DCC888 – Worklist Algorithms

Nome: _____ Matrícula: _____

1. Considere o algoritmo a esquerda, que aplicado as restrições mostradas à direita produz a tabela abaixo:

$x_1 = \bot, x_2 = \bot, \ldots, x_n = \bot$
w = $[v_1, \ldots, v_n]$
while (w ≠ [])
  v_i = extract(w)
  $y = F_i(x_1, \ldots, x_n)$
  if $y \neq x_i$
    for $v \in \text{dep}(v_i)$
      w = insert(w, v)
    $x_i = y$

x1 = []

x2 = x1 ∪ (x3 \ [3, 5, 6]) ∪ {3}

x3 = x2

x4 = x1 ∪ (x5 \ [3, 5, 6]) ∪ {5}

x5 = x4

x6 = x2 ∪ x4

w	x1	x2	x3	x4	x5	x6
[x1, x2, x3, x4, x5, x6]	⊥	⊥	⊥	⊥	⊥	⊥
[x2, x4, x2, x3, x4, x5, x6]	[]	⊥	⊥	⊥	⊥	⊥
[x3, x6, x4, x2, x3, x4, x5, x6]	[]	[3]	⊥	⊥	⊥	⊥
[x2, x6, x4, x2, x3, x4, x5, x6]	[]	[3]	[3]	⊥	⊥	⊥
[x6, x4, x2, x3, x4, x5, x6]	[]	[3]	[3]	⊥	⊥	⊥
[x4, x2, x3, x4, x5, x6]	[]	[3]	[3]	⊥	⊥	[3]
[x5, x6, x2, x3, x4, x5, x6]	[]	[3]	[3]	⊥	⊥	[3]
[x4, x6, x2, x3, x4, x5, x6]	[]	[3]	[3]	[5]	⊥	[3]
[x6, x2, x3, x4, x5, x6]	[]	[3]	[3]	[5]	[5]	[3]
[x2, x3, x4, x5, x6]	[]	[3]	[3]	[5]	[5]	[3,5]
[x3, x4, x5, x6]	[]	[3]	[3]	[5]	[5]	[3,5]
[x4, x5, x6]	[]	[3]	[3]	[5]	[5]	[3,5]
[x5, x6]	[]	[3]	[3]	[5]	[5]	[3,5]
[x6]	[]	[3]	[3]	[5]	[5]	[3,5]
[]	[]	[3]	[3]	[5]	[5]	[3,5]

(a) Se, em vez de começássemos com a lista $[x_1, x_2, x_3, x_4, x_5, x_6]$, nós houvéssemos começado com a lista $[x_6, x_5, x_4, x_3, x_2, x_1]$, quantas iterações seriam necessárias para que atingíssemos um ponto fixo?

(b) A tabela mostrada assume que o primeiro elemento inserido na lista de trabalho é o próximo elemento a ser removido pela função *extract*. Esta ordem chama-se LIFO, do inglês *Last-In, First-Out*. Quantas iterações seriam necessárias para que atingíssemos um ponto fixo, se a ordem de inserção e extração fosse FIFO (*First-In, First-Out*)? Note que neste caso teríamos uma fila, em vez de uma pilha de processamento a ser feito.

(c) O nosso algoritmo não verifica se já existe um elemento na lista de trabalho, antes de inseri-lo lá. Podemos, assim, ter várias versões da mesma variável na lista. Embora essa seja uma abordagem por demais simplista, mesmo em implementações reais de listas de trabalho, podemos encontrá-la. Quais as vantagens de permitirmos que elementos repetidos permaneçam na lista de trabalho? Fundamente sua resposta em termos da complexidade computacional do algoritmo que estamos discutindo.

(d) Assume, agora, que um elemento é inserido na lista de trabalho somente se ele lá já não se encontre. Nesse caso, quantas iterações de nosso algoritmo seriam necessárias, até que atingíssemos um ponto fixo?

2. Quando formos estudar otimizações de *loops*, veremos que um nodo n_1 *domina* outro nodo n_2 em um grafo direcionado (N, A) com raiz H se todo caminho de H até n_2 passa por n_1. O conjunto de dominadores de um nodo pode ser aproximado pelas equações

$$Dom(n) = \begin{cases} \{n\} & if n \in H, \\ \{n\} \cup \bigcap_{(n',n)\in A} Dom(n') & \text{otherwise} \end{cases}$$

Escreva um algoritmo, baseado em lista de trabalho, que compute o conjunto de dominadores dos nodos de um grafo direcionado com raiz.