a mesma
- ▶ 5° - Testar situações de exceção
  - ▶ Difícil de prever
- ▶ 6° - Não manter coisas desnecessárias ou perigosas.
- ▶ 7° - Tamanho do defeito não reflete tamanho da falha

---

▶ 12

## Pathfinder

---

- ▶ Sonda de exploração do solo marciano.
- ▶ Inaugurou diversos conceitos
  - ▶ Uso de airbags para permitir o pouso.
- ▶ Colhia dados por um longo tempo e depois os transmitia para a Terra.
- ▶ Ou melhor: colhia dados por um longo tempo, resetava sozinho e perdia todos os dados

---

▶ 13

## Pathfinder

---

- ▶ Software do robô é concorrente
  - ▶ Com escalonamento preemptivo.
  - ▶ Cada thread possui uma prioridade.
- ▶ “Information bus” é uma memória compartilhada que serve para trocar informação entre diversas partes do sistema.
  - ▶ Acesso controlado por mutex.
  - ▶ Gerenciador do I.B.: roda freqüentemente, com alta prioridade.
- ▶ Thread meteorológica
  - ▶ Roda de vez em quando, com baixa prioridade e publica dados no I.B.
- ▶ Thread de comunicação
  - ▶ Longa, e com média prioridade

---

▶ 14

## Pathfinder

---

- ▶ Essa combinação geralmente funciona bem.
  - ▶ Situação de erro: Gerenciador do I.B. bloqueado no mutex.
- ▶ Comunicação é escalonada e ganha processador pois tem prioridade maior que a meteorologia.
  - ▶ Comunicação demora quanto tempo quiser.
- ▶ Timer expira indicando que gerenciador do I.B. não foi executado por um longo período de tempo.
  - ▶ Ação corretiva: reset.

---

▶ 15

## Pathfinder

---

- ▶ Horas de execução em “debug mode” salvaram o projeto
  - ▶ Além de sorte
- ▶ Alterando valor de uma constante no programa, o problema foi resolvido.
  - ▶ Herança de prioridade: quando o gerenciado do I.B. ficou bloqueado no mutex, a meteorologia iria herdar sua prioridade.
  - ▶ Isso evitaria que a thread de comunicação executasse por muito tempo e que o gerenciador ficasse sem executar.

---

▶ 16



## Lições

- ▶ 8º - Mecanismos de depuração em alguns casos são essenciais.
- ▶ 9º - Meio para corrigir o problema de maneira fácil é benvindo (flexibilidade)
- ▶ 10º - Aplicar técnicas de teste adequadas ao domínio.
  - ▶ O que diferencia o laboratório de teste do ambiente real?
  - ▶ Não desprezar indício de defeitos “it was probably caused by a hardware glitch”.
- ▶ Portanto:

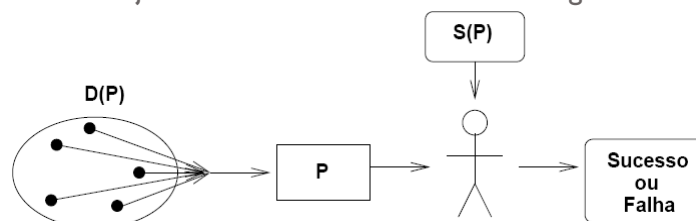
**Software vai falhar!**

Objetivo do Teste: Minimizar as chances de falha

▶ 17

## Então...

- ▶ Como testar?
  - ▶ Executar todas as entradas possíveis!
- ▶ Teste exaustivo
  - ▶ Programa P:  $x^y$
  - ▶ Entradas: todos os possíveis pares de inteiros  $(x, y)$
  - ▶ Saídas: conjunto de números inteiros e mensagens de erros



▶ 18

## Algumas definições

- ▶ **Dado de teste**
  - ▶ Um elemento do domínio de entrada do programa P
- ▶ **Caso de teste**
  - ▶ Par formado por um dado de teste mais o resultado esperado pela execução do programa com aquele dado de teste
  - ▶ Programa:  $x^y$ 
    - ▶  $\langle(2, 3), 8\rangle, \langle(4, 3), 64\rangle, \langle(3, -1), \text{"Erro"}\rangle$
- ▶ **Conjunto de Teste**
  - ▶ ou Conjunto de Casos de Teste
  - ▶ Conjunto de todos os casos de teste usados durante uma determinada atividade de teste

