SSC 0721 Teste e Inspeção de Software

Critérios de fluxo de controle

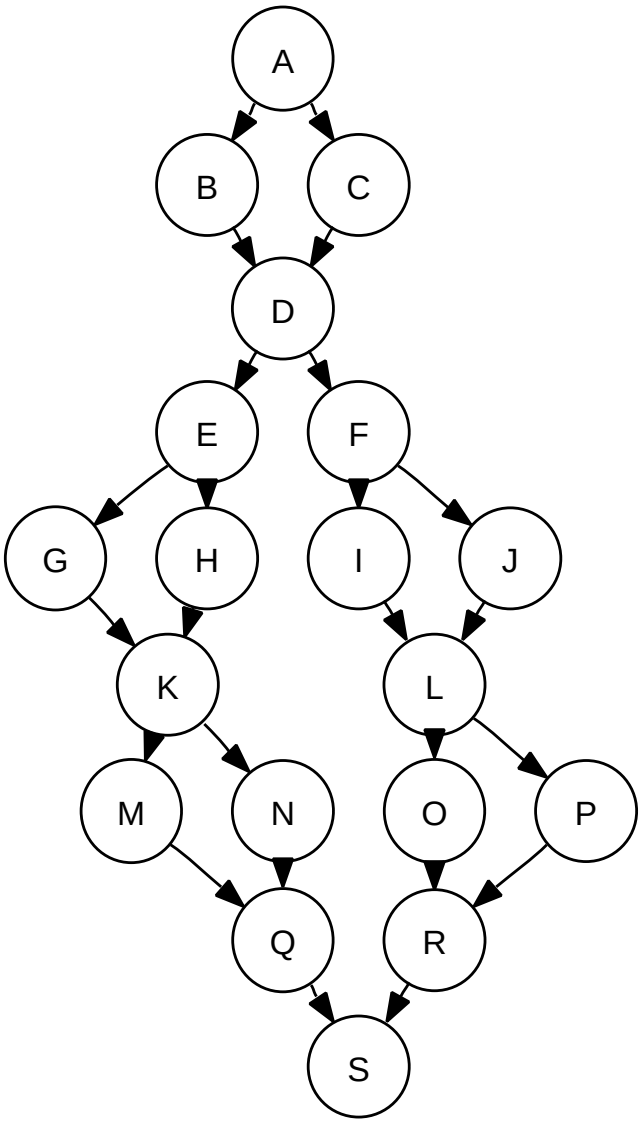
Prof. Marcio E. Delamaro

delamaro@icmc.usp.br

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

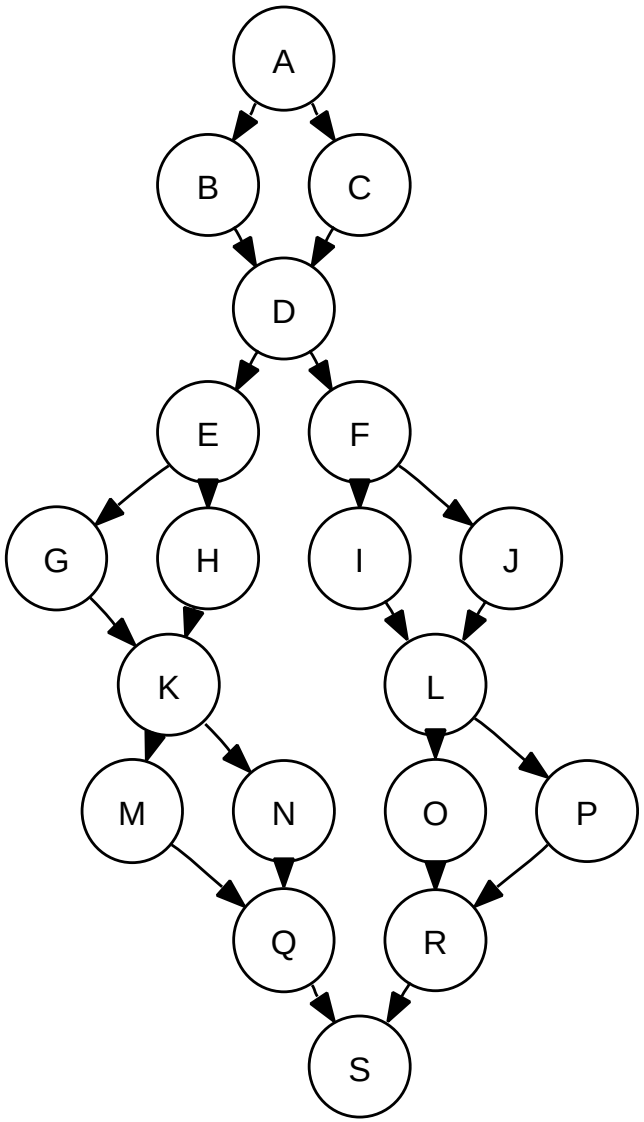


- C = 24 - 19 + 2 = 7
- C = 6 + 1 = 7
- Nesse grafo podemos identificar 7 caminhos “básicos” como descrito a seguir

Teste de caminhos básicos

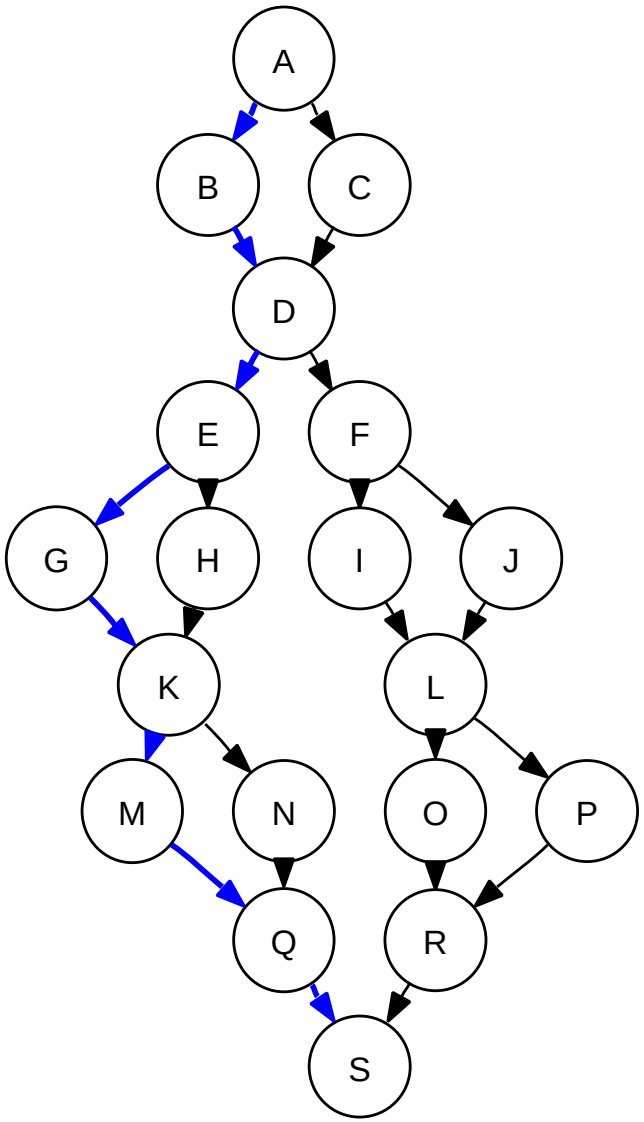
- Construir o GFC para o módulo do produto em teste.
- Calcular a Complexidade Ciclomática ($\mathcal{C}$).
- Selecionar um conjunto de $\mathcal{C}$ caminhos básicos.
- Criar um caso de teste para cada caminho básico.
- Executar os casos de testes.

