SSC0721 – Teste e inspeção de software

Teste Estrutural

Prof. Marcio E. Delamaro

delamaro@icmc.usp.br

Relembrando

- Teste estrutural utiliza o código do programa para derivar requisitos de teste

Relembrando

- Teste estrutural utiliza o código do programa para derivar requisitos de teste
- Por isso requer conhecimento da linguagem de programação e obviamente do programa implementado

Relembrando

- Teste estrutural utiliza o código do programa para derivar requisitos de teste
- Por isso requer conhecimento da linguagem de programação e obviamente do programa implementado
- De qualquer forma, o conhecimento da especificação é utilizado

Relembrando

- Teste estrutural utiliza o código do programa para derivar requisitos de teste
- Por isso requer conhecimento da linguagem de programação e obviamente do programa implementado
- De qualquer forma, o conhecimento da especificação é utilizado
- Apropriado principalmente para o teste de unidade

Tipos de critérios

- Baseados na complexidade
- No fluxo de controle
- No fluxo de dados

Grafo de fluxo de controle (GFC)

- Grafo de programa
- É uma forma de abstrair a estrutura de uma unidade do programa
- Uma função, procedimento ou método
- Grafo: vértices e arestas

GFC – definição

- Vértices são formados por blocos indivisíveis de código
  - Cada instrução é executada em seqüência. Uma vez que a primeira instrução seja executada, todas as demais são executadas também. Não existe, desvio para o meio do bloco.
- Arestas representam a possível transferência da execução entre um bloco e outro.
- Existem um único nó que é chamado de nó de entrada, que corresponde ao bloco da primeira instrução da unidade
- Podem existir diversos nós de saída, ou seja, nós que não têm sucessores

GFC – exemplo

The program determines if a given identifier is valid or not in a variant of Pascal language, called Silly Pascal. A valid identifier must begin with a letter and contain only letters or digits. Moreover, it has at least one and no more than six character length.

Exemplos de identificadores:

abc12 (valid);
cont*1 (invalid); 1soma (invalid); a123456 (invalid)

GFC – exemplo

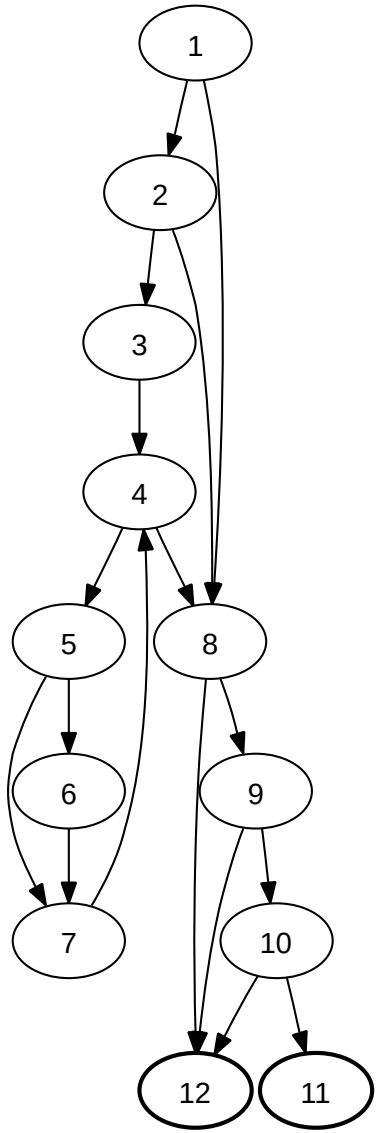
```
public boolean validateIdentifier(String s) {
        char achar;
        boolean valid_id = false;
        if (s.length() > 0) {
           achar = s.charAt(0);
           valid_id = valid_s(achar);
           if (s.length() > 1) {
              achar = s.charAt(1);
              int i = 1;
              while (i < s.length() – 1) {
                 achar = s.charAt(i);
                 if (!valid_f(achar))
                    valid_id = false;
                 i++;
              }
           }
        }
        if (valid_id && (s.length() >= 1) && (s.length() < 6))
           return true;
        else
           return false;
}
```

GFC – blocos

```
public boolean validateIdentifier(String s) {
       char achar;
/*01*/ boolean valid_id = false;
/*01*/ if (s.length() > 0) {
/*02*/    achar = s.charAt(0);
/*02*/    valid_id = valid_s(achar);
/*02*/    if (s.length() > 1) {
/*03*/       achar = s.charAt(1);
/*03*/       int i = 1;
/*04*/       while (i < s.length() - 1) {
/*05*/          achar = s.charAt(i);
/*05*/          if (!valid_f(achar))
/*06*/             valid_id = false;
/*07*/          i++;
             }
          }
       }      /*08*/          /*09*/              /*10*/
       if (valid_id && (s.length() >= 1) && (s.length() < 6))
/*11*/    return true;
       else
/*12*/    return false;
}
```

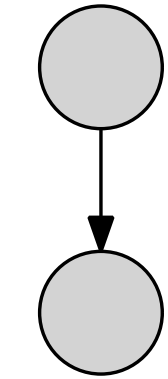
GFC – o grafo

```
public boolean validateIdentifier(String s) {
        char achar;
/*01*/  boolean valid_id = false;
/*01*/  if (s.length() > 0) {
/*02*/     achar = s.charAt(0);
/*02*/     valid_id = valid_s(achar);
/*02*/     if (s.length() > 1) {
/*03*/        achar = s.charAt(1);
/*03*/        int i = 1;
/*04*/        while (i < s.length() – 1) {
/*05*/           achar = s.charAt(i);
/*05*/           if (!valid_f(achar))
/*06*/              valid_id = false;
/*07*/           i++;
              }
           }
        }       /*08*/            /*09*/              /*10*/
        if (valid_id && (s.length() >= 1) && (s.length() < 6))
/*11*/     return true;
        else
/*12*/     return false;
}
```