▶ 19

## CrITÉrios e TÉcnicas de Seleção

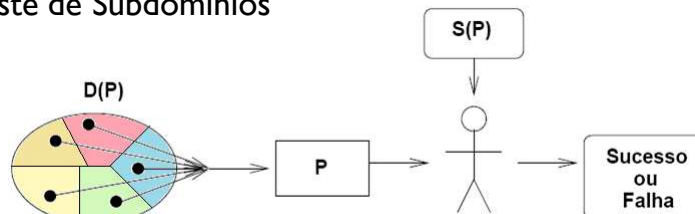
- ▶ **Teste Exaustivo**
  - ▶ Inviável
  - ▶ Programa P:
    - ▶  $2^n * 2^n$  (n é o número de bits usado para representar um inteiro)
    - ▶ Arquitetura de 32 bits:  $2^{64} = 18.446.744.073.709.551.616$
  - ▶ Cada teste executado em 1ms
    - ▶ 5.849.424 séculos para executar todos
- ▶ **Teste Randômico**
  - ▶ Não garante quais trechos serão executados
    - ▶ Trechos críticos



▶ 20

## Critérios e Técnicas de Seleção

### ▶ Teste de Subdomínios



### ▶ Como particionar?

- ▶ Definir regras
  - ▶ Requisitos de Teste
  - ▶ Como executar uma determinada estrutura do programa
  - ▶ Os dados que satisfazem esse requisito pertencem ao mesmo subdomínio

▶ 21

## Teste de Subdomínios

### ▶ Regras

- ▶ Critérios de Teste
- ▶ Geram Requisitos de Teste



### ▶ Critérios

- ▶ Funcionais
- ▶ Estruturais
- ▶ Baseados em defeitos (ou erros)

### ▶ Um conjunto de teste que satisfaz todos os requisitos de um critério de teste C

- ▶ C-adequado

▶ 22

## Objetivo

---

- ▶ **Mostrar que um programa está correto?**
  - ▶ Não!
- ▶ **Revelar a presença de defeitos**
  - ▶ Caso existam
- ▶ **Teste criterioso e embasado tecnicamente**
  - ▶ “Confiança”
  - ▶ Comportamento correto para grande parte do domínio de entrada



---

▶ 23

## Técnicas

---

- ▶ **Funcional**
  - ▶ Teste baseado na especificação
  - ▶ Confronta: saída obtida x saída esperada
  - ▶ Teste Caixa Preta
- ▶ **Estrutural**
  - ▶ Teste baseado na estrutura interna do programa
  - ▶ Execução de partes ou módulos elementares do código
  - ▶ Teste Caixa Branca
- ▶ **Baseado em defeitos**
  - ▶ Teste baseado nos erros típicos cometidos durante o processo de desenvolvimento

---

▶ 24

## Teste Estrutural

```

void Matriz_AtribuiElemento (Matriz* pMatriz, int iLinha, int
                             iColuna, float fValor, int *iErro) {
    int iIndice;
    if(pMatriz != NULL){
        if (iLinha < 0 || iLinha >= pMatriz->iLin ||
            iColuna < 0 || iColuna >= pMatriz->iCol) {
            *iErro = ERRO_ENDERECO_INVALIDO;
        }
        else{
            iIndice = (iLinha - 1) * pMatriz->iCol + iColuna;
            pMatriz->fVet[iIndice] = fValor;
            *iErro = ERRO_SUCESSO;
        }
    }
    else{
        *iErro = ERRO_PONTEIRO_NULO;
    }
}

```

▶ 25

## Conceitos

- ▶ **Grafo de Fluxo de Controle (GFC)**
  - ▶ Ou Grafo de Programa
  - ▶ Representação gráfica de um programa
- ▶ **Nós**
  - ▶ Um nó no grafo representa um bloco indivisível de comandos
- ▶ **Arcos**
  - ▶ Cada aresta representa um possível desvio de um bloco para outro



