Técnica de Teste Estrutural - Fluxo de Controle

PPGCC12-Teste de Software

Reginaldo Ré

Aula 05

Introdução

Introdução

- Critérios pertencentes à Técnica de Teste Caixa-Branca.
  - Em oposição ao Teste de Caixa-Preta.
  - Baseado na **estrutura interna** (implementação) do programa.
- Os critérios dessa técnica geralmente utilizam uma representação chamada de **Grafo de Programa**
  - Identificam requisitos de testes (caminhos de execução) a partir da implementação do SUT (Software Under Test).
  - Requer a criação e execução de casos de testes que exercitem tais requisitos.
  - Definição:
    - **Caminho**: sequência de execução de comandos que se inicia em um ponto de entrada e termina em um ponto de saída do SUT.

Exemplo de problemas

- O teste exaustivo de todos os possíveis caminhos de fluxo de controle possui várias desvantagens:
  - O número de caminhos pode ser infinito ou muito grande, ex: laços aninhados.
  - Caminhos presentes na especificação podem ser esquecidos na implementação, ex: um `else` esquecido.
  - Defeitos podem existir mesmo com o fluxo de controle correto, ex: uma equação matemática incorreta.
  - Uma parte do código pode executar corretamente para diversos casos de testes e falhar para alguns, ex: uma falha de I/O ou divisão por zero.
- Mesmo com tais limitações o teste caixa branca é de grande importância e complementar ao teste caixa preta.

Passos da Técnica Estrutural

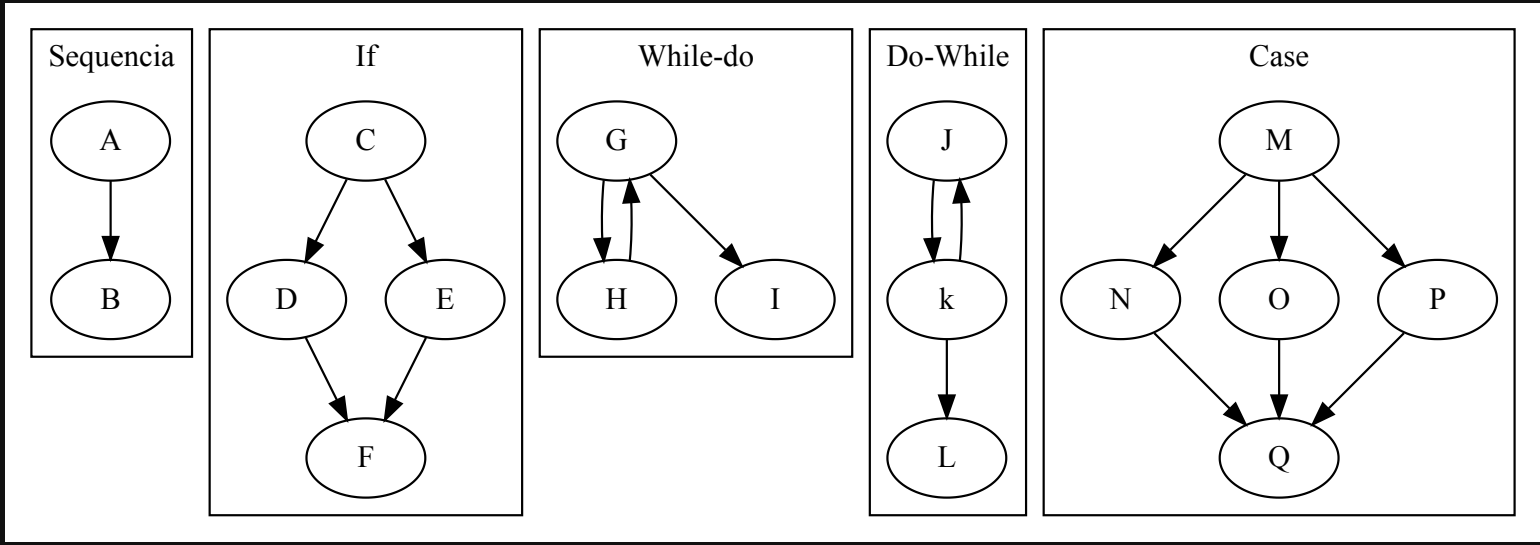
- Passos básicos para aplicar um critério de teste estrutural:
  - A implementação do produto em teste é analisada.
  - **Caminhos** através da implementação são escolhidos.
    - **Requisitos de Teste**
  - Valores de entrada são selecionados de modo que os caminhos escolhidos sejam executados.
  - As saídas esperadas para as entradas selecionadas são determinadas.
  - Os casos de testes são construídos.
  - As saídas obtidas são comparadas às saídas esperadas.
  - Um relatório é gerado para avaliar o resultado dos testes.

Grafo de Fluxo de Controle

O que é um GFC (Grafo de Fluxo de Controle)

- Notação utilizada para abstrair o fluxo de controle lógico de um programa.
  - Auxilia na geração de requisitos de teste
- Seja $GFC = (N, E, s)$ um grafo no qual N representa o conjunto de nós, E o conjunto de arcos, e s o nó de entrada.
  - **Nós**: bloso de comandos **indivisíveis**
    - Não existe desvio para o meio do bloco.
    - Uma vez que o primeiro comando do bloco é executado, os demais são executados na sequencia.
  - **Arestas ou Arcos**: representam o fluxo de controle entre os nós.
  - **Caminhos**: sequencias de execução de comandos que iniciam em um nó de entrada e terminam em um nós de saída.