Escolha dos caminhos



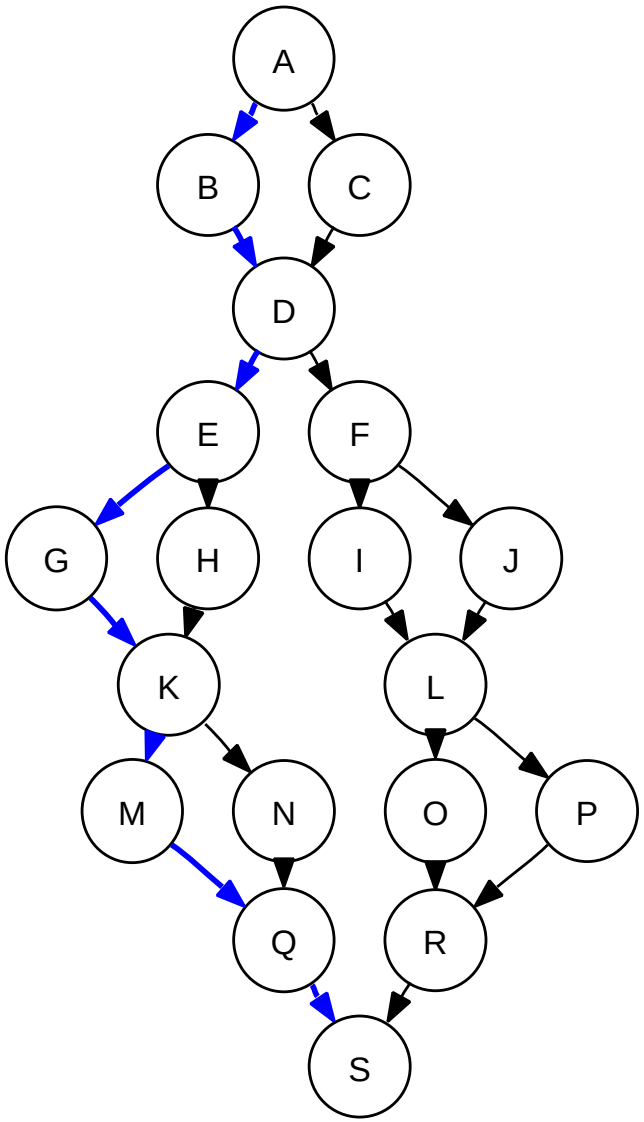
- Escolha um caminho básico. Esse caminho pode ser:
  - Caminho mais comum.
  - Caminho mais crítico.
  - Caminho mais importante do ponto de vista de teste.

Escolha dos caminhos



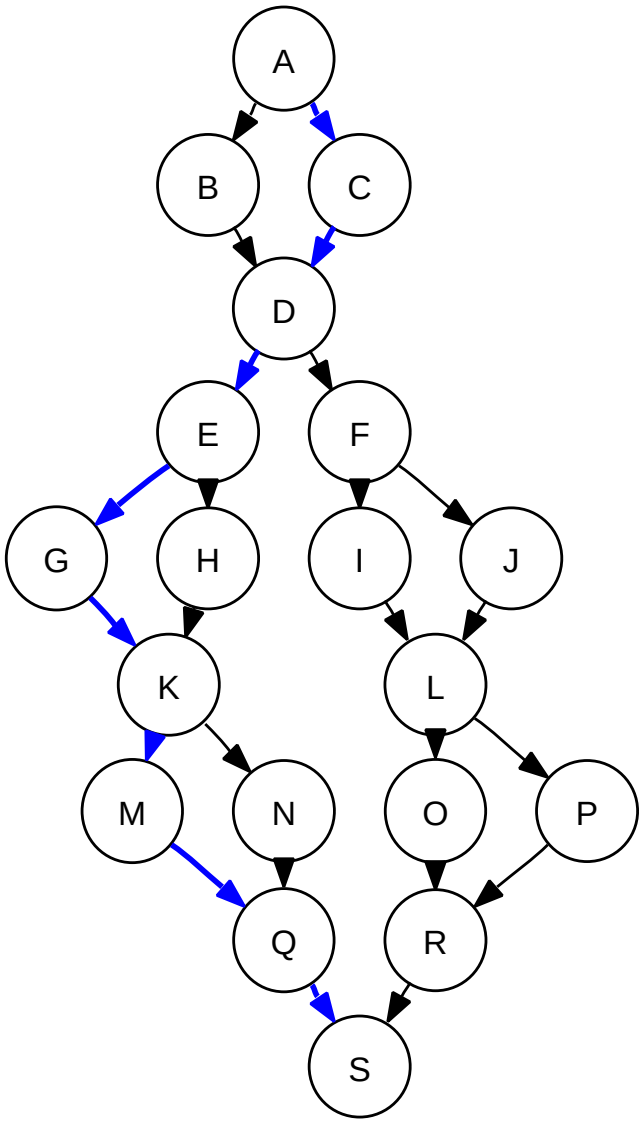
- Escolha um caminho básico. Esse caminho pode ser:
  - Caminho mais comum.
  - Caminho mais crítico.
  - Caminho mais importante do ponto de vista de teste.
- Caminho 1: ABDEGKMQS

Escolha dos caminhos



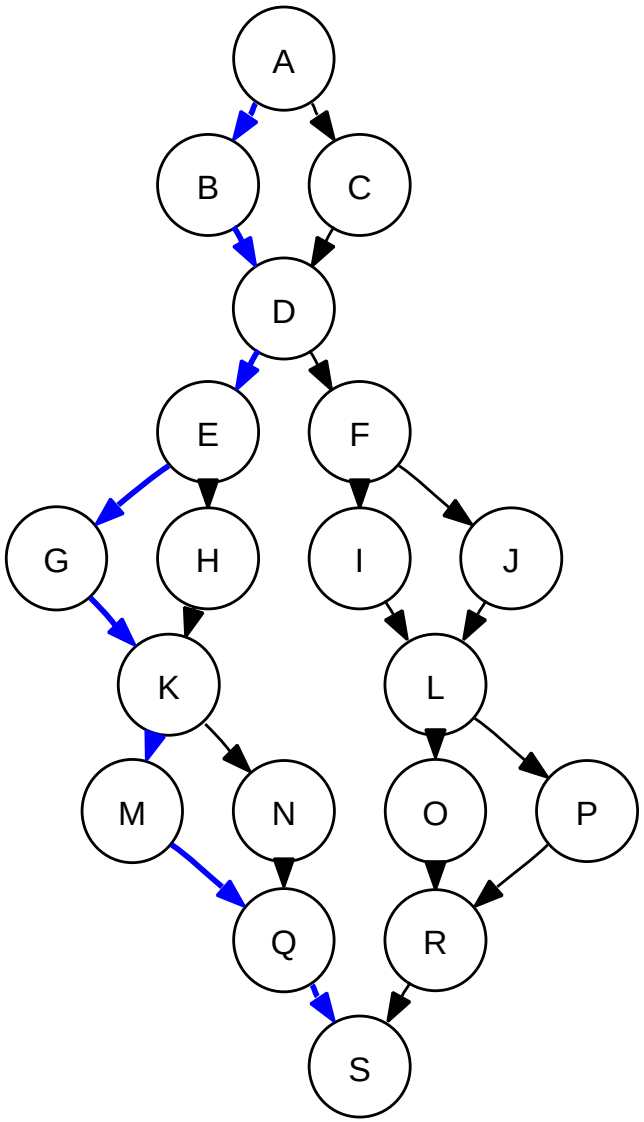
- Altere a saída só do primeiro comando de decisão