▶ 26

## Definição

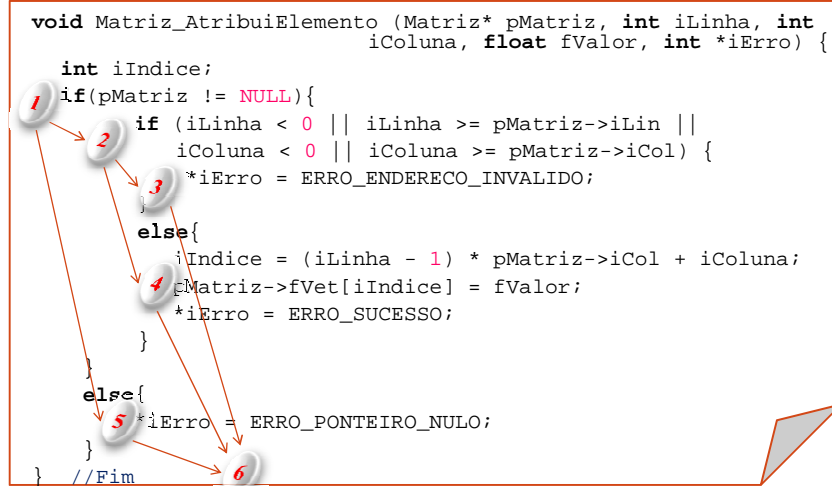
- ▶ “Blocos de comandos são trechos de código em que uma vez executado seu primeiro comando, todos os demais comandos pertencentes a este bloco são executados sequencialmente”

(Delamaro, et. al., 2007)

- ▶ Ou seja, não existe nenhum comando interno com desvio de execução para outro bloco, e nenhum comando externo possui um desvio de execução para um comando interno deste

▶ 27

## Grafo de Fluxo de Controle



▶ 28

## Organizando

### ▶ GFC

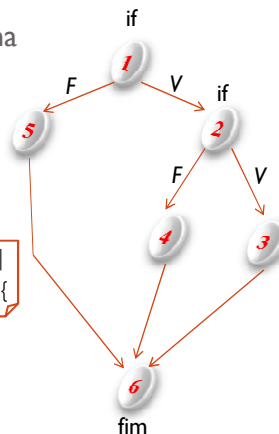
- ▶ Representa o comportamento do programa
- ▶ A ordem de execução dos blocos

### ▶ Dependente da linguagem

#### ▶ Em C

```
if (iLinha < 0 || iLinha >= pMatriz->iLin ||
    iColuna < 0 || iColuna >= pMatriz->iCol) {
```

- ▶ Dividir cada operação lógica como um nó
  - ▶ 4 nós



▶ 29

## GFC

### ▶ Comando *if* da segunda linha

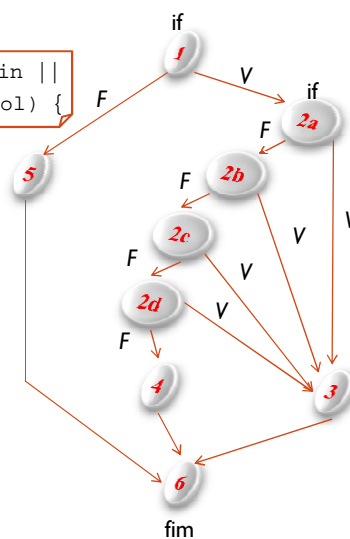
```
if (iLinha < 0 || iLinha >= pMatriz->iLin ||
    iColuna < 0 || iColuna >= pMatriz->iCol) {
```

- ▶ 4 nós
- ▶ Testa-se na sequência
- ▶ Se uma condição falhar
  - ▶ Passa direto para o nó 3

```
*iErro = ERRO_ENDERECO_INVALIDO;
```

### ▶ Importante

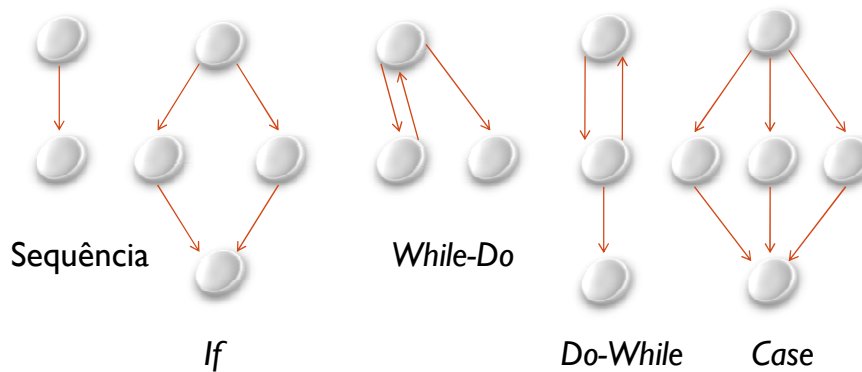
- ▶ Definir um padrão e segui-lo



▶ 30

## Outros elementos do GFC

### ▶ Comandos



For? `for(int i; i<max; i++){ /*Comandos*/ }`

▶ 31

## Grafo pronto! E agora?

### ▶ Técnica Estrutural

- ▶ Quais requisitos cobrir?

