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

Método da árvore binária balanceada

O problema é decomposto em 2 subproblemas menores que são decompostos, cada um, em dois subproblemas menores, e assim sucessivamente, até chegarmos a subproblemas triviais.

A decomposição do problema é representada por uma árvore binária e balanceada

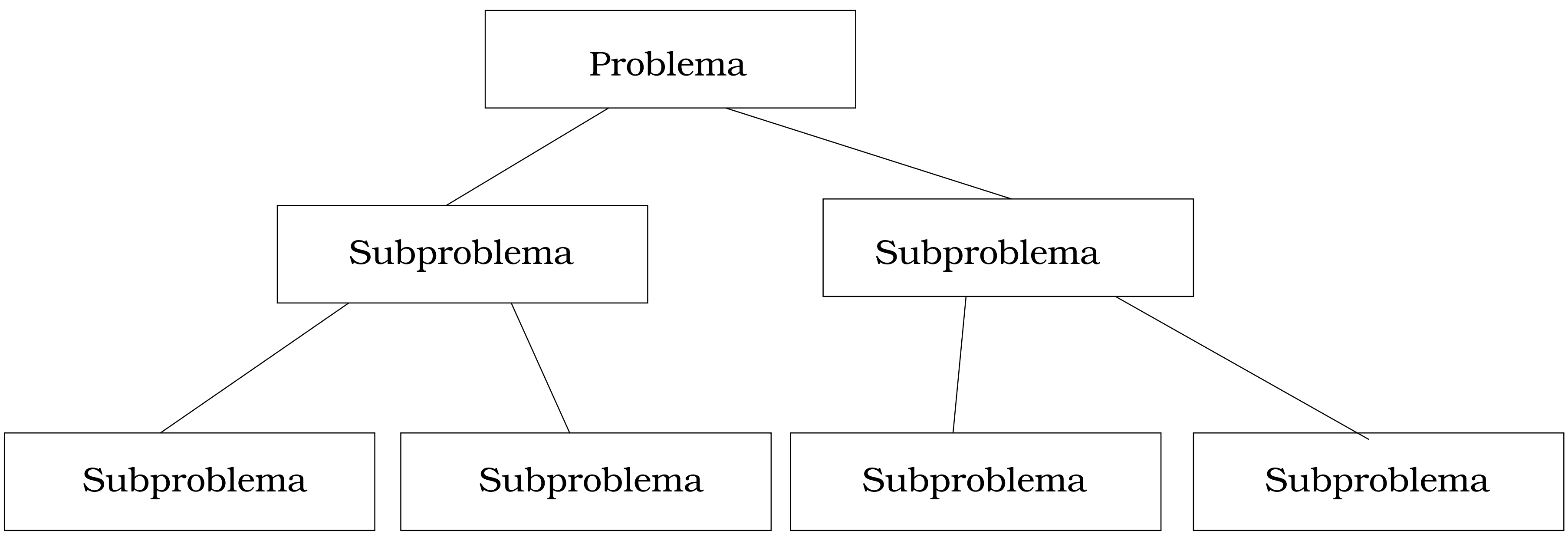
Método da árvore binária balanceada

O problema é resolvido de baixo para cima na árvore (dos sub-problemas menores para os maiores).

Temos um passo de tempo no algoritmo para cada nível da árvore.

Método da árvore binária balanceada

Todos os subproblemas de um mesmo nível da árvore são resolvidos em paralelo.

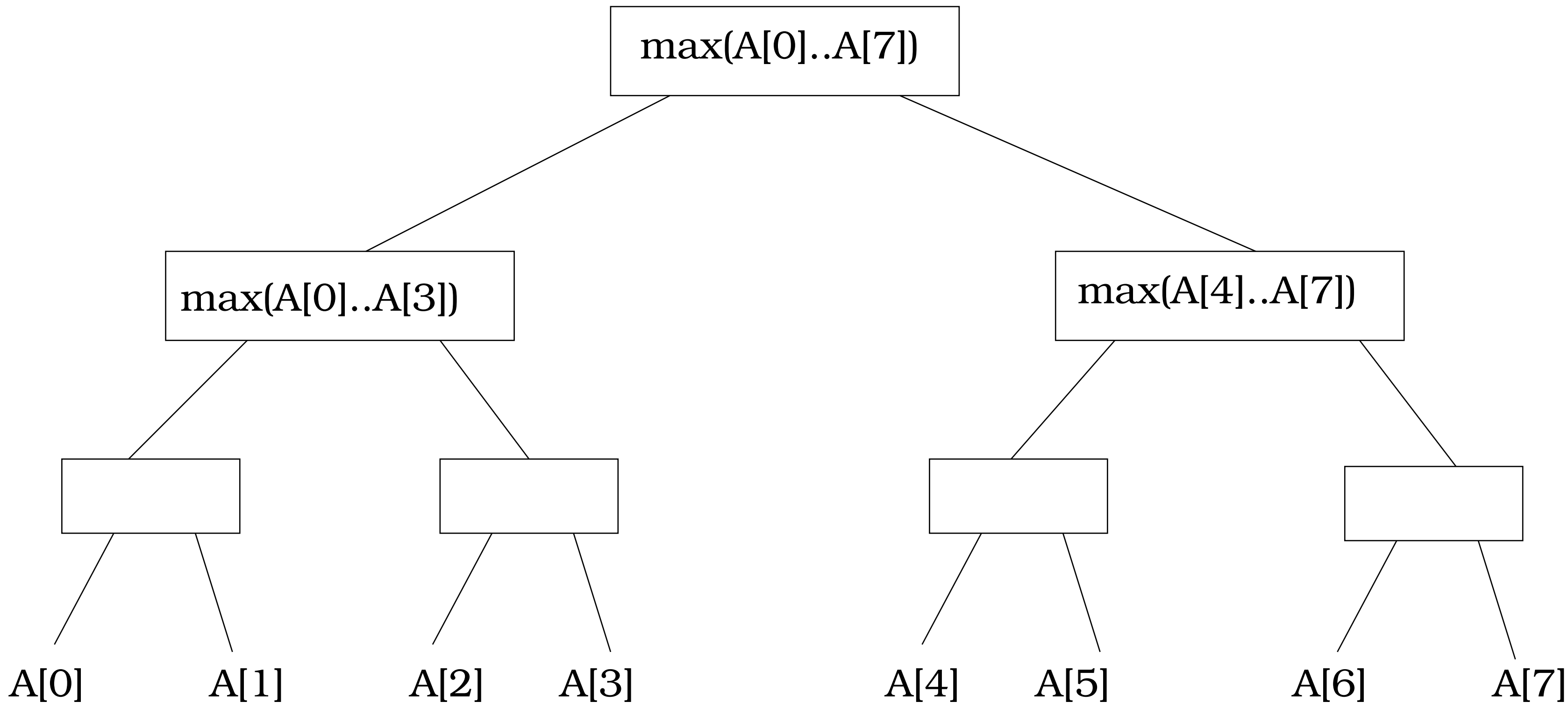


Algoritmo para determinar o elemento de valor máximo de um vetor

Idéia:

- ▷ Na base da árvore (folhas) ficam os elementos do vetor
- ▷ O algoritmo determina o máximo de 2 em 2 elementos

Algoritmo para determinar o elemento de valor máximo de um vetor



Algoritmo para determinar o elemento de valor máximo de um vetor

Entrada:

- $\triangleright$ n: número de elementos do vetor $= 2^n$ (senão, ajeitar)
- $\triangleright$ A: vetor de n elementos $A[0] \cdots A[n-1]$

Algoritmo para determinar o elemento de valor máximo de um vetor

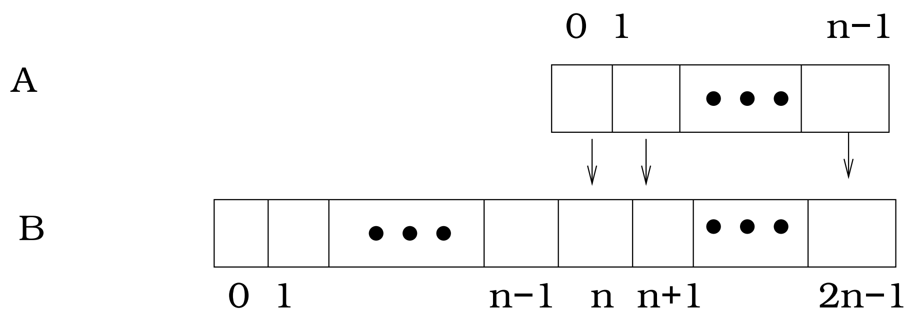
Estrutura auxiliar:

- ▷ B: vetor com $2 \times n$ posições $B[0], \cdots B[2n-1]$
- ▷ As n posições finais de B terão uma cópia de A.
- ▷ As n posições iniciais de B terão máximos intermediários.
- ▷ Ao final do algoritmo, $B[1]$ terá o máximo de A.

Algoritmo para determinar o elemento de valor máximo de um vetor

Saída:

▷ máximo: valor do elemento máximo de A



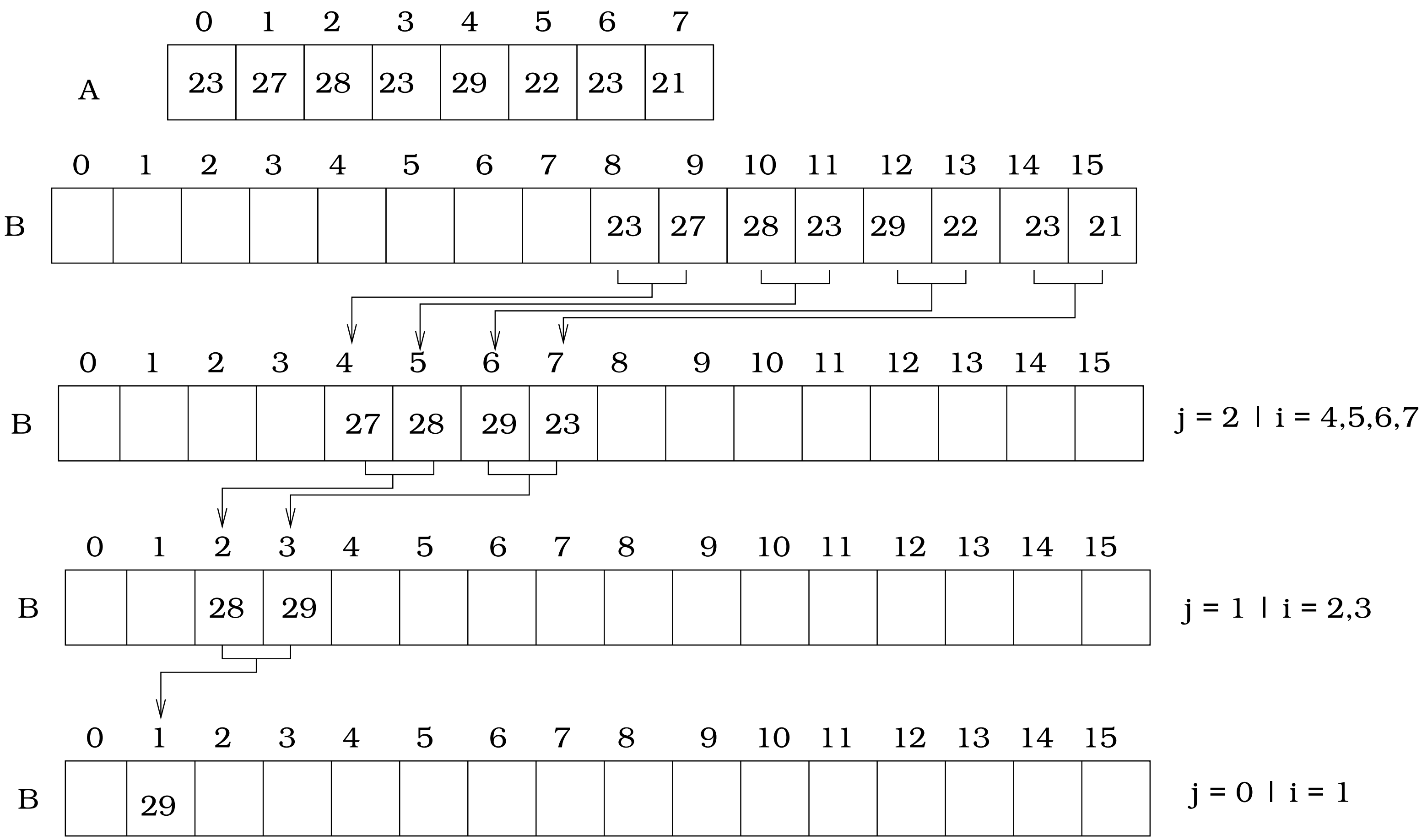
Algoritmo

$\triangleright$ Vetor A é copiado para a 2a. metade de B
para $0 \leq i \leq n-1$ **faça em paralelo**
$B[n+i] := A[i]$