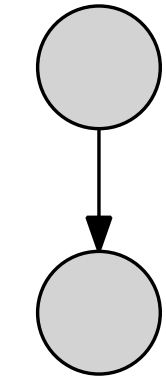
Alguns elementos do GFC

Alguns elementos do GFC

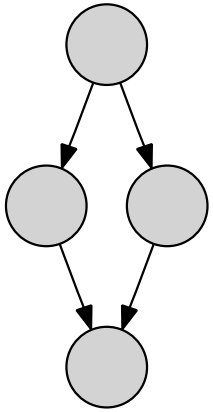


Sequence

Alguns elementos do GFC

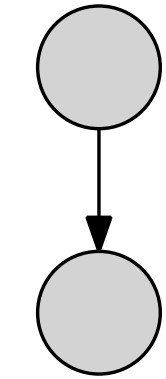


Sequence

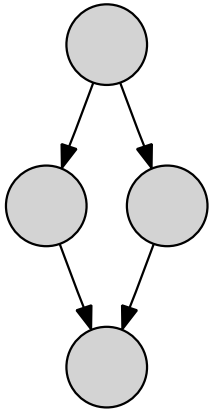


If

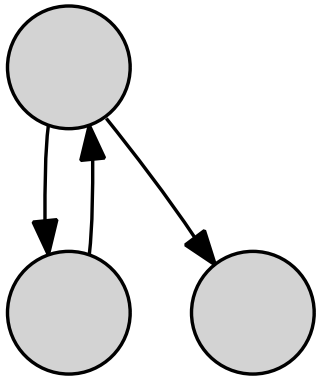
Alguns elementos do GFC



Sequence

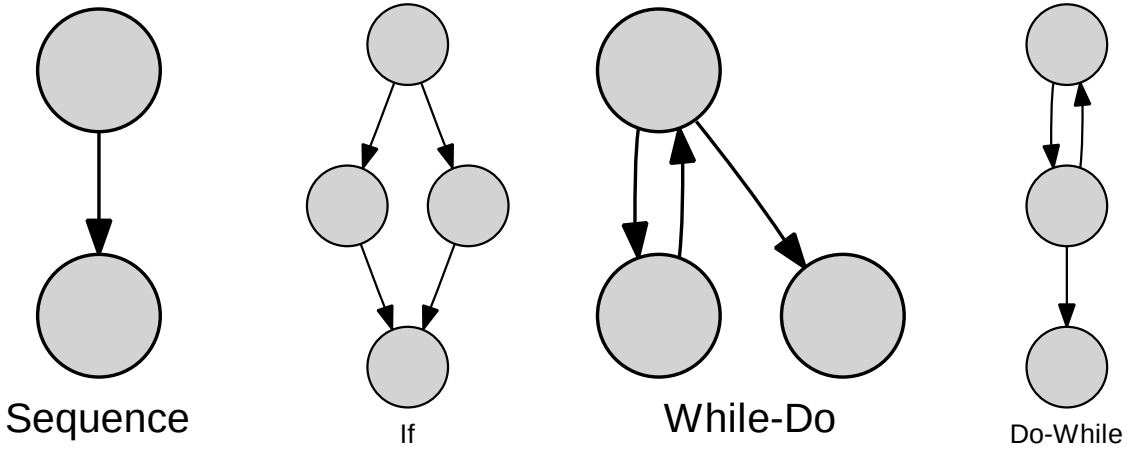


If

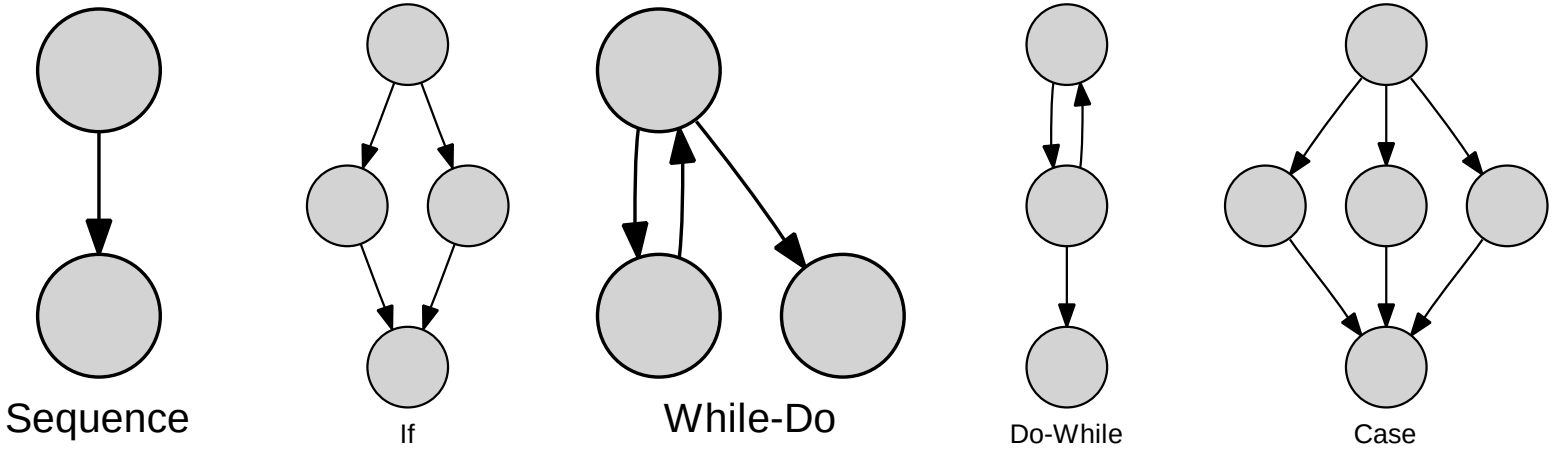


While-Do

Alguns elementos do GFC



Alguns elementos do GFC

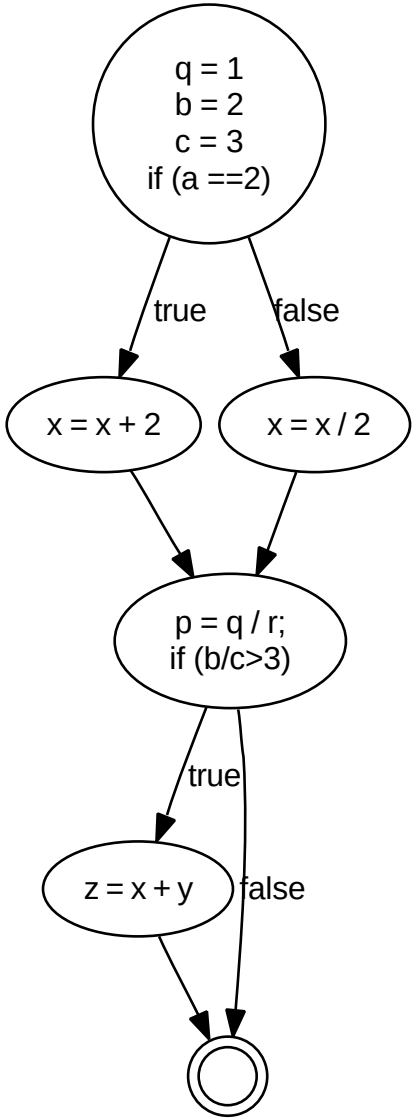


Mais exemplo

```
1  q = 1;
2  b = 2;
3  c = 3;
4  if (a ==2) {
5      x = x + 2;
6  }  else {
7      x = x / 2;
8  }
9  p = q / r;
10 if (b/c>3) {
11     z = x + y;
12 }
```

Mais exemplo

```
1   q = 1;
2   b = 2;
3   c = 3;
4   if (a ==2) {
5       x = x + 2;
6   }  else {
7       x = x / 2;
8   }
9   p = q / r;
10  if (b/c>3) {
11      z = x + y;
12  }
```

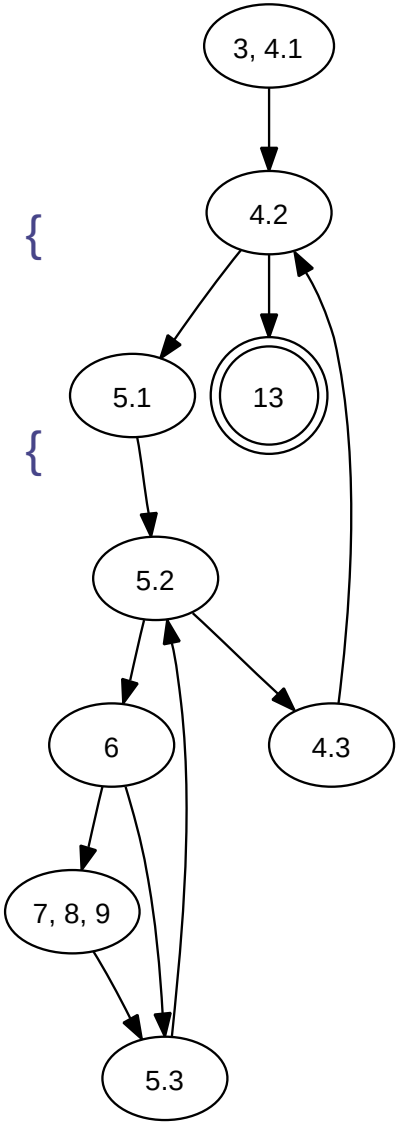


Mais exemplo – Bubble Sort

```
1  public class Sort {
2     public void bolha(int[] a, int size) {
3        int i, j, aux;
4        for (i = 0; i < size; i++) {
5           for (j = size – 1; j > i; j––) {
6              if (a[j – 1] > a[j]) {
7                 aux = a[j – 1];
8                 a[j – 1] = a[j];
9                 a[j] = aux;
10             }
11          }
12       }
```

Mais exemplo – Bubble Sort

```
1  public class Sort {
2     public void bolha(int[] a, int size) {
3        int i, j, aux;
4        for (i = 0; i < size; i++) {
5           for (j = size – 1; j > i; j––) {
6              if (a[j – 1] > a[j]) {
7                 aux = a[j – 1];
8                 a[j – 1] = a[j];
9                 a[j] = aux;
10             }
11          }
12       }
```



Exercícios (1)

```
15      void insercao(int a[], int size) {
16          int i, j, aux;
17          for (i = 1; i < size; i++) {
18              aux = a[i];
19              j = i - 1;
20              while (j >= 0 && a[j] >= aux) {
21                  a[j + 1] = a[j];
22                  j--;
23              }
24              a[j + 1] = aux;
25          }
26      }
```

Exercícios (2)

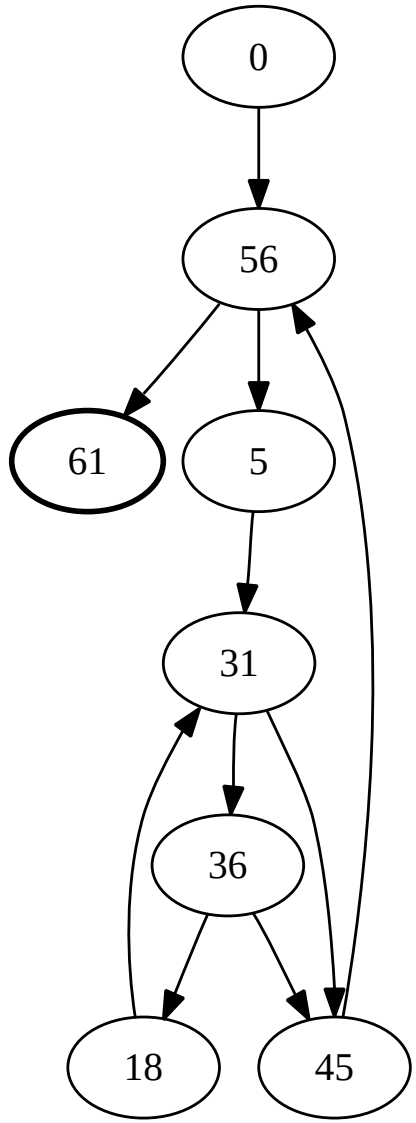
```
28      public static void heapsort(int n, double ra[]) {
29          int l, j, ir, i;
30          double rra;
31
32          l = (n >> 1) + 1;
33          ir = n;
34          for (;;) {
35              if (l > 1) {
36                  rra = ra[--l];
37              } else {
38                  rra = ra[ir];
39                  ra[ir] = ra[1];
40                  if (--ir == 1) {
41                      ra[1] = rra;
42                      return;
43                  }
44              }
45              i = l;
46              j = l << 1;
47              while (j <= ir) {
48                  if (j < ir && ra[j] < ra[j + 1]) {
49                      ++j;
50                  }
51                  if (rra < ra[j]) {
52                      ra[i] = ra[j];
53                      j += (i = j);
54                  } else {
55                      j = ir + 1;
56                  }
57              }
58              ra[i] = rra;
59          }
60      }
```

Exercícios (3)

```
62      void quicksort(int a[], int lo0, int hi0) {
63          int lo = lo0;
64          int hi = hi0;
65          int mid;
66
67          // pause for redraw
68          if (hi0 > lo0) {
69              mid = a[(lo0 + hi0) / 2];
70
71              while (lo <= hi) {
72                  while ((lo < hi0) && (a[lo] < mid))
73                      ++lo;
74
75                  while ((hi > lo0) && (a[hi] > mid))
76                      --hi;
77
78                  if (lo <= hi) {
79                      swap(a, lo, hi);
80                      ++lo;
81                      --hi;
82                  }
83              }
84
85              if (lo0 < hi)
86                  quicksort(a, lo0, hi);
87
88              if (lo < hi0)
89                  quicksort(a, lo, hi0);
90
91          }
92      }
```

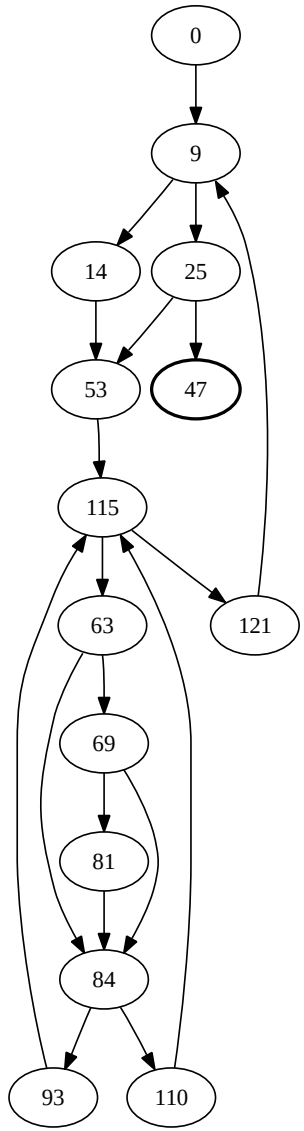
Solução (1)

```
15  void insercao(int a[], int size) {
16      int i, j, aux;
17      for (i = 1; i < size; i++) {
18          aux = a[i];
19          j = i - 1;
20          while (j >= 0 && a[j] >= aux) {
21              a[j + 1] = a[j];
22              j--;
23          }
24          a[j + 1] = aux;
25      }
26  }
```