Elementos presentes no GFC



Exemplo de GFC: programa *Identifier* (1/4)

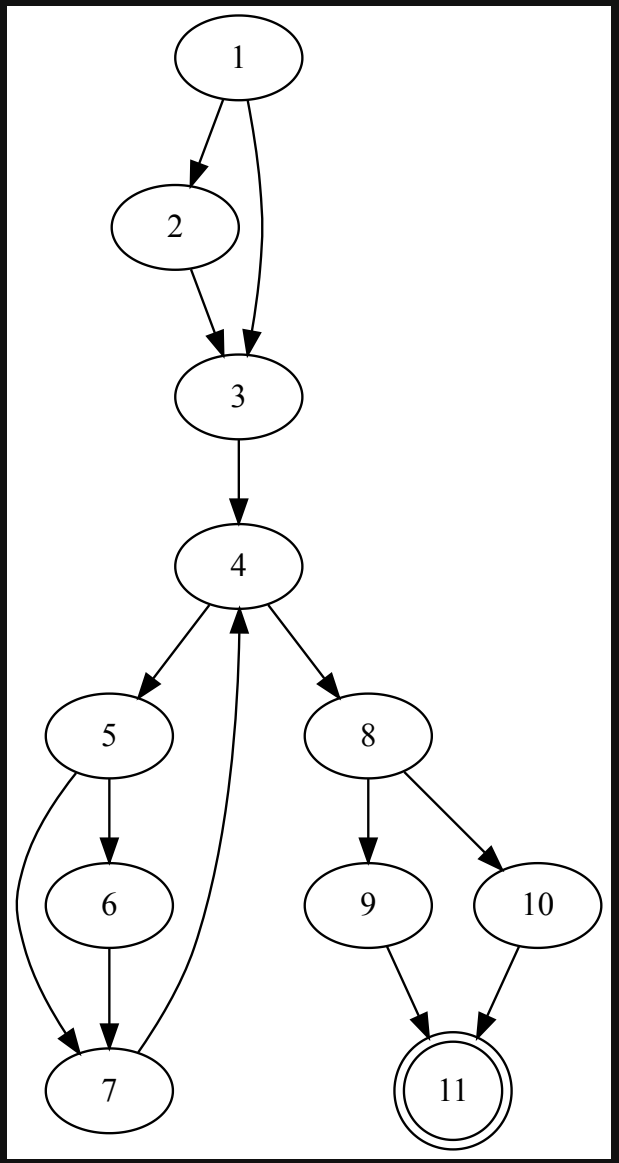
O programa Identifier determina se um identificador é válido ou não. Um identificador válido deve começar com uma letra e conter apenas letras ou dígitos. Além disso, deve ter no mínimo um caractere e no máximo seis caracteres de comprimento.

Exemplo de GFC: programa *Identifier* (2/4)

```
/* 01 */ {
/* 01 */    char achar;
/* 01 */    int length, valid_id;
/* 01 */    length = 0;
/* 01 */    printf("Identificador: ");
/* 01 */    achar = fgetc(stdin);
/* 01 */    valid_id = valid_s(achar);
/* 01 */    if (valid_id)
/* 02 */       length = 1;
/* 03 */    achar = fgetc(stdin);
/* 04 */    while (achar != '\n') {
/* 05 */       if (!(valid_f(achar)))
/* 05 */          valid_id = 0;
/* 06 */       length++;
/* 07 */       achar = fgect(stdin);
/* 07 */    }
/* 08 */    if (valid_id && (length >= 1) && length<6))
/* 09 */       printf("Valido\n");
/* 10 */    else
/* 10 */       printf("Invalido\n");
/* 11 */ }
```

- `valid_s()`: determina se o primeiro caractere é válido.
- `valid_f()`: determina se o próxima caractere é válido.

Exemplo de GFC: programa *Identifier* (3/4)



- Nós: 1,2,3, …
- Arcos: <1,2>, <1,3>, …
- Arcos primitivos: <1,2>,<1,3>,<5,6>, <5,7>,<8,9>,<8,10>
  - Arcos que nunca são herdeiros de outros arcos do grafo fluxo de controle.
    - Um arco é considerado herdeiro sempre que sua execução ocorrer após a execução de outro arco.
  - Importantes porque herdam as informações da execução .
- Caminhos:
  - Simples: (2,3,4,5,6,7)
  - Completo: (1,2,3,4,5,7,4,8,9,11)
- Definição de variável: `length=0`
- Usos:
  - Predicativo: `achar != '\n'`
  - Computacional: `length++`

Exemplo de GFC: programa *Identifier* (4/4)

```
/* 01 */ {
/* 01 */    char achar;
/* 01 */    int length, valid_id;
/* 01 */    length = 0;
/* 01 */    printf("Identificador: ");
/* 01 */    achar = fgetc(stdin);
/* 01 */    valid_id = valid_s(achar);
/* 01 */    if (valid_id)
/* 02 */       length = 1;
/* 03 */    achar = fgetc(stdin);
/* 04 */    while (achar != '\n') {
/* 05 */       if (!(valid_f(achar)))
/* 05 */          valid_id = 0;
/* 06 */       length++;
/* 07 */       achar = fgect(stdin);
/* 07 */    }
/* 08 */    if (valid_id && (length >= 1)
  && length<6))
/* 09 */       printf("Valido\n");
/* 10 */    else
/* 10 */       printf("Invalido\n");
/* 11 */ }
```

- Caminho não-executável
  - **Executabilidade**
- (1,3,4,8,9) é não executável
  - Quaisquer outros caminhos que incluem este, também são não executáveis