Escolha dos caminhos



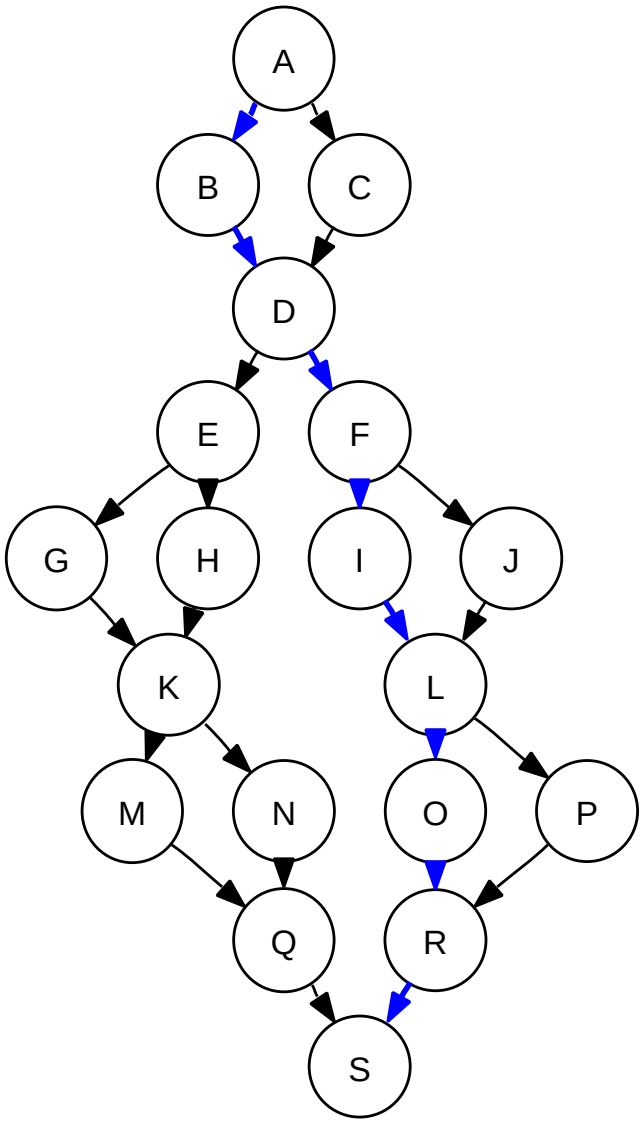
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS

Escolha dos caminhos



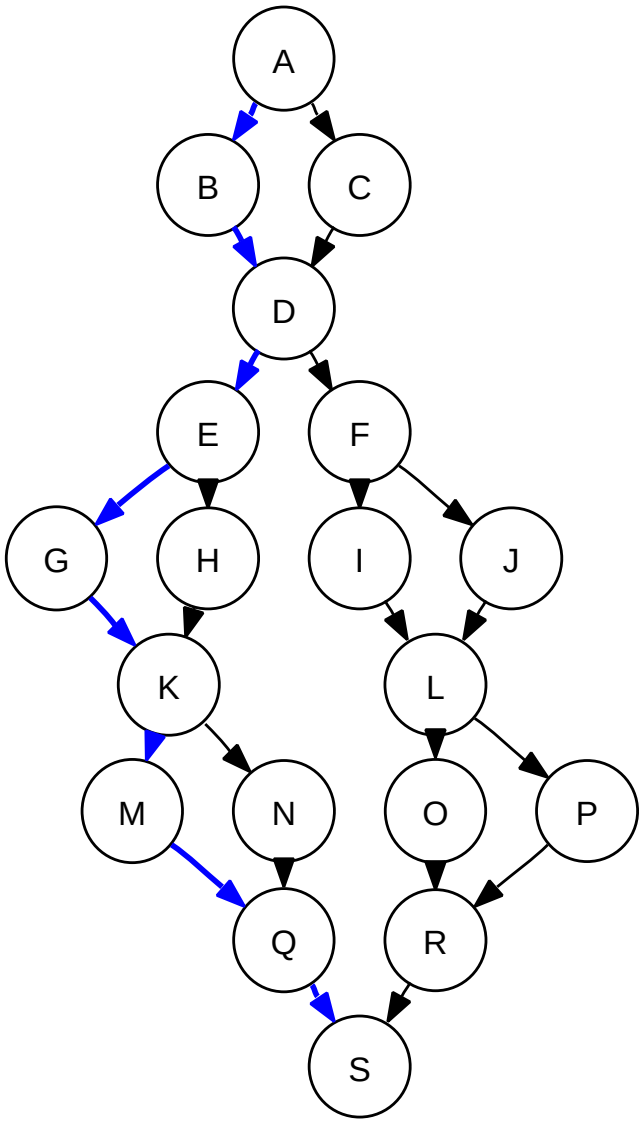
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.

Escolha dos caminhos



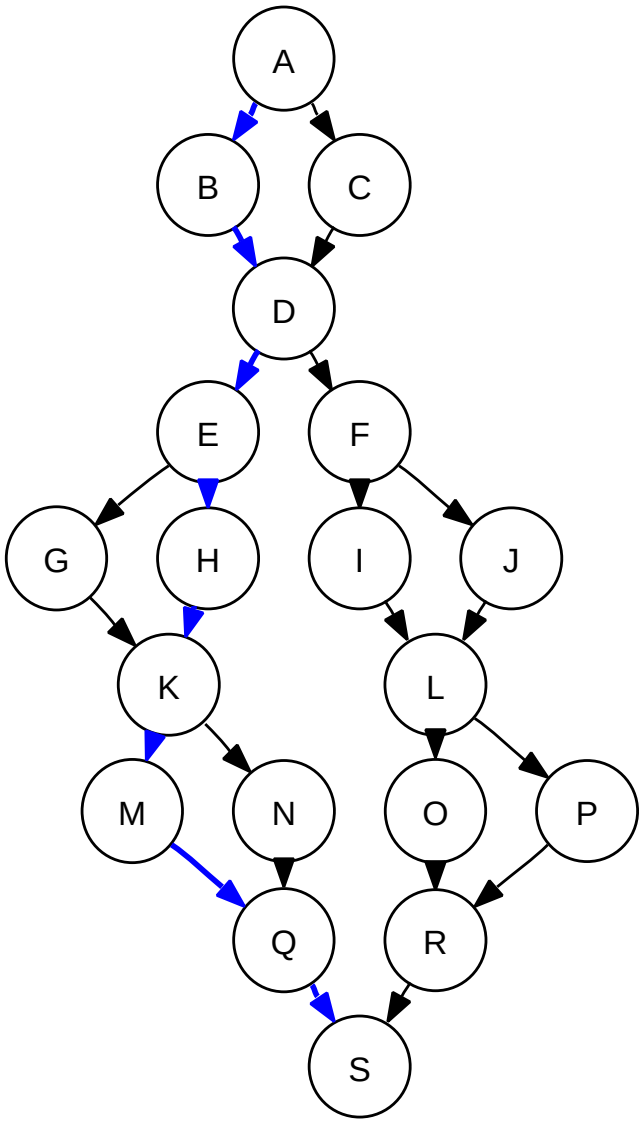
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.
- Caminho 3: ABDFILORS

Escolha dos caminhos



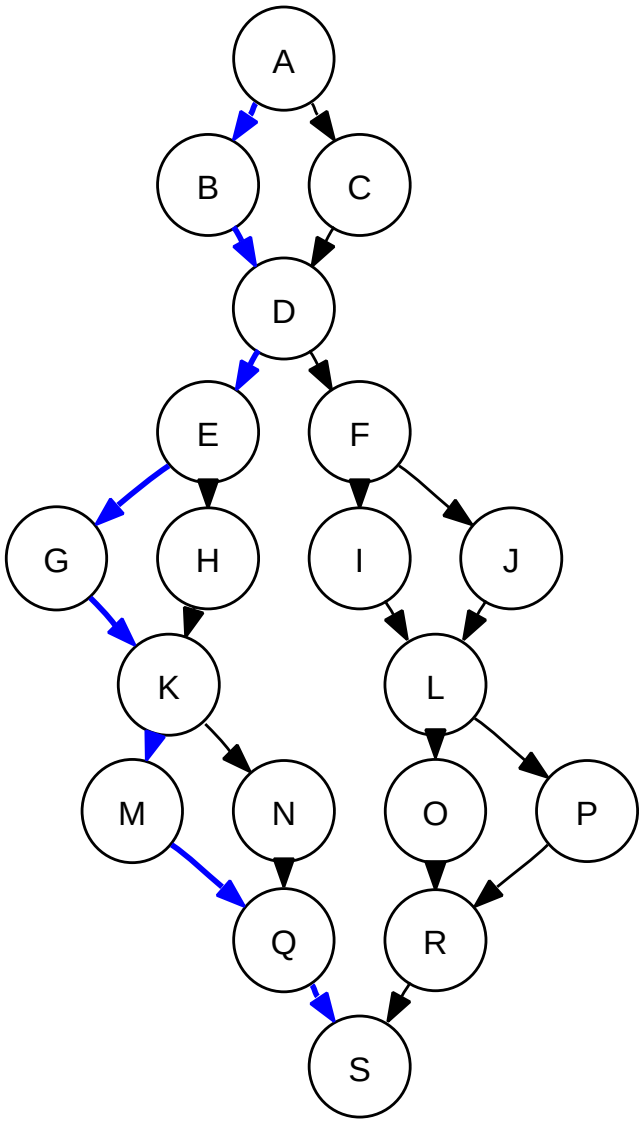
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.
- Caminho 3: ABDFILORS
- Alterar a saída do terceiro comando de decisão.