### ▶ Critérios Baseados

- ▶ Fluxo de Controle
- ▶ Fluxo de Dados

### ▶ Fluxo de Controle

- ▶ Todos-nós
- ▶ Todos-arcos
- ▶ Todos-caminhos

▶ 32



## Critérios de Rapps-Weyuker

---

- ▶ Proposto na década de 1980
- ▶ Estabelece precisamente os requisitos de teste
  - ▶ Inclui também critérios de fluxo de dados
- ▶ Todos-nós
  - ▶ Requer que todos os vértices sejam executados pelo menos uma vez
  - ▶ Equivale a executar cada comando (cada linha de código) um vez
- ▶ Todas-arcos
  - ▶ Requer que todas as arestas sejam executadas pelo menos uma vez
  - ▶ Equivale a dizer que todos os desvios devem ser executados pelo menos uma vez
- ▶ Todos-caminhos
  - ▶ Requer que todos os possíveis caminhos do grafo sejam executados pelo menos uma vez

---

▶ 33

## Observações

---

- ▶ Apesar de muito simples, critérios são efetivos, em particular o todas-arestas
- ▶ Maioria dos projetos de software não alcança esse nível mínimo de cobertura
- ▶ Diversas ferramentas de teste de apoio a esses critérios
  - ▶ JaBUTi – *Java Bytecode Understanding and Testing*
    - ▶ ICMC/USP
  - ▶ Poke-Tool – *Potencial Uses Criteria Tool for Program Testing*
    - ▶ FEEC/UNICAMP em colaboração com o ICMC/USP
  - ▶ ATAC
    - ▶ Telcordia Technologies

---

▶ 34

## Ferramenta Gcov

- ▶ Programa de Teste de Cobertura
- ▶ *GNU Compiler Collection (GCC)*
  - ▶ Gcov é uma ferramenta disponibilizada em conjunto ao gcc
  - ▶ Linux e Windows
- ▶ Dentre outros teste
  - ▶ Análise de cobertura
  - ▶ Critério Todos-nós

▶ 35

## Como utilizar

- ▶ Adicionar as flags `-fprofile-arcs -ftest-coverage` à linha de compilação do projeto
  - ▶ `gcc -fprofile-arcs -ftest-coverage -o matriz Cliente.c Matriz.c`

```

C:\Windows\system32\cmd.exe
C:\Users\TOSHIBA\Documents\Arineiza\USP\PAE\Sources\MatrizAula>dir
Volume in drive C is SQ004680V03
Volume Serial Number is 30CD-F497

Directory of C:\Users\TOSHIBA\Documents\Arineiza\USP\PAE\Sources\MatrizAula
23/08/2010  09:41    <DIR>          .
23/08/2010  09:41    <DIR>          ..
19/08/2010  15:45                1.122 Cliente.c
19/08/2010  15:47                300 LibError.h
19/08/2010  15:49                8.054 Matriz.c
19/08/2010  15:43                6.098 Matriz.h
               4 File(s)              15.574 bytes
               2 Dir(s)  74.222.419.968 bytes free

C:\Users\TOSHIBA\Documents\Arineiza\USP\PAE\Sources\MatrizAula>gcc -fprofile-arcs
-fptest-coverage -o matriz Cliente.c Matriz.c
  