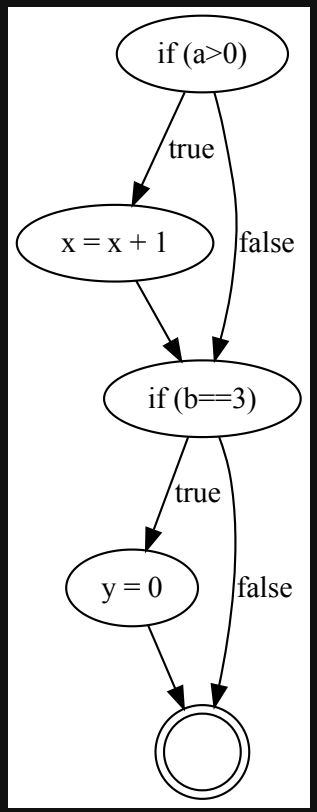
Níveis de Cobertura (1/11)

- Diferentes níveis de cobertura podem ser definidos a partir dos elementos do GFC.
- Cobertura: porcentagem dos requisitos de teste que foram testados *versus* o total de requisitos gerados.
- Existem oito diferentes níveis de cobertura.
- Quanto maior o nível, maior o rigor do critério de teste
  - Ou seja, mais casos de teste ele exige para ser satisfeito

Níveis de Cobertura (2/11)

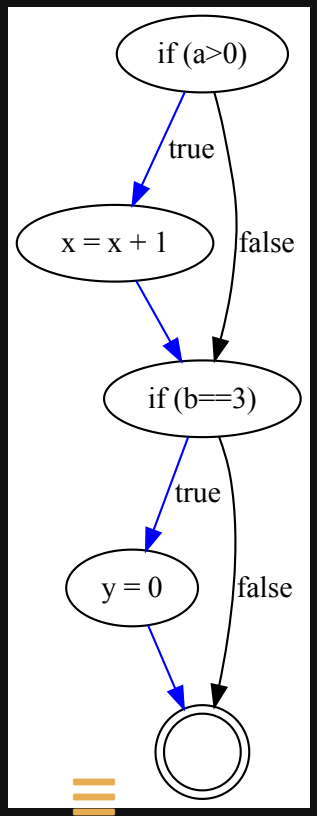
- **Nível 0**: Qualquer valor de cobertura inferior a 100% da cobertura de todos os **comandos**
- **Nível 1**: 100% de cobertura de comandos
  - Chamado de cobertura de nós (**critério todos-nós**)

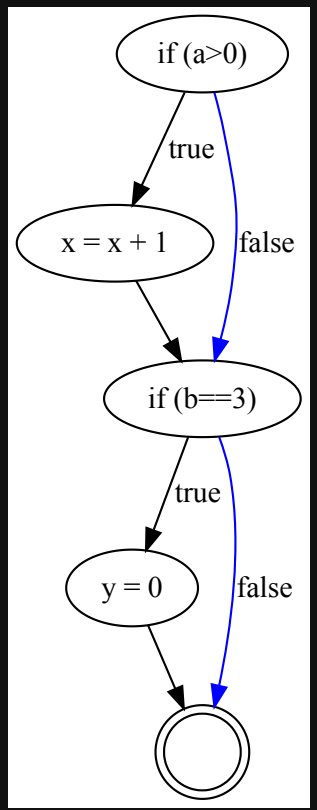
```
if (a>0){
  x = x + 1;
}
if (b==3){
  y = 0;
}
```

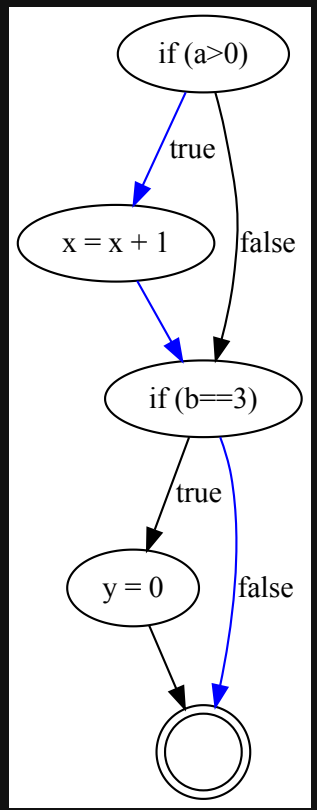


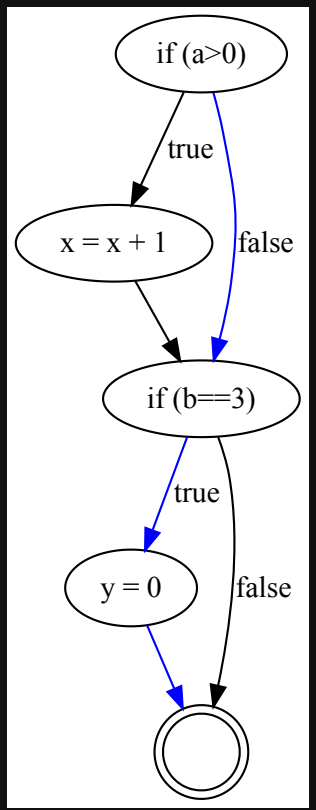
Níveis de Cobertura (3/11)

- **Nível 1**: 100% de cobertura de comandos (requisíto mínimo de teste)
  - Um caso de teste é suficiente para cobrir todos os comandos.
    - Por exemplo, a=6 e b=3.
  - Mas não todos os caminhos.
    - Por exemplo, a=6 e b=3 não cobre o caminho 4.