Escolha dos caminhos



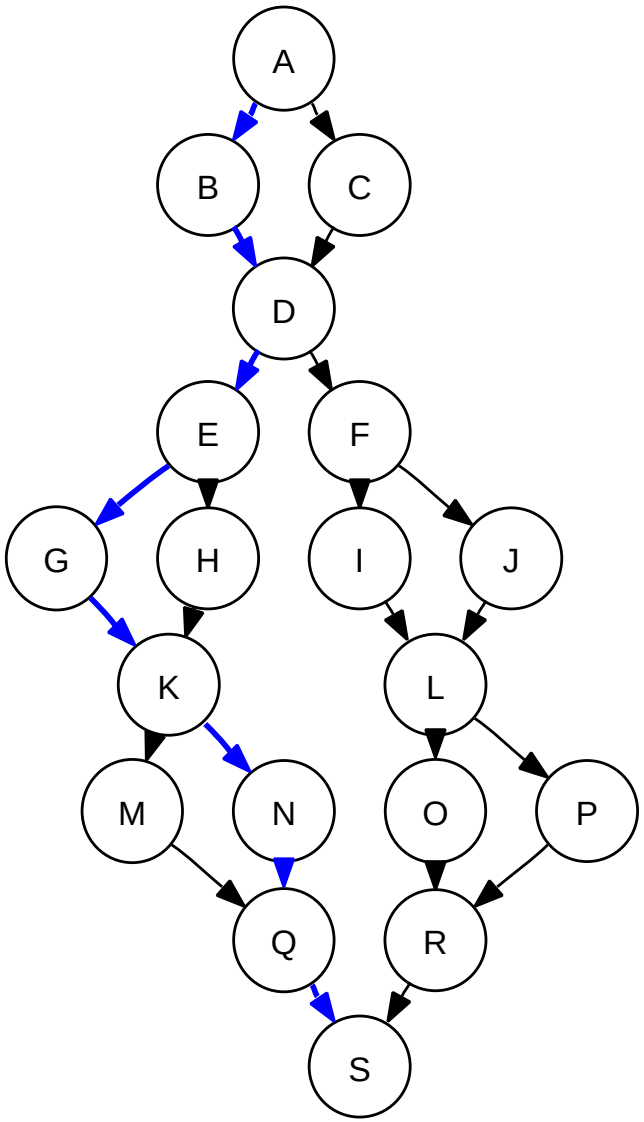
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.
- Caminho 3: ABDFILORS
- Alterar a saída do terceiro comando de decisão.
- Caminho 4: ABDEHKMQS

Escolha dos caminhos



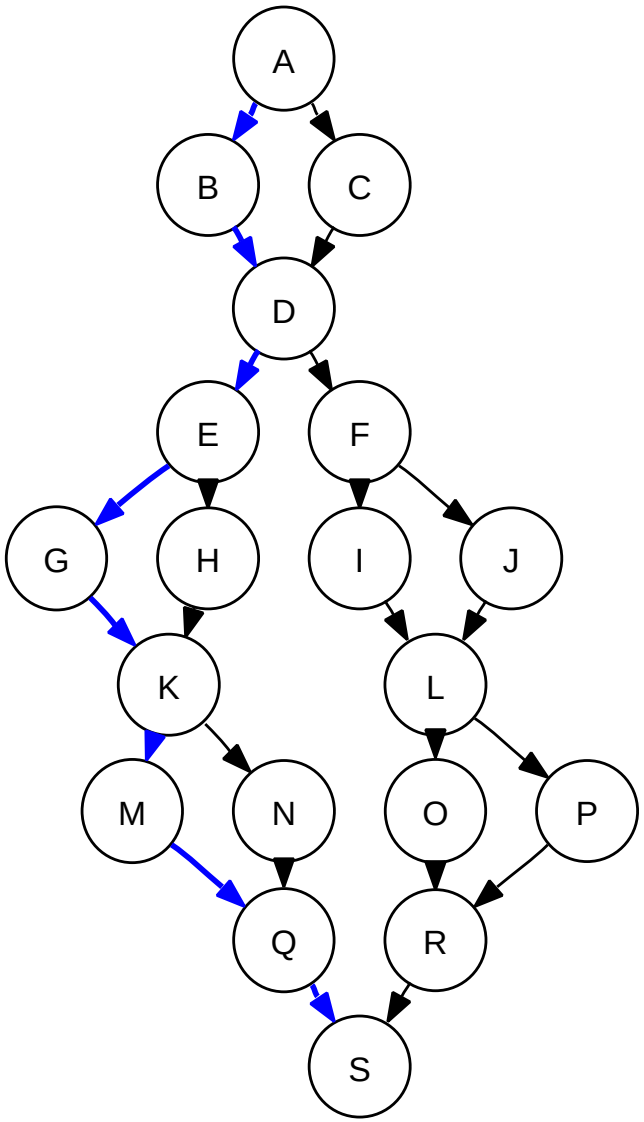
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.
- Caminho 3: ABDFILORS
- Alterar a saída do terceiro comando de decisão.
- Caminho 4: ABDEHKMQS
- Alterar a saída do quarto comando de decisão.

Escolha dos caminhos



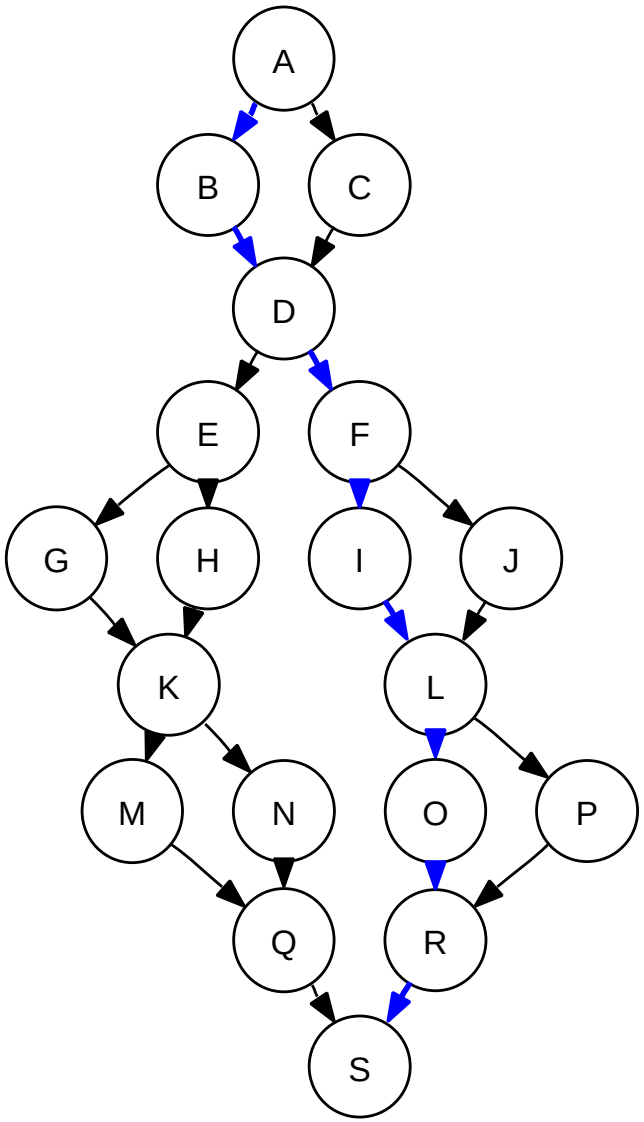
- Altere a saída só do primeiro comando de decisão
- Caminho 2: ACDEGKMQS
- Alterar a saída do segundo comando de decisão.
- Caminho 3: ABDFILORS
- Alterar a saída do terceiro comando de decisão.
- Caminho 4: ABDEHKMQS
- Alterar a saída do quarto comando de decisão.
- Caminho 5: ABDEGKNQS