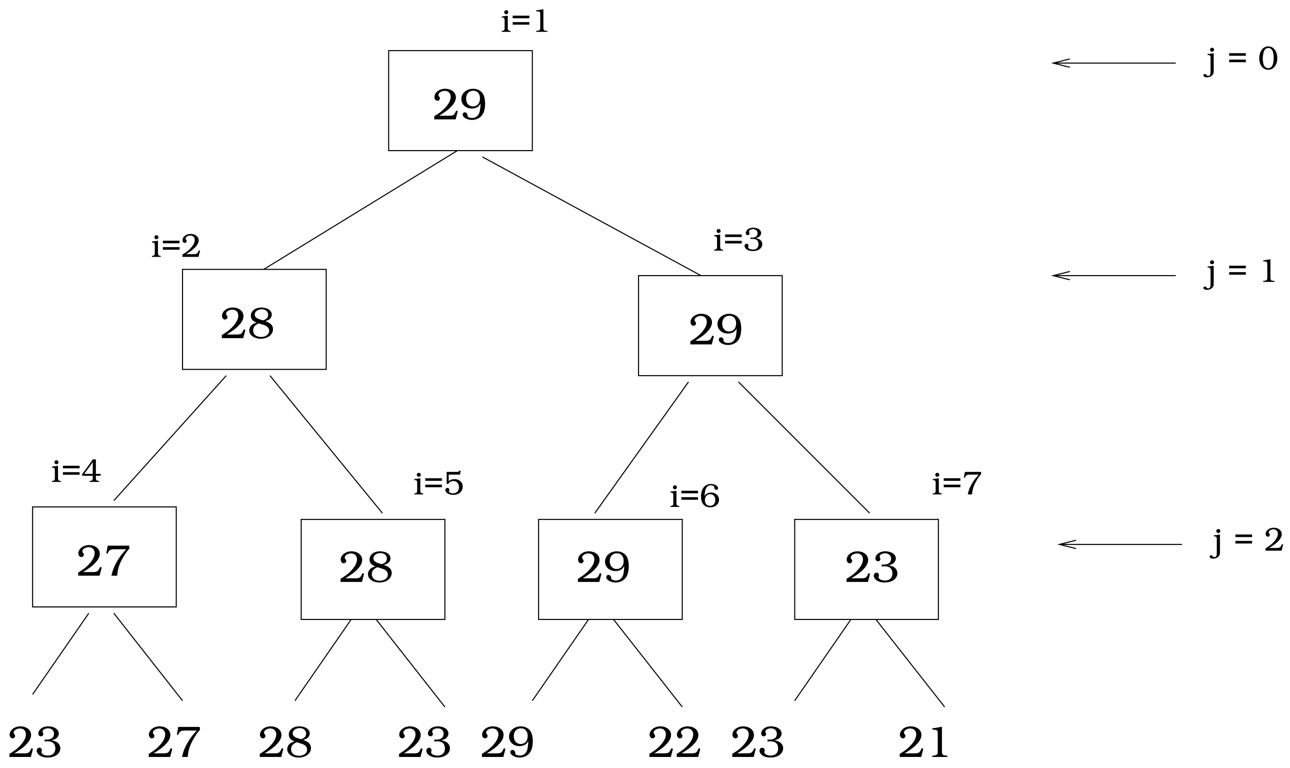
$\triangleright$ Loop sequencial, para cada nível da árvore
para $j := (\log_2 n) - 1$ **até** 0 **faça**
$\triangleright$ Loop paralelo, alocando um processador
$\triangleright$ para cada subproblema deste nível
para $2^j \leq i \leq 2^{j+1} - 1$ **faça em paralelo**
$B[i] := max(B[2i], B[2i+1])$

$maximo := B[1]$

Exemplo: $n = 8 = 2^3$



Exemplo: $n = 8 = 2^3$



Algoritmo para determinar o elemento de valor máximo de um vetor

Submodelos e complexidade

▷ EREW

▷ Tempo: $O(\log_2 n)$ (loop mais externo, número de níveis da árvore)

▷ Processadores: $O(n)$ (número de vértices no último nível da árvore $= n/2$)

▷ Custo: $O(n \log_2 n)$

▷ Eficiente: sim

▷ Ótimo: Não, pois o tempo sequencial é $O(n)$

Algoritmo para determinar o elemento de valor máximo de um vetor

Solução genérica, para qualquer n:

▷ n não é potência de 2
▷ Obter n', menor potência de 2 maior que n: $n' = 2^{\lceil \log_2 n \rceil}$.
▷ Ajeitar o vetor A, preenchendo posições de n a $n' - 1$, com valores neutros em relação à operação a ser realizada.
▷ Neste caso, o valor neutro é qualquer valor que seja menor que o máximo.

Algoritmo para determinar o elemento de valor máximo de um vetor

Algoritmo

$n' := 2^{\lceil \log_2 n \rceil}$
para $n \leq i \leq n' - 1$ **faça em paralelo**
    $A[i] := A[i - n] - 1$

Algoritmo para determinar o elemento de valor máximo de um vetor

Submodelo e Complexidade

- ▷ EREW
- ▷ Tempo: $O(1)$
- ▷ Processadores: $O(n)$
- ▷ Não altera a complexidade do algoritmo original.