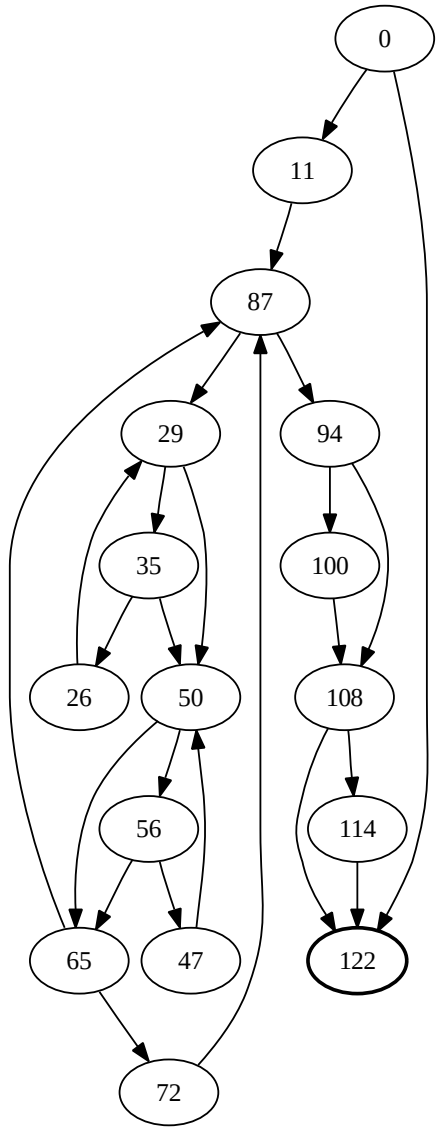
Solução (2)

```
28    public static void heapsort(int n, double ra[]) {
29        int l, j, ir, i;
30        double rra;
31
32        l = (n >> 1) + 1;
33        ir = n;
34        for (;;) {
35            if (l > 1) {
36                rra = ra[--l];
37            } else {
38                rra = ra[ir];
39                ra[ir] = ra[1];
40                if (--ir == 1) {
41                    ra[1] = rra;
42                    return;
43                }
44            }
45            i = l;
46            j = l << 1;
47            while (j <= ir) {
48                if (j < ir && ra[j] < ra[j + 1]) {
49                    ++j;
50                }
51                if (rra < ra[j]) {
52                    ra[i] = ra[j];
53                    j += (i = j);
54                } else {
55                    j = ir + 1;
56                }
57            }
58            ra[i] = rra;
59        }
60    }
```



Solução (3)

```
62      void quicksort(int a[], int lo0, int hi0) {
63          int lo = lo0;
64          int hi = hi0;
65          int mid;
66
67          // pause for redraw
68          if (hi0 > lo0) {
69              mid = a[(lo0 + hi0) / 2];
70
71              while (lo <= hi) {
72                  while ((lo < hi0) && (a[lo] < mid))
73                      ++lo;
74
75                  while ((hi > lo0) && (a[hi] > mid))
76                      --hi;
77
78                  if (lo <= hi) {
79                      swap(a, lo, hi);
80                      ++lo;
81                      --hi;
82                  }
83              }
84
85              if (lo0 < hi)
86                  quicksort(a, lo0, hi);
87
88              if (lo < hi0)
89                  quicksort(a, lo, hi0);
90
91          }
92      }
```



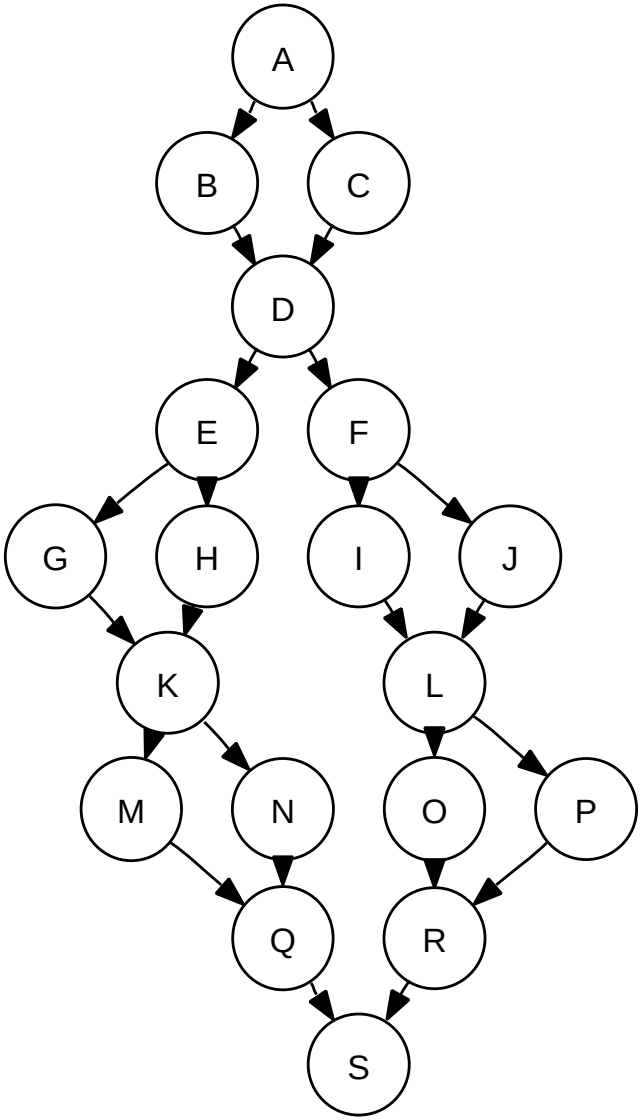
Algumas definições

- **Caminho**: seqüência de vértices conectados por arestas
- **Caminho completo**: caminho que inicia no nó de entrada e termina em um nó de saída
  - Representa uma execução da unidade
- **Caminho livre de laço**: caminho em que um nó não se repete
  - Possivelmente o primeiro e o último

Complexidade Ciclomática

- É uma medida de complexidade de código que é baseada no GFC
- Ela indica também o número de caminhos independentes (sem laço) que existem no grafo
- Por isso pode ser usada para selecionar casos de teste (caminhos)
- Cálculo:
  - C = arestas - vértices + 2
  - C = p + 1, onde p é o número de nós com decisão binária

Complexidade Ciclomática

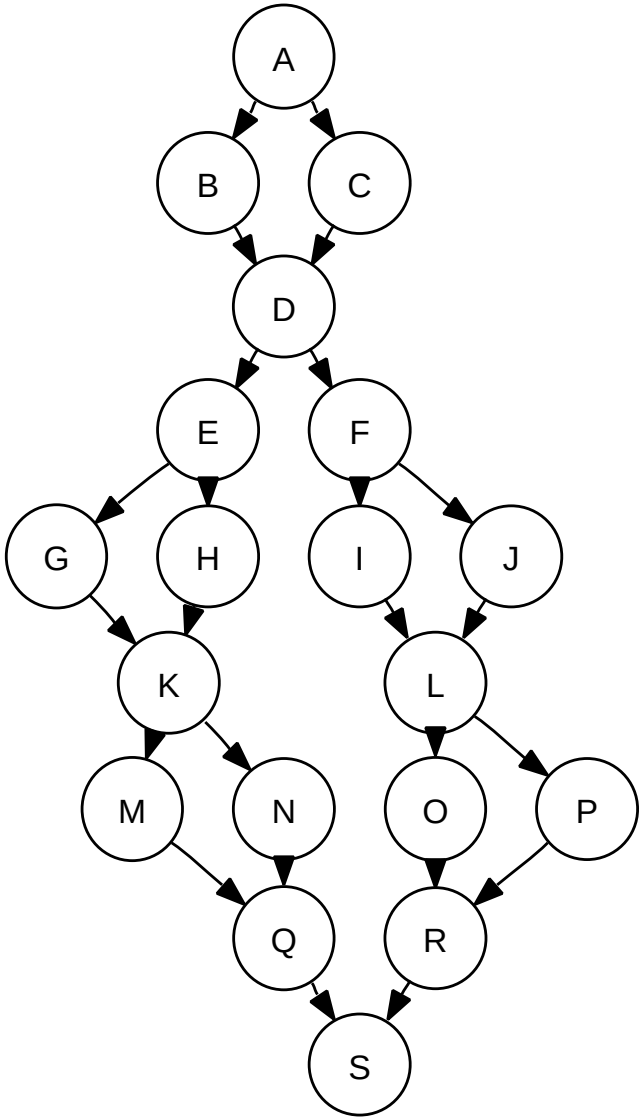


- C = 24 - 19 + 2 = 7
- C = 6 + 1 = 7
- Nesse grafo podemos identificar 7 caminhos “básicos” como descrito a seguir

Teste de caminhos básicos

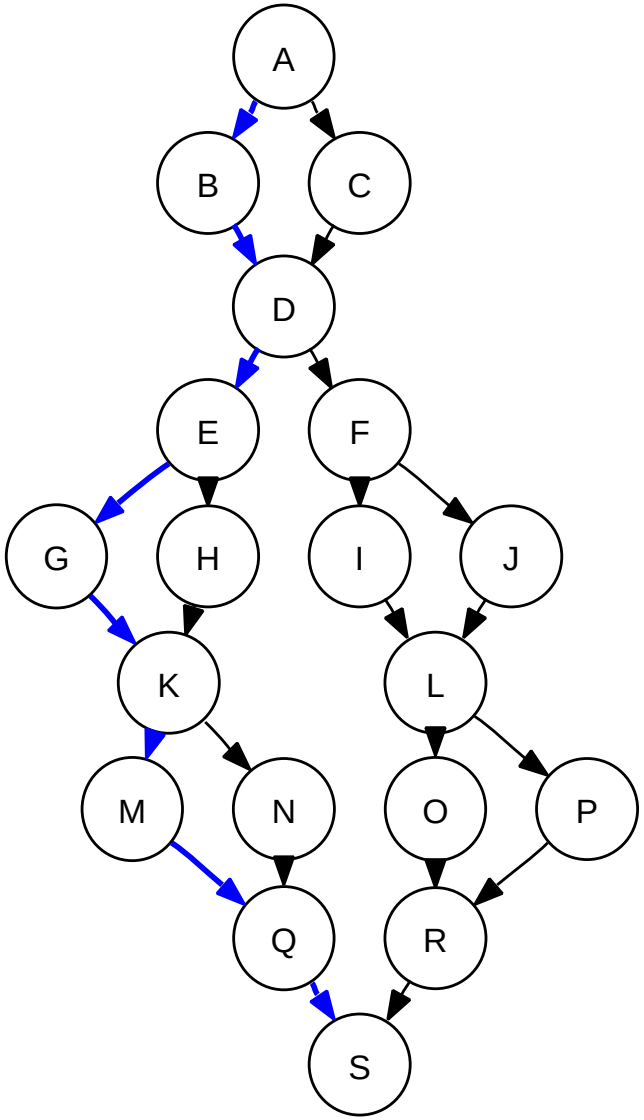
- Construir o GFC para o módulo do produto em teste.
- Calcular a Complexidade Ciclomática ($\mathcal{C}$).
- Selecionar um conjunto de $\mathcal{C}$ caminhos básicos.
- Criar um caso de teste para cada caminho básico.
- Executar os casos de testes.

