Programação_Lógica_com_ProLog

:-

Integrante(‘Eduardo Reis’).
Integrante(‘Felipe de Morais’).
Integrante(‘Gabriel Fischer’).
Integrante(‘Rodrigo Smiderle’).

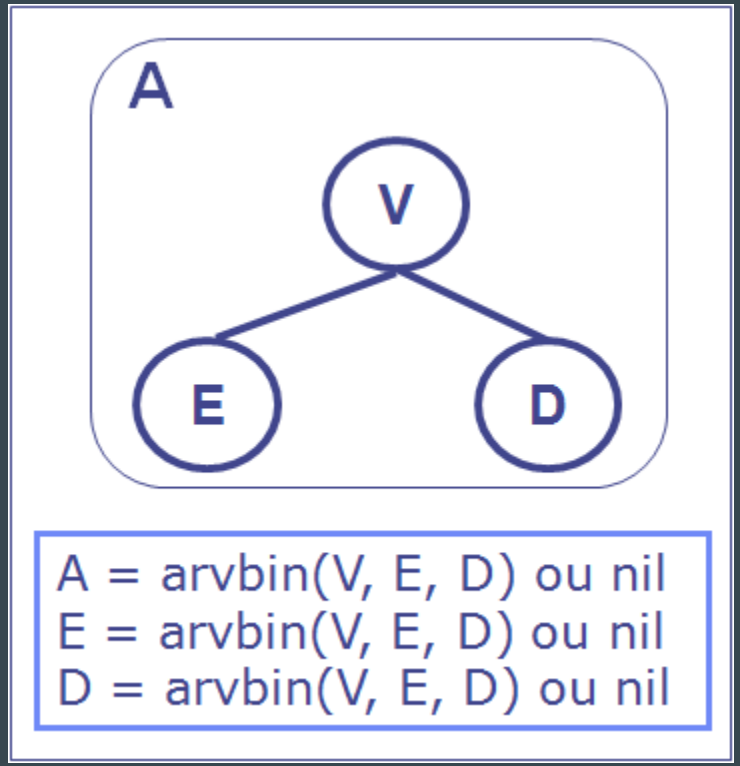
Árvores Binárias

Árvores Binárias

Verificar se é uma Árvore - isarvore(A).

arvbin(V, E. D) é uma árvore binária se E e D forem árvores binárias e V um valor.

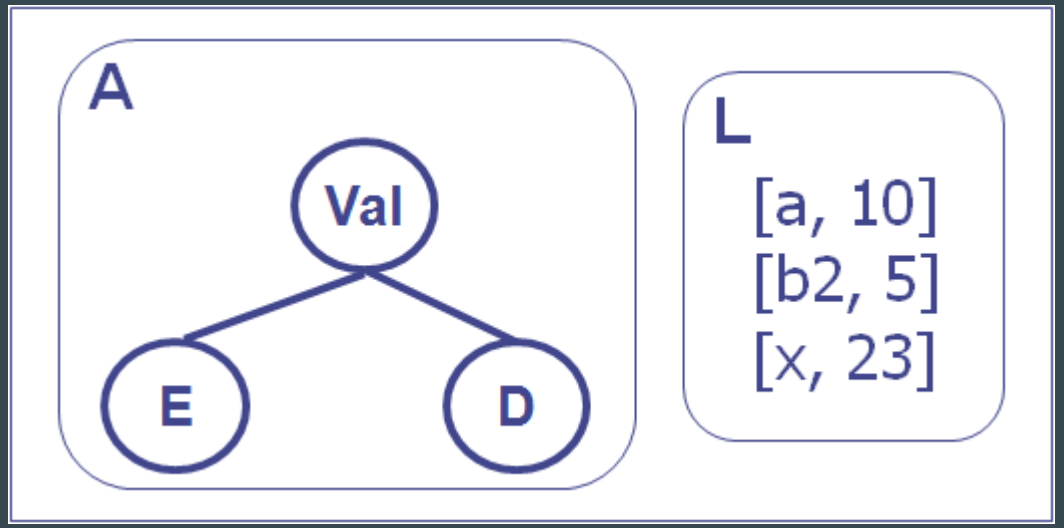
Um árvore binária pode ser ou uma estrutura arvbin(V, E, D) ou o valor terminal nil



Árvores Binárias

Calcular - calc(L, A, V).

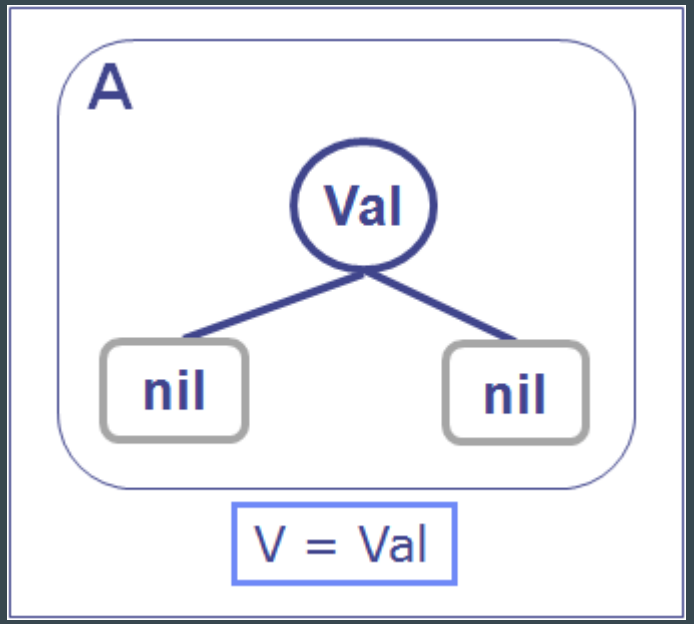
Verifica se A é uma árvore Binária e se L é uma lista de variáveis de memória



Árvores Binárias

Calcular - calc(L, A, V).

Se e E e D são nil, logo a saída V é Val.



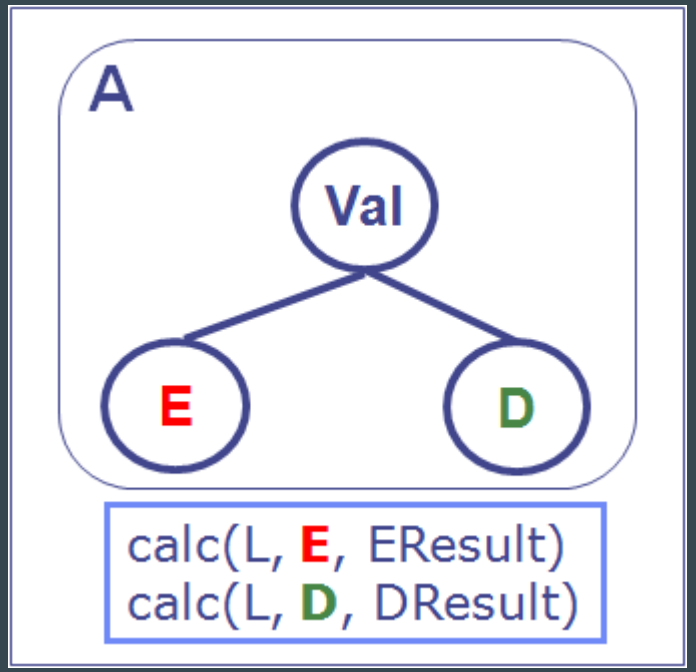
Árvores Binárias

Calcular - calc(L, A, V).

Senão executa:

calc(L, E, ER)

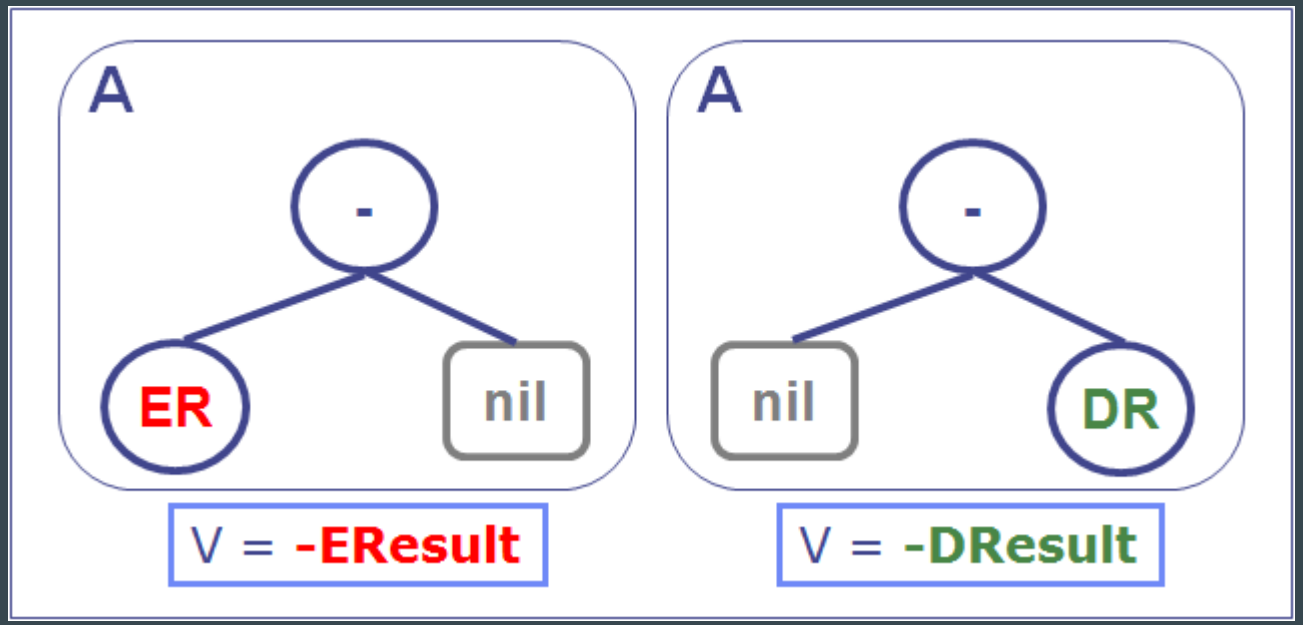
calc(L, D, DR)



Árvores Binárias

Calcular - calc(L, A, V).

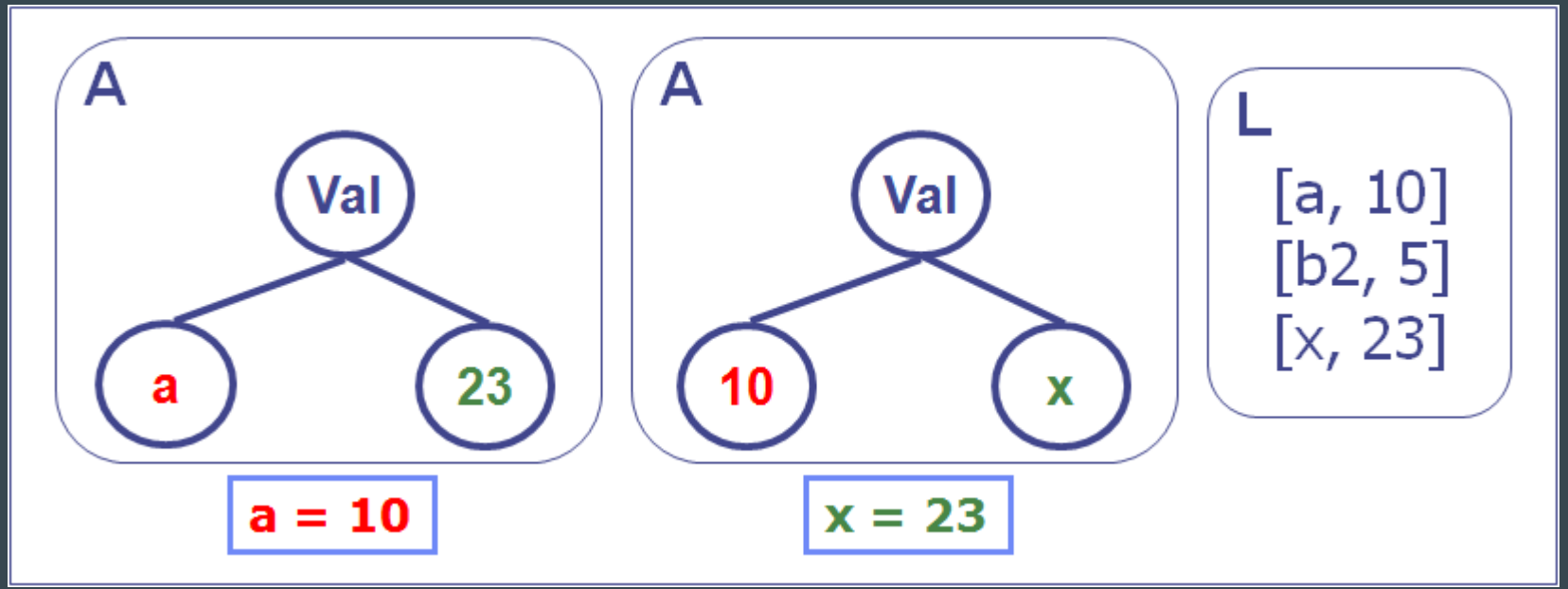
Depois verifica-se se V é '-' e se um dos EResult ou DResult é nil, caso seja:



Árvores Binárias

Calcular - calc(L, A, V).