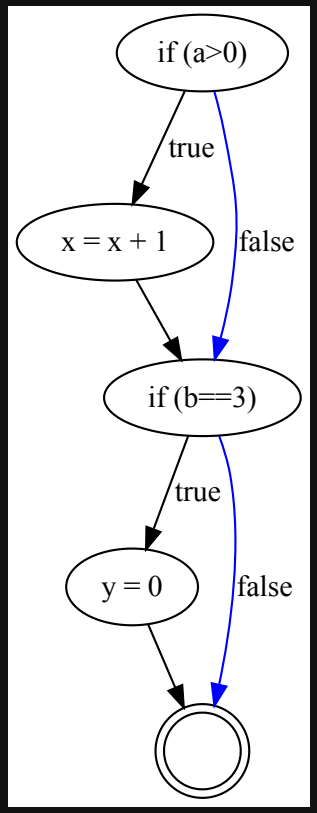


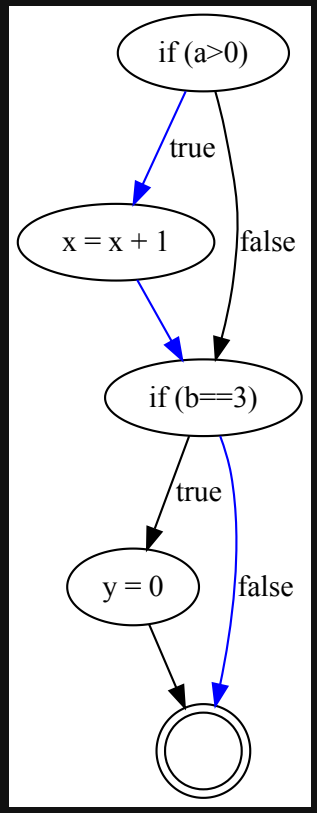
Níveis de Cobertura (4/11)

- **Nível 1**: 100% de cobertura de comandos (requisíto mínimo de teste)
  - Embora seja o nível mais baixo de cobertura, pode ser difícil de ser atingido
    - Dificuldade ou impossibilidade de cobrir código para situações excepcionais: falta de memória, disco cheio, arquivos ilegíveis, perda de conexão, dentre outras.

Níveis de Cobertura (5/11)

- **Nível 2**: 100% de cobertura de decisões
  - Chamado de cobertura de arcos/arestas (**critério todos-arcos**)
  - Fazer cada comando de decisão assumir os valores TRUE e FALSE.





- Dois casos de teste são suficientes para cobrir todos os arcos do GFC (critério **todos-arcos**).
  - Por exemplo, a=2, b=2 e a=4,b=3.

Níveis de Cobertura (6/11)

- **Nível 3**: 100% de cobertura de condições.
  - Nem todos os comandos de decisão são simples, como o anterior.
  - Considere o exemplo a seguir:

```
if (a>0 && c==1){
  x = x+1
}
if (b==3 || d<0){
  y = 0;
}
```

- Dois casos de teste necessários para cobrir todas as condições:
  - {a>0, c=1, b=3, d<0} e {$a \leq 0$, $c \neq 1$, $b \neq 3$, $d \geq 0$}
- Cobertura de condição é, em geral, melhor que cobertura de decisão.

Níveis de Cobertura (7/11)

- **Nível 4**: 100% de cobertura de decisões/condições.
  - Considere o exemplo a seguir:

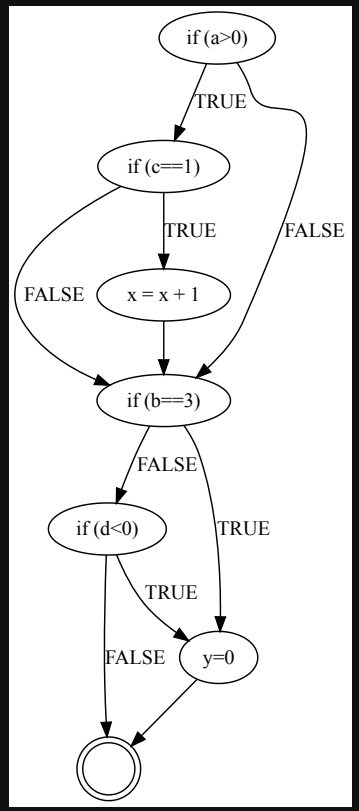
```
if (x && y){
  x = x+1
}
```

- Dois casos de teste necessários para cobrir todas as condições:
  - {x=TRUE, y=FALSE} e {x=FALSE, y=TRUE}.
  - Mas `x = x+1;` não será executado.
- Todas as combinações devem ser testadas.

Níveis de Cobertura (8/11)

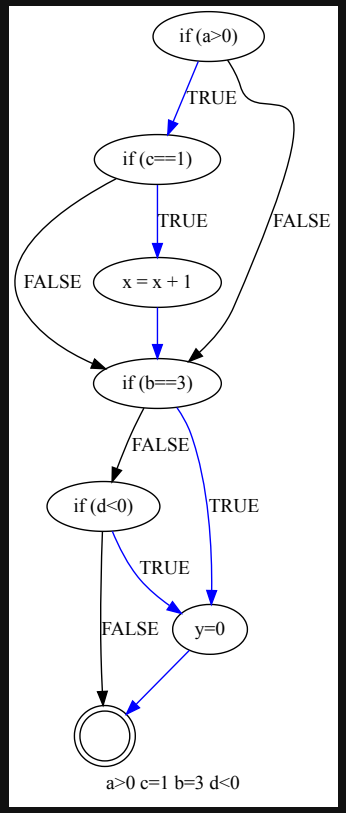
- **Nível 5**: 100% de cobertura de condições múltiplas.
- Consiste em utilizar o conhecimento de como o compilador avalia condições múyltiplas de determinado comando de decisão e usar essa informação na geração de casos de teste.
  - Obter 100% de cobertura de condições múltiplas implica cobrir 100% dos critérios anteriores, mas não garante cobertura do critério **todos-caminhos**.
  - Considere um compilador que avalia condições múltiplas conforme ilustrado a seguir:

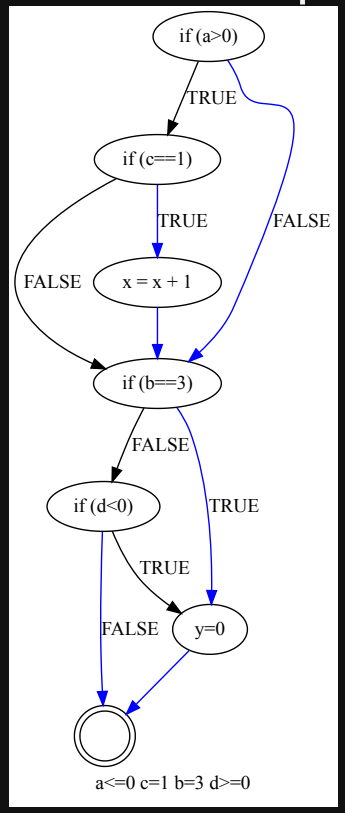
```
if (a>0 && c==1){
  x = x+1
}
if (b==3 || d<0){
  y = 0;
}
```

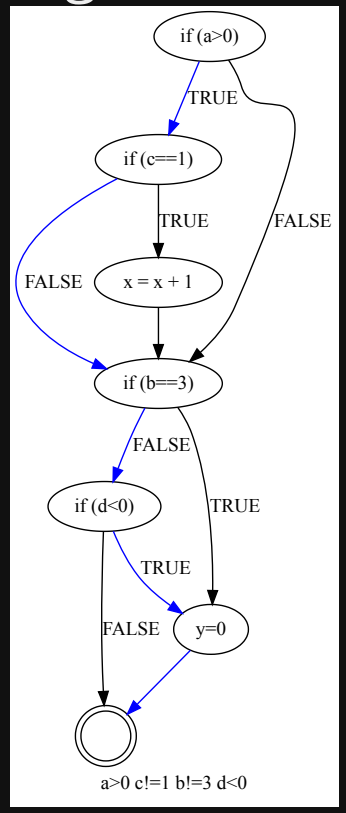


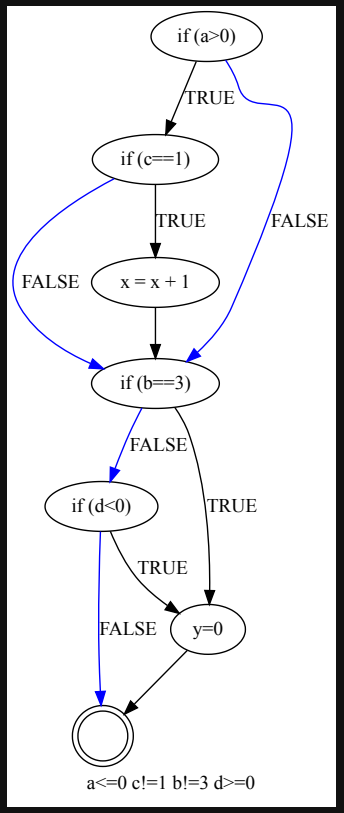
Níveis de Cobertura (9/11)

- **Nível 5**: 100% de cobertura de condições múltiplas.
  - Cobrir os arcos do GFC requer os seguintes casos de teste:









Níveis de Cobertura (10/11)

- **Nível 6**: cobertura de laço.
  - Em presença de laços, o número de caminhos pode ser infinito.
  - Pode-se reduzir usando a seguinte estratégia:
    - 0 vezes.
    - 1 vez.
    - 2 vezes.
    - n vezes, no qual n é um número de vezes padrão que o laço é executado.
    - m vezes, no qual m é número máximo de vezes que o loop pode ser executado.
    - m - 1 vezes.
    - m + 1 vezes.

Níveis de Cobertura (11/11)

- **Nível 7**: 100% de cobertura de caminhos.
  - Conhecido como critério **todos-caminhos**.
  - Sem a presença de laços, o número de caminhos pode ser pequeno, e podem ser desenvolvidos casos de teste para cobrí-los.
  - Com a presença de laços, o número de caminhos pode ser grande ou infinito, tornando a cobertura impossível.

Técnica Estrutural

Aplicabilidade

- Critérios da técnica estrutural podem ser utilizados em todas as fases de teste.
- Teste de caminhos:
  - Caminhos dentro de uma unidade.
  - Caminhos entre unidades.
  - Caminhos entre sub-sistemas.
  - Caminhos entre o sistema todo.

Problemas da Técnica Estrutural

- O número de **caminhos** a serem executados pode ser infinito (semelhante ao teste exaustivo).
- Assume o fluxo de controle **correto** (ou próximo do correto).
  - Caminhos inexistentes não podem ser descobertos.
- Impossibilidade, em geral, de determinar se um **caminho é executável ou não**.
  - Intervenção do testador.
  - Dificuldade de automatização.
  - Habilidades de programação avançadas são exigidas para compreender o código e decidir pela executabilidade ou não de um caminho.

Vantagens da Técnica Estrutural

- É possível garantir que **partes essenciais** ou críticas do programa tenham sido executadas.
  - Requisito mínimo de teste: garantir que o programa foi liberado tendo seus **comandos** executados ao menos uma vez por pelo menos um caso de teste.
- Implementação dos critérios de teste é mais fácil de ser **automatizada**.
  - Ferramentas de teste.