Escolha dos caminhos



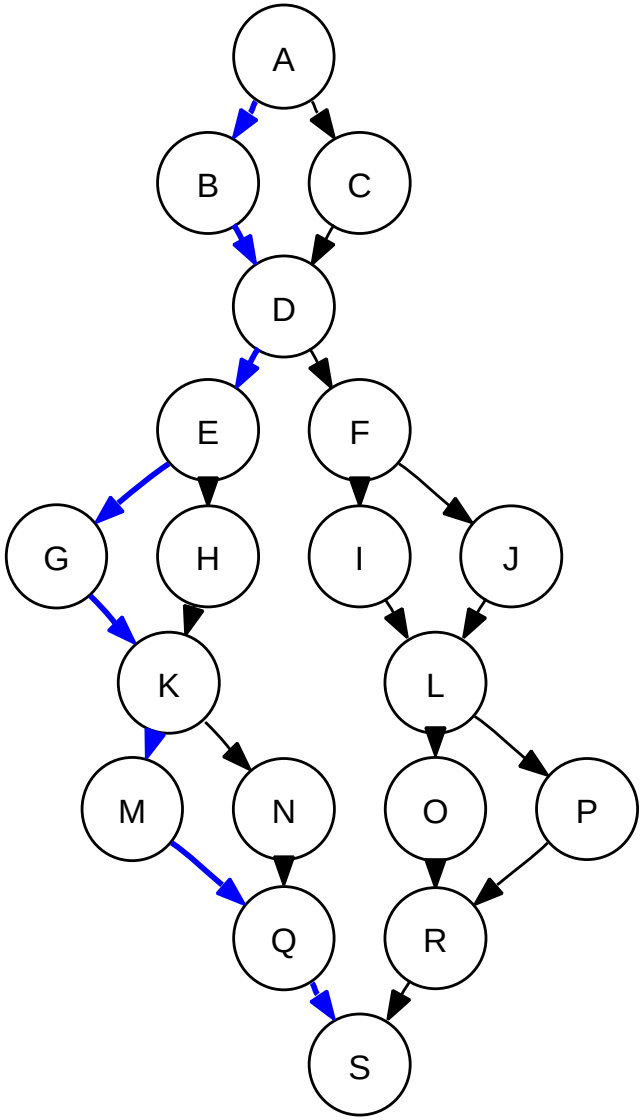
- Escolha um caminho básico. Esse caminho pode ser:
  - Caminho mais comum.
  - Caminho mais crítico.
  - Caminho mais importante do ponto de vista de teste.

Escolha dos caminhos



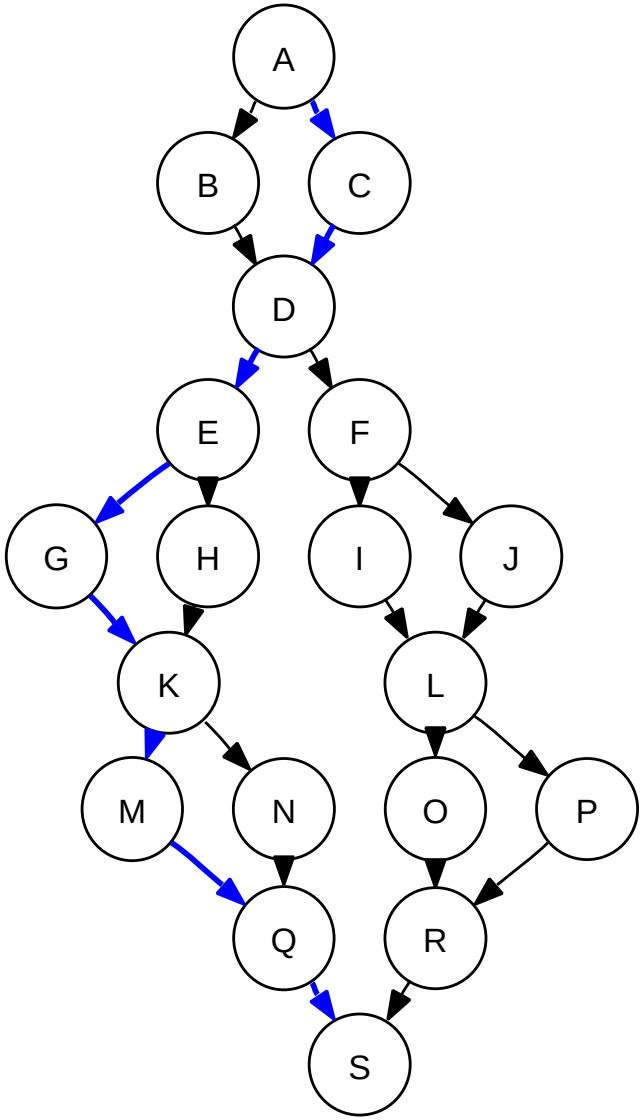
- Escolha um caminho básico. Esse caminho pode ser:
  - Caminho mais comum.
  - Caminho mais crítico.
  - Caminho mais importante do ponto de vista de teste.
- Caminho 1: ABDEGKMQS

Escolha dos caminhos



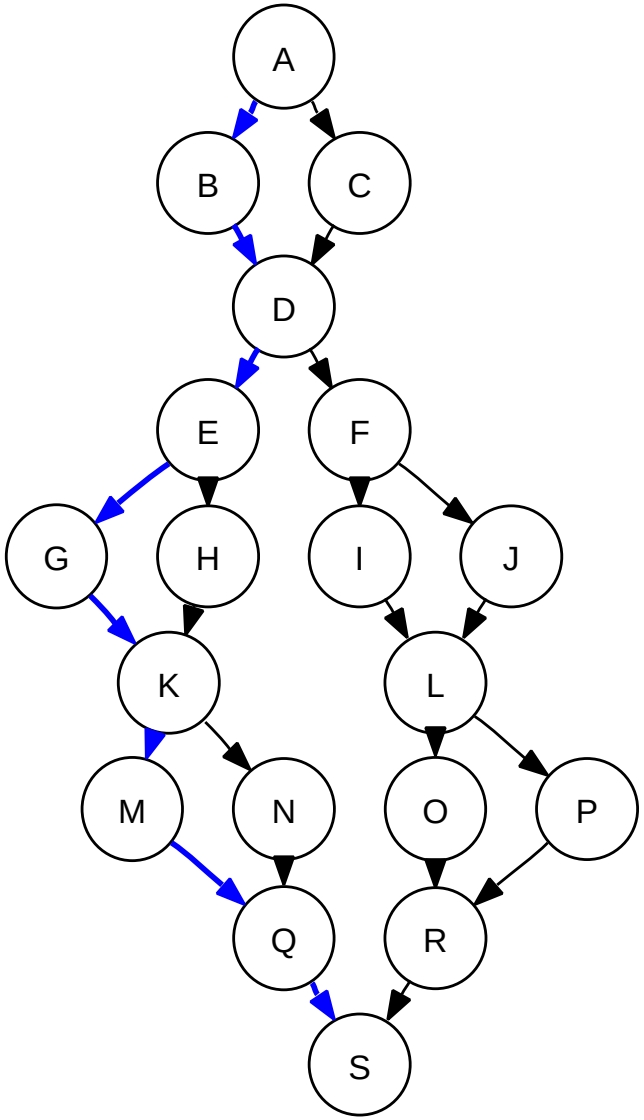
- Altere a saída só do primeiro comando de decisão

Escolha dos caminhos



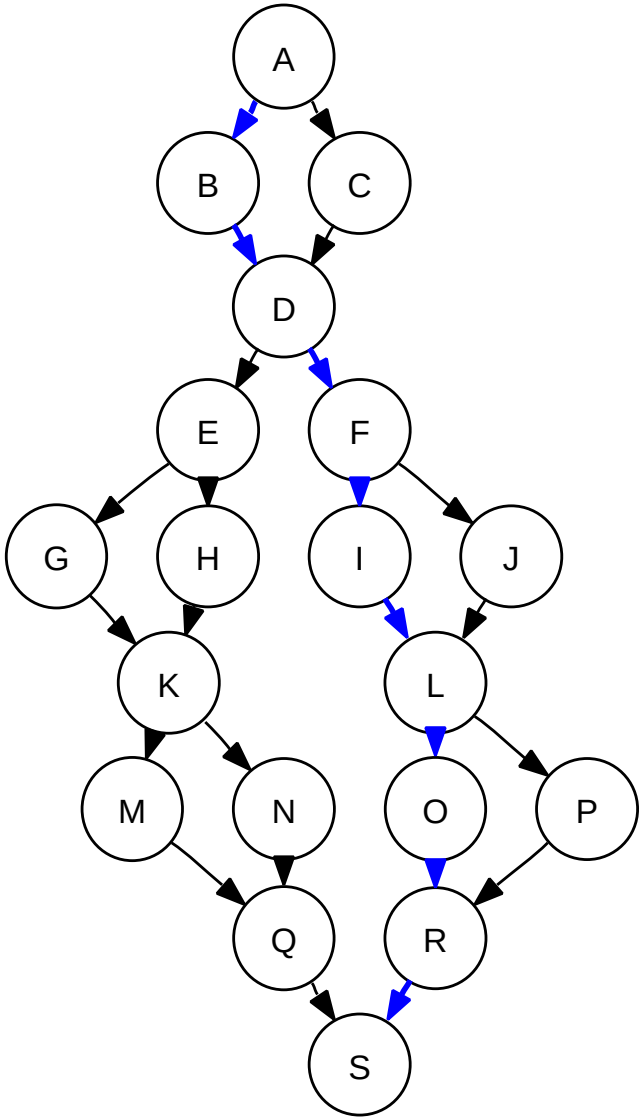
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS

Escolha dos caminhos



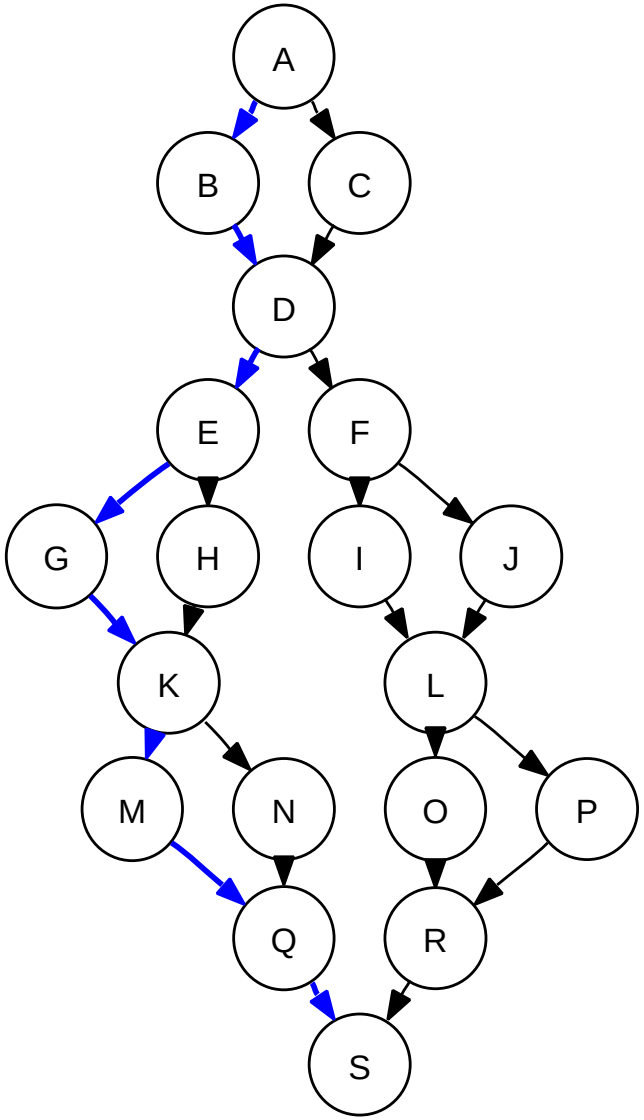
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.