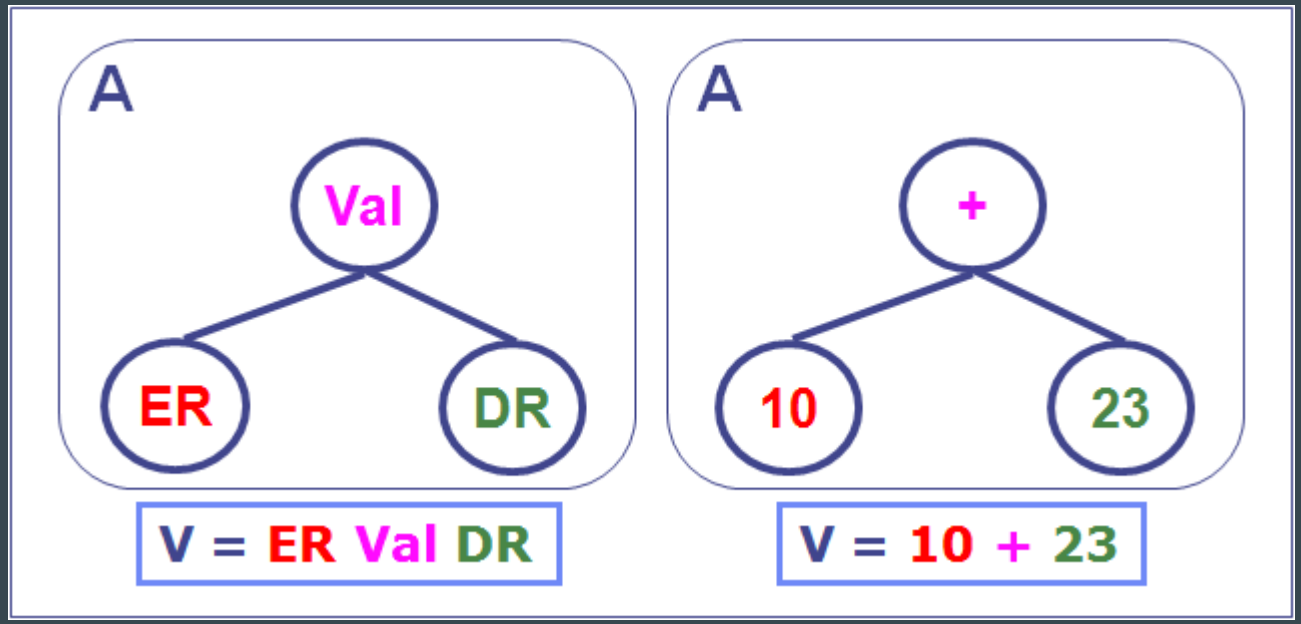
Senão verifica-se:



Árvores Binárias

Calcular - calc(L, A, V).

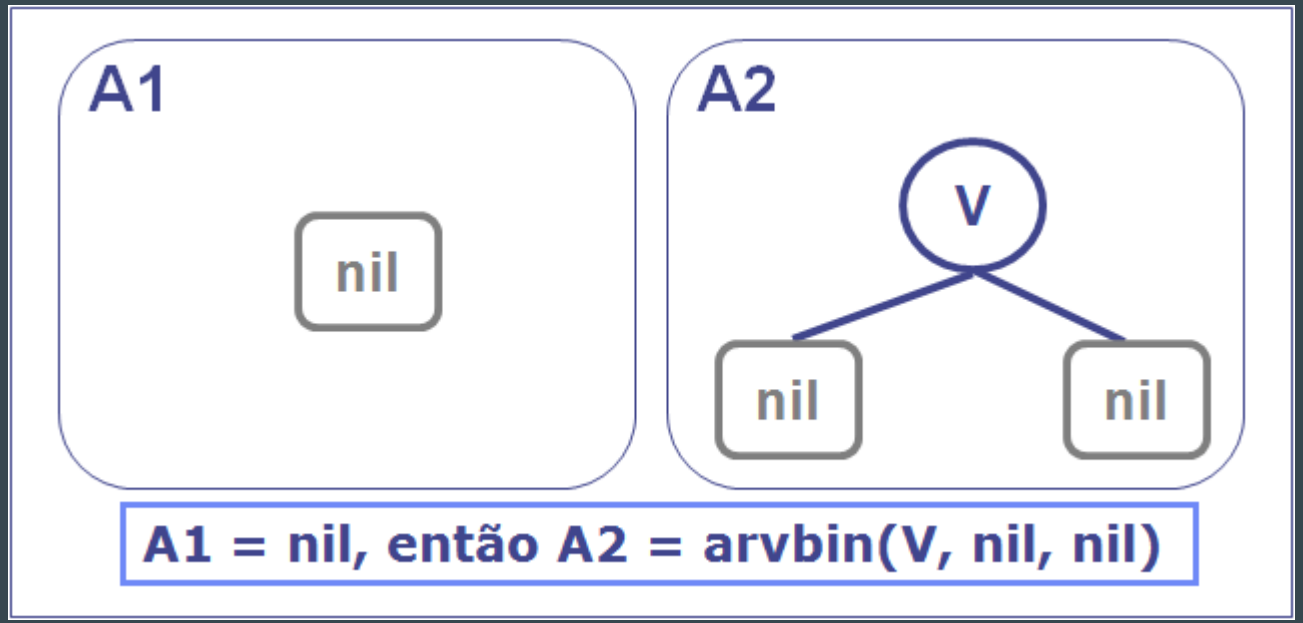
Por fim, faz o cálculo EResult Val DResult, onde Val é tratado como a operação a ser usada.



Árvores Binárias

Inserir Valor em uma árvore - insereabb(A1, V, A2).

Se A1 é nil, logo a árvore A2 é arvbin(V, nil, nil).

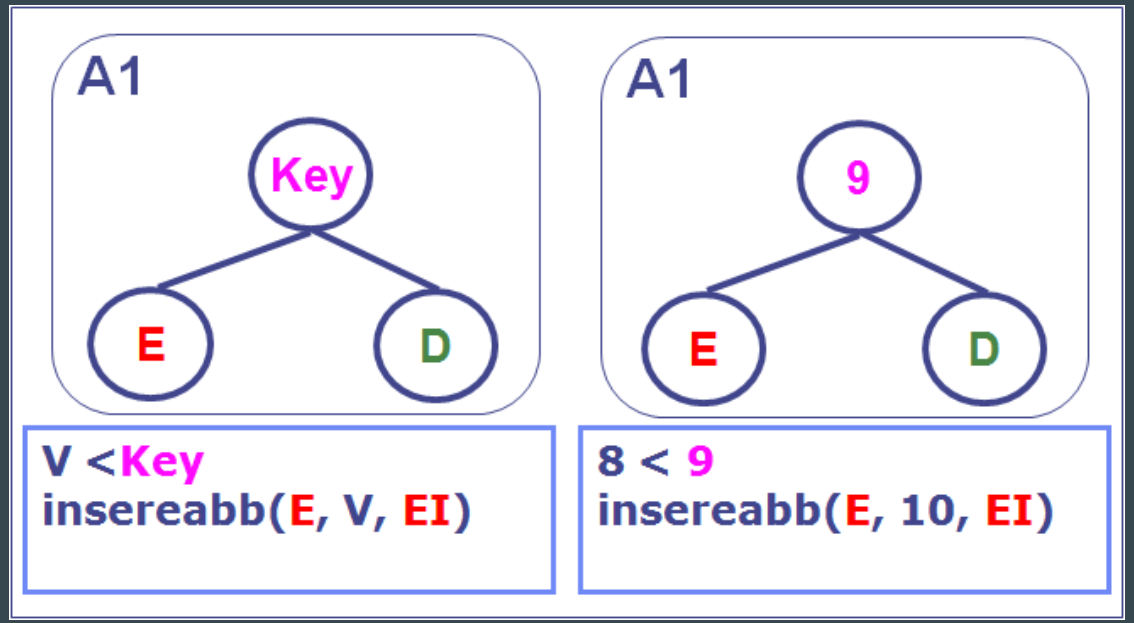


Árvores Binárias

Inserir Valor em uma árvore - insereabb(A1, V, A2).

Verifica-se se Key é maior ou menor que V

Se for menor executa o insereabb para E e a key

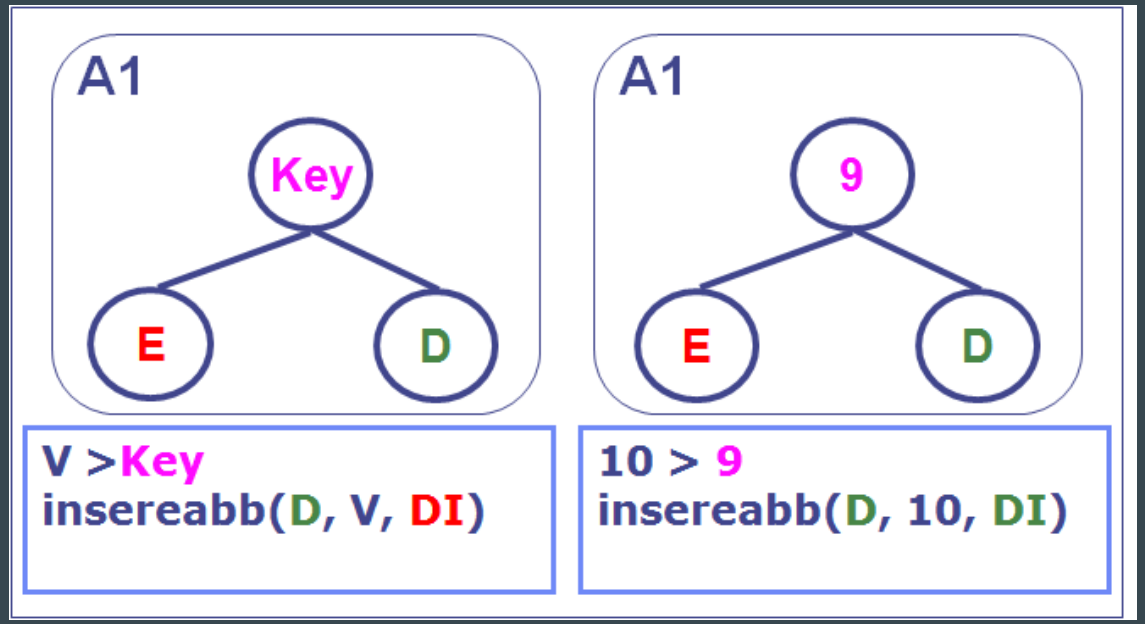


Árvores Binárias

Inserir Valor em uma árvore - insereabb(A1, V, A2).

Verifica-se se Key é maior ou menor que V

Se for maior executa o insereabb para D e a key



Árvores Binárias

Inserir Valor em uma árvore - insereabb(A1, V, A2).

Se for igual a Key, ignora-se a inserção, para não ter valores duplicados na árvore

Após é realizado o balanceamento da árvore.