Critérios da Técnica Estrutural

- Baseados em Fluxo de Controle
  - Todos-Nós
  - Todas-Arestas
  - Todos-Caminhos: Simples, Completo, Livre de Laço, …
- Baseados em Complexidade
  - Critério de McCabe (teste do caminho base).
- Baseados em Fluxo de Dados
  - Critérios de Rapps & Weyuker
    - Todas-Defs, Todos-Usos, Todos-P-Usos e outros.
  - Critérios Potenciais-Usos (Maldonado)
    - Todos-Potenciais-Usos, Todos-Potenciais-Usos/DU e outros.

Critérios Baseados em Fluxo de Controle

Critérios Baseados em Fluxo de Controle

- Utilizam características de controle da execução do programa para determinar os **requisitos de teste**.
  - Comandos
  - Desvios de Execução
- Alguns critérios de fluxo de controle:
  - Todos-Nós
  - Todos-Arcos
  - Todos-Caminhos
  - Todos-Caminhos-Simples
  - Todos-Caminhos-Completos
  - Todos-Caminhos-Livre-Laço

Critério Todos-Nós

- Exige que a execução do programa passe, ao menos uma vez, em cada nó do grafo de programa.
- Ou seja,

> *cada comando do programa seja executado pelo menos uma vez.*

- Elementos requeridos (**requisitos de teste**) pelo critério no programa *Identifier*:
  - Nós: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

Critério Todos-Arcos

- Exige que a execução do programa passe, ao menos uma vez, em cada arco do grafo de programa.
- Ou seja,

> *cada desvio de fluxo de controle do programa seja exercitado pelo menos uma vez.*

- Elementos requeridos (**requisitos de teste**) pelo critério no programa *Identifier*:
  - Arcos:
    - Arcos Primitivos: <1,2>, <1,3>, <5,6>, <5,7>, <8,9>, <8,10>

Critério Todos-Caminhos-Completos

- Exige que todos os **caminhos completos** sejam exercitados pelo menos uma vez.
  - Um caminho completo é um caminho P em que o primeiro nó de P é o nó inicial e o último nó de P é o nó final do grafo.
- Ou seja,

> *todos os **caminhos completos** sejam exercitados pelo menos uma vez*

- Elementos requeridos (**requisitos de teste**) pelo critério no programa *Identifier*:
  - (1,2,3,4,8,10,11)
  - (1,2,3,4,5,7,4,5,7,4,8,10,11)
  - ...

Critério Todos-Caminhos-Simples

- Exige que todos os **caminhos simples** sejam exercitados pelo menos uma vez.
  - Um **caminho simples** é formado por nós **distintos**, exceto possivelmente o primeiro e o último.
- Ou seja,

> *todos os **caminhos simples** sejam exercitados pelo menos uma vez*

- Elementos requeridos (**requisitos de teste**) pelo critério no programa *Identifier*:
  - (1,2,3,4,8,10,11)
  - (4,5,6,7,4)
  - ...

Critério Todos-Caminhos-Livre-Laços

- Exige que todos os **caminhos livre de laço** sejam exercitados pelo menos uma vez.
  - Um **caminho livre de laço** é um caminho simples em que **todos os nós são distintos**, inclusive o primeiro e o último.
- Ou seja,

> *todos os **caminhos livre de laço** sejam exercitados pelo menos uma ve*

- Elementos requeridos (**requisitos de teste**) pelo critério no programa *Identifier*:
  - (1,2,3,4,8,10,11)
  - (1,2,3,4,5,7)
  - ...

Critérios Baseados em Complexidade

Critério de McCabe (1/2)

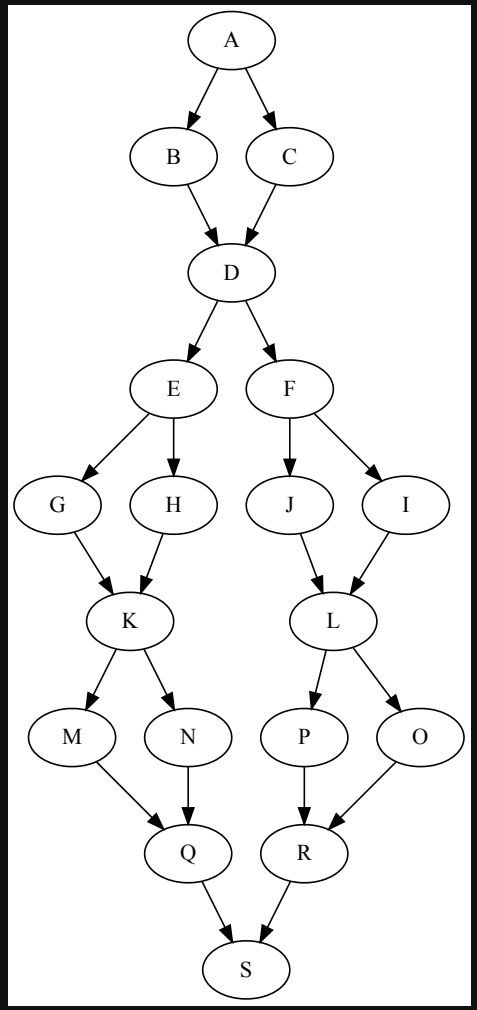
- Conhecido como **teste do caminho básico** ou critério de McCabe
- Baseado no conceito de **Complexidade Ciclomática**, calculada a partir do GFC.
- A complexidade ciclomática define o número de **caminhos independentes** de um programa.
  - Um caminho independente é qualquer caminho que introduz no mínimo uma **nova aresta** ainda não atravessada pelos caminhos anteriores.

Critério de McCabe (2/2)

- Passos para aplicação do critério:
  1. Construir o GFC para o módulo do produto em teste.
  2. Calcular a **Complexidade Ciclomática** (C).
  3. Selecionar um conjunto de P caminhos básicos.
  4. Criar um caso de teste para cada caminho básico.
  5. Executar os casos de testes.