Escolha dos caminhos



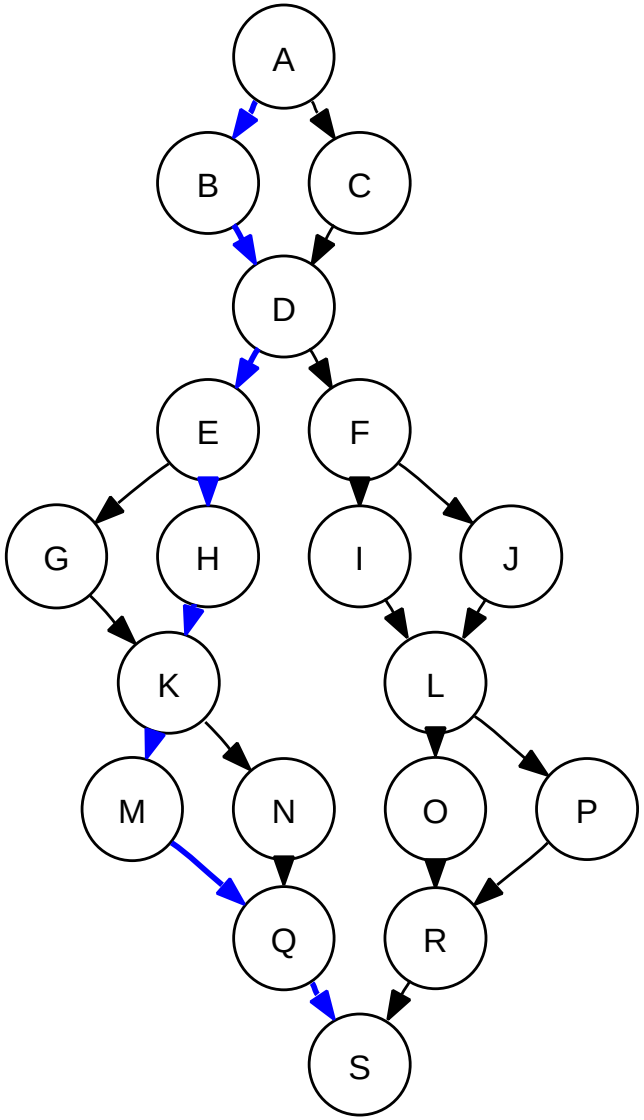
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.
- Caminho 3: ABDFILORS

Escolha dos caminhos



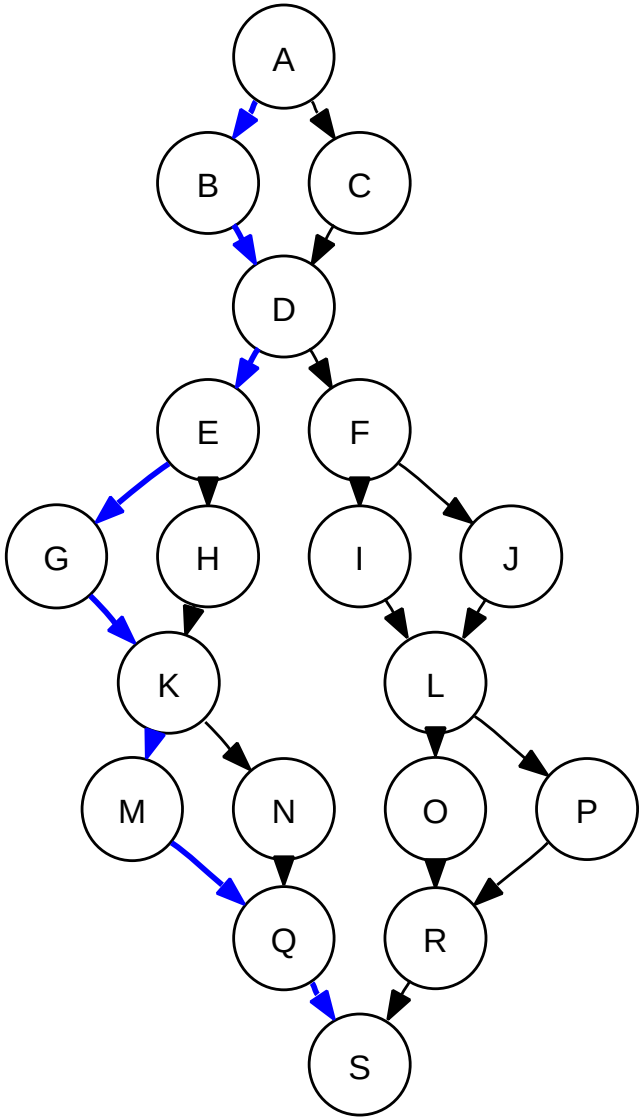
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.
- Caminho 3: ABDFILORS
- Alterar a saída do terceiro comando de decisão.

Escolha dos caminhos



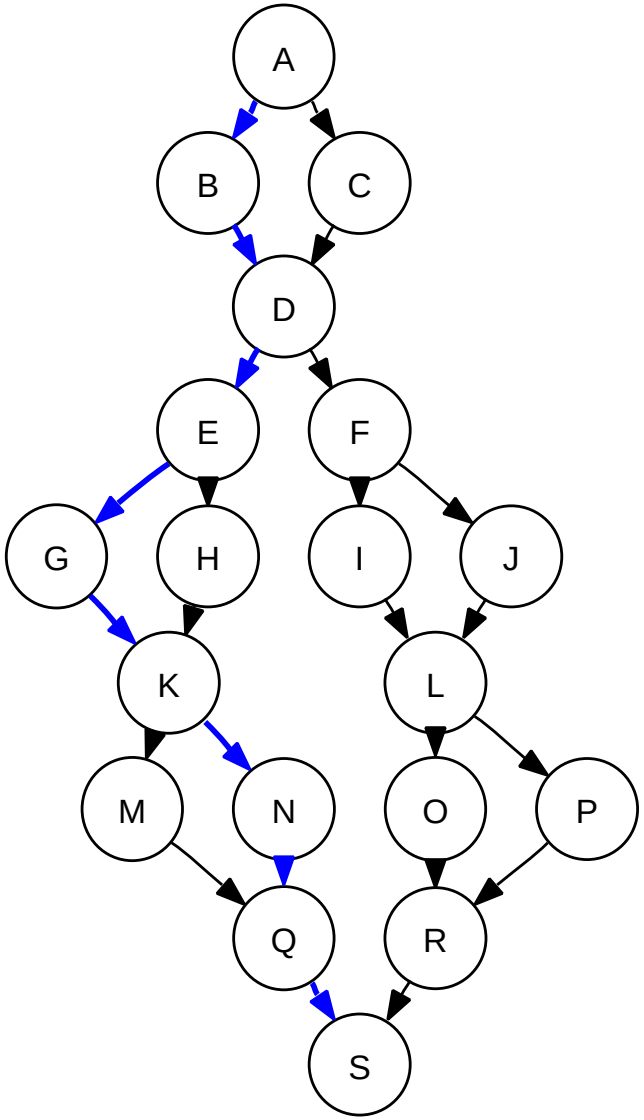
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.
- Caminho 3: ABDFILORS
- Alterar a saída do terceiro comando de decisão.
- Caminho 4: ABDEHKMQS

Escolha dos caminhos



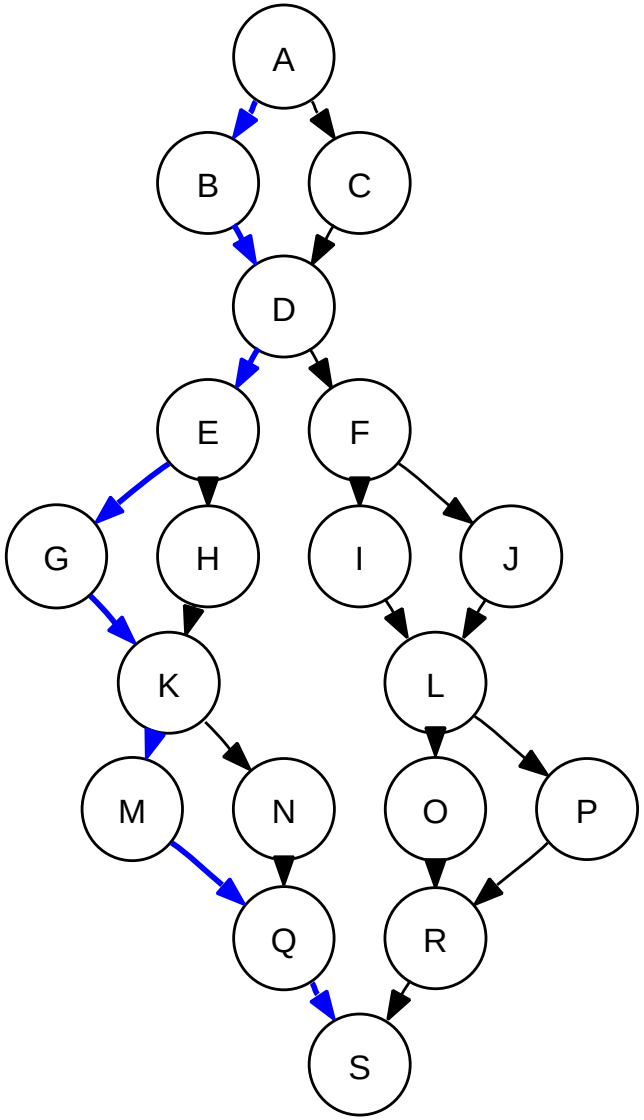
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.
- Caminho 3: ABDFILORS
- Alterar a saída do terceiro comando de decisão.
- Caminho 4: ABDEHKMQS
- Alterar a saída do quarto comando de decisão.