Árvores Binárias

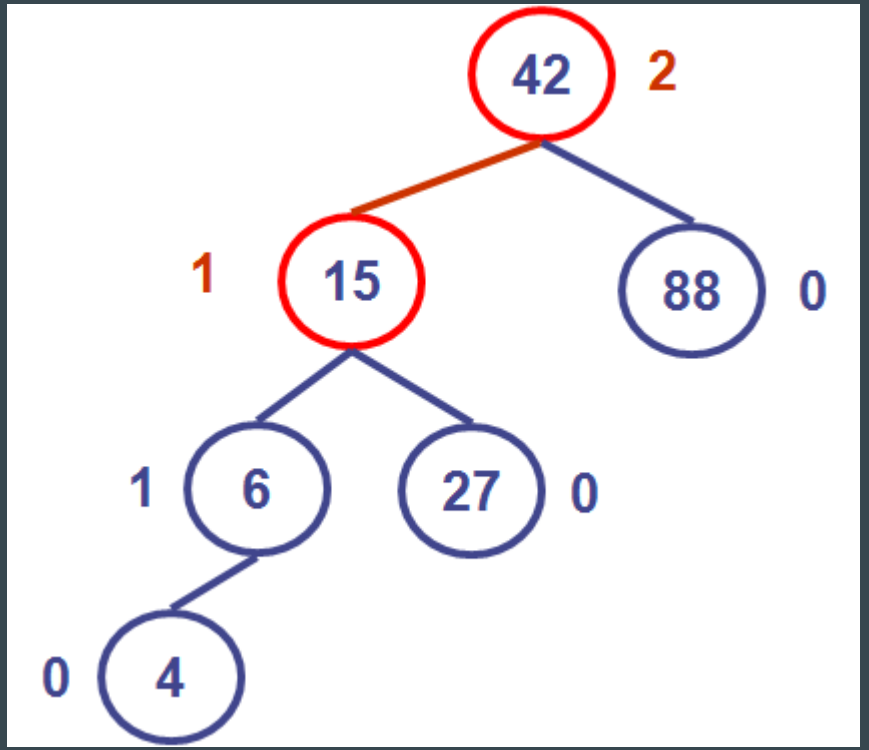
Balanceamento da Árvore

Verifica-se a altura da árvore

Se a altura for -2 ou 2 são realizadas as rotações, caso contrário é desnecessário.

Se for 2, Rotação à Direita

Se for -2, Rotação à Esquerda



Árvores Binárias

Rotação à Direita.

Verifica-se se a subárvore E tem altura 0 ou negativo

Se a altura for negativa, realiza-se uma rotação dupla à Direita

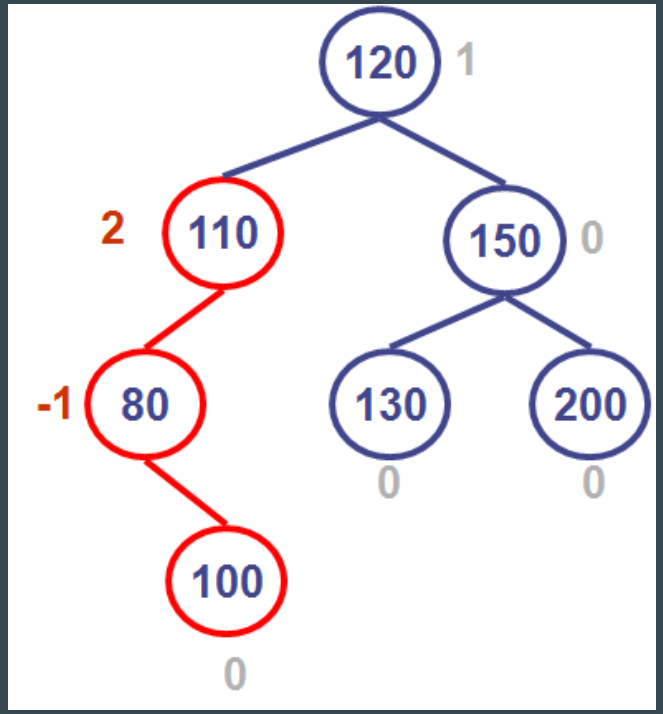
Caso contrário realiza-se uma rotação simples à Direita.

Rotação à Esquerda

Verifica-se se a subárvore D tem altura 0 ou positiva.

Se a altura for positiva, realiza-se uma rotação dupla à Esquerda

Caso contrário realiza-se uma rotação simples à Esquerda



Árvores Binárias

Rotação à Direita.

Verifica-se se a subárvore E tem altura 0 ou negativo

Se a altura for negativa, realiza-se uma rotação dupla à Direita

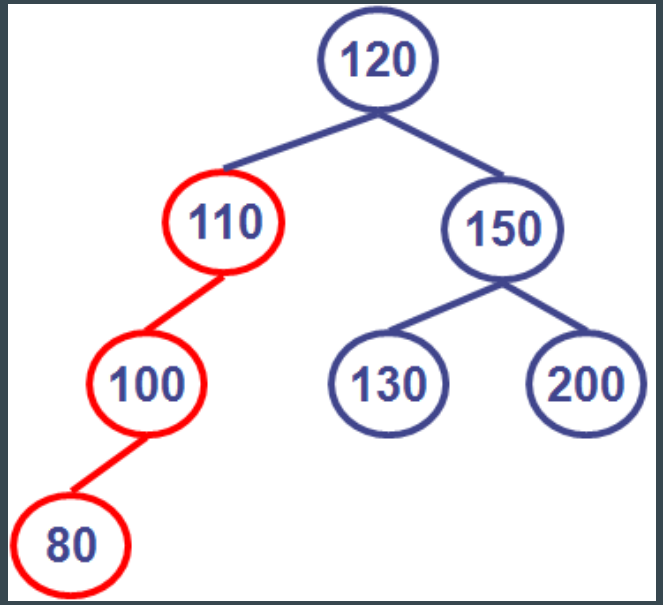
Caso contrário realiza-se uma rotação simples à Direita.

Rotação à Esquerda

Verifica-se se a subárvore D tem altura 0 ou positiva.

Se a altura for positiva, realiza-se uma rotação dupla à Esquerda

Caso contrário realiza-se uma rotação simples à Esquerda



Árvores Binárias

Rotação à Direita.

Verifica-se se a subárvore E tem altura 0 ou negativo

Se a altura for negativa, realiza-se uma rotação dupla à Direita

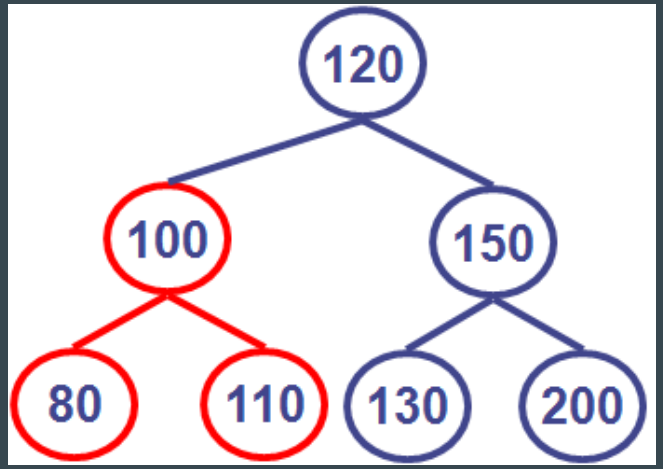
Caso contrário realiza-se uma rotação simples à Direita.

Rotação à Esquerda

Verifica-se se a subárvore D tem altura 0 ou positiva.

Se a altura for positiva, realiza-se uma rotação dupla à Esquerda

Caso contrário realiza-se uma rotação simples à Esquerda



Árvores Binárias

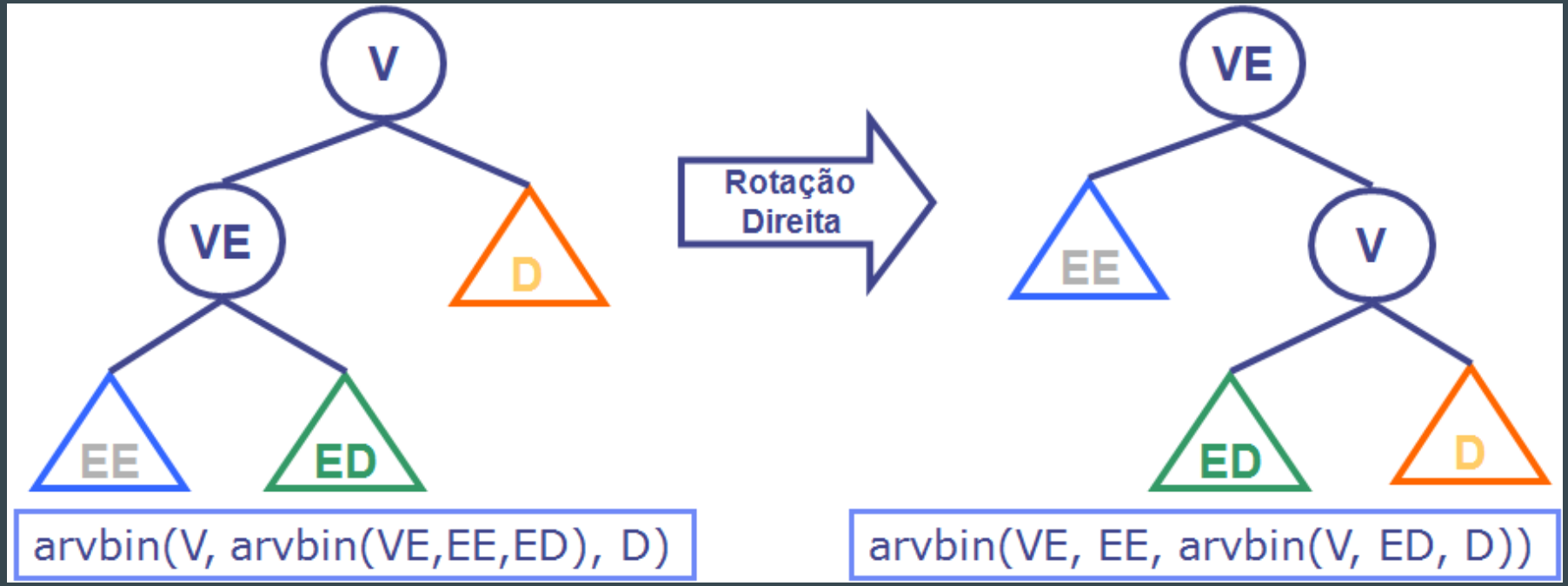
Dupla Rotação

Na dupla rotação à Direita, executa-se primeiro uma rotação simples à Esquerda e sobre o resultado uma rotação simples à Direita

Na dupla rotação à Esquerda, executa-se primeiro uma rotação simples à Direita e sobre o resultado uma rotação simples à Esquerda