Passos para aplicação do Critério de McCabe (1/8)

Construir o GFC para o módulo do produto em teste



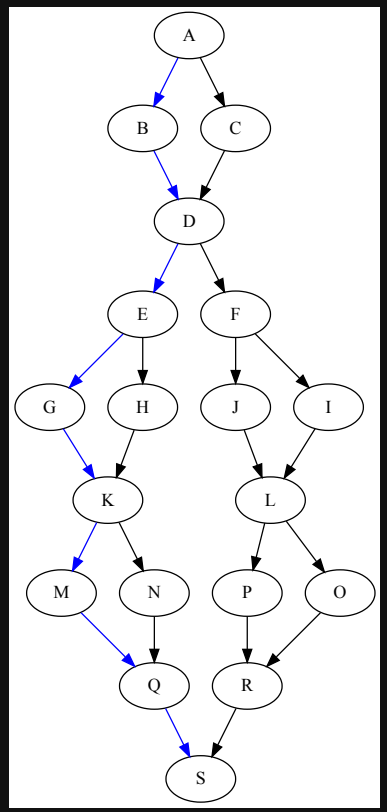
Passos para aplicação do Critério de McCabe (2/8)

Calcular a Complexidade Ciclomática (C♪)

- Para o GFC anterior
  - $C = numero.arcos - numero.nos + 2$
  - $C = 24 - 19 + 2$
  - $C = 7$
- Para um GFC com p decisões binárias (dois arcos saindo):
  - $C = p + 1$
- Cada caminho básico percorre **pelo menos** um arco que ainda não foi percorrido.

Passos para aplicação do Critério de McCabe (3/8)

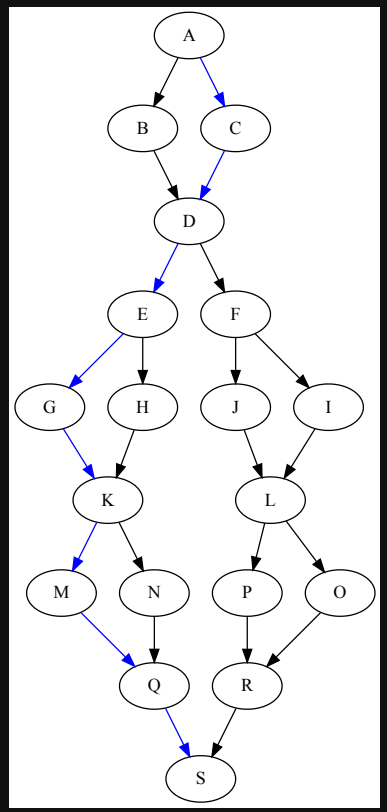
Selecionar um conjunto de P caminhos básicos (Passo 1)



- Escolha um caminho básico, que pode ser:
  - Caminho mais comum.
  - Caminho mais crítico.
  - Caminho mais importante do ponto de vista de teste.
- Caminho 1: (A, B, D, E, G, K, M, Q, S)

Passos para aplicação do Critério de McCabe (4/8)

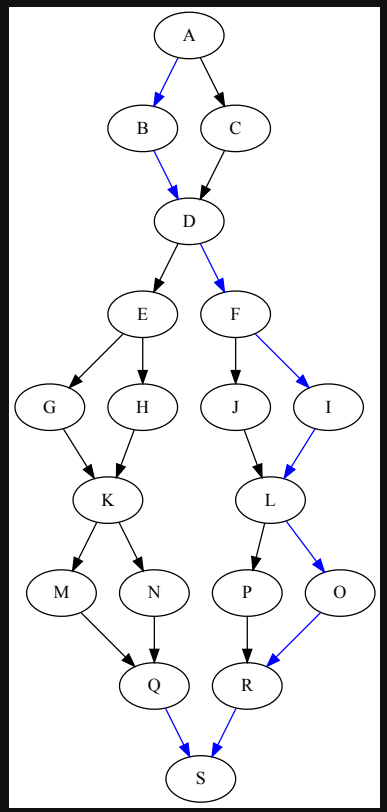
Selecionar um conjunto de P caminhos básicos (Passo 2)



- Altere a saída do primeiro comando de decisão e mantenha o máximo possível do caminho inalterado.
- Caminho 2: (A, C, D, E, G, K, M, Q, S)

Passos para aplicação do Critério de McCabe (5/8)

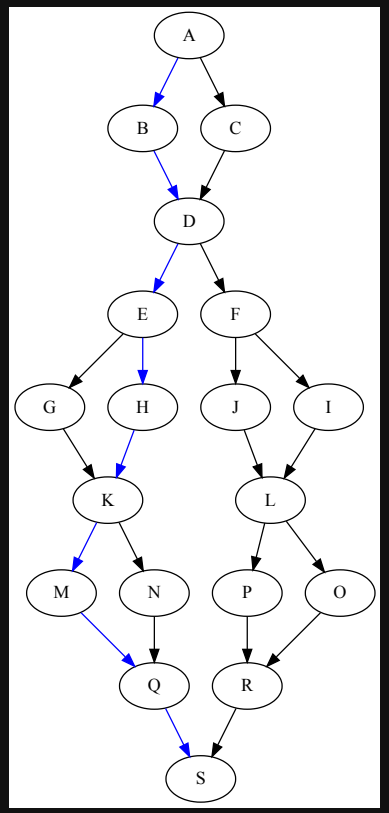
Selecionar um conjunto de P caminhos básicos (Passo 3)