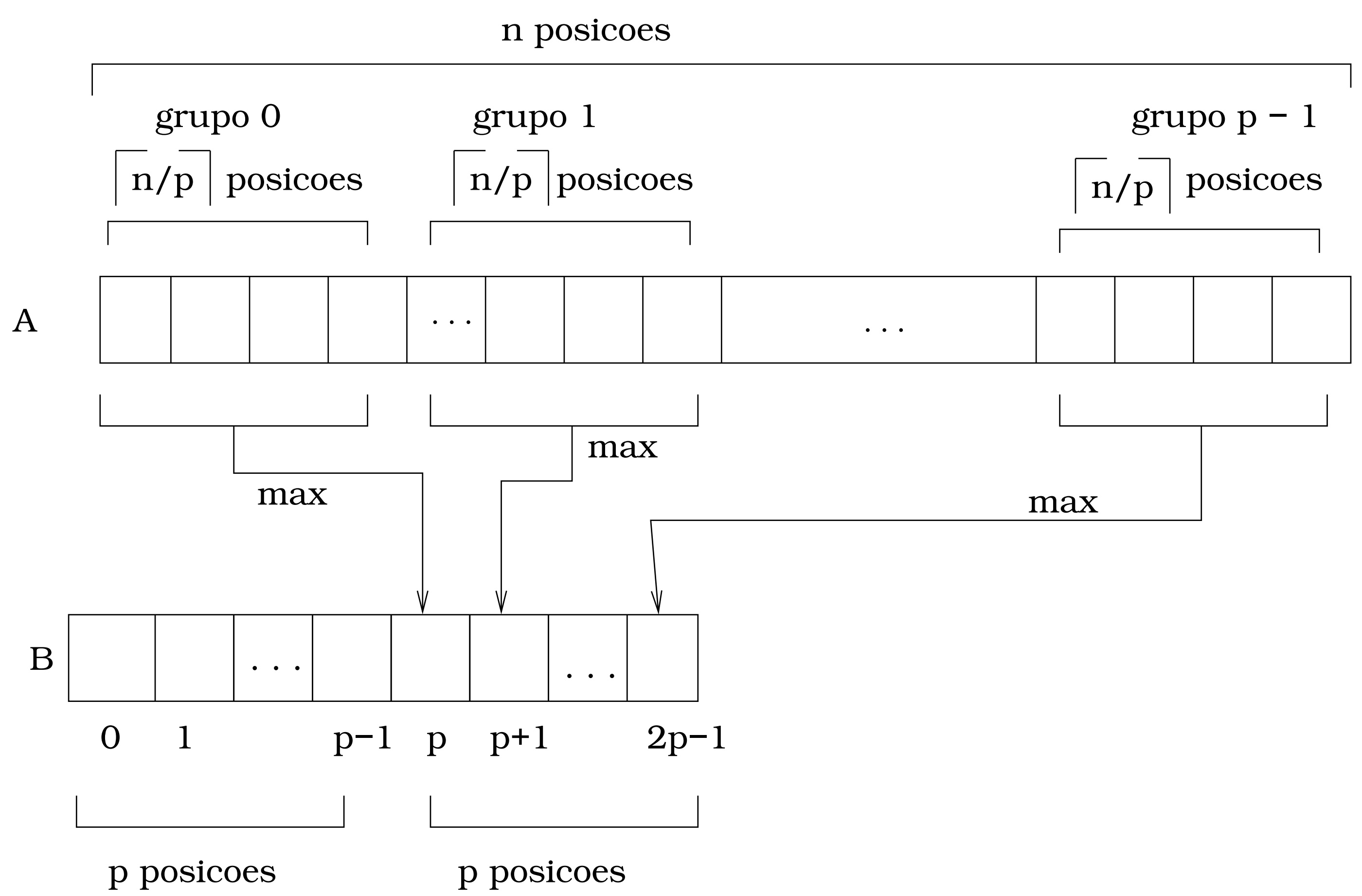
Aplicando o Teorema de Brent

Aplicando o Teorema de Brent

Idéia do algoritmo

- ▷ Utilizar $p < n$ processadores.
- ▷ Os n elementos de A são divididos em p grupos de $\lceil n/p \rceil$ elementos.
- ▷ Cada processador determina o máximo de um grupo, sequencialmente em paralelo com os demais processadores.

Aplicando o Teorema de Brent



Aplicando o Teorema de Brent

$\triangleright$ Para determinar o máximo dos p números resultantes, aplicamos o algoritmo visto, substituindo n por p.

Entrada:

$\triangleright$ n: número de elementos do vetor, n é potência de 2.

$\triangleright$ p: número de processadores

$\triangleright$ A : vetor de n elementos $A[0], \cdots, A[n-1]$

Aplicando o Teorema de Brent

Estrutura auxiliar:

▷ B: vetor com $2 \times p$ posições $B[0], \cdots, B[2 \times p - 1]$

Armazena máximos intermediários. As p posições finais de B terão o elemento máximo de cada grupo de A. Ao final do algoritmo, $B[1]$ conterá o valor do elemento máximo de A

Saída

▷ máximo: valor do elemento máximo de A.

Algoritmo

$\triangleright$ Loop paralelo, alocando um processador para cada grupo de elementos

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

$B[p+i] := A[i \times \lceil n/p \rceil]$

$\triangleright$ Loop sequencial, para determinar o máximo de

$\triangleright$ cada grupo de A e copiá-lo para o vetor B

para $j := 1$ **até** $\lceil n/p \rceil - 1$ **faça**

se $A[i \times \lceil n/p \rceil + j] > B[p+i]$ **então**

$B[p+i] := A[i \times \lceil n/p \rceil + j]$

continuação ...

$\triangleright$ Loop sequencial, para cada nível da árvore

para $j :=$ ($\log_2 p$) $- 1$ **até** 0 **faça**

$\triangleright$ Loop paralelo, alocando um processador

$\triangleright$ para cada subproblema deste nível

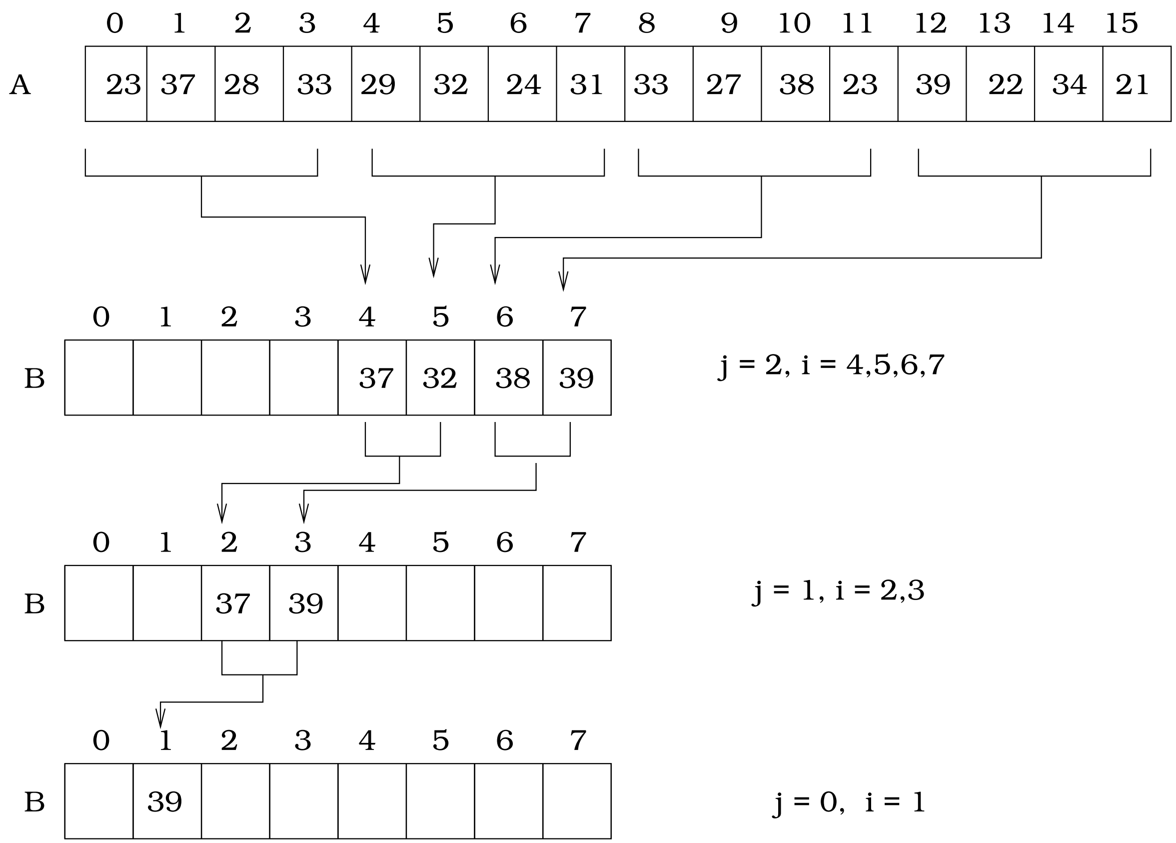
para $2^j \leq i \leq 2^{j+1} - 1$ **faça em paralelo**