Escolha dos caminhos



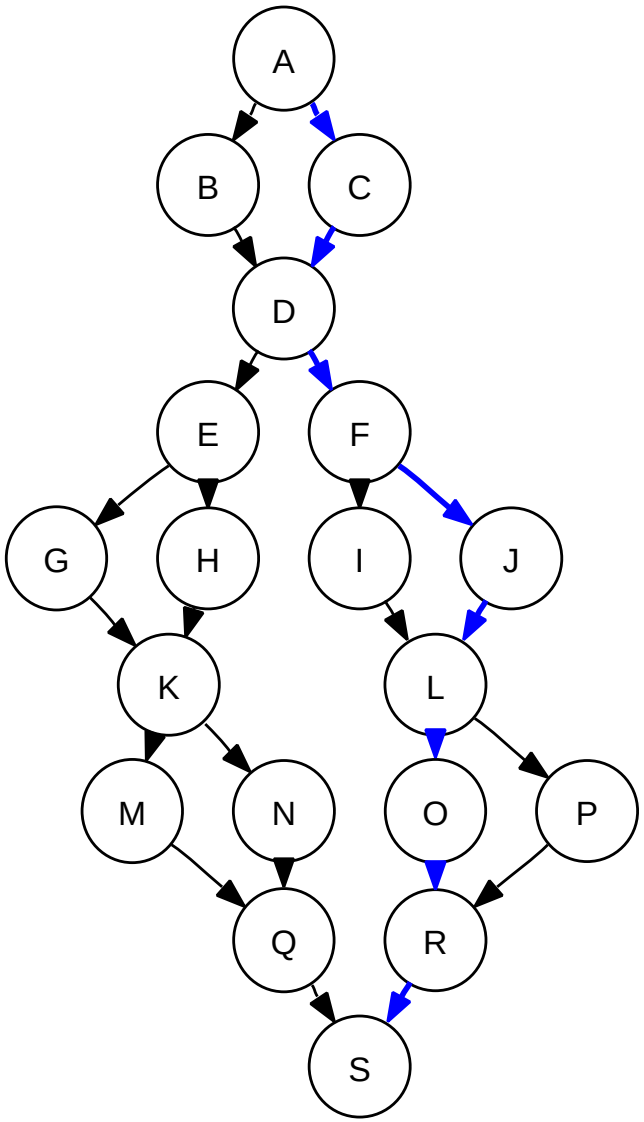
- Todos os comandos de decisão do caminho base foram modificados

Escolha dos caminhos



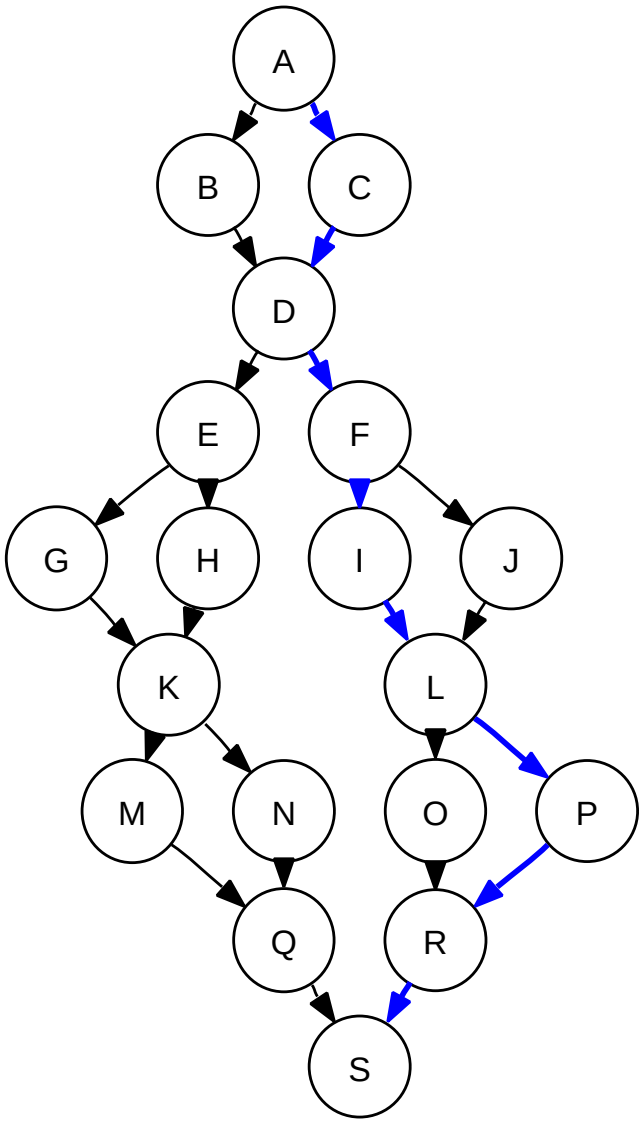
- Todos os comandos de decisão do caminho base foram modificados
- Fazer o mesmo para o caminho 3 (o 2 já foi tratado)

Escolha dos caminhos



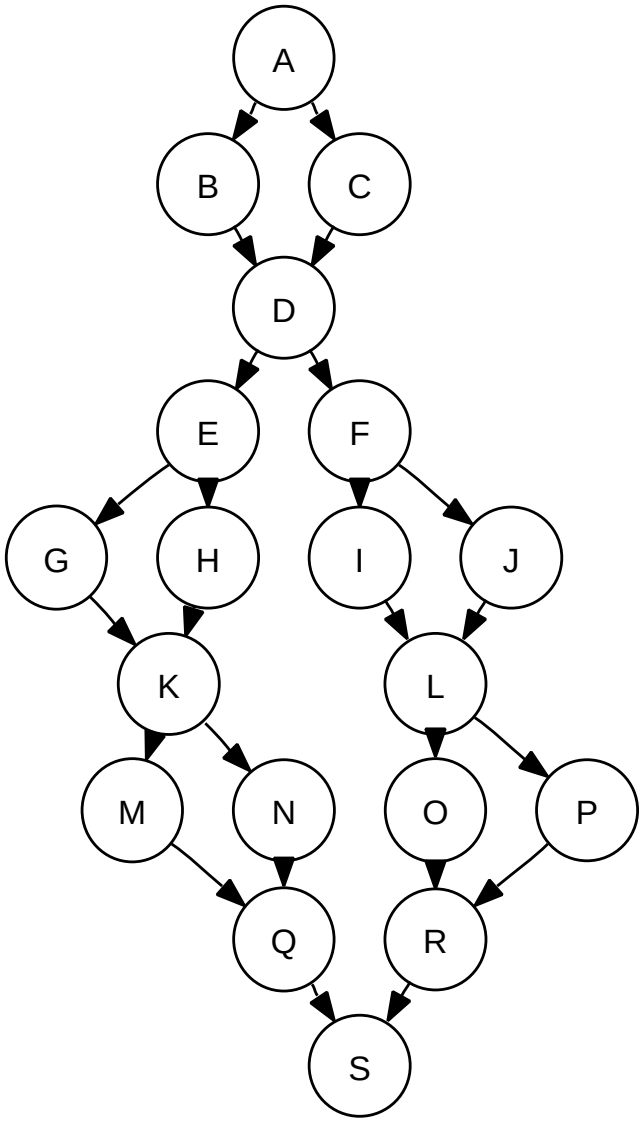
- Todos os comandos de decisão do caminho base foram modificados
- Fazer o mesmo para o caminho 3 (o 2 já foi tratado)
- Caminho 6: ACDFJLORS