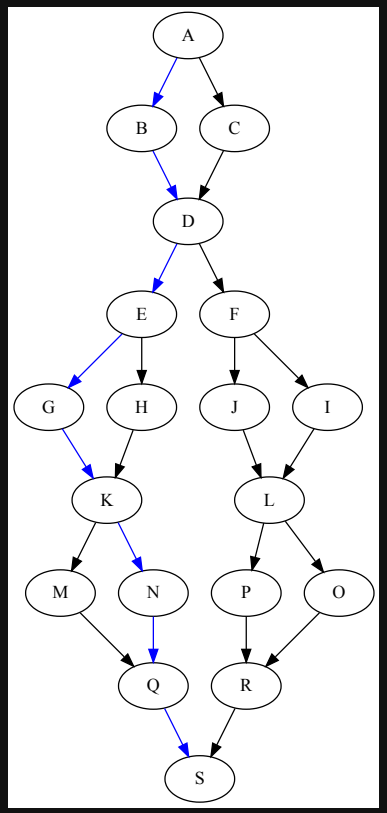


- A partir do caminho básico alterar a saída do segundo comando de decisão.
- Caminho 3: (A, B, D, F, I, L, O, R, S)

Passos para aplicação do Critério de McCabe (6/8)

Selecionar um conjunto de P caminhos básicos (Passo 4)

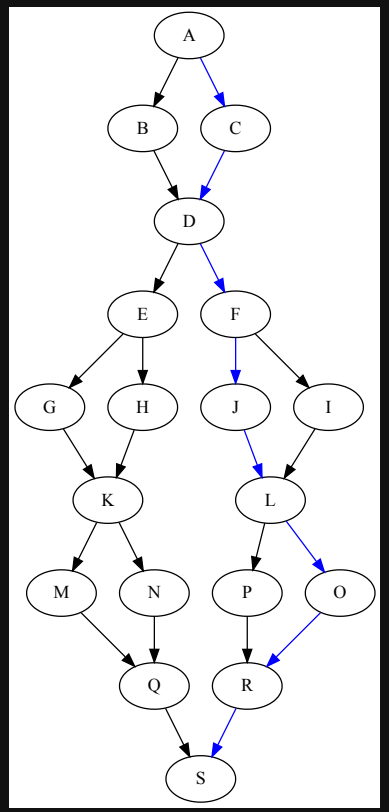


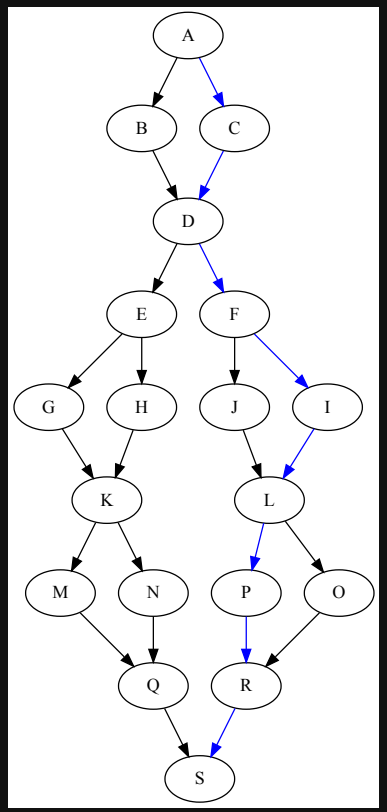


- A partir do caminho básico alterar a saída do terceiro comando de decisão. Repetir esse processo até atingir o final do GFC.
- Caminho 4: (A, B, D, E, H, K, M, Q, S)
- Caminho 5: (A, B, D, E, G, K, N, Q, S)

Passos para aplicação do Critério de McCabe (7/8)

Selecionar um conjunto de P caminhos básicos (Passo 5)





- Todas as decisões do caminho básico foram contempladas.
- A partir do segundo caminho, fazer as inversões dos comandos de decisão até o final do GFC.
- Esse padrão é seguido até que o conjunto completo de caminhos seja atingido.
- Caminho 6: (A, C, D, F, J, L, O, R, S)
- Caminho 7: (A, C, D, F, I, L, P, R, S)

Passos para aplicação do Critério de McCabe (8/8)

Conjunto Completo de Caminhos Básicos

- Requisitos de testes derivados pelo critério:
  - (A, B, D, E, G, K, M, Q, S)
  - (A, C, D, E, G, K, M, Q, S)
  - (A, B, D, F, I, L, O, R, S)
  - (A, B, D, E, H, K, M, Q, S)
  - (A, B, D, E, G, K, N, Q, S)
  - (A, C, D, F, J, L, O, R, S)
  - (A, C, D, F, I, L, P, R, S)
- O conjunto criado **não é único.**
- **Propriedade**: o conjunto de teste que exercita os caminhos básicos também exercita todos os nós e todos os arcos do programa.
  - Garante a cobertura dos critérios **todos-nós** e **todos-arcos**.

Considerações Finais

Pontos importantes

- Critérios da técnica estrutural identificam **caminhos** que devem ser percorridos no **código** do programa.
- O **GFC** é a base a partir do qual os requisitos de testes são derivados.
- Critérios de **Fluxo de Controle** são a pedra fundamental do teste de unidade.
  - Podem ser aplicados em todos os módulos do software, em especial, nos mais **críticos**.
  - Exigem **habilidades de programação** do testador para compreender o fluxo de controle do programa.
  - Podem consumir **tempo** e **recursos** significativos para sua aplicação.
- A complexidade ciclomática define o número **mínimo** de conjuntos de caminhos independentes livres de laço (caminho básico).
  - O conjunto de caminhos básicos inclui todos os nós e arcos do GFC.
  - Casos de testes que exercitem todos os caminhos básicos do conjunto também executam todos os **comandos** e todas as **decisões** do programa.