$$B[i] := max(B[2i], B[2i+1])$$

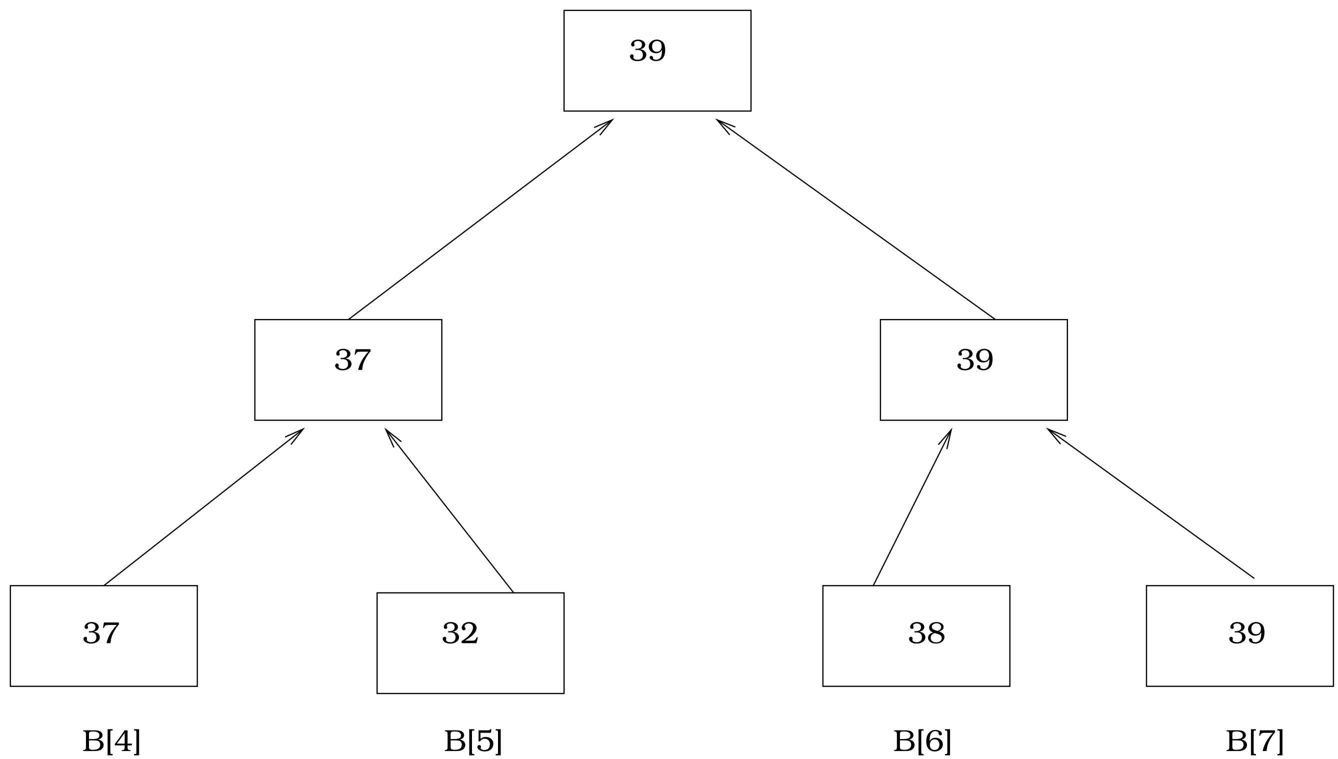
$maximo := B[1]$

Aplicando o Teorema de Brent

Exemplo: $n = 16, p = 4, \lceil n/p \rceil = 4$



Aplicando o Teorema de Brent



Aplicando o Teorema de Brent

Submodelo: EREW

Complexidades:

▷ Tempo: $O(\lceil n/p \rceil + \log_2 p)$
▷ Processadores: $O(p)$

Aplicando o Teorema de Brent

Fazendo $p = \dfrac{n}{\log_2 n}$

Complexidades:

▷ Tempo: $O\left(\left\lceil \dfrac{n}{\frac{n}{\log_2 n}} \right\rceil + \log_2 \dfrac{n}{\log_2 n}\right) = O\left(\log_2 n + \log_2 \dfrac{n}{\log_2 n}\right)$

▷ Tempo: $O(\log_2 n)$

▷ Processadores: $O\left(\dfrac{n}{\log_2 n}\right)$

▷ Custo: $O(n)$

▷ É eficiente e ótimo.

Máximo de um vetor para CRCW Forte

Máximo de um vetor para CRCW Forte

para $0 \leq i \leq n-1$ **faça em paralelo**
$maximo := A[i]$

- ▷ Tempo: $O(1)$
- ▷ Processadores: $O(n)$

Máximo de um vetor para CRCW Fraco

Máximo de um vetor para CRCW Fraco

Algoritmo

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

para $0 \leq i, j \leq n-1$, $i < j$ **faça em paralelo**
    se $A[i] < A[j]$ **então**
        $F[i] := 0$
    senão se $A[i] > A[j]$ **então**
        $F[j] := 0$
    senão
        $F[j] := 0$

para $0 \leq i \leq n-1$ **faça em paralelo**
    se $F[i] = 1$ **então**
        $maximo := A[i]$

Máximo de um vetor para CRCW Fraco

Exemplo: $n = 3$

A	23	27	28	23	29	22	25	21

F	1	1	1	1	1	1	1	1

F	0	0	0	0	1	0	0	0

Maximo = 29

Máximo de um vetor para CRCW Fraco

- ▷ tempo : $O(1)$
- ▷ processadores: $O(n^2)$
- ▷ custo: $O(n^2)$

É eficiente, não é ótimo.

Comparação dos Algoritmos

Modelo:	tempo	processadores	eficiente	ótimo
CRCW Forte	$O(1)$	$O(n)$	Sim	Sim
CRCW Fraco	$O(1)$	$O(n^2)$	Sim	Não
EREW	$O(\log_2 n)$	$O(n)$	Sim	Não
EREW com Teorema de Brent	$O(\log_2 n)$	$O(n/\log_2 n)$	Sim	Sim

Exemplos Análogos

▷ Algoritmo para determinar o elemento de valor mínimo de um vetor.

  ▷ Valor neutro: $A[i]+1$

▷ Algoritmo para determinar a soma dos elementos de um vetor.

  ▷ Valor neutro: 0

▷ Algoritmo paralelo para determinar o índice do elemento de valor máximo (ou mínimo) de um vetor.

Computação de Prefixos/Sufixos

Computação de Prefixos/Sufixos

Dado um vetor A de n elementos $A[0], \cdots, A[n-1]$, a computação de prefixos calcula valores

```
A[0]
A[0] op A[1]
A[0] op A[1] op A[2]
...
A[0] op A[1] ... op A[n-1]
```

onde `op` é uma operação binária associativa.

Computação de Prefixos/Sufixos

Este método utiliza a árvore binária balanceada em 2 passos.

▷ **passo 1:** sobe na árvore equivalente ao método da árvore binária balanceada.

Computação de Prefixos/Sufixos

▷ **passo 2:** desce na árvore calculando as operações `op` parciais.