```

▶ 36

## Como utilizar

- ▶ Arquivos do tipo .gcno são gerados
- ▶ Rodar o arquivo executável gerado
  - ▶ ./matriz (Linux) ou matriz.exe (Windows)

```

C:\Windows\system32\cmd.exe
C:\Users\TOSHIBA\Documents\Arineiza\USP\PAE\Sources\MatrizAula>dir
Volume in drive C is SQ004680V03
Volume Serial Number is 30CD-F497

Directory of C:\Users\TOSHIBA\Documents\Arineiza\USP\PAE\Sources\MatrizAula
23/08/2010 09:46 <DIR> .
23/08/2010 09:46 <DIR> ..
19/08/2010 15:46      1.122 Cliente.c
23/08/2010 09:46      1.152 Cliente.gcno
19/08/2010 15:47      300 LibError.h
19/08/2010 15:49      8.054 Matriz.c
23/08/2010 09:46     55.345 matriz.exe
23/08/2010 09:46      2.724 Matriz.gcno
19/08/2010 15:43      6.098 Matriz.h
                7 File(s)      74.795 bytes
                2 Dir(s)  74.222.407.680 bytes free

C:\Users\TOSHIBA\Documents\Arineiza\USP\PAE\Sources\MatrizAula>matriz.exe
M11F2I: 40.00
M21F3I: 3.00
M31F0I: 15.00
M41F1I: 21.00
Press any key to continue . . .
  
```

▶ 37

## Como utilizar

- ▶ Arquivos do tipo .gcda serão gerados
- ▶ Executar a ferramenta com o arquivo .gcda que deseja-se analisar a cobertura do Critério Todos-nós
  - ▶ gcov -a Matriz.gcda

```

C:\Windows\system32\cmd.exe
Directory of C:\Users\TOSHIBA\Documents\Arineiza\USP\PAE\Sources\MatrizAula
23/08/2010 09:46 <DIR> .
19/08/2010 15:46 <DIR> ..
23/08/2010 09:46      1.122 Cliente.c
19/08/2010 15:46      268 Cliente.gcda
23/08/2010 09:46      1.152 Cliente.gcno
19/08/2010 15:47      300 LibError.h
19/08/2010 15:49      8.054 Matriz.c
23/08/2010 09:46     55.345 matriz.exe
23/08/2010 09:46      460 Matriz.gcda
23/08/2010 09:46      2.724 Matriz.gcno
19/08/2010 15:43      6.098 Matriz.h
                9 File(s)      75.523 bytes
                2 Dir(s)  74.222.407.680 bytes free

C:\Users\TOSHIBA\Documents\Arineiza\USP\PAE\Sources\MatrizAula>gcov -a Matriz.gcda
da
File 'Matriz.c'
Lines executed:47.17% of 53
Matriz.c:creating 'Matriz.c.gcov'
  
```

▶ 38

## Como utilizar

- ▶ Será gerado um arquivo texto com extensão .gcov
- ▶ gedit Matriz.c.gcov (Linux)
- ▶ notepad Matriz.c.gcov (Windows)

```

C:\Windows\system32\cmd.exe
Directory of C:\Users\TOSHIBA\Documents\Arineiza\USP\PAE\Sources\MatrizAula
23/08/2010 09:47 <DIR> .
23/08/2010 09:47 <DIR> ..
19/08/2010 15:45      1.122 cliente.c
23/08/2010 09:46      268 cliente.gcda
23/08/2010 09:46      1.152 cliente.gcno
19/08/2010 15:47      300 libError.h
19/08/2010 15:49      8.054 Matriz.c
23/08/2010 09:47     12.414 Matriz.c.gcov
23/08/2010 09:46     55.345 Matriz.exe
23/08/2010 09:46      460 Matriz.gcda
23/08/2010 09:46      2.724 Matriz.gcno
19/08/2010 15:43      6.098 Matriz.h
10 File(s)          87.297 bytes
2 Dir(s)          74.222.317.568 bytes free

C:\Users\TOSHIBA\Documents\Arineiza\USP\PAE\Sources\MatrizAula>notepad Matriz.c.gcov
  
```

▶ 39

## Matriz.c.gcov

```

function Matriz_AcessaElemento called 4 returned 100% blocks executed 80%
4: 76:float Matriz_AcessaElemento (Matriz* pMatriz, int iLinha, int iColuna, int *iErro) {
-: 77:
4: 78:  int iIndice; /* índice do elemento no vetor */
4: 79:  if(pMatriz != NULL){
4: 79-block 0
4: 80:    if (iLinha < 0 || iLinha >= pMatriz->Lin || iColuna < 0 || iColuna >= pMatriz->iCol) {
4: 80-block 0
4: 80-block 1
4: 80-block 2
4: 80-block 3
4: 80-block 4
#####: 81:      *iErro = ERRO_ENDERECO_INVALIDO;
#####: 82:      return 0;
$$$$$: 82-block 0
-: 83:    }
  