Escolha dos caminhos



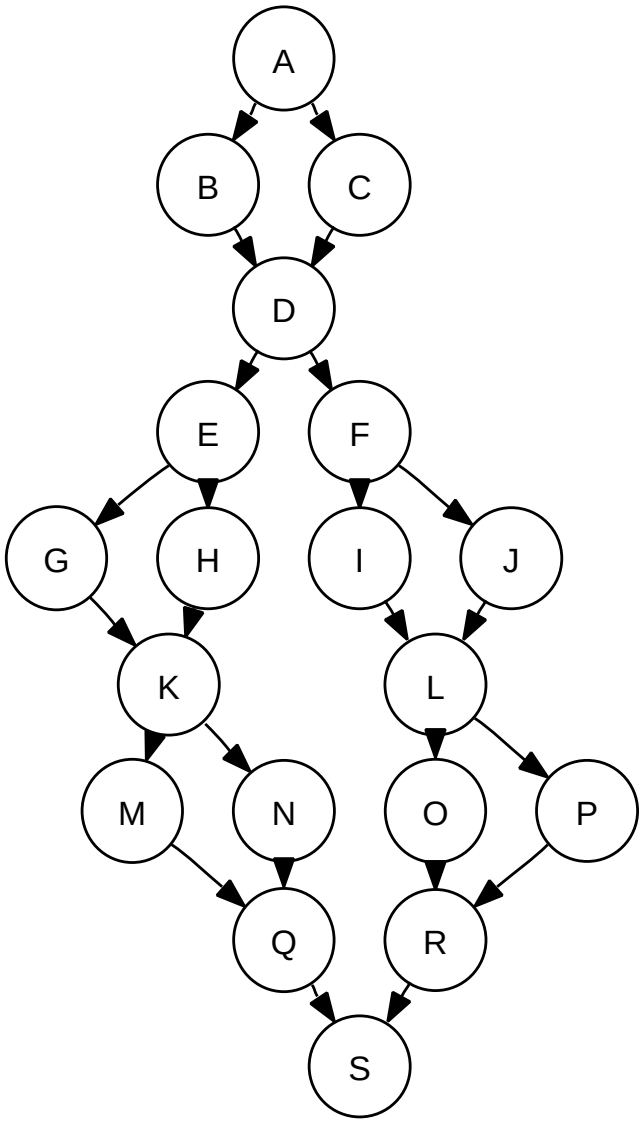
- Todos os comandos de decisão do caminho base foram modificados
- Fazer o mesmo para o caminho 3 (o 2 já foi tratado)
- Caminho 6: ACDFJLORS
- Caminho 7: ACDFILPRS

Requisitos de teste



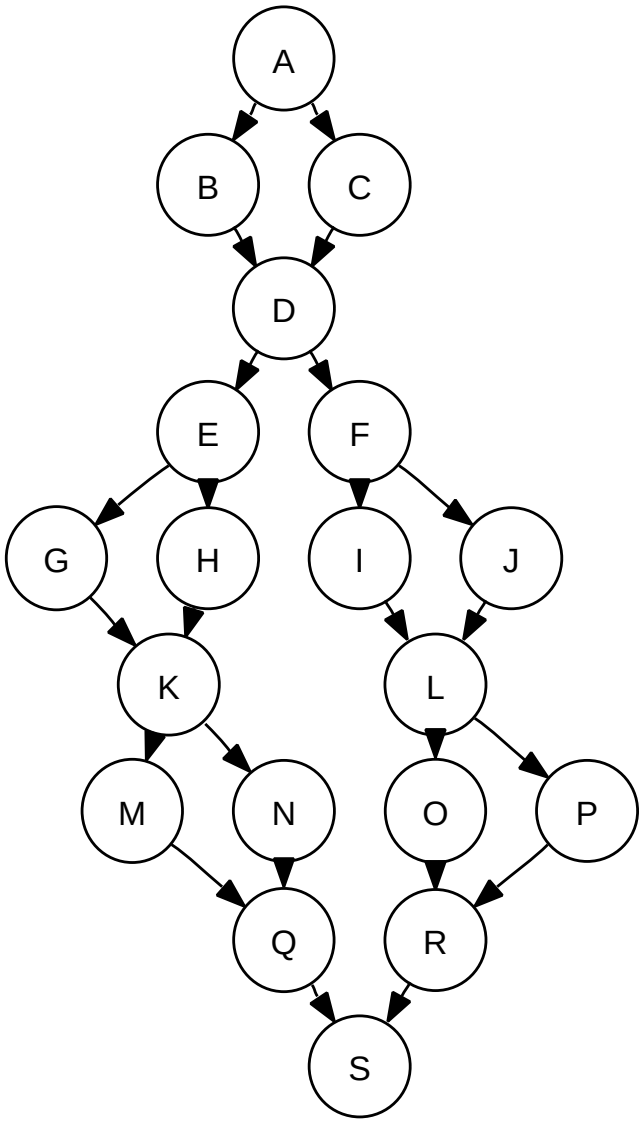
- Requisitos de testes derivado pelo critério.
  - ABDEGKMQS
  - ACDEGKMQS
  - ABDFILORS
  - ABDEHKMQS
  - ABDEGKNQS
  - ACDFJLORS
  - ACDFILPRS