Escolha dos caminhos



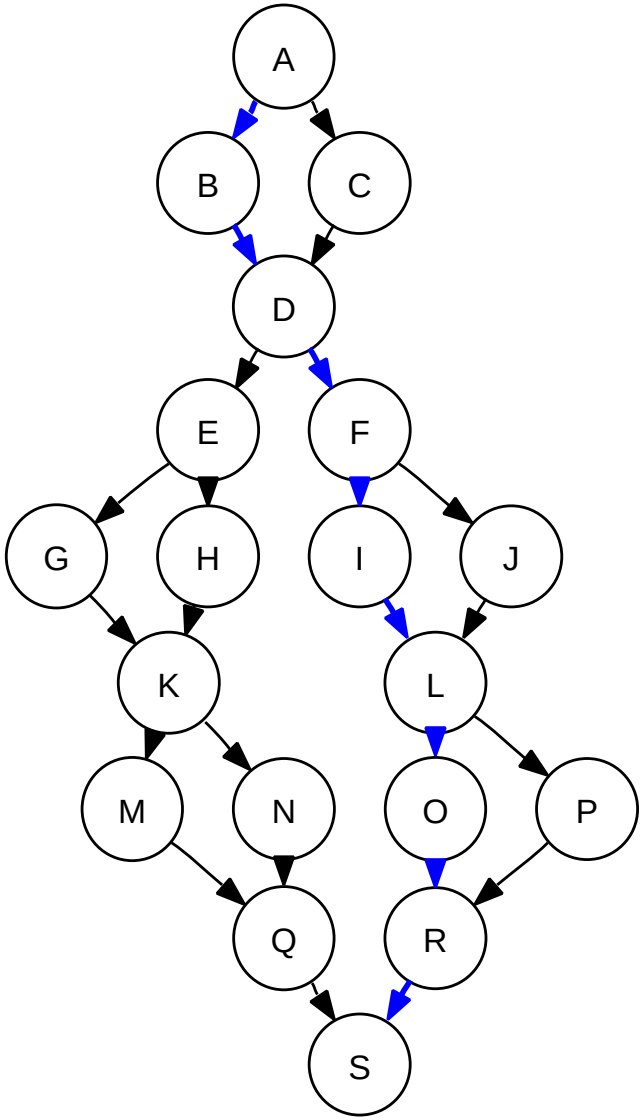
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.
- Caminho 3: ABDFILORS
- Alterar a saída do terceiro comando de decisão.
- Caminho 4: ABDEHKMQS
- Alterar a saída do quarto comando de decisão.
- Caminho 5: ABDEGKNQS

Escolha dos caminhos



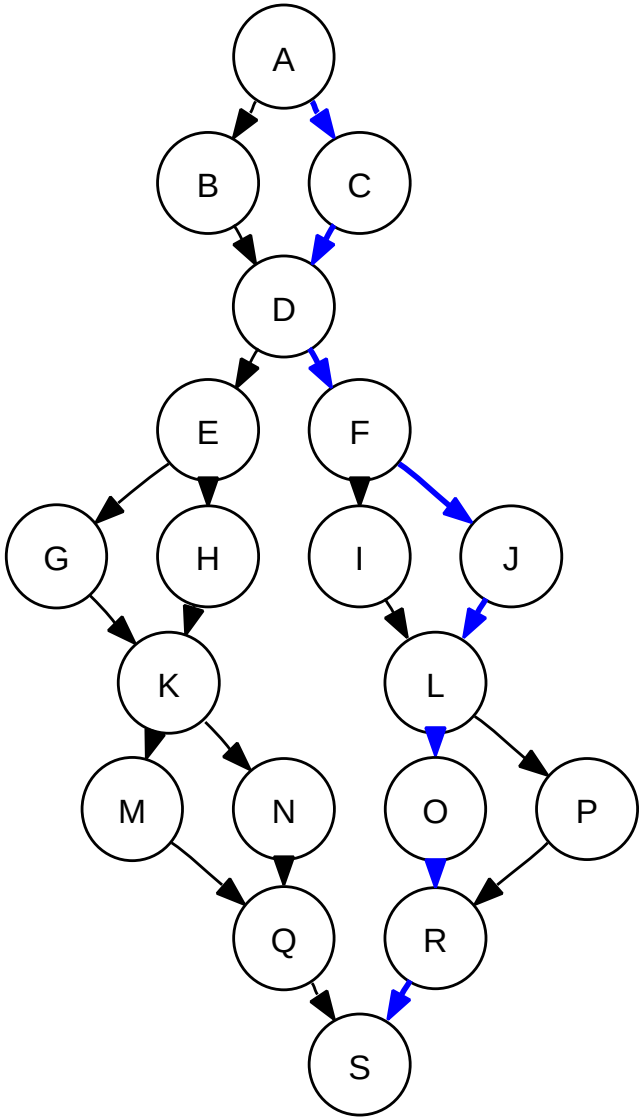
- Todos os comandos de decisão do caminho base foram modificados

Escolha dos caminhos



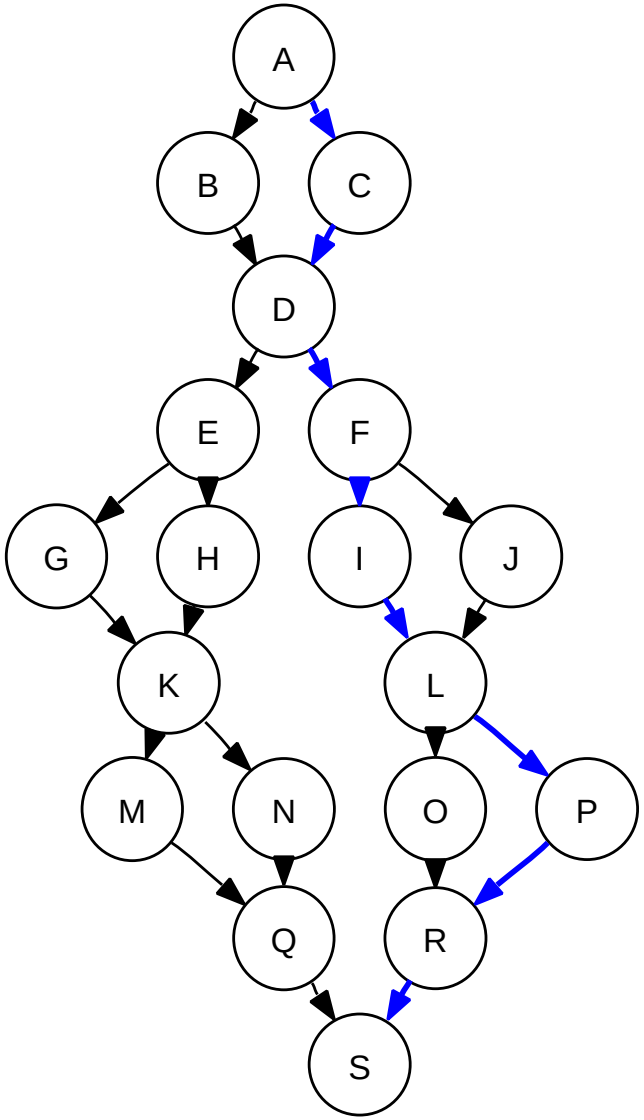
- Todos os comandos de decisão do caminho base foram modificados
- Fazer o mesmo para o caminho 3 (o 2 já foi tratado)

Escolha dos caminhos



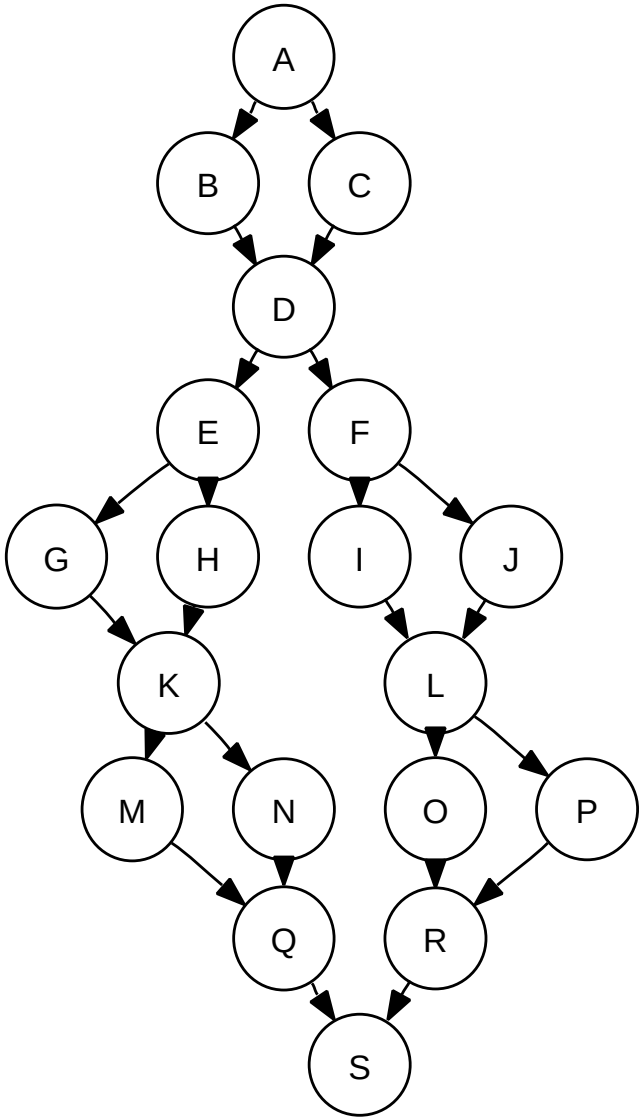
- Todos os comandos de decisão do caminho base foram modificados
- Fazer o mesmo para o caminho 3 (o 2 já foi tratado)
- Caminho 6: ACDFJLORS

Escolha dos caminhos



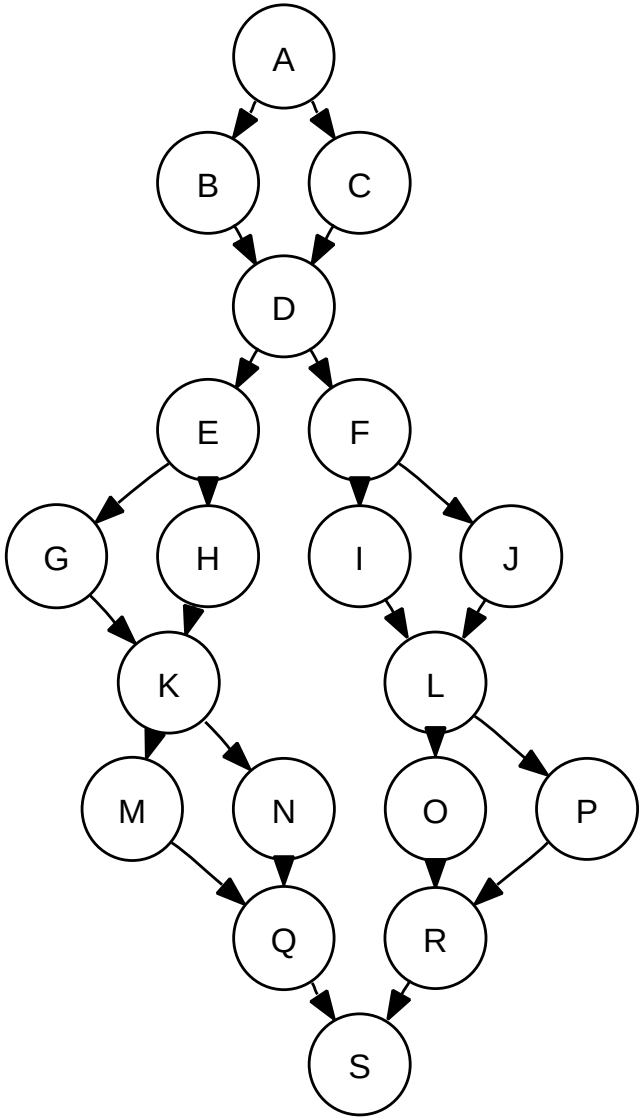
- Todos os comandos de decisão do caminho base foram modificados
- Fazer o mesmo para o caminho 3 (o 2 já foi tratado)
- Caminho 6: ACDFJLORS
- Caminho 7: ACDFILPRS