```

- ▶ #####: Linhas não executadas

▶ 40

## Matriz.c.gcov

---

### ▶ Continua...

```

4: 84:   iIndice = (iLinha - 1) * pMatriz->iCol + iColuna;
4: 85:   *iErro = ERRO_SUCESSO;
4: 86:   return pMatriz->fVet[iIndice];
4: 86-block 0
-: 87: }
-: 88: else{
#####: 89:   *iErro = ERRO_PONTEIRO_NULO;
4: 90:   return 0;
$$$$$: 90-block 0
4: 90-block 1
-: 91: }
-: 92:}

```

---

▶ 41

## Matriz.c.gcov

---

### ▶ Algumas linhas não podem ser executadas

#### ▶ Por exemplo: linha 47

```

function Matriz_Cria called 1 returned 100% blocks executed 67%
1: 43:Matriz* Matriz_Cria (int iLinha, int iColuna, int *iErro) {
-: 44:
1: 45:   Matriz* pMatriz = (Matriz*) malloc(sizeof(Matriz));
1: 45-block 0
1: 46:   if (pMatriz == NULL) {
#####: 47:     *iErro = ERRO_MEMORIA_INSUFICIENTE;
#####: 48:     return NULL;
$$$$$: 48-block 0
-: 49: }

```

---

▶ 42

## Objetivo

---

- ▶ Executar o máximo possível do código
  - ▶ Criar um conjunto de testes **Todos-nós-adequado**
  - ▶ Considerando os nós “**não-executáveis**”



---

▶ 43

## Executabilidade

---

- ▶ Um dos problemas no teste estrutural, em geral, é a executabilidade
  - ▶ Um caminho é dito não executável quando não existe um dado de entrada que faça com que esse caminho seja executado
- ▶ Ao se determinarem os requisitos de teste é impossível determinar se são executáveis ou não
  - ▶ Esse é um problema provado indecível
- ▶ É um problema para a automatização da atividade de teste

---

▶ 44

## Dúvidas?



Arineiza Cristina Pinheiro

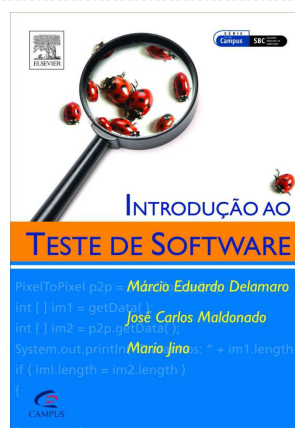
LabES – CISC – 2-208

[arineiza@icmc.usp.br](mailto:arineiza@icmc.usp.br)

▶ 45

## Referências

- ▶ MALDONADO, J. C.; JINO M.; DELAMARO, M. E. *Conceitos Básicos*. In: MALDONADO, J. C.; JINO M.; DELAMARO, M. E. (eds). *Introdução ao Teste de Software*. 1 ed. São Paulo: Elsevier Editora Ltda., 2007, v. 1, p. 1-7.
- ▶ BARBOSA, E. F.; CHAIM, M. L.; VINCENZI, A.M.R.; DELAMARO, M. E.; JINO M.; MALDONADO, J.C. *Teste Estrutural*. In: MALDONADO, J. C.; JINO M.; DELAMARO, M. E. (eds). *Introdução ao Teste de Software*. 1 ed. São Paulo: Elsevier



▶ 46

## Referências

---

- ▶ Editora Ltda., 2007, v. 1, p. 48-76. VINCENZI, A. M. R.; WONG, W. E.; DELAMARO, M. E.; MALDONADO J. C. *JaBUTi - Java Bytecode Understanding and Testing. Version 1.0 – Java. Manual do Usuário*. São Carlos, Brasil, Março, 2003. Disponível em: <http://incubadora.fapesp.br/projects/jabuti/>. Acesso em: 23 de agosto de 2010
  
- ▶ VILELA, P. R. S.; VERGILIO, S. R.; MALDONADO, J. C.; JINO, M. *Introdução aos Critérios Potenciais Usos e à POKE-TOOL*. Disponível em: <http://www.inf.ufpr.br/silvia/topicos/pokemanual.ps>. Acesso em: 23 de agosto de 2010
  
- ▶ Grace Murray Hopper
  - ▶ <http://www.inf.ufg.br/~eduardo/lp/alunos/cobol/LPCobol.htm>
  
- ▶ Gcov
  - ▶ <http://gcc.gnu.org/onlinedocs/gcc/Gcov.html>