Soma de Prefixos

Soma de Prefixos

- ▷ Exemplo: op é a adição
- ▷ Vetor de entrada:
  $[3, 7, 5, 1, 2, 4, 0, 9]$
- ▷ Vetor de resultante:
  $[3, 10, 15, 16, 18, 22, 22, 31]$

Soma de Prefixos - **Algoritmo Sequencial**

$\triangleright$ passo 1: out[0] recebe a soma de prefixos de in[0].

$soma$:= in[0]

out[0] := $soma$

$\triangleright$ passo 2: Calcule os demais prefixo.

para $1 \leq i \leq n - 1$ **faça**

$\triangleright$ passo 2.1: A soma do i-ésimo prefixo é a i-ésima posição

$\triangleright$ do vetor de entrada mais soma.

$\triangleright$ Soma contém a soma do (i-ésimo)-1 prefixo.

$out[i] := in[i] + soma$

$soma := out[i]$

Complexidade: $O(n)$

Soma de Prefixos

▷ **passo 1:** equivalente a técnica da árvore binária balanceada.

▷ **passo 2:** desce na árvore subtraindo alguns elementos das somas intermediárias para outras somas de prefixos. Quando desce para o filho esquerdo subtrai o valor do filho direito, e quando desce para o filho direito passa o valor direto

Soma de Prefixos

Entrada:

▷ n: número de elementos (n potência de 2, senão, ajeitar..)
▷ A: vetor de n elementos

Soma de Prefixos

Estrutura auxiliar:

- ▷ B: vetor de $2 \times n$ elementos
- ▷ As n posições finais de B terão cópias de A

Soma de Prefixos

Saída:

▷ P vetor de $2 \times n$ posições

▷ As n posições finais de P terão somas de prefixos

Soma de Prefixos

Algoritmo

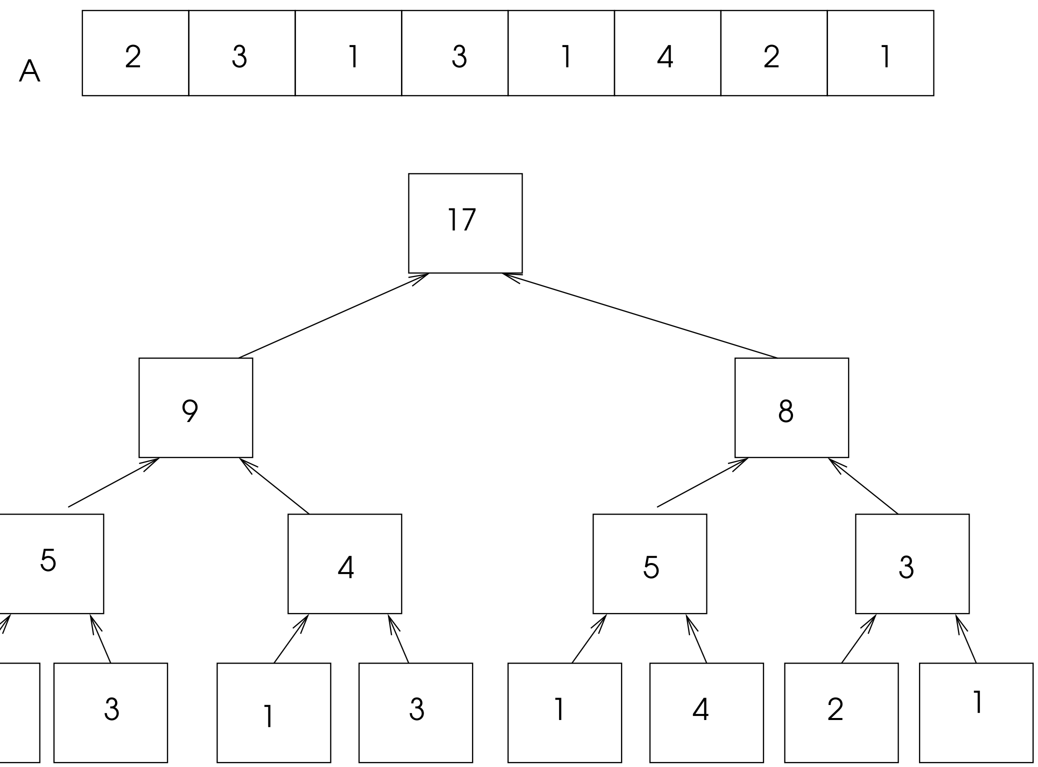
$\triangleright$ Vetor A é copiado para a segunda metade de B

para $0 \leq i \leq n-1$ **faça em paralelo**
$B[n+i] := A[i]$

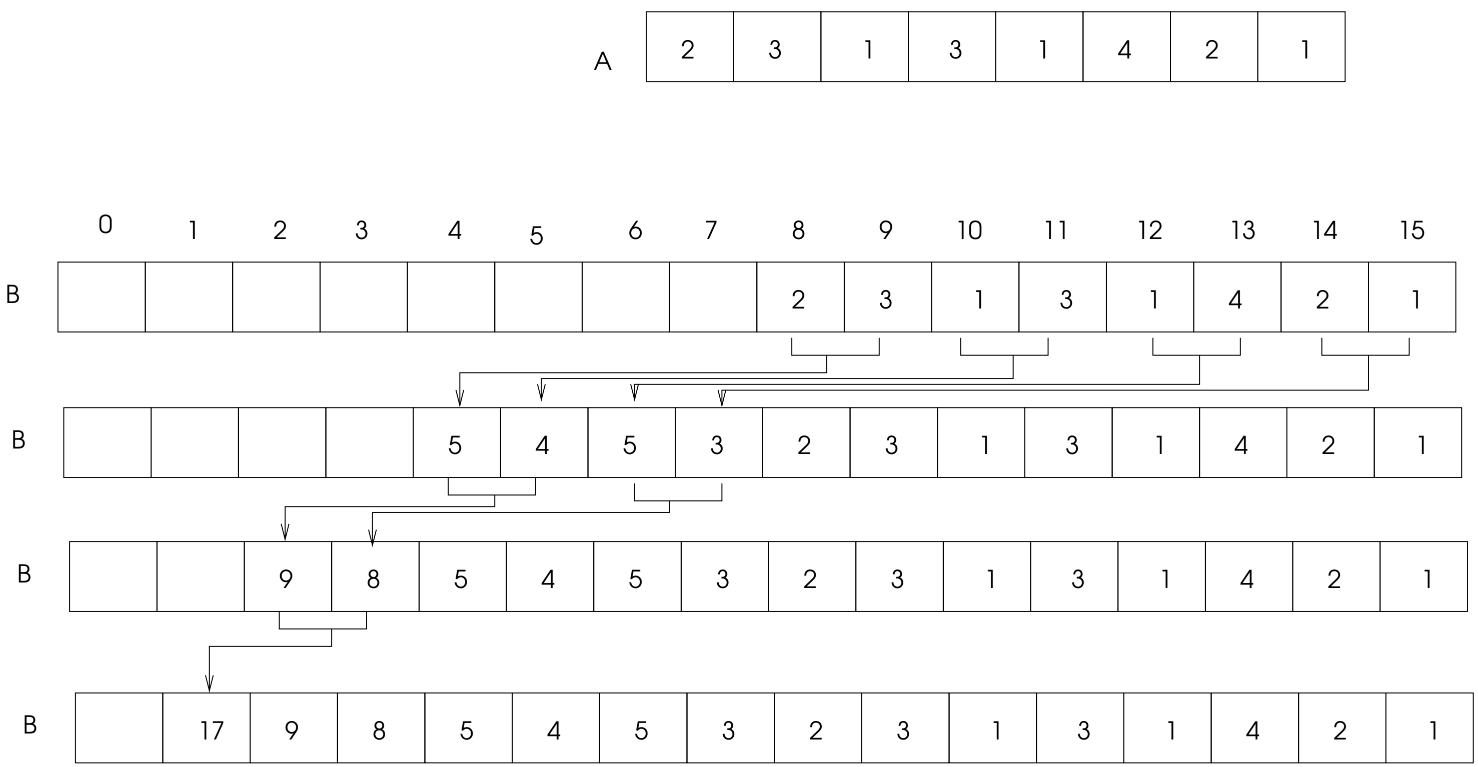
$\triangleright$ passo 1: subida na árvore