Requisitos de teste



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

Requisitos de teste

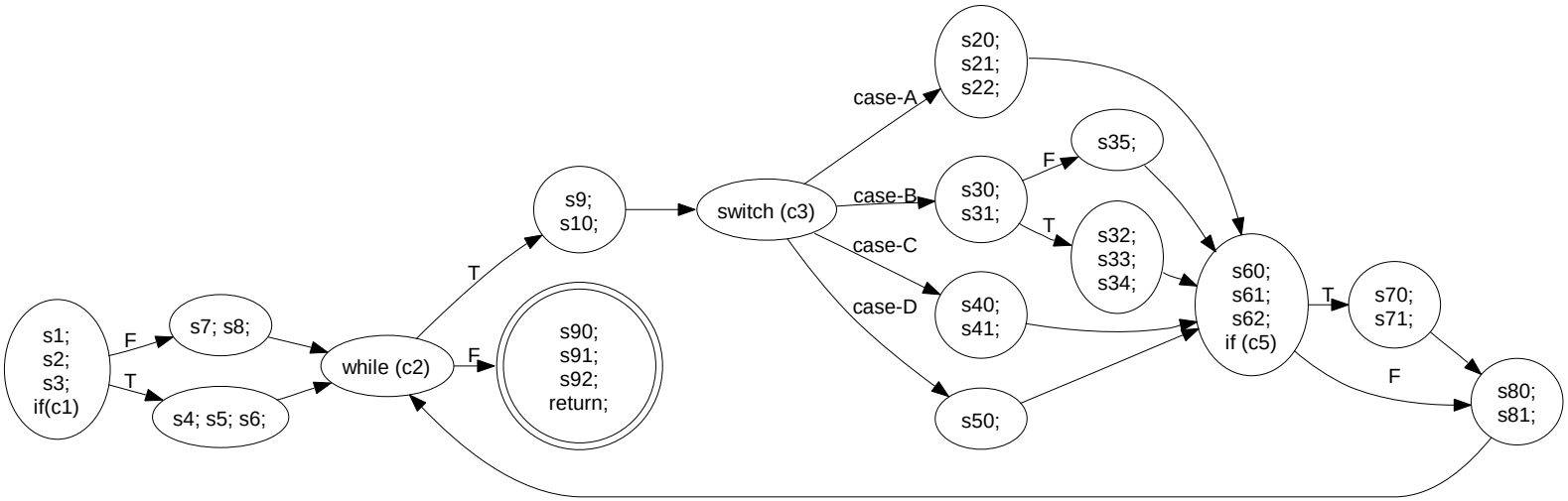


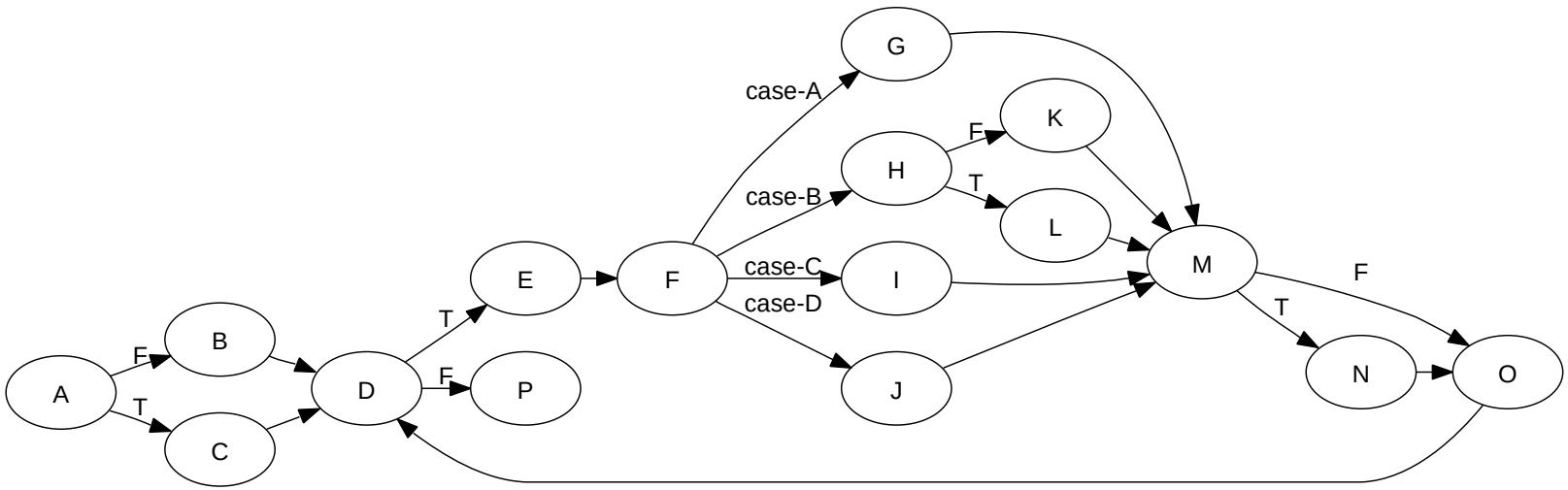
- Requisitos de testes derivado pelo critério.
  - ABDEGKMQS
  - ACDEGKMQS
  - ABDFILORS
  - ABDEHKMQS
  - ABDEGKNQS
  - ACDFJLORS
  - ACDFILPRS
- Conjunto criado não é único.
- Propriedade: o conjunto de teste que exercita os caminhos básicos também exercita todos-nós e todos-arcos do programa.