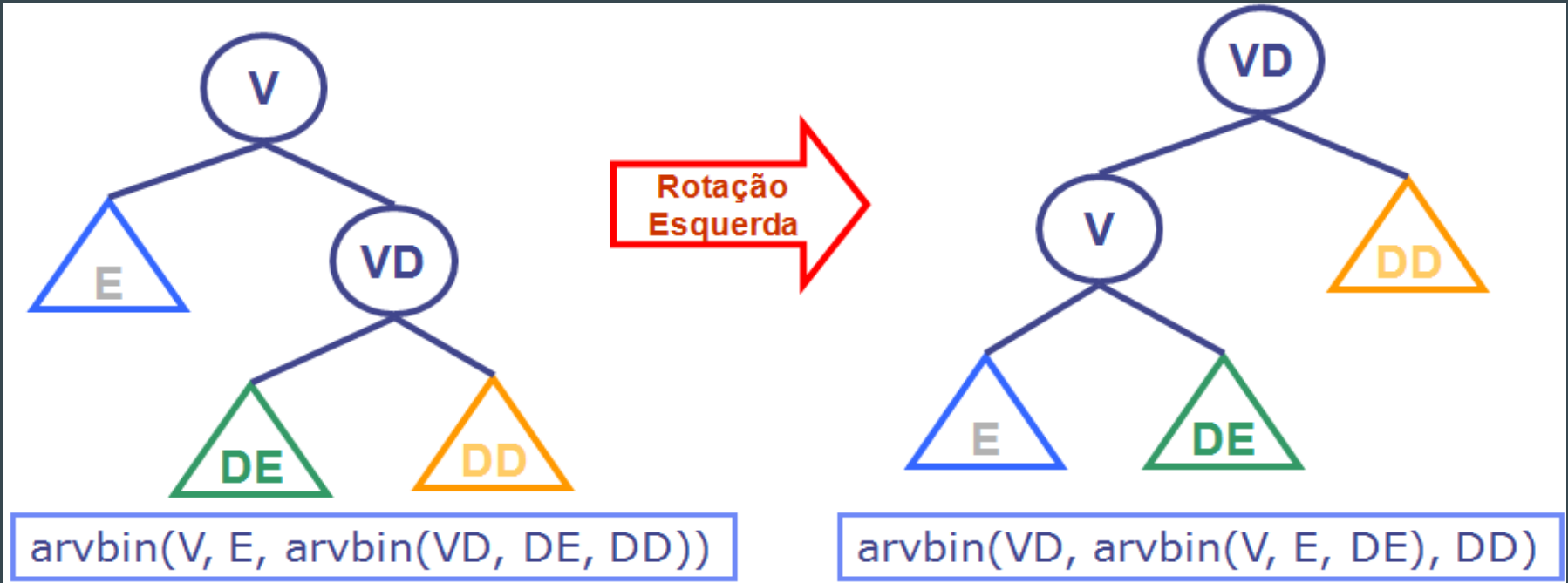
Árvores Binárias

Rotação Simples à Direita



Árvores Binárias

Rotação Simples à Esquerda



Árvores Genéricas

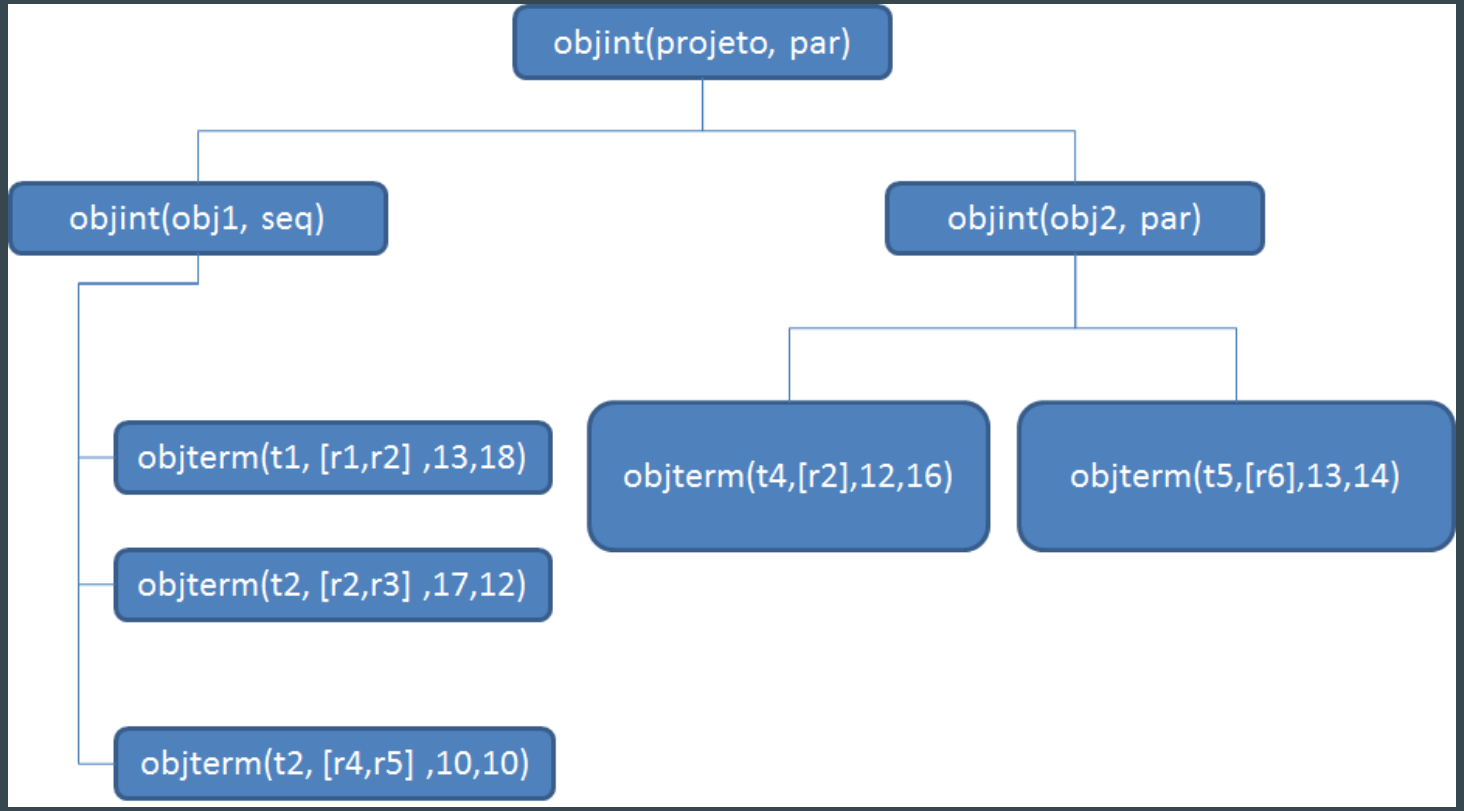
Árvores Genéricas

- São árvores em que os nodos podem possuir um número indeterminado de filhos

Aplicações

- Estrutura de dados para Jogos
- Armazenamento de Arquivos
- Armazenamento de dados hierárquicos
- Podem representar uma Estrutura Analítica de Projeto (EAP)

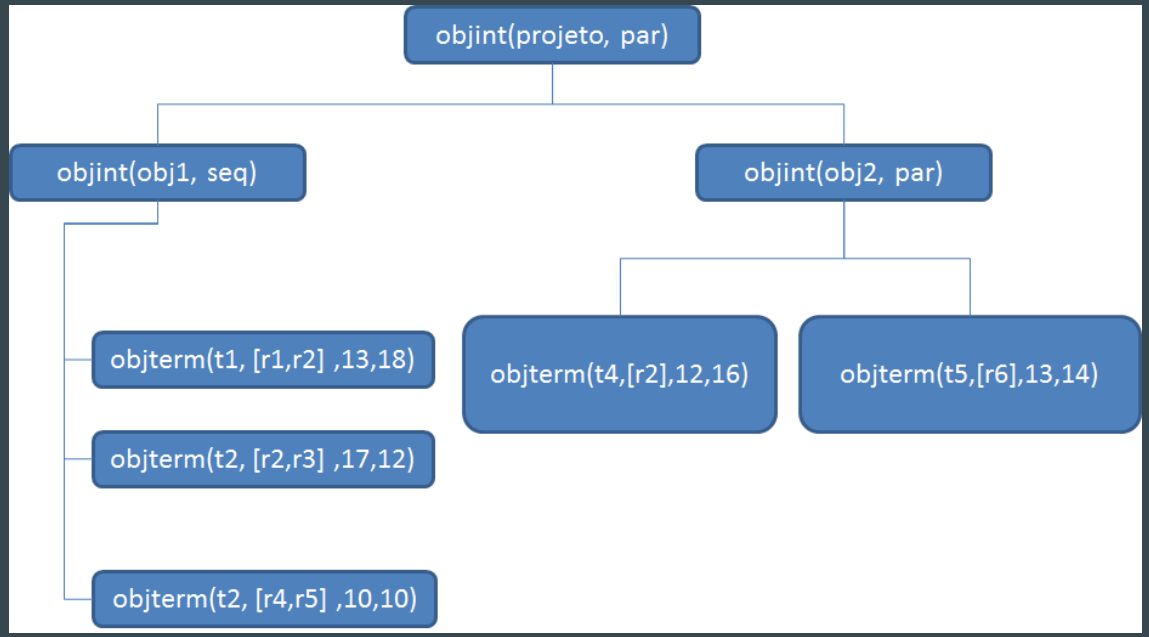
Exemplo: EAP


objint(projeto, par)
objint(obj1, seq)
objint(obj2, par)
objterm(t1, [r1,r2] ,13,18)
objterm(t2, [r2,r3] ,17,12)
objterm(t2, [r4,r5] ,10,10)
objterm(t4,[r2],12,16)
objterm(t5,[r6],13,14)

EAP

- total_recursos
- recursos_obj
- custo_total
- custo_obj
- prazo_total
- prazo_obj

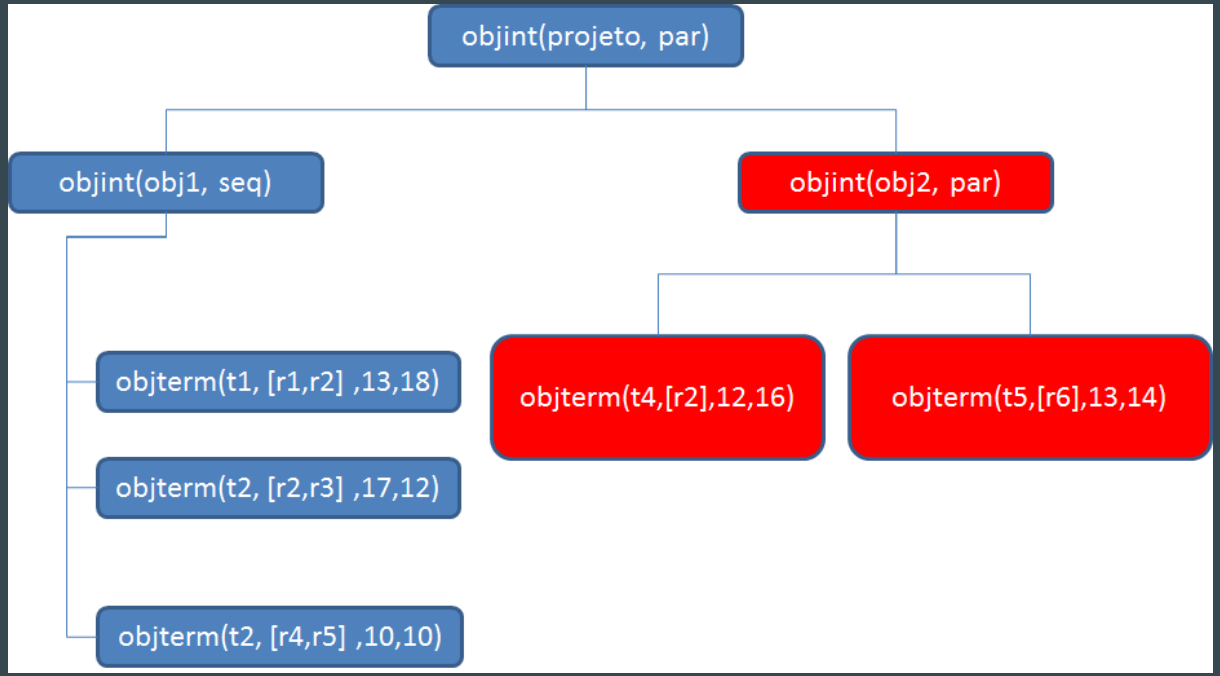
Exemplo: Recursos



- total_recursos(P, R)
- Verifica se é árvore
- Recursos do nodo
- Se nodo for terminal
- Adiciona recursos único a lista
- E adicionada nodos term

[r1,r2,r3,r4,r5,r6]

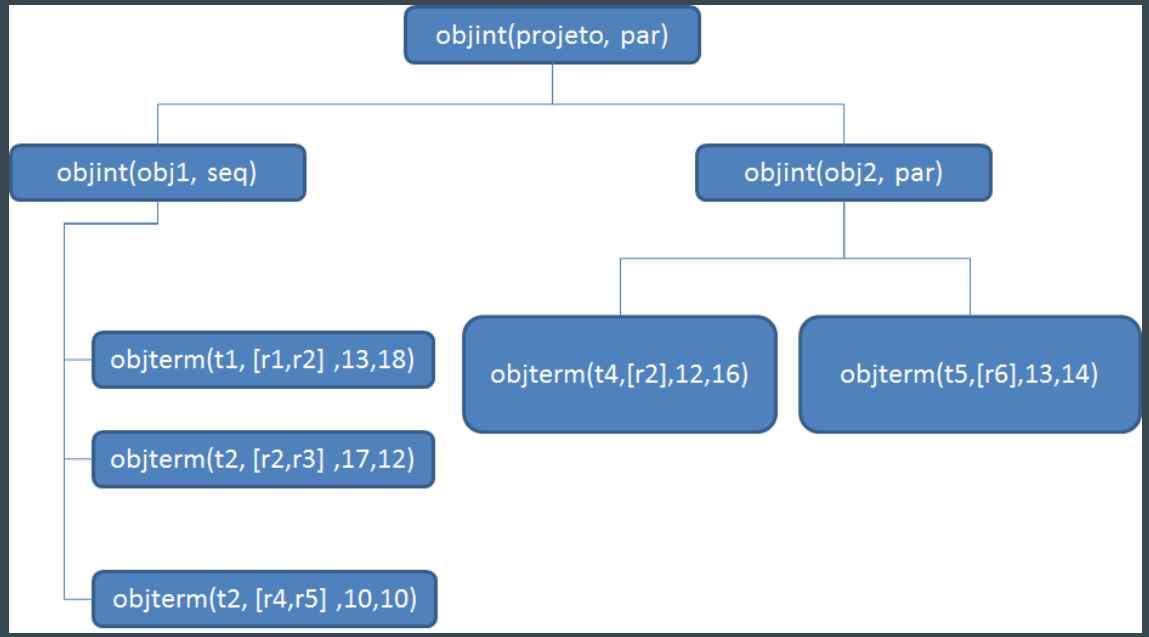
Exemplo: Recurso por objetivo



- recursos_obj
- Verifica se é árvore
- Encontra objeto
- Objeto Terminal
- Objeto Intermediário
- Expande lista nodos intermediários
- Encontra Obj
- Chama total_recursos

[r2,r3]

Exemplo: Custo Total

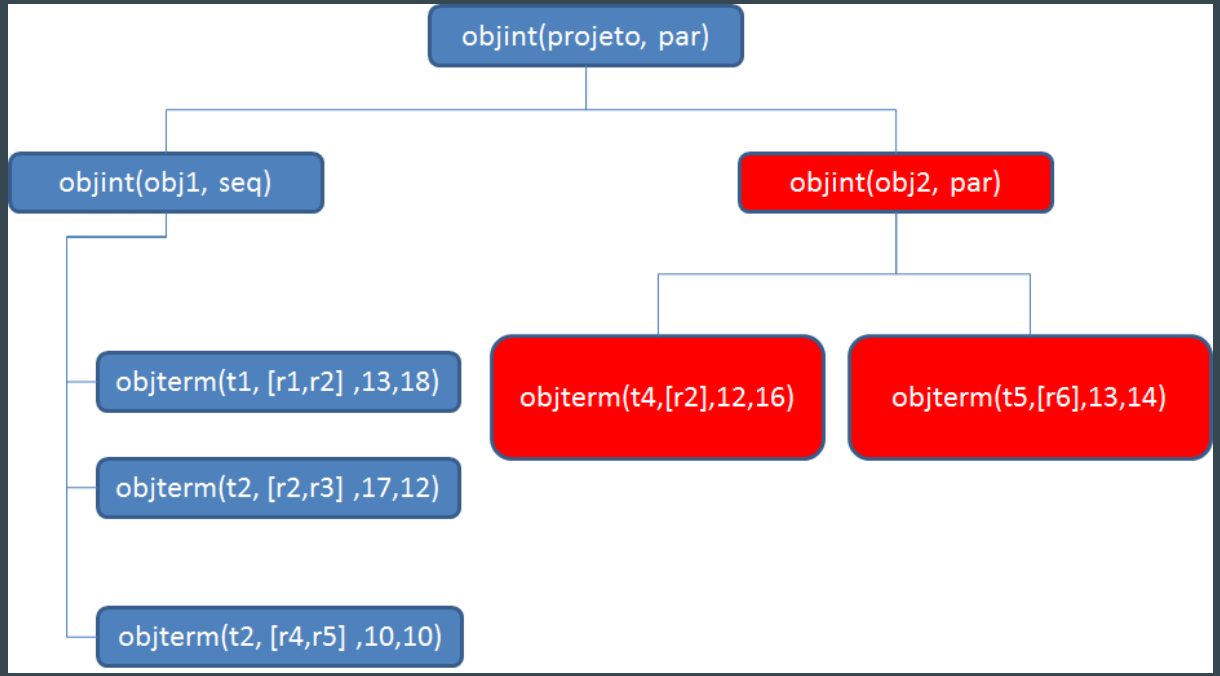


- custo_total(P, C)
- Verifica se é árvore
- Nodo é terminal adiciona custo
- Expande lista de nodos filhos

obj1 = 40.
obj2 = 25.

projeto = 65.

Exemplo: Custo por objetivo

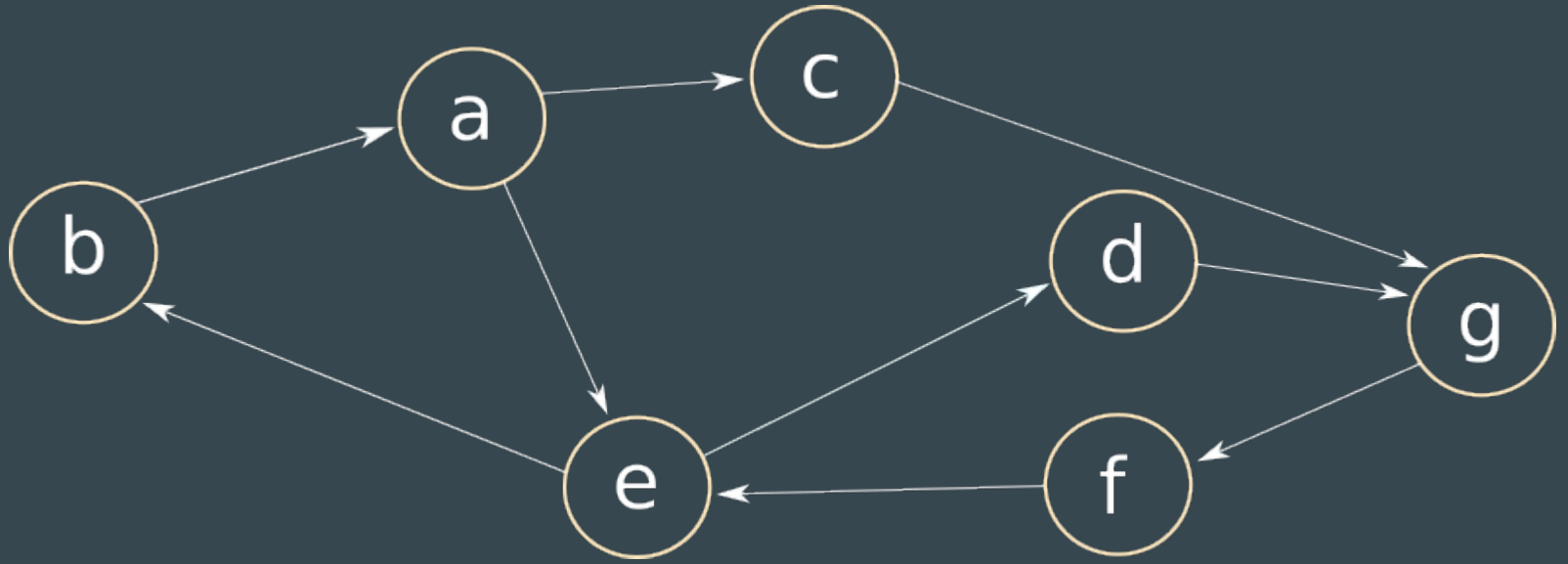


- custo_obj(P,O,R)
- Verifica se é árvore
- Encontra objetivo
- Objeto Terminal
- Objeto Intermediário
- Expande lista nodos intermediários
- Encontra Obj
- Chama custo_total

obj = 25.

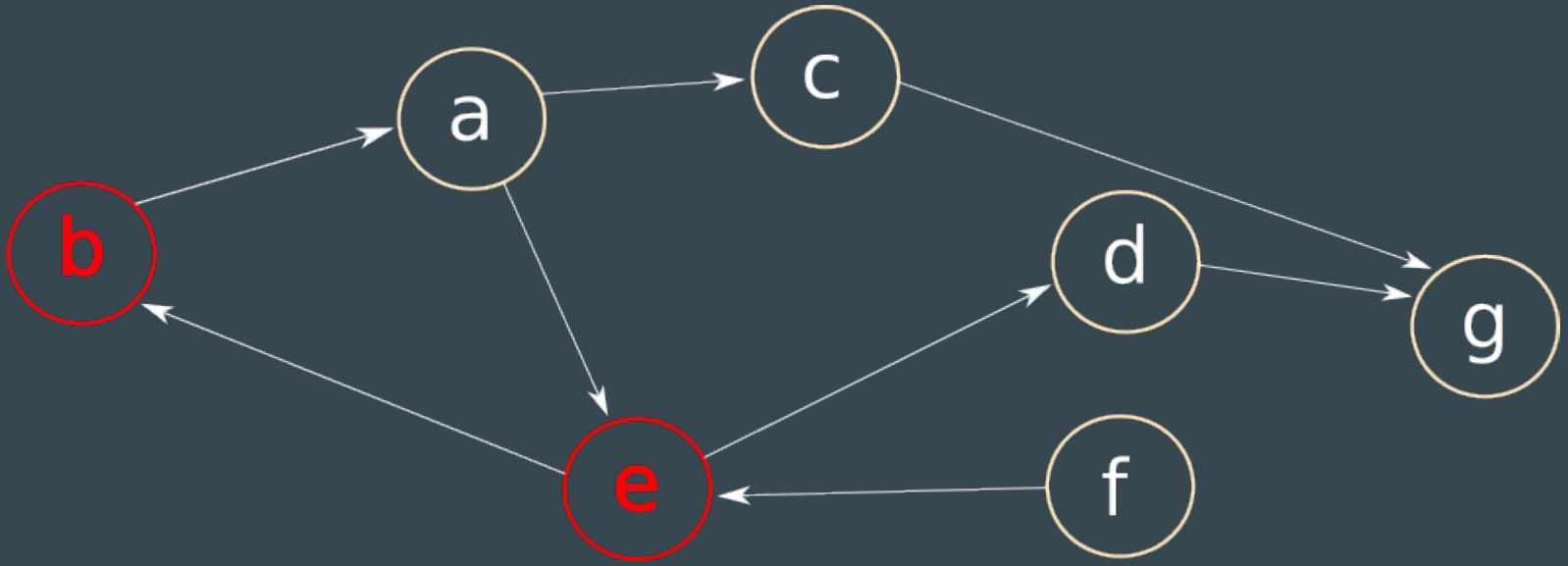
Grafos

Exemplo



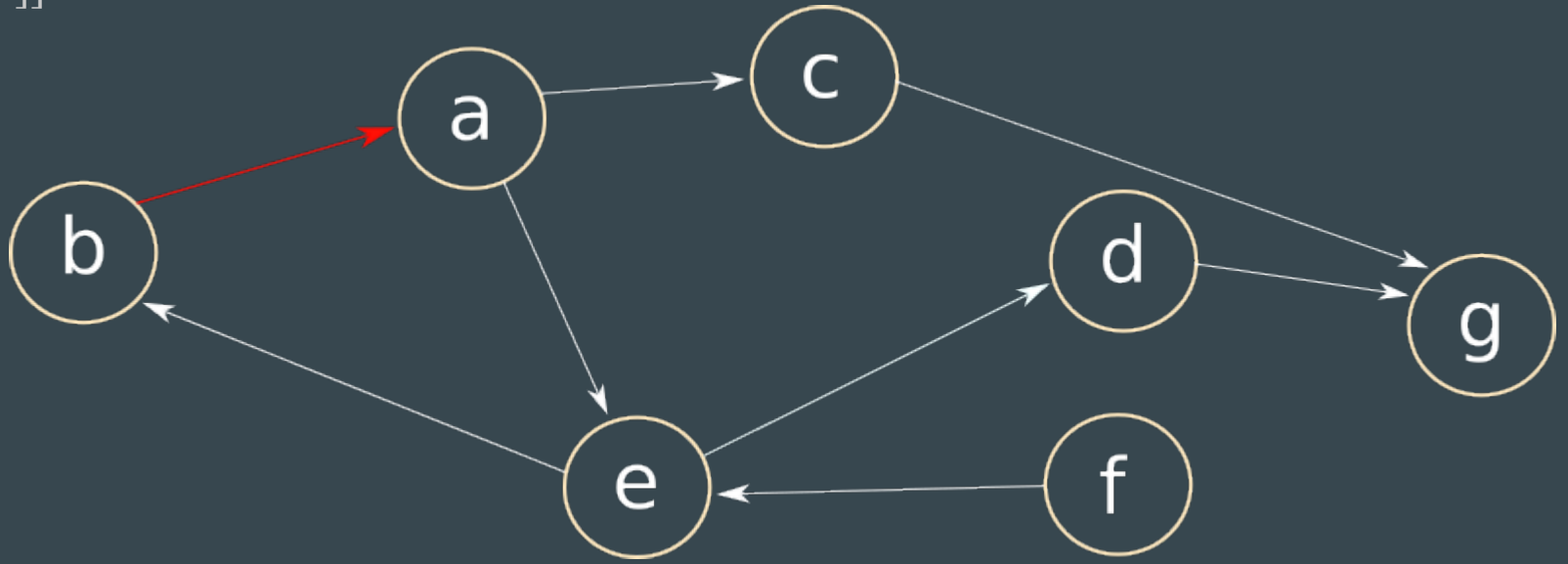
conectado(G, N1, N2):-

[]



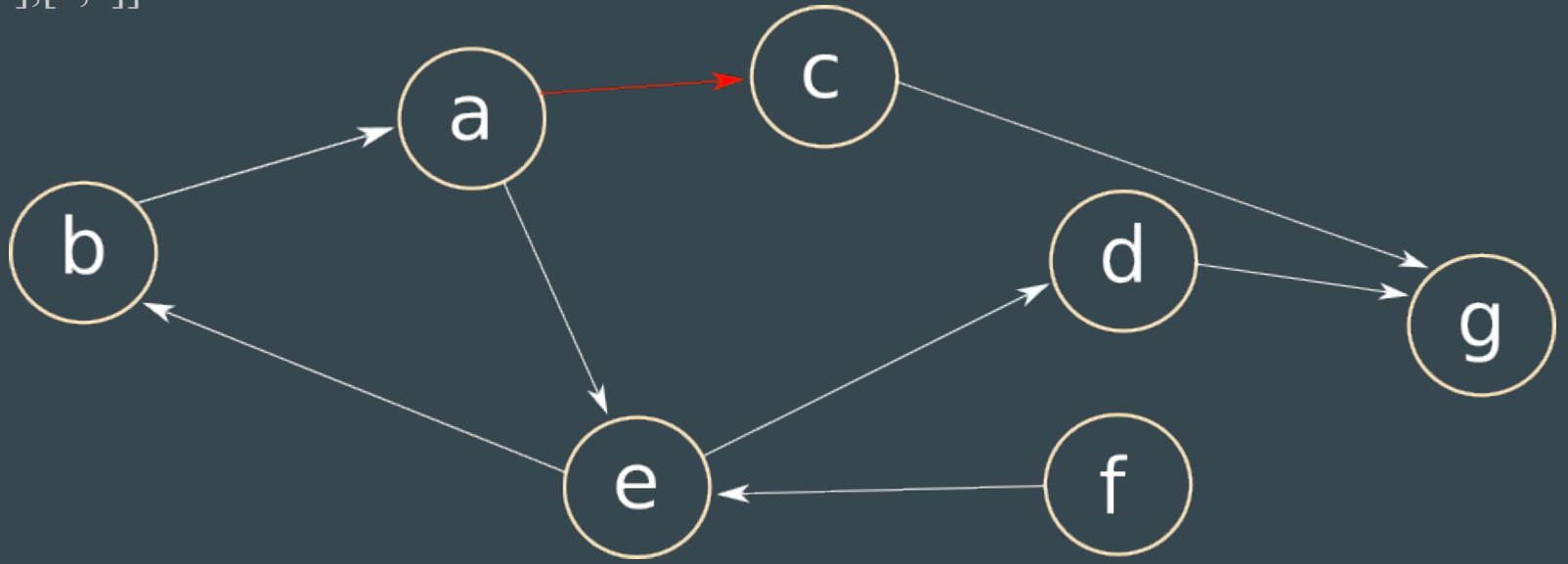
conectado(G, N1, N2):-

[[b,a]]



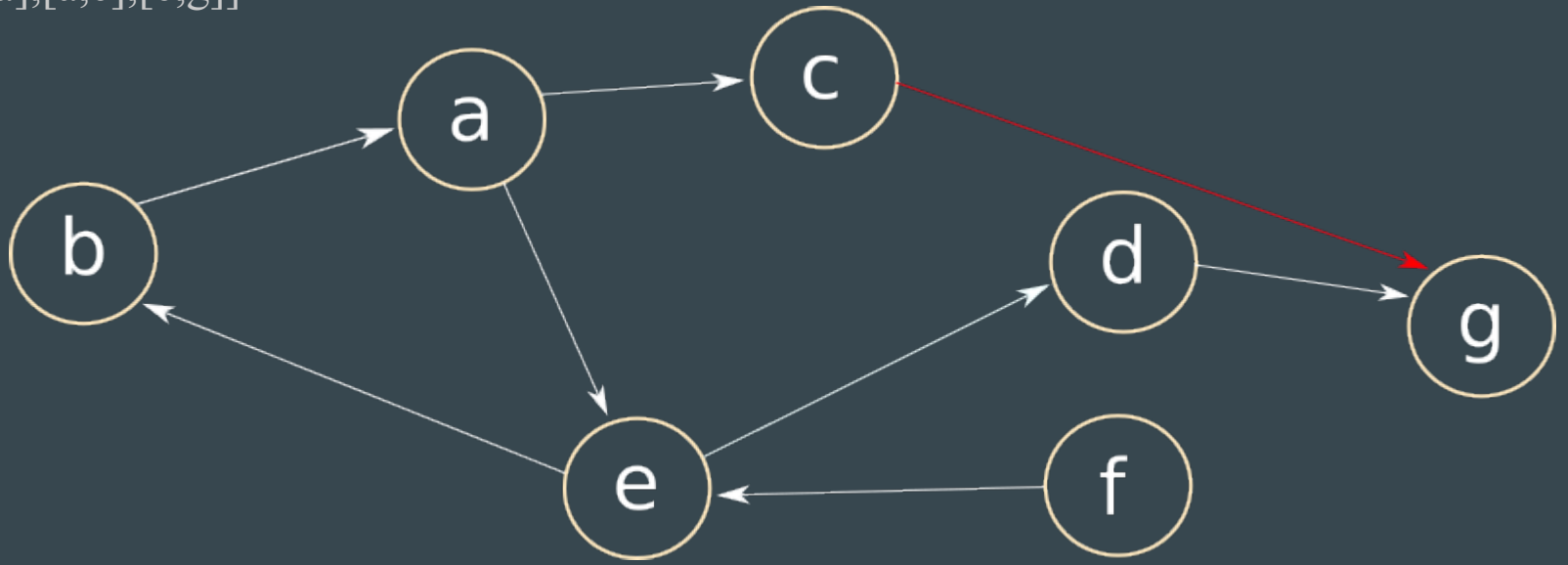
conectado(G, N1, N2):-

[[b,a],[a,c]]



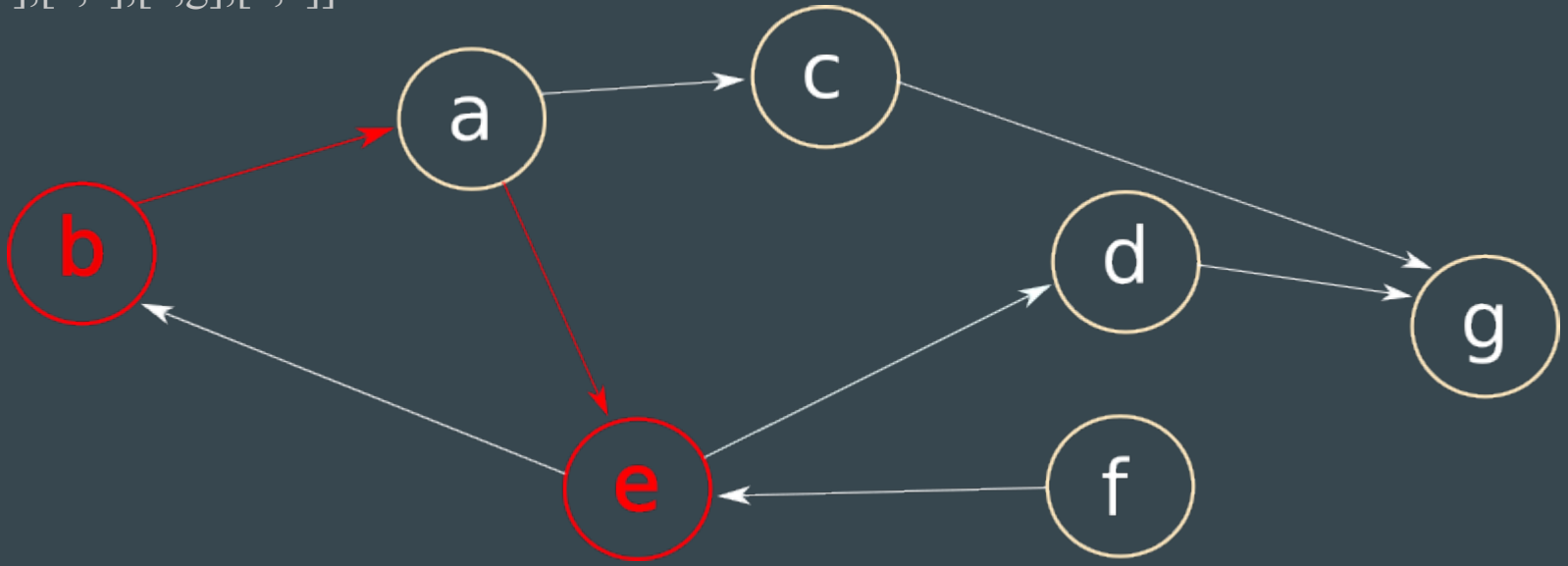
conectado(G, N1, N2):-

[[b,a],[a,c],[c,g]]

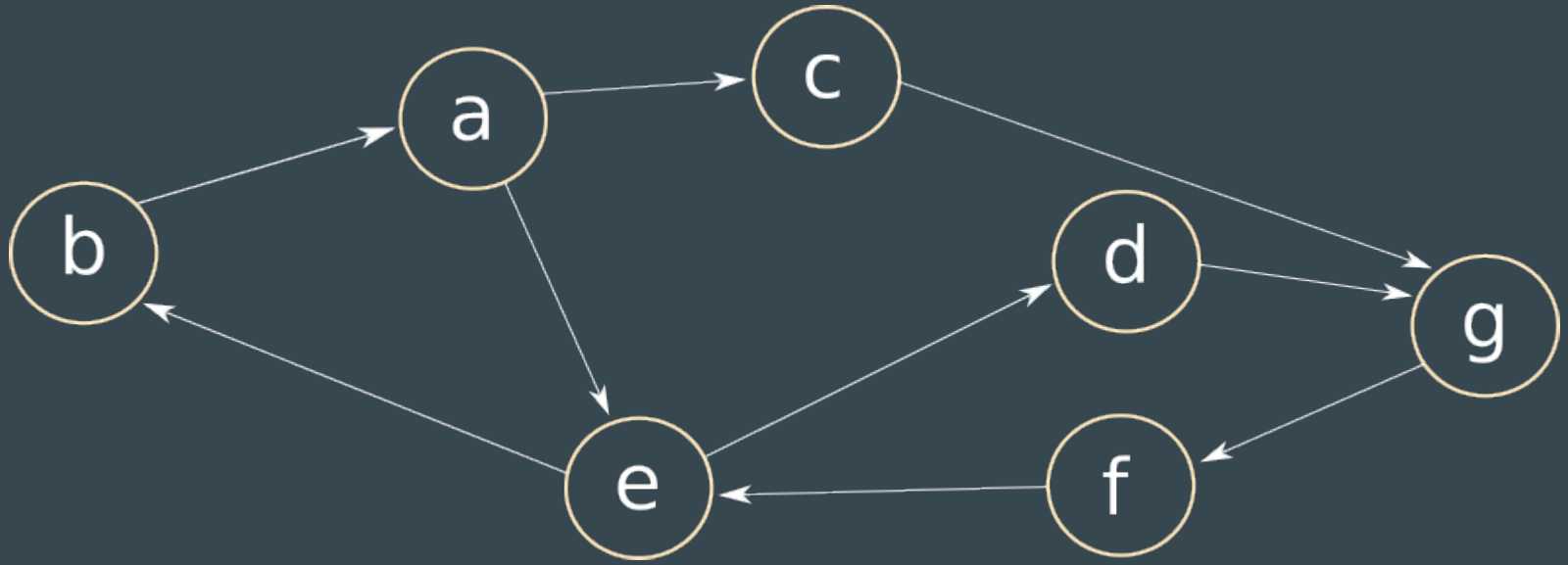


conectado(G, N1, N2):-

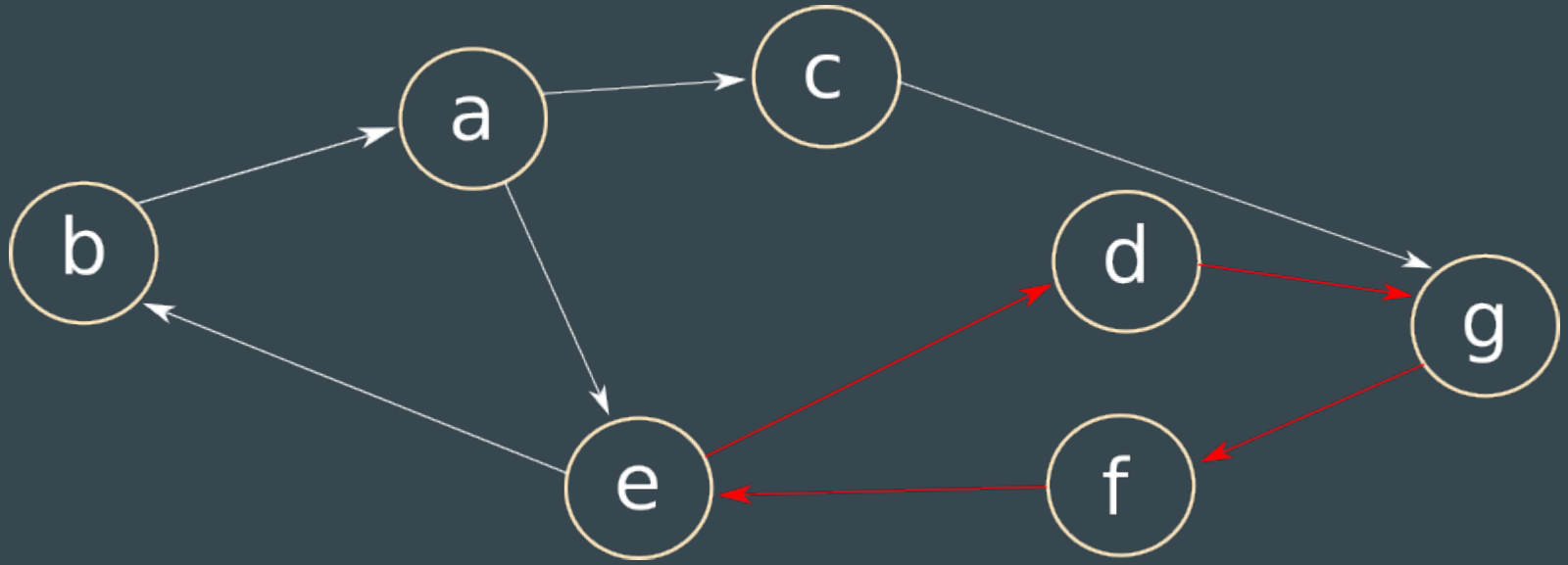
[[b,a],[a,c],[c,g],[a,e]]