para $j := (\log_2 n) - 1$ **até** 0 **faça**
para $2^j \leq i \leq 2^{j+1} - 1$ **faça em paralelo**
$B[i] := B[2 \times i] + B[2 \times i + 1]$

Subida na árvore



Subida na árvore



$\triangleright$ passo 2: descida na árvore

$P[1] := B[1]$

$\triangleright$ loop sequencial para cada nível da árvore

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

  $\triangleright$ loop paralelo alocando um processador para

  $\triangleright$ cada vértice deste nível

  para $2^j \leq i \leq 2^{j+1} - 1$ **faça em paralelo**

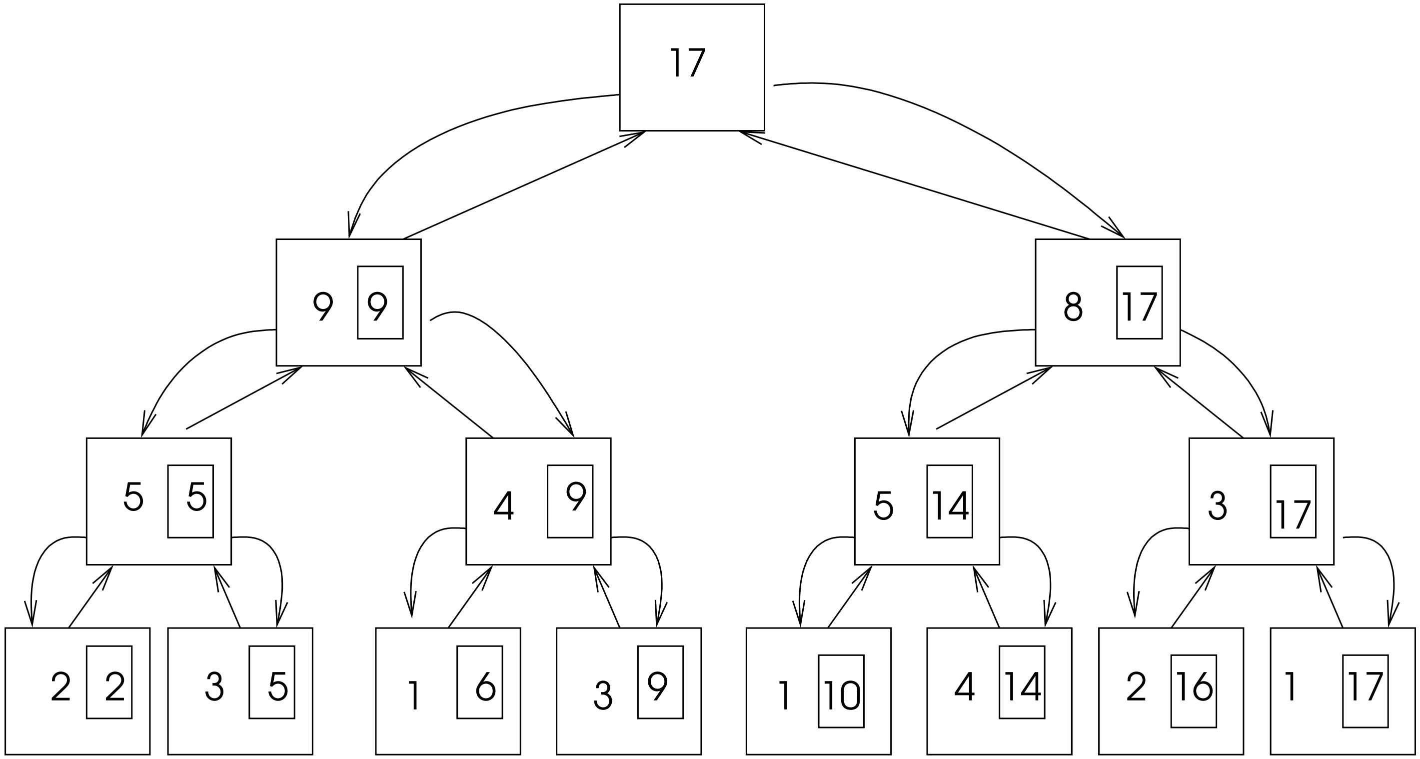
    se $i \ mod \ 2 = 0$ **então**

      $P[i] := P[i/2] - B[i+1]$ $\triangleright$ filho esquerdo

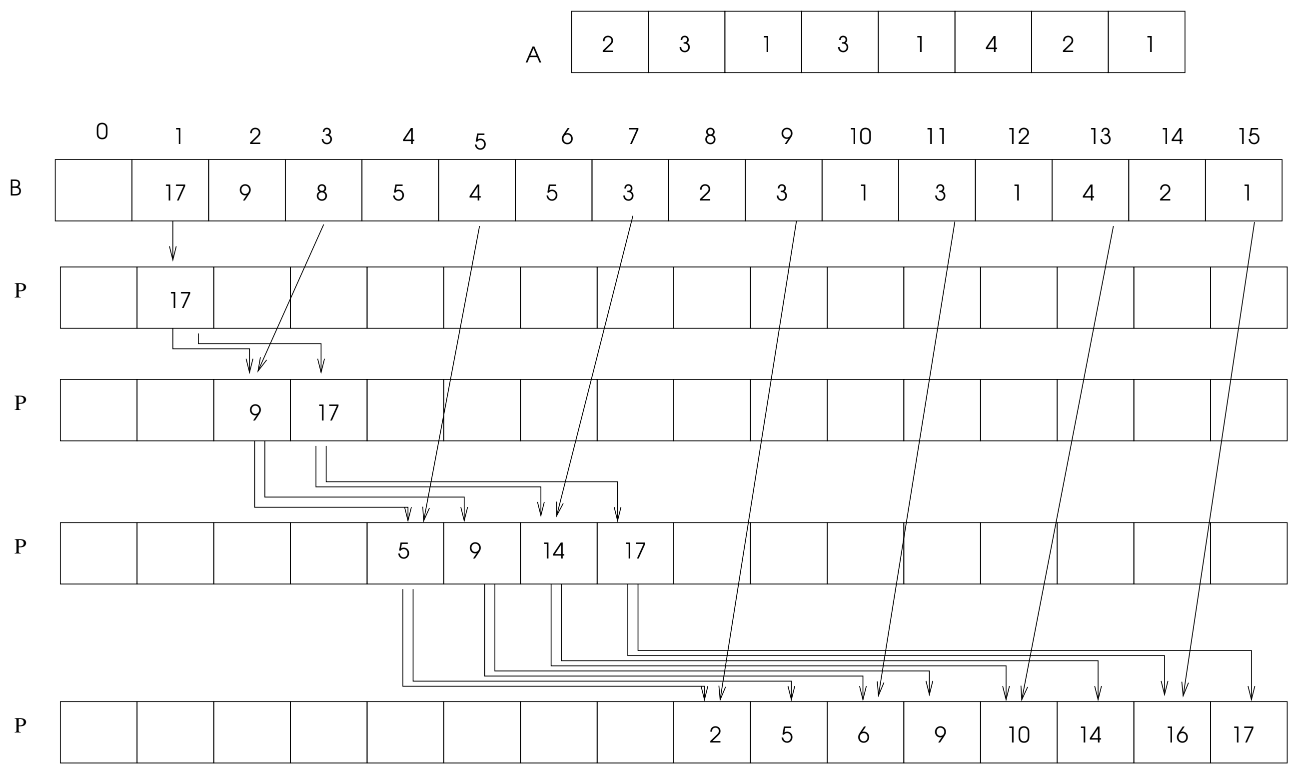
    senão

      $P[i] := P[(i-1)/2]$ $\triangleright$ filho direito

Descida na árvore



Descida na árvore



Soma de Prefixos

Submodelo: EREW

Complexidade

tempo: $O(\log_2 n)$

processadores: $O(n)$

custo: $O(n \log_2 n)$

É eficiente. Não é ótimo

Soma de Prefixos

Aplicando o Teorema de Brent

processadores: $O(n/\log_2 n)$

tempo: $O(\log_2 n + \log_2 p) = O(\log_2 n)$

custo: $O(n)$

É eficiente. É ótimo

Avaliação de Polinômios

Avaliação de Polinômios

Polinômio de grau $n-1$:

$$P(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \cdots + c_{n-1} x^{n-1}$$

Avaliação de Polinômios

Algoritmo sequencial:

para $i := 0$ **até** $n - 1$ **faça**
    $P := P + c[i] * x^i$

Problema: Executa muitas exponenciações!

Avaliação de Polinômios

Algoritmo sequencial mais eficiente:

para $i := n - 1$ **até** 1 **faça**