Requisitos de teste



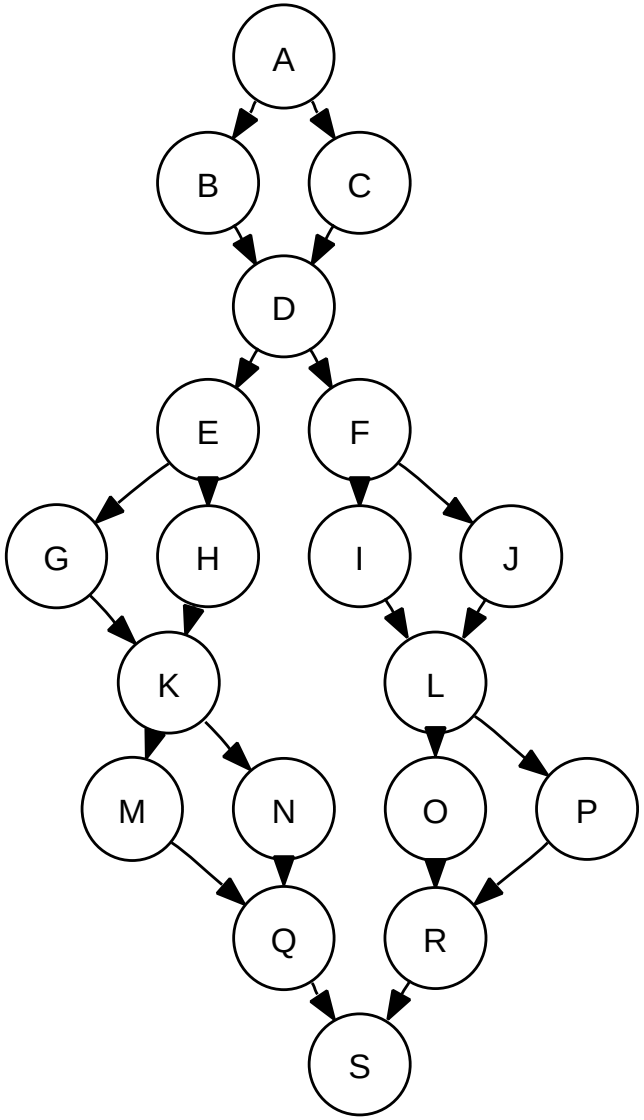
- Requisitos de testes derivado pelo critério.
  - ABDEGKMQS
  - ACDEGKMQS
  - ABDFILORS
  - ABDEHKMQS
  - ABDEGKNQS
  - ACDFJLORS
  - ACDFILPRS

Requisitos de teste



- Requisitos de testes derivado pelo critério.
  - ABDEGKMQS
  - ACDEGKMQS
  - ABDFILORS
  - ABDEHKMQS
  - ABDEGKNQS
  - ACDFJLORS
  - ACDFILPRS
- Conjunto criado não é único.

Requisitos de teste

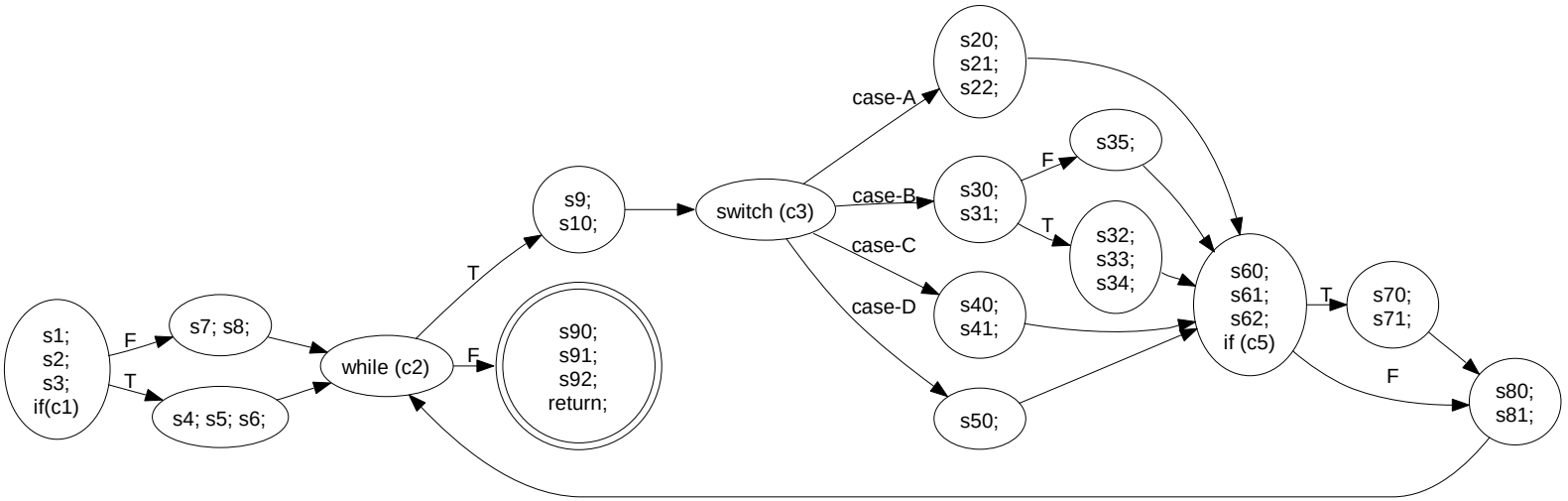


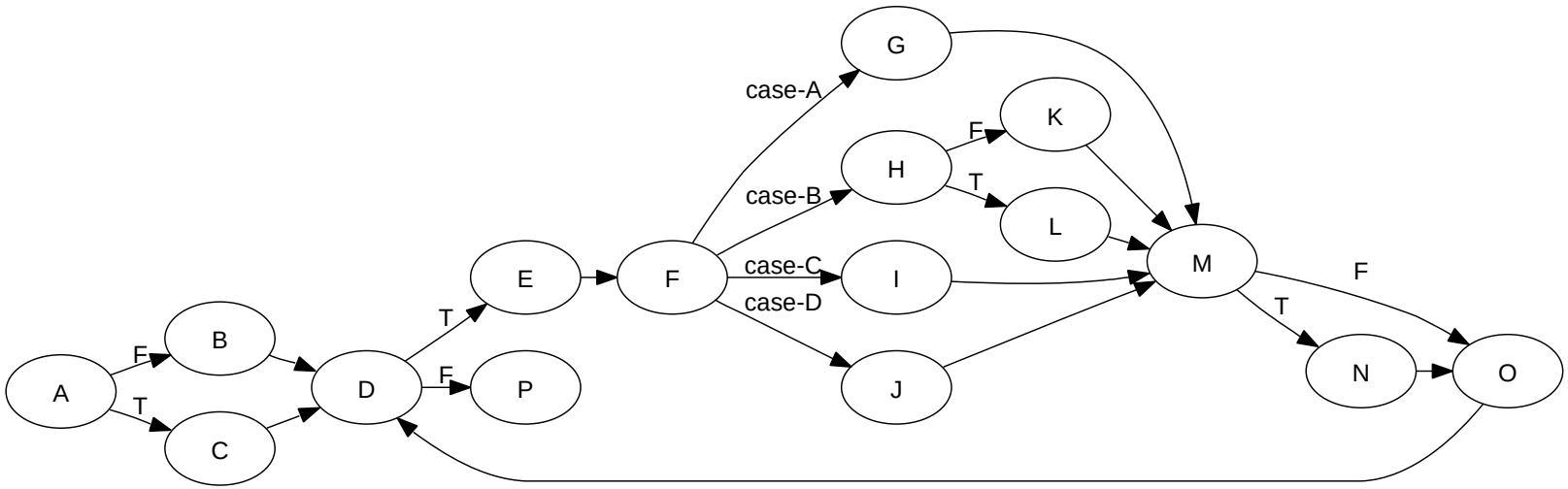
- Requisitos de testes derivado pelo critério.
  - ABDEGKMQS
  - ACDEGKMQS
  - ABDFILORS
  - ABDEHKMQS
  - ABDEGKNQS
  - ACDFJLORS
  - ACDFILPRS
- Conjunto criado não é único.
- Propriedade: o conjunto de teste que exercita os caminhos básicos também exercita todos-nós e todos-arcos do programa.

Exercício

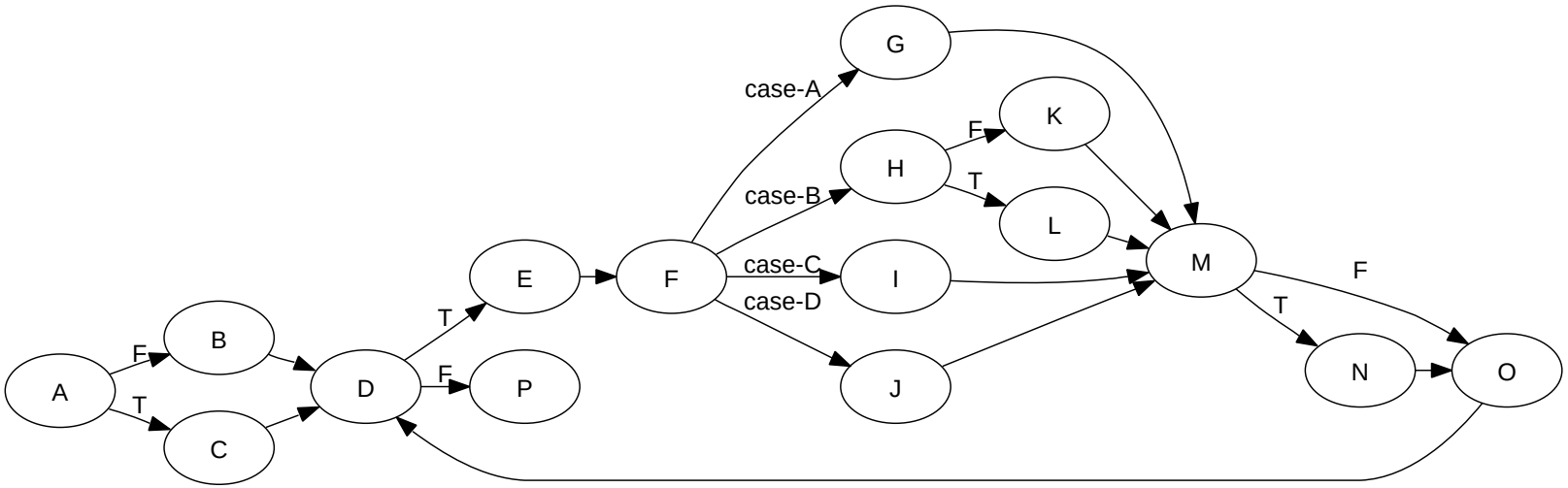
```
1  boolean evaluateBuySell (TickerSymbol ts) {
2  s1;
3  s2;
4  s3;
5  if (c1) {s4; s5; s6;}
6  else {s7; s8;}
7  while (c2) {
8     s9;
9     s10;
10    switch (c3) {
11       case–A:
12          s20;
13          s21;
14          s22;
15          break; // End of Case–A
16       case–B:
17          s30;
18          s31;
19          if (c4) {
20             s32;
21             s33;
22             s34;
23          }
24          else {
25             s35;
26          }
27          break; // End of Case–B
28       case–C:
29          s40;
30          s41;
31          break; // End of Case–C
32       case–D:
33          s50;
34          break; // End of Case–D
35       } // End Switch
36       s60;
37       s61;
38       s62;
39       if (c5) {s70; s71; }
40       s80;
41       s81;
42 } // End While
43 s90;
44 s91;
45 s92;
46 return result;
```

