

Programação Vetorial

Modificação de *rank*, ordenação e reduções

Prof. Edson Alves

Faculdade UnB Gama

Sumário

1. Modificação do *rank* e da profundidade
2. Ordenação
3. Reduções e varreduras

Mistura

- ▶ A função monádica \uparrow (*mix*, *mistura*) permite a criação de *arrays* de *rank* maior

Mistura

- ▶ A função monádica \uparrow (*mix*, *mistura*) permite a criação de *arrays* de *rank* maior
- ▶ Ela rearranja os elementos de um *array* em um novo *array* de *rank* uma unidade maior, com um nível a menos de aninhamento

```
      ↑(2 3 5) (7 9 11)
2 3 5
7 9 11
```

Mistura

- ▶ A função monádica \uparrow (*mix*, mistura) permite a criação de *arrays* de *rank* maior
- ▶ Ela rearranja os elementos de um *array* em um novo *array* de *rank* uma unidade maior, com um nível a menos de aninhamento

```
      ↑(2 3 5) (7 9 11)
2 3 5
7 9 11
```

- ▶ Se os elementos tem formas distintas, serão utilizados elementos para preencher os espaços em branco, de acordo com o tipo do primeiro componente do elemento

Mistura

- ▶ A função monádica \uparrow (*mix*, mistura) permite a criação de *arrays* de *rank* maior
- ▶ Ela rearranja os elementos de um *array* em um novo *array* de *rank* uma unidade maior, com um nível a menos de aninhamento

```

      ↑(2 3 5) (7 9 11)
2 3 5
7 9 11

```

- ▶ Se os elementos tem formas distintas, serão utilizados elementos para preencher os espaços em branco, de acordo com o tipo do primeiro componente do elemento
- ▶ Em geral, inteiros são preenchidos com zero e caracteres com o espaço em branco

```

      ↑(2 3) 5 (7 9 11) ♦ ↑'string' 'abc'
2 3 0
5 0 0
7 9 11
string
abc

```

Novo símbolo

Símbolo	Aridade	Descrição
\uparrow (<i>mix</i>)	monádico	Rearranja os elementos do <i>array</i> em um novo <i>array</i> com <i>rank</i> uma unidade maior
Unicode	TAB	APL
U+2191	\wedge <code><tab></code>	APL + <i>y</i>

Cisão

- ▶ A função monádica \downarrow (*split*, *cisão*) gera, a partir do argumento, um novo *array* aninhado, com *rank* uma unidade menor do que o argumento

Cisão

- ▶ A função monádica \downarrow (*split*, *cisão*) gera, a partir do argumento, um novo *array* aninhado, com *rank* uma unidade menor do que o argumento
- ▶ Caso o argumento seja escalar, o retorno terá *rank* zero

Cