Fontes principais

1. J. Jaja, An introduction to Parallel Algorithms, Addison Wesley, 92

    ▷ Algoritmos paralelos

2. E. Cáceres, H. Mongeli, S. Song: Algoritmos paralelos usando CGM/PVM/MPI: uma introdução
   `http://www.ime.usp.br/~song/papers/jai01.pdf`

Técnicas de desenvolvimento de algoritmos paralelos

2

Pointer Jumping

Descrição da técnica

▷ Técnica normalmente aplicada sobre uma lista encadeada de elementos.

▷ Para cada elemento da lista é associado um processador.

Pointer Jumping

▷ A técnica de pointer jumping consiste de atualizar o sucessor (ou seguinte) de cada vértice com o sucessor do sucessor.

Pointer Jumping

▷ Em um determinado passo do algoritmo, resolvemos o problema para todos os elementos da lista que estão até uma certa distância de um elemento individual da lista.

▷ Esta distância dobra a cada passo. Logo, depois de k passos, o problema foi resolvido para todos os elementos da lista que estão até uma distância 2^k.

Exemplo: List Ranking

Distância dos elementos da lista ao fim da lista.

▷ Dada uma lista encadeada de n elementos, determinar a distância de cada elemento da lista ao fim da lista.

Idéia:

▷ Associar um processador para cada elemento da lista

▷ A cada passo, o processador responsável pelo elemento i duplica sua estimativa da distância até o fim da lista.

Exemplo: List Ranking

Entrada:

▷ n: número de elementos da lista

▷ L : lista encadeada

▷ $prox[i]$: elemento de L seguinte ao elemento i. Se i é o último, então $prox[i] = nil$.

Estrutura auxiliar:

▷ $p[i]$: inicialmente terá cópia de $prox[i]$

Saída:

▷ $dist[i]$: distância do elemento i ao fim de L.

List Ranking

Algoritmo

para $i \in L$ **faça em paralelo**

$p[i] := prox[i]$

se $p[i] = nil$ **então**

$dist[i] := 0$

senão

$dist[i] := 1$

List Ranking

continuação...

> **para** $j := 1$ **até** $\lceil \log_2 n \rceil$ **faça**
>   **para** $i \in L$ **faça em paralelo**
>   **se** $p[i] \neq nil$ **então**
> $dist[i] := dist[i] + dist[p[i]]$
> $p[i] := p[p[i]]$

List Ranking

Submodelo e complexidades:

Submodelo: EREW

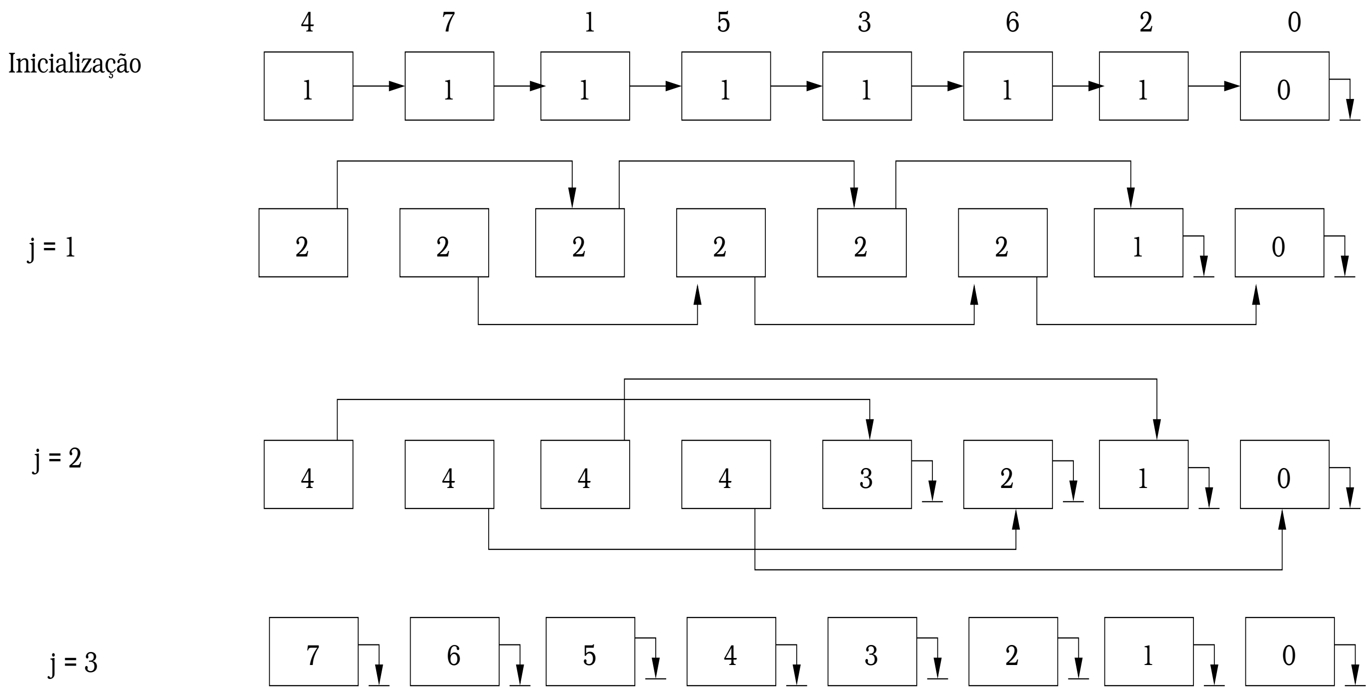
Complexidades

- ▷ Tempo: $O(\log_2 n)$
- ▷ Processador: $O(n)$
- ▷ É eficiente. Não é ótimo.

Exemplo: $n = 8$

		0	1	2	3	4	5	6	7
Inicialização:	prox	nil	5	0	6	7	3	2	1
	p	nil	5	0	6	7	3	2	1
	dist	0	1	1	1	1	1	1	1
j = 1	dist	0	2	1	2	2	2	2	2
	p	nil	3	nil	2	1	6	0	5
j = 2	dist	0	4	1	3	4	4	2	4
	p	nil	2	nil	nil	3	0	nil	6
j = 3	dist	0	5	1	3	7	4	2	6
	p	nil	nil	nil	nil	nil	nil	nil	nil
		0	1	2	3	4	5	6	7

Exemplo: $n = 8$


4 7 1 5 3 6 2 0
Inicialização
1 1 1 1 1 1 1 0
j = 1
2 2 2 2 2 2 1 0
j = 2
4 4 4 4 3 2 1 0
j = 3
7 6 5 4 3 2 1 0

Algoritmo genérico

para $i \in L$ **faça em paralelo**
$p[i] := prox[i]$
$v[i] := valor[i]$

para $j := 1$ **até** $\lceil \log_2 n \rceil$ **faça**
para $i \in L$ **faça em paralelo**
se $p[i] \neq nil$ **então**
$v[i] := v[i] \ op \ v[p[i]]$
$p[i] := p[p[i]]$

Exemplos análogos: Máximo (ou mínimo)

Determinar o máximo (ou mínimo) dos dados dos elementos da lista.

- $\triangleright$ op: máximo (ou mínimo)
- $\triangleright$ $valor[i]$: dado do elemento i.
- $\triangleright$ $v[i]$: máximo (ou mínimo) dos dados dos elementos desde o elemento i até o último elemento.

Exemplos análogos: Soma

Determinar a soma dos dados dos elementos da lista.

- $\triangleright$ op: soma
- $\triangleright$ $valor[i]$: dado do elemento i.
- $\triangleright$ $v[i]$: soma dos dados dos elementos desde o elemento i até o último elemento.

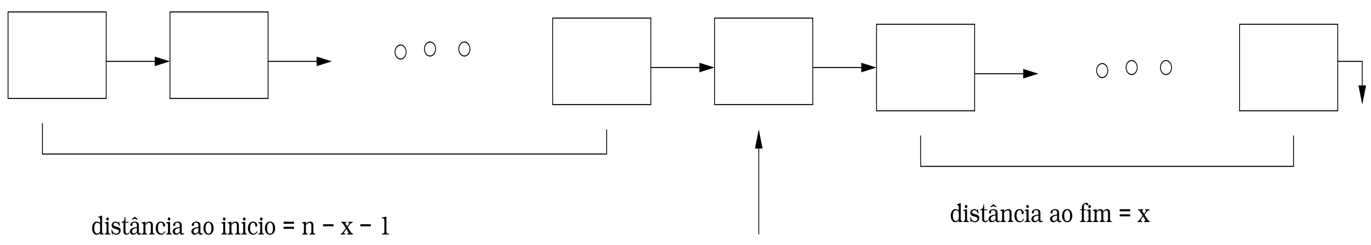
Exemplos análogos: distância dos elementos ao fim da lista

Determinar a distância dos elementos ao fim da lista.

- ▷ op: soma
- ▷ $valor[i]$:1, exceto para o último elemento, para o qual $valor[i] = 0$.
- ▷ $v[i]$: distância do elemento i ao fim da lista.

Exemplos análogos: distância dos elementos ao início da lista

Ideia:



Análogo ao anterior

No final: $v[i] = n - v[i] - 1$, para todo elemento i da lista.

Construção de Sub-listas

Construção de Sub-listas

Alguns elementos da lista estão marcados e outros não. Deseja-se construir uma sub-lista apenas com os elementos marcados.

Construção de Sub-listas

Ideia:

▷ Aplicar duplicação recursiva, pulando apenas elementos não marcados.

▷ Ao final deste passo, a lista só possui elementos marcados, com a possível exceção do primeiro elemento.

▷ Se necessário então, corrige o primeiro elemento da sub-lista.

Construção de Sub-listas

Entrada:

$\triangleright$ n : o número de elementos da lista

$\triangleright$ L: lista de n elementos

$\triangleright$ $inicio$: ponteiro para o primeiro elemento da lista.

$\triangleright$ $prox[i]$: ponteiro para o elemento seguinte ao elemento i em L. Se i é o último elemento de L, $prox[i] = nil$

$\triangleright$ $marca[i]$: flag representando se o elemento i está marcado ou não.

Construção de Sub-listas

Saída:

$\triangleright$ $inicioSL$: ponteiro para o primeiro elemento da sub-lista.

$\triangleright$ $p[i]$: inicialmente terá cópia de $prox[i]$. Ao final do algoritmo, terá o ponteiro para o elemento seguinte ao elemento i na sub-lista. Se i é o último elemento da sub-lista, $p[i] = nil$

Construção de Sub-listas

Algoritmo

para $i \in L$ **faça em paralelo**
$p[i] := prox[i]$

$inicioSL = inicio$
para $j := 1$ **até** $\lceil \log_2 n \rceil$ **faça**
para $i \in L$ **faça em paralelo**
se $p[i] \neq nil$ e not $marca[i]$ **então**
$p[i] := p[p[i]]$

continuação...

se not $marca[inicioSL]$ **então**
    $inicioSL := p[inicioSL]$

para $i \in L$ **faça em paralelo**
    se not $marca[i]$ **então**
        $p[i] := nil$

Construção de Sub-listas

Submodelo e complexidades:

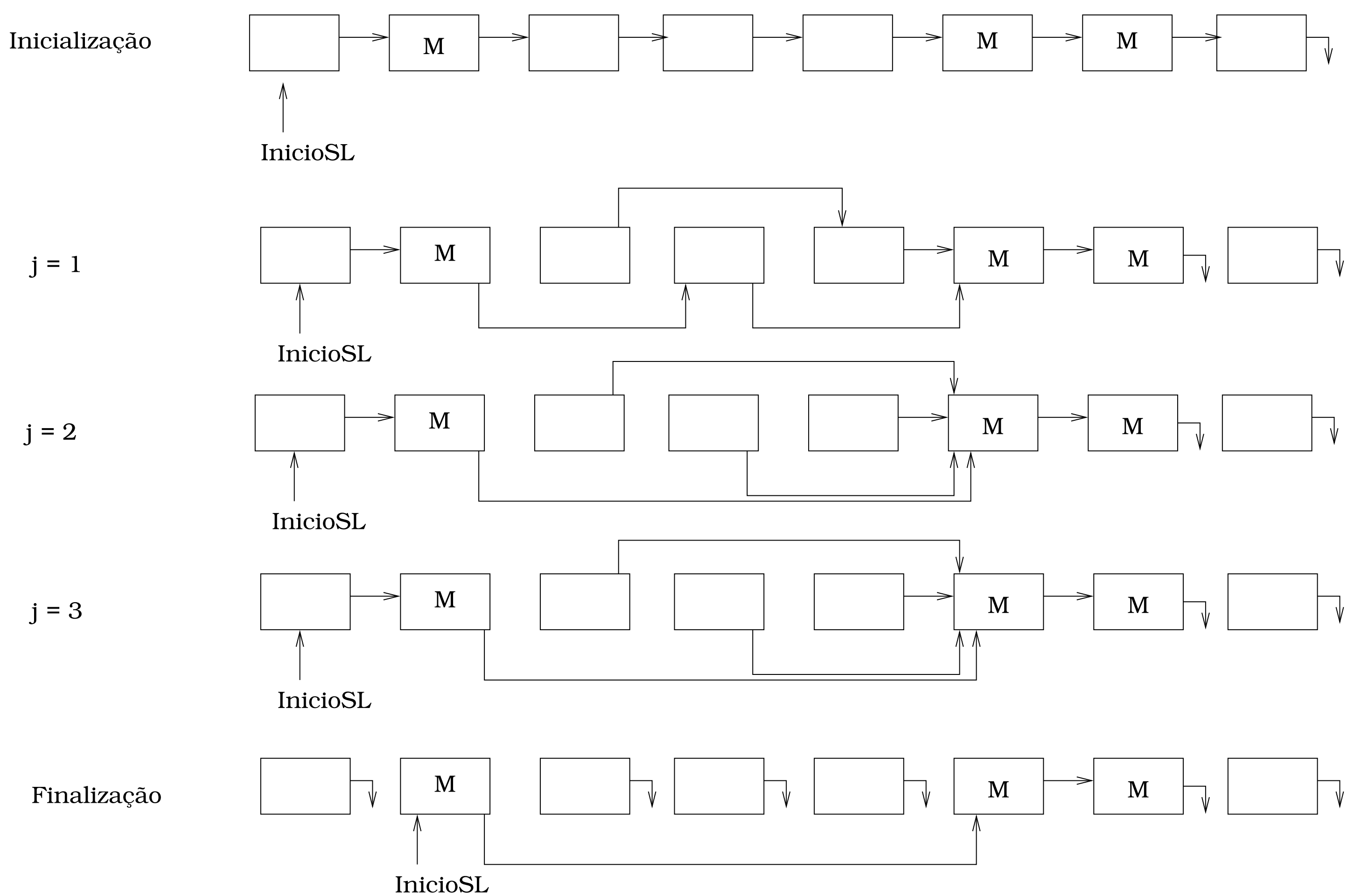
Submodelo: CREW

- ▷ leitura concorrente em marca.

Complexidades

- ▷ Tempo: $O(\log_2 n)$
- ▷ Processador: $O(n)$
- ▷ É eficiente. Não é ótimo.

Exemplo: $n = 8$


Inicialização
M
M
M
InicioSL
j = 1
M
M
M
InicioSL
j = 2
M
M
M
InicioSL
j = 3
M
M
M
InicioSL
Finalização
M
M
M
InicioSL

Algoritmos em Árvores Utilizando Duplicação Recursiva

Determinar raízes de uma floresta

Descrição:

$\triangleright$ Dada uma floresta F de árvores enraizadas, deseja-se determinar, para cada vértice i de F, a raiz da árvore que contém i.

Determinar raízes de uma floresta

Entrada:

$\triangleright$ n : número de vértices de F.

$\triangleright$ $pai[i]$: vértice pai do vértice i em F. Se i é uma raiz, $pai[i] = -1$

Saída:

$\triangleright$ $raiz[i]$: vértice raiz da árvore que contém o vértice i.

Determinar raízes de uma floresta

Idéia:

▷ Aplicar duplicação recursiva sobre o pai. Inicialmente cada vértice sabe o seu pai. Depois de um passo, cada vértice sabe o pai do seu pai, e assim por diante, até chegar a raiz.

Determinar raízes de uma floresta

Algoritmo

para $0 \leq i \leq n-1$ **faça em paralelo**
$raiz[i] := pai[i]$
se $raiz[i] \neq -1$ **então**
$raiz[i] := i$

▷ Duplicação recursiva

para $0 \leq i \leq n-1$ **faça em paralelo**
enquanto $raiz[i] \neq raiz[raiz[i]]$ **faça**
$raiz[i] := raiz[raiz[i]]$

Outra maneira de escrever o algoritmo

Algoritmo

para $0 \leq i \leq n-1$ **faça em paralelo**

$raiz[i] := pai[i]$

se $raiz[i] \neq -1$ **então**

$raiz[i] := i$

▷ Duplicação recursiva

para $j := 1$ **até** $\lceil \log_2 n \rceil$ **faça**

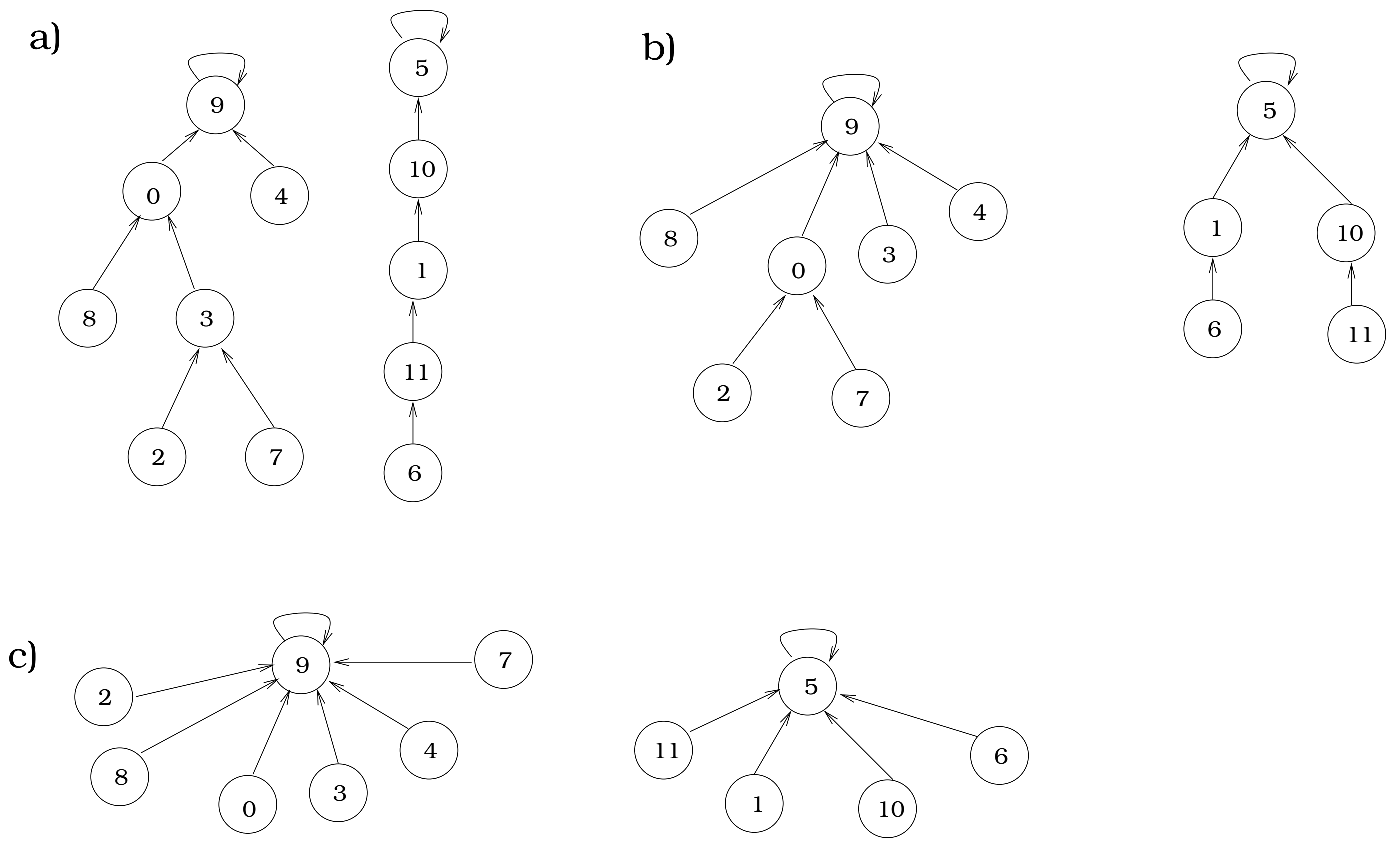
para $0 \leq i \leq n-1$ **faça em paralelo**

se $raiz[i] \neq raiz[raiz[i]]$ **então**

$raiz[i] := raiz[raiz[i]]$

Exemplo: $n = 12$, $\lceil \log_2 n \rceil = 4$

Raízes : 5, 9



		0	1	2	3	4	5	6	7	8	9	10	11
	pai	9	10	3	0	9	−1	11	3	0	−1	5	1
Inicialmente	raiz	9	10	3	0	9	5	11	3	0	9	5	1
passo 1:	raiz	9	5	0	9	9	5	1	0	9	9	5	10
passo 2:	raiz	9	5	9	9	9	5	5	9	9	9	5	5

Passos 3 e 4 (se executados) não fazem nada.

Determinar raízes de uma floresta

Submodelo e complexidades

▷ CREW: Leitura concorrente em Raiz, quando dois vértices apontam para um mesmo pai.

▷ Tempo: $O(\log_2 n)$. Gasta $\lceil \log_2 h \rceil$ passos, onde h é o máximo das alturas das árvores de F. No pior caso, temos uma única árvore que é uma lista, logo com altura $\lceil \log_2 n \rceil$.

▷ Processadores: $O(n)$. Um processador para cada vértice.

▷ É eficiente. Não é ótimo.

Computação de Prefixos em uma Floresta

Computação de Prefixos em uma Floresta

Dada um floresta F de árvores enraizadas, deseja-se determinar, para cada vértice i de F, uma computação com os pesos dos vértices no caminho de i até a raiz da árvore que contém i.

Soma de Prefixos em uma Floresta

Soma de Prefixos em uma Floresta

Entrada:

- $\triangleright$ n: número de vértices de F.
- $\triangleright$ $pai[i]$: vértice pai do vértice em F. Se i é uma raiz $pai[i] = nil$.
- $\triangleright$ $peso[i]$: peso do vértice i

Soma de Prefixos em uma Floresta

Estrutura auxiliar:

$\triangleright$ $p[i]$: inicalmente terá uma cópia de $pai[i]$

Saída:

$\triangleright$ $soma[i]$: soma dos pesos dos vértices no caminho do vértice i até a raiz de sua árvore.

Soma de Prefixos em uma Floresta

Idéia:

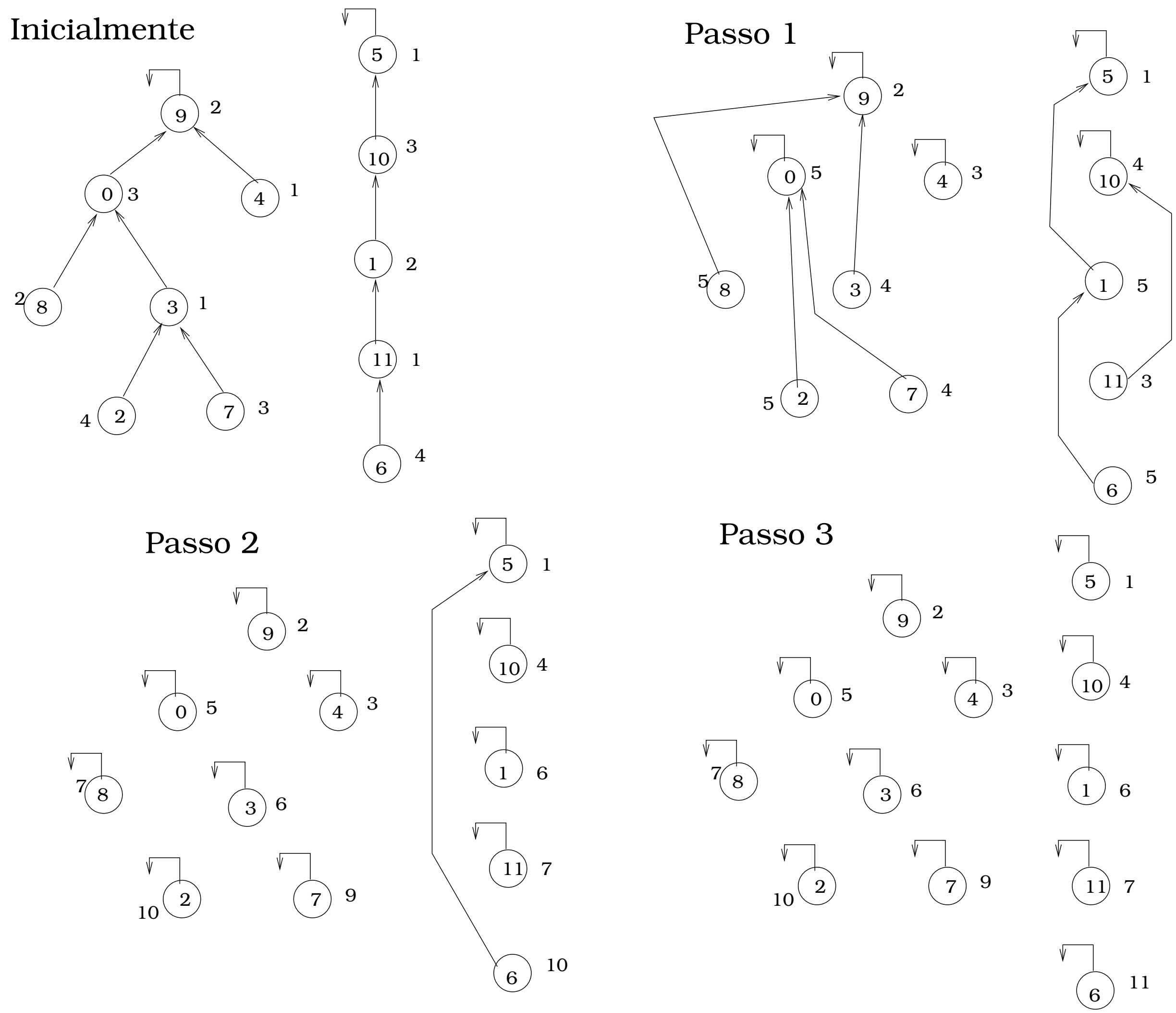
▷ Aplicar duplicação recursiva sobre o pai. Inicialmente cada vértice sabe o seu peso. Depois de um passo, cada vértice sabe a soma do seu peso com o peso do seu pai, e assim por diante, até chegar a raiz.

Soma de Prefixos em uma Floresta

Algoritmo

para $0 \leq i \leq n-1$ **faça em paralelo**
$p[i] := pai[i]$
$soma[i] := peso[i]$

para $0 \leq i \leq n-1$ **faça em paralelo**
enquanto $p[i] \neq nil$ **faça**
$soma[i] := soma[i] + soma[p[i]]$
$p[i] := p[p[i]]$



Passo 4 (se executado) não faz nada

Soma de Prefixos em uma Floresta

Submodelo e complexidades:

▷ CREW: Leitura concorrente em $soma$ e p, quando dois vértices apontam para um mesmo pai.

▷ Tempo: $O(\log_2 n)$. Gasta $\lceil \log_2 n \rceil$ passos, onde h é o máximo das alturas das árvores de F. No pior caso, temos uma única árvore que é uma lista, logo com altura $\lceil \log_2 n \rceil$.

▷ Processadores: $O(n)$. Um processador para cada vértice.

▷ É eficiente.

Soma de Prefixos em uma Floresta

Observação:

Se fazemos $peso[i] = 1$, para todo vértice i de F, o algoritmo calculará o nível de cada vértice na sua árvore.

Fim