Exercício

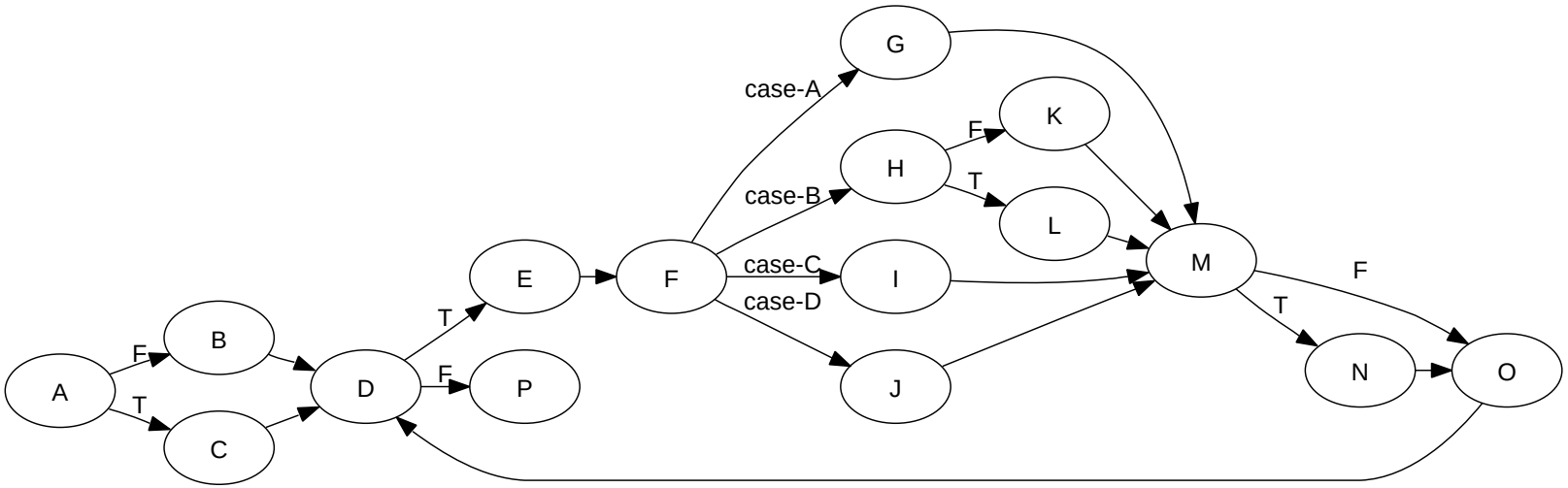
```
1  boolean evaluateBuySell (TickerSymbol ts) {
2  s1;
3  s2;
4  s3;
5  if (c1) {s4; s5; s6;}
6  else {s7; s8;}
7  while (c2) {
8     s9;
9     s10;
10    switch (c3) {
11       case_A:
12          s20;
13          s21;
14          s22;
15          break; // End of Case_A
16       case_B:
17          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



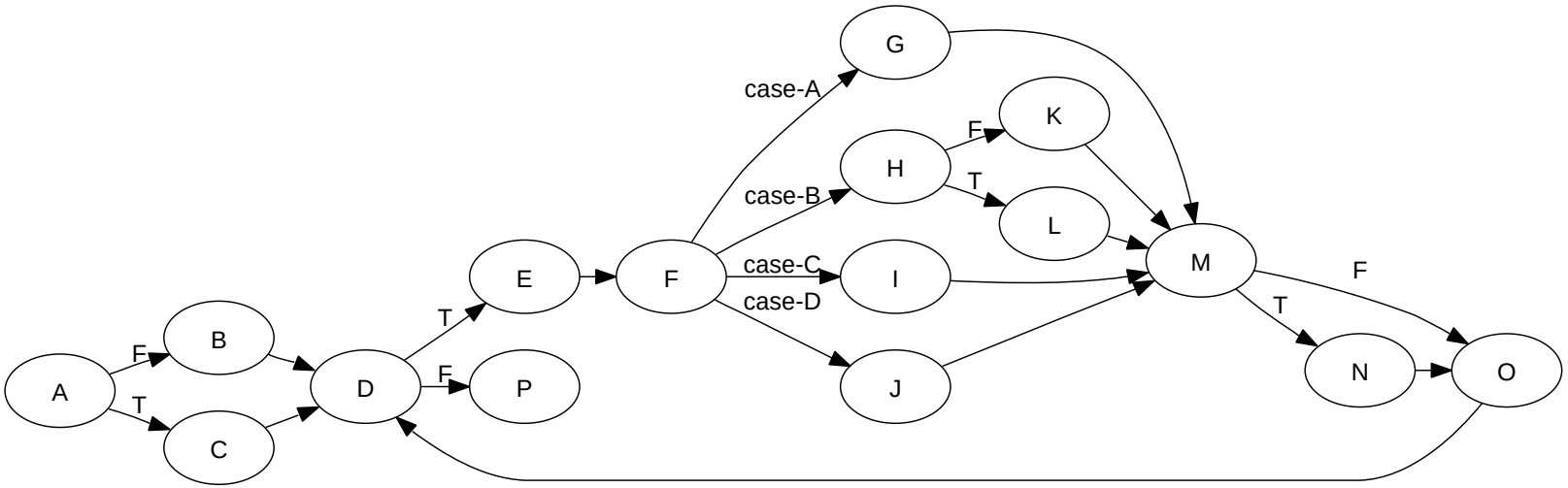
- Cálculo da complexidade ciclomática para o GFC acima:

  C = arcos - nós + 2

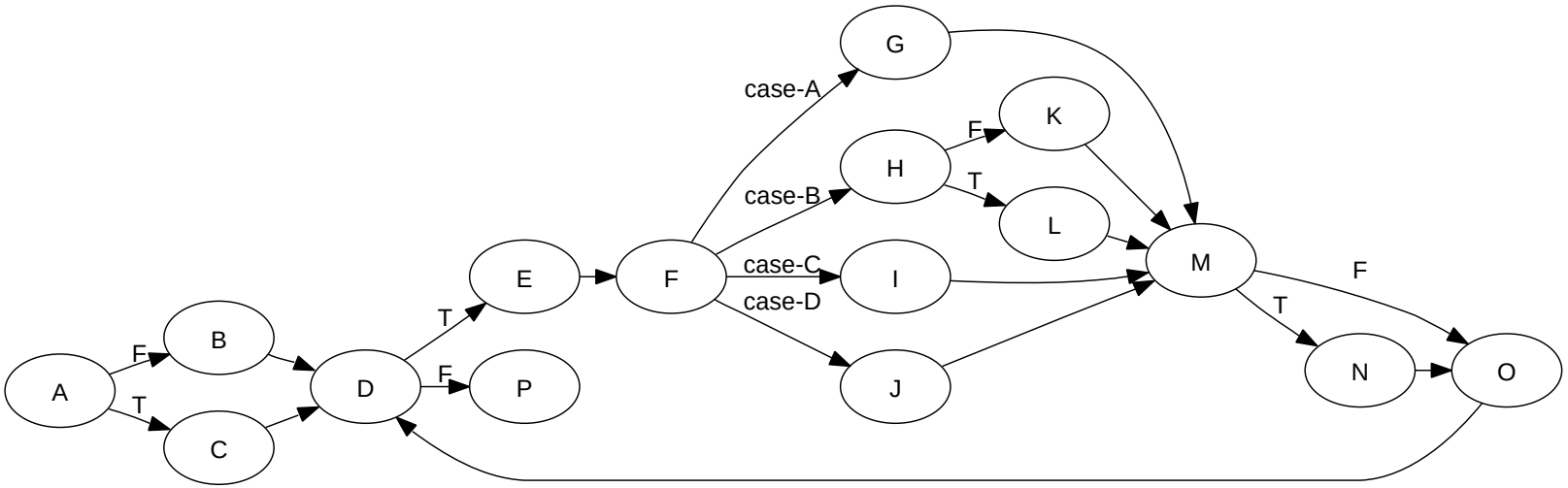
  C = 22-16+2

  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

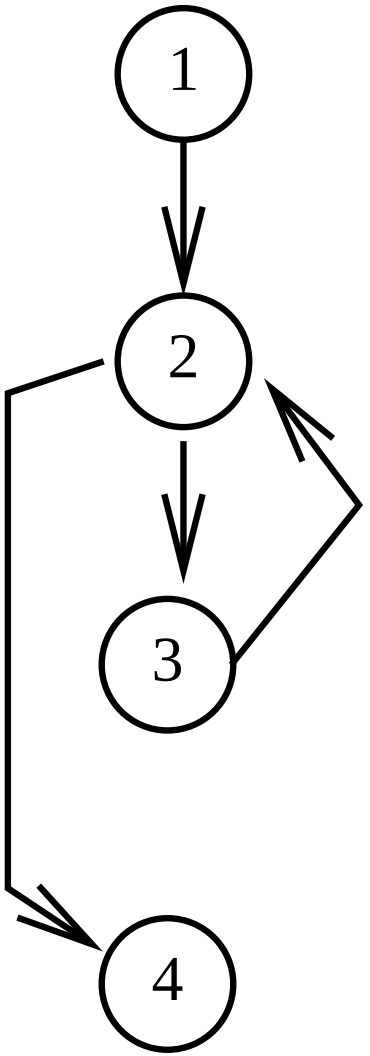
- Um dos problemas no teste estrutural, em geral, é a executabilidade
- Um caminho π é dito não executável quando não existe um dado de entrada que faça com que esse caminho seja executado
- Ao se determinarem os requisitos de teste é impossível determinar se são executáveis ou não
- Esse é um problema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

Executabilidade

```
for (i = 0; i < 10; i++)
{
   printf("\%d", i);
}
return;
```



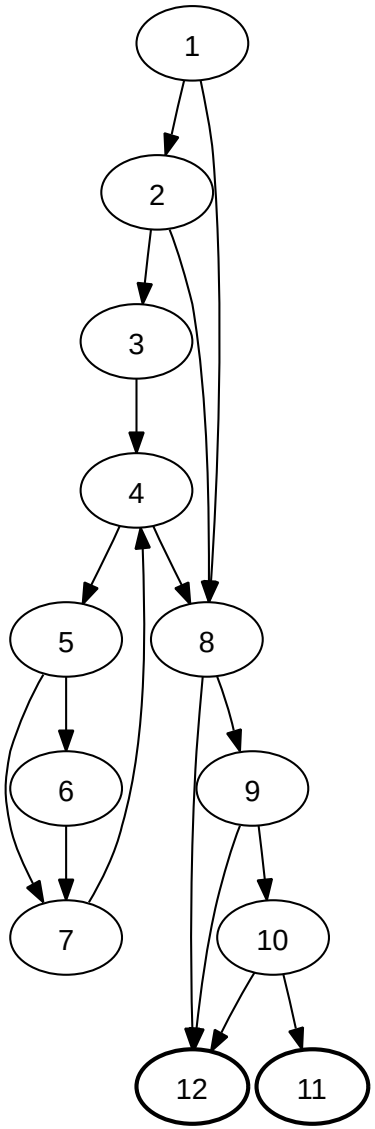
1 2 4

1 2 3 2 4

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

Exercício

Para os seguintes programas ache:

- Requisitos de teste para os critérios todos-nós e todas-arestas
- Requisitos não executáveis
- Conjunto de teste que seja todos-usos adequados mas que não seja todas-arestas adequado
- Conjunto de teste que seja todas-arestas adequado

Exercício – numZero

```
public static int numZero (int[] x) {
// Funcao: se x==null lanca NullPointerException
// senao retorna o número de ocorrências de 0 em x
   int count = 0;
   for (int i = 0; i < x.length; i++)
   {
      if (x[i] == 0)
      {
         count++;
      }
   }
   return count;
}
```

Exercício – oddOrPos

```
public static int oddOrPos(int[] x) {
// Funcao: se x==null lanca NullPointerException
// senao retorna o numero de elementos em x que
// sao impar ou positivo (ou ambos)
   int count = 0;
   for (int i = 0; i < x.length; i++) {
     if (x[i]%2 == 1 || x[i]%2 == -1 || x[i] > 0)
     {
         count++;
      }
   }
   return count;
}
```

Exercício – insertion sort

```
void insercao(int a[], int size) {
    int i, j, aux;
    for (i = 1; i < size; i++) {
        aux = a[i];
        j = i - 1;
        while (j >= 0 && a[j] >= aux) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = aux;
    }
}
```