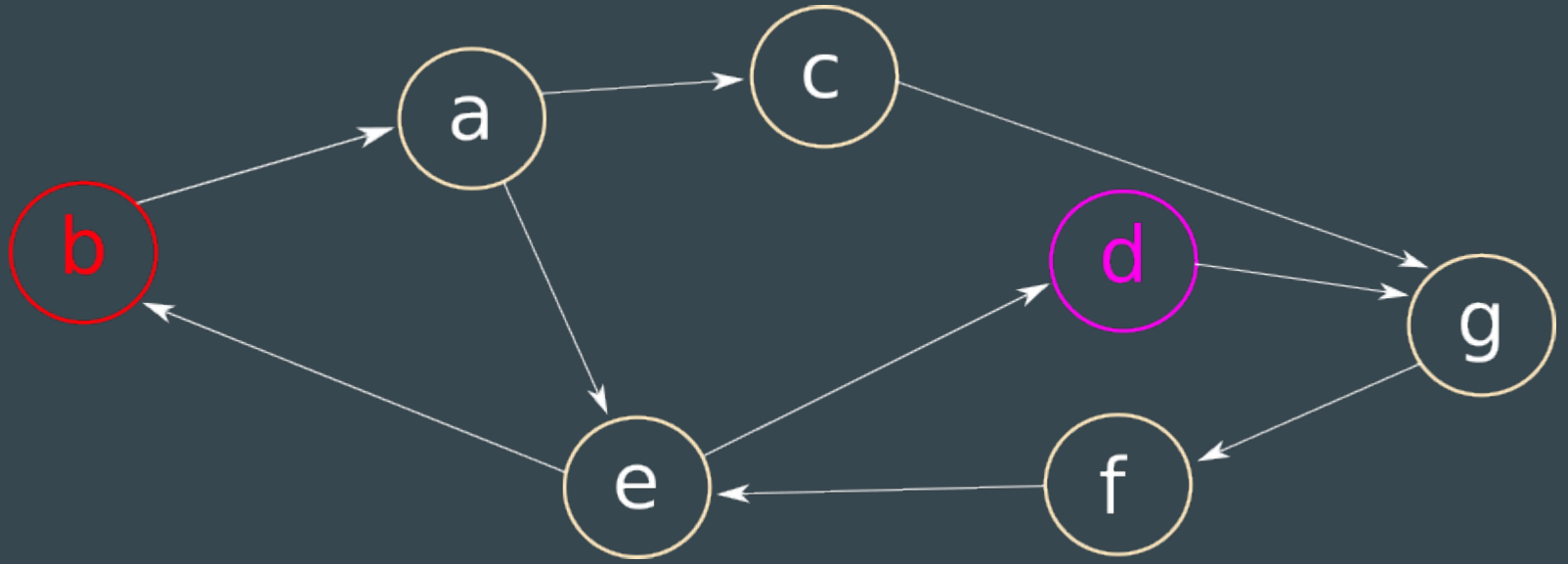
grafo_ciclico(G):-



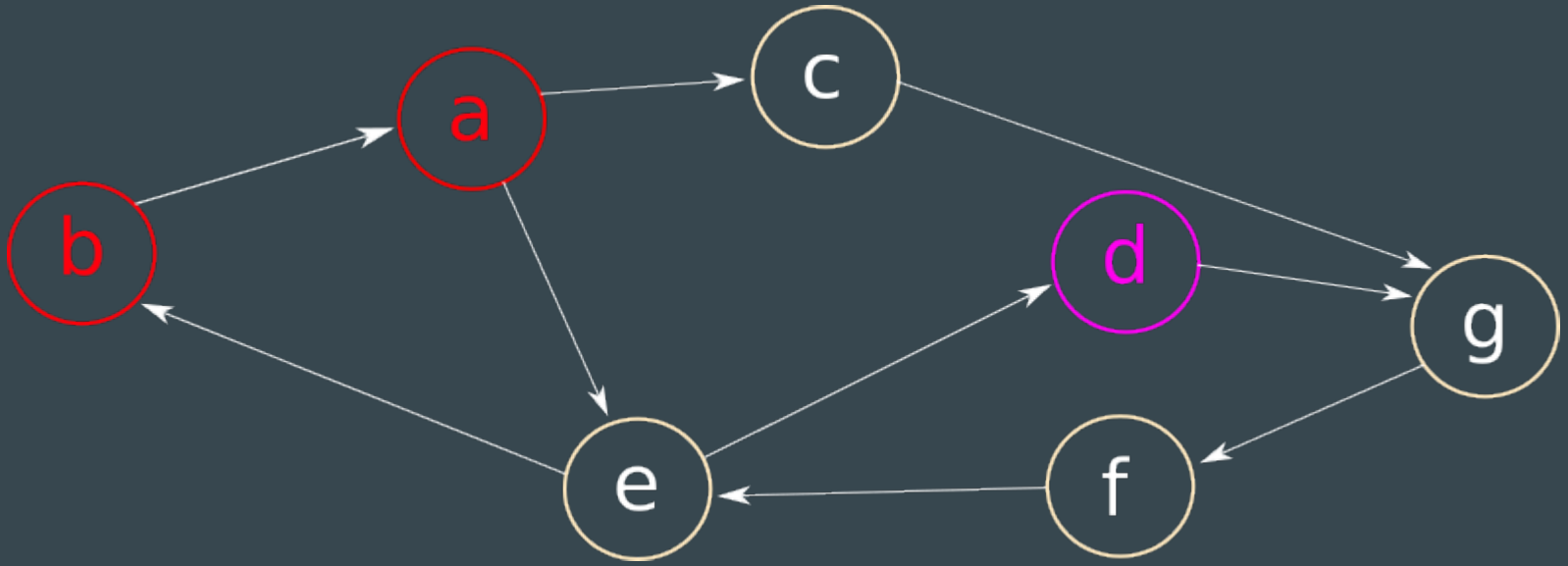
grafo_ciclico(G):-



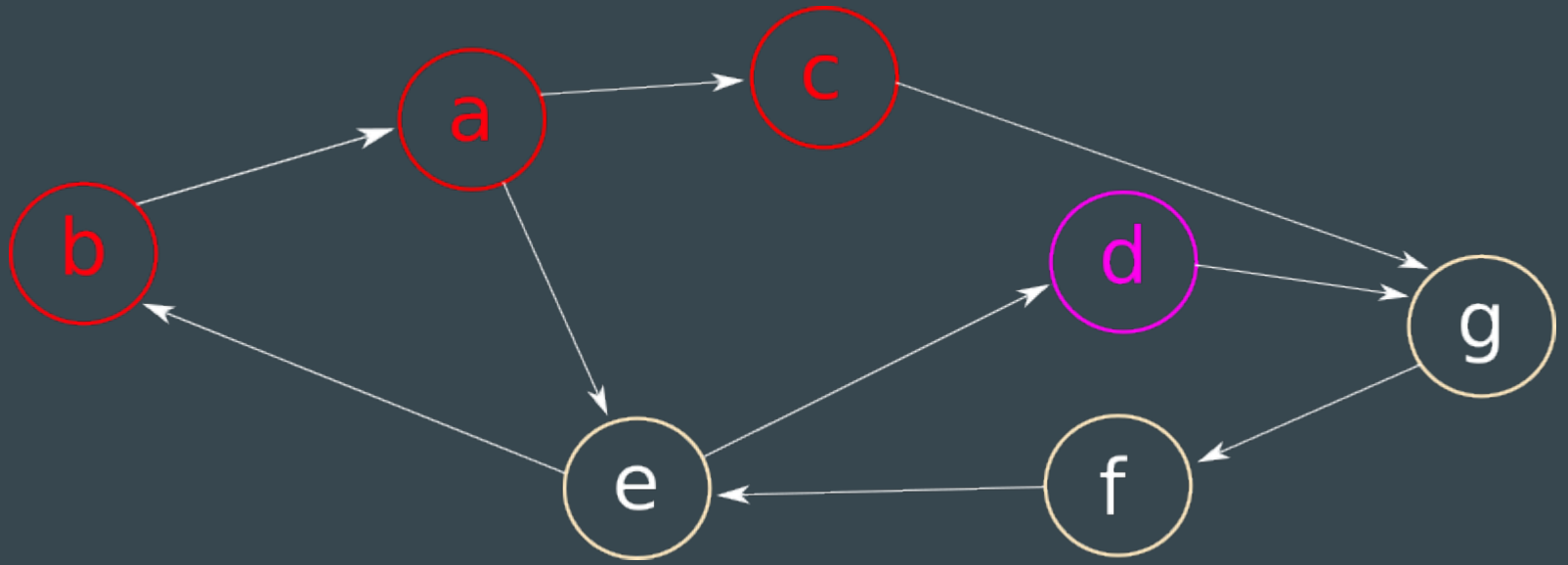
caminho_mais_curto(G,N1,N2,C):-



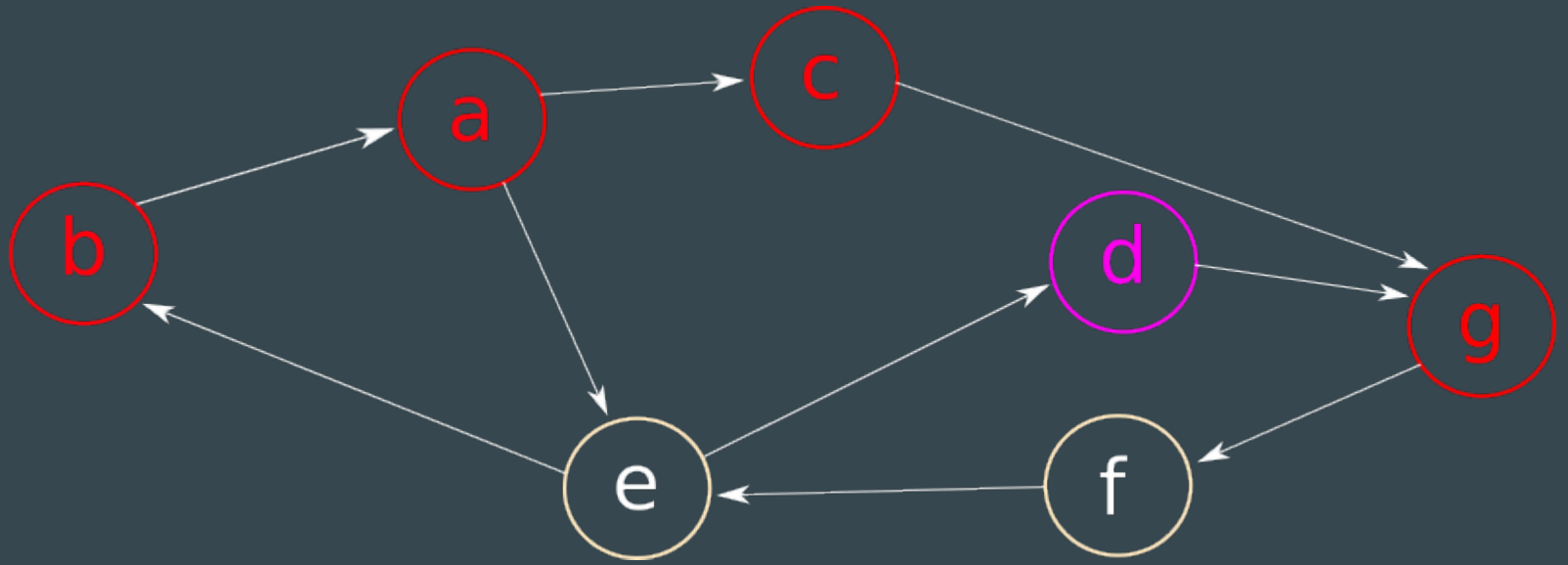
caminho_mais_curto(G,N1,N2,C):-



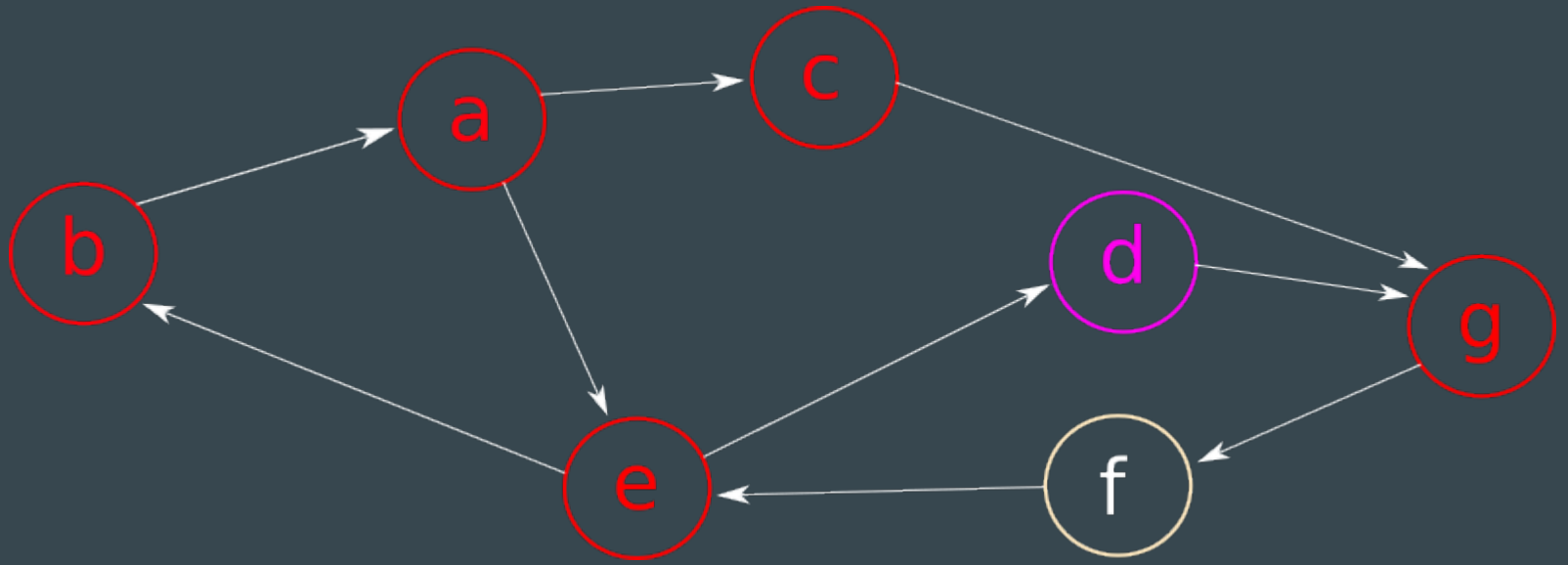
caminho_mais_curto(G,N1,N2,C):-



caminho_mais_curto(G,N1,N2,C):-



caminho_mais_curto(G,N1,N2,C):-



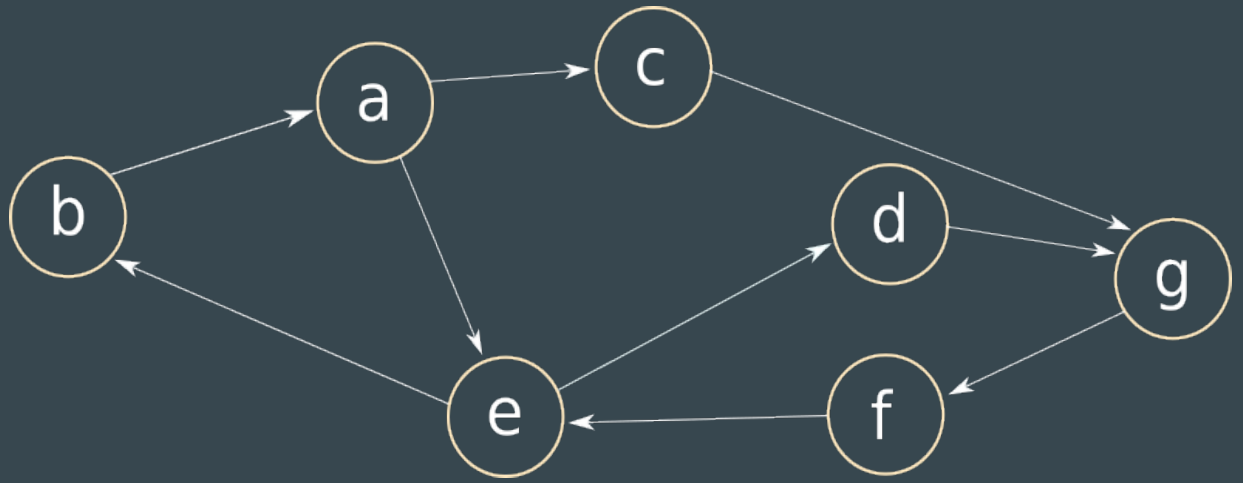
hamiltoniano(G):-

-nodos_do_grafo

- busca em profundidade

-lista de entrada
[a,c,g,f,e,b]

-testa se último está
conectado com primeiro



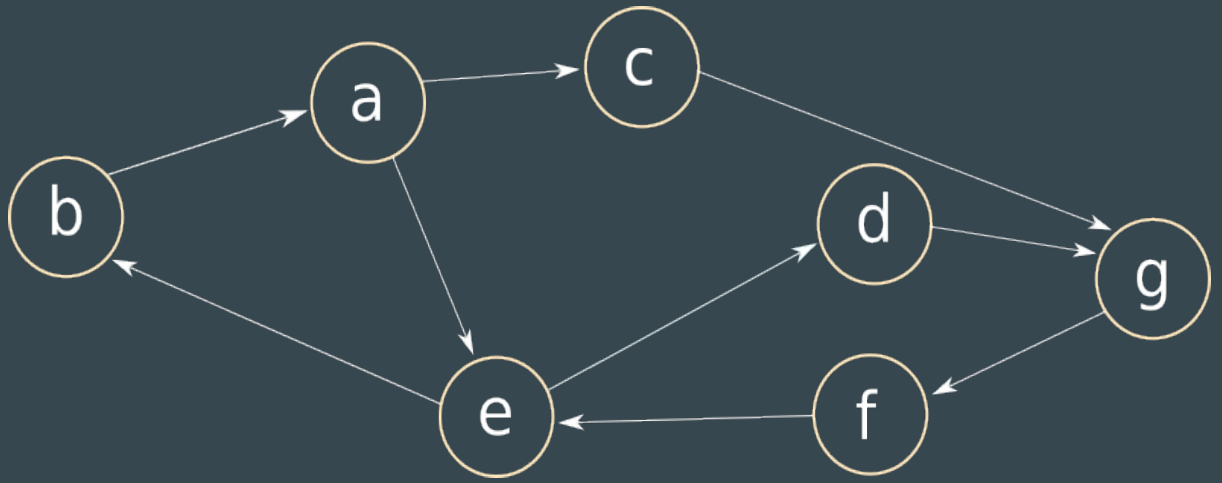
eulariano(G):-

- arestas_do_grafo

- busca em profundidade

-lista de entrada
[[a,c],[c,g],[g,f],[f,e],[e,b],[b,a],[a,e],[e,d],[d,g]]

-testa se último está conectado com primeiro