    $P := (P + c[i]) * x$

$P := P + c[0]$

Mesma complexidade de tempo, mas não faz exponenciação.

Avaliação de Polinômios

Polinômio de grau $n-1$:

$$P(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \cdots + c_{n-1} x^{n-1}$$

Idéia: Representar $P(x)$ da forma:

$$P(x) = R(x) + Q(x) \times x^{n/2}$$

onde $R(x)$ e $Q(x)$ são polinômios de graus $\frac{n}{2} - 1$

Avaliação de Polinômios

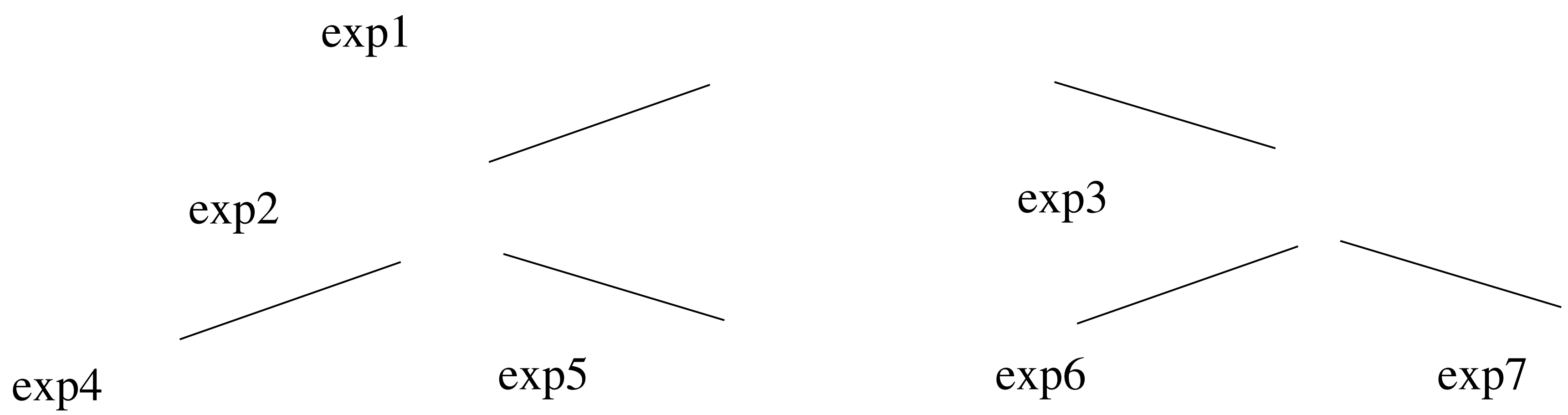
Recursivamente aplicamos a mesma idéia para determinar $R(x)$ e $Q(x)$, e assim sucessivamente até chegarmos a polinômios de grau 1, que são a base da recursão.

Ex.: $n = 8$

$c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5 + c_6x^6 + c_7x^7$

Avaliação de Polinômios

Ex.: $n = 8$



Avaliação de Polinômios

Entrada:

- n: Supomos que n é potência de 2.
- c : Vetor de n coeficientes $c[0], \cdots, c[n-1]$
- x: Variável do polinômio.

Saída

- $resultado$: resultado do polinômio.

Algoritmo

$$resultado := avalia(0, n-1)$$

função avalia(i, j:inteiros)

 variáveis locais: a, b

 se $i = j - 1$ **então**

  retorna $c[i] + c[j]x$

 senão

  para $0 \leq k \leq 1$ **faça em paralelo**

   se $k = 0$ **então**

    $a := avalia(i, \frac{i+j-1}{2})$

   senão

    $b := avalia(\frac{i+j+1}{2}, j)$

  retorna $a + bx^{\left\lceil \frac{j-i}{2} \right\rceil}$

fim

Avaliação de Polinômios

Submodelo: EREW

Complexidades:

- ▷ Tempo: $O(\log_2 n)$
- ▷ Processadores: $O(n)$
- ▷ Custo: $O(n \log_2 n)$

É eficiente. Não é ótimo.

Podemos aplicar o teorema de Brent para obter o algoritmo ótimo.

Fim