Solução – GFC


s1;
s2;
s3;
if(c1)
F
T
s7; s8;
s4; s5; s6;
while (c2)
T
F
s9;
s10;
s90;
s91;
s92;
return;
switch (c3)
case-A
case-B
case-C
case-D
s20;
s21;
s22;
s30;
s31;
s40;
s41;
s50;
F
T
s35;
s32;
s33;
s34;
s60;
s61;
s62;
if (c5)
T
F
s70;
s71;
s80;
s81;


A
F
T
B
C
D
T
F
E
P
F
case-A
case-B
case-C
case-D
G
H
I
J
F
T
K
L
M
F
T
N
O

Solução – complexidade



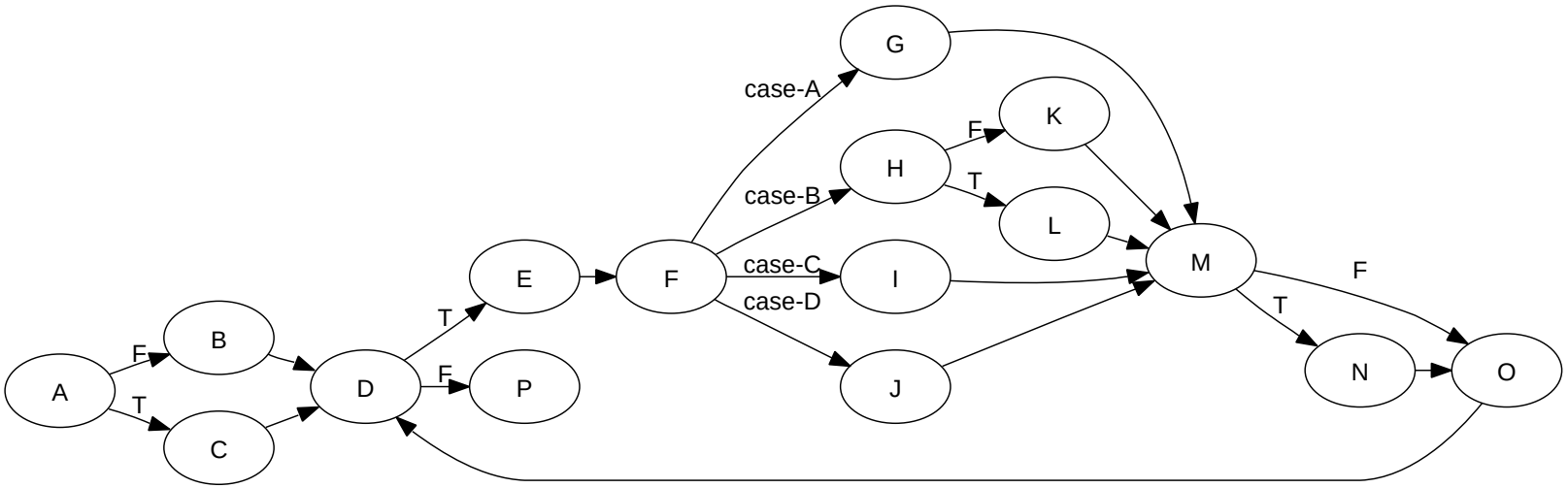
- Cálculo da complexidade ciclomática para o GFC acima:

  C = arcos - nós + 2

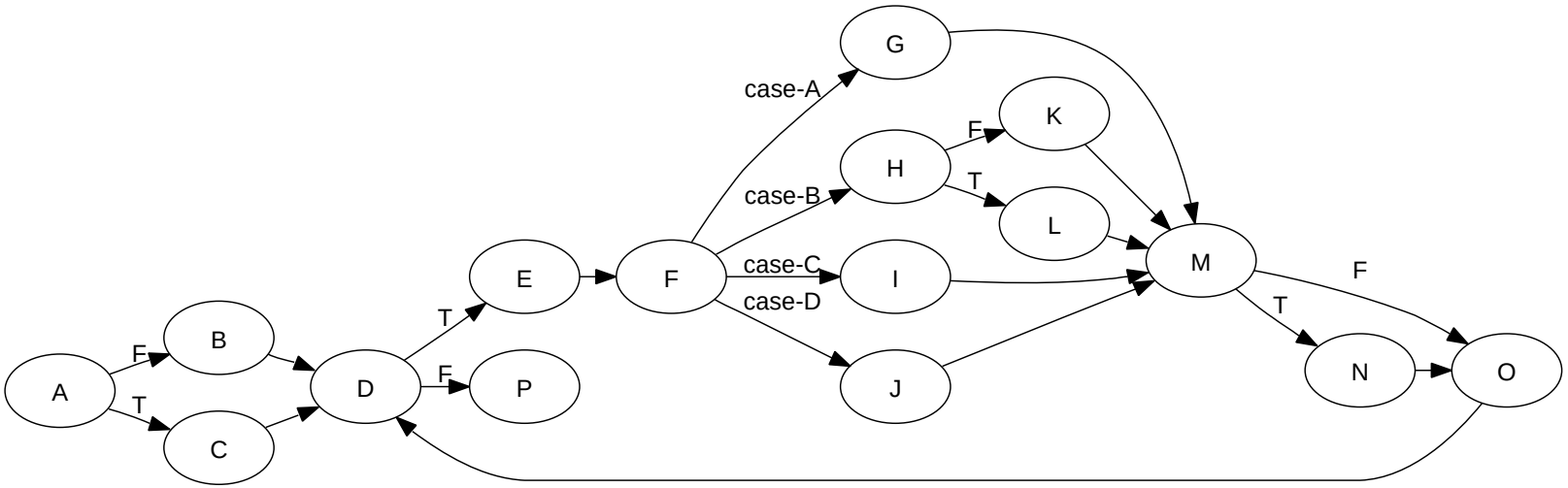
  C = 22-16+2

  C = 8

Solução – requisitos



Solução – requisitos



1. ABDP
2. ACDP
3. ABDEFGMODP
4. ABDEFHKMODP
5. ABDEFIMODP
6. ABDEFJMODP
7. ABDEFGMNODP
8. ABDEFHLMODP

Solução – casos de teste

1. ABDP
2. ACDP
3. ABDEFGMODP
4. ABDEFHKMODP
5. ABDEFIMODP
6. ABDEFJMODP
7. ABDEFHLMODP
8. ABDEFIMNODP

Caso Teste	C1	C2	C3	C4	C5
1	False	False	N/A	N/A	N/A
2	True	False	N/A	N/A	N/A
3	False	True	A	N/A	False
4	False	True	B	False	False
5	False	True	C	N/A	False
6	False	True	D	N/A	False
7	False	True	A	N/A	True
8	False	True	B	True	False

Exercício

- Mostre que se o GFC tem mais do que um nó de saída, a fórmula C = arestas - nós + 2 pode dar um resultado incorreto
- Sugira uma outra forma de computar o valor da CC para esses casos

Executabilidade

- Um dos problemas no teste estrutural, em geral, é a executabilidade

Executabilidade

- Um dos problemas no teste estrutural, em geral, é a executabilidade
- Um caminho π é dito não executável quando não existe um dado de entrada que faça com que esse caminho seja executado

Executabilidade

- Um dos problemas no teste estrutural, em geral, é a executabilidade
- Um caminho π 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

Executabilidade

- Um dos problemas no teste estrutural, em geral, é a executabilidade
- Um caminho π 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 indecidível

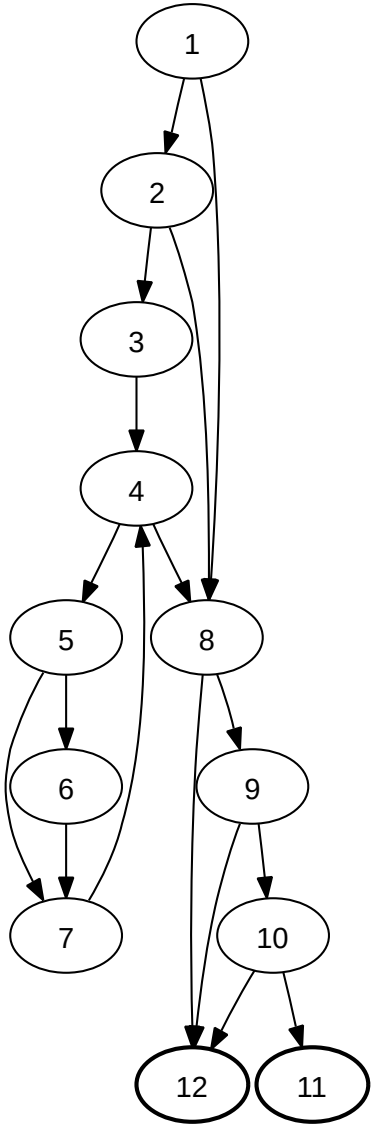
Executabilidade

- Um dos problemas no teste estrutural, em geral, é a executabilidade
- Um caminho π 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 indecidível
- É um problema para a automatização da atividade de teste

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 um vez
- Todas-aresta: 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

Todos-nós, Todos-arcos



- Todos-nós
  - (1 #@, Inválido) – executa nós (1,2,3,4,5,6,7,4,8,12)
  - (`i`, Válido) – executa nós (1,2,8,9,10,11)
- $T_{all-nodes}$ = {(1 #@, Inválido),(`i`, Válido)} é todos-nós-adequado.
- Não é todos-arcos-adequate: arestas $(1, 8)$, $(5, 7)$, $(9, 12)$ and $(10, 12)$ não são executadas por nenhum caso de teste em $T_{all-nodes}$.
- Edge $(9, 12)$ é não executável
- $T_{all-edges} = T_{all-nodes} \cup$ {(`A1b2C3d`, Inválido), ("", Inválido) } é todas-arestas-adequado.

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

A ferramenta JaBUTi

- `http://incubadora.fapesp.br/projects/jabuti/`
- `Jabuti-bin.zip` possui tudo que precisa para executar a ferramenta
- `java -cp Jabuti-bin.zip br.jabuti.gui.JabutiGUI`
- Programa “dot” (GraphViz) deve estar instalado em:
  - `\Arquivos de programas\ATT\GraphViz\bin\dot.exe`
  - `/usr/bin/dot`
- Se não estiver o programa pede o endereço do programa, quando for feita a visualização do grafo