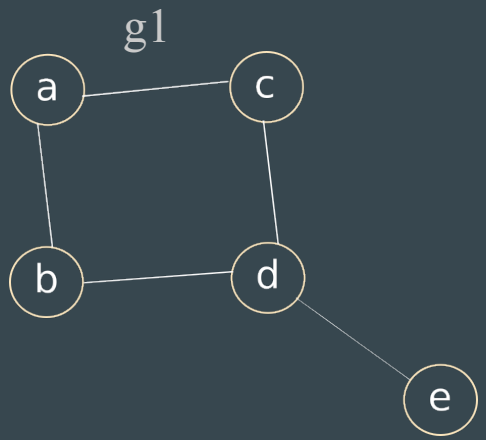
isomorfos(G1, G2)

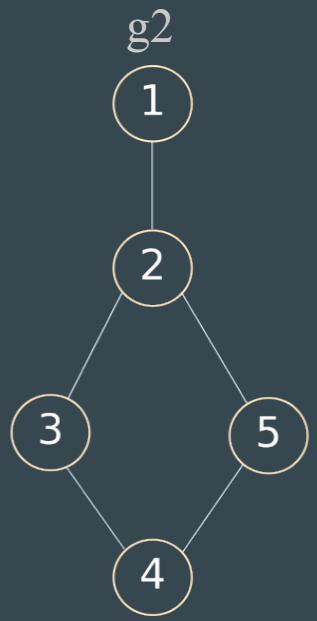
- compara número de vértices e arestas

- permutação

a | 1 - equiv(g1, a, g2, b)
b | 2
c | 3
d | 4
e | 5

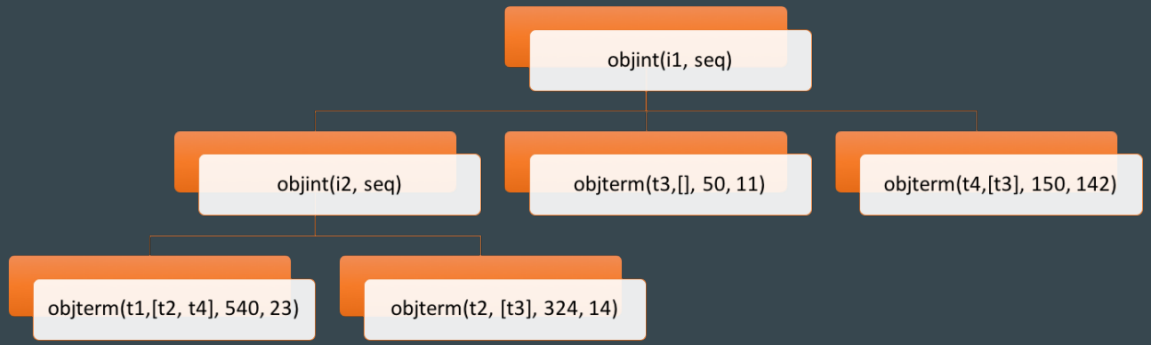
- compara 'a' com '1'
  [b,c] == [2]





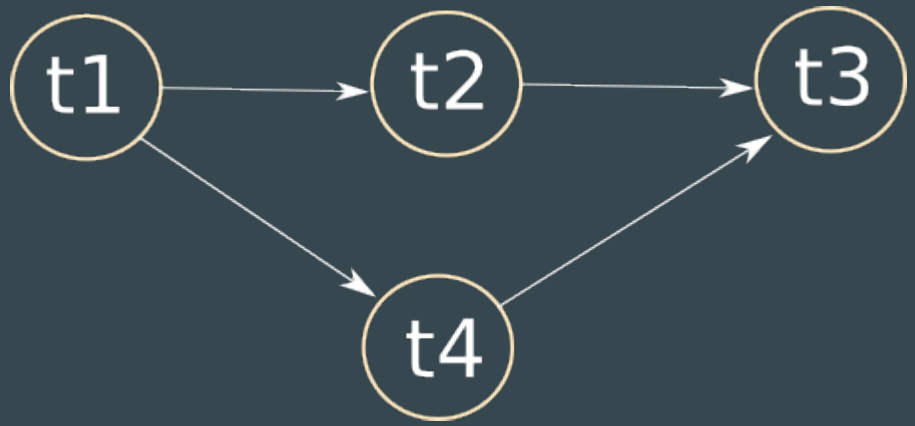
Lista Extra

extrai_grafo_aon(AG, G):-



- verifica se AG é uma árvore
- obtém uma lista de objetos terminais
- insere nodo(G, O, D)
- insere arco(G, O, R)

caminho_critico e prazo_caminho_critico(G, L):-



[t1, t4, t3] => 23+142+11 = 176

[t1, t2, t3] => 23+14+11 = 48

- acha a lista de nodos iniciais e finais
- acha a lista de todos os caminhos possíveis entre os nodos iniciais e finais
- calcula a soma das durações (prazos) de cada caminho e retorna o caminho com maior soma

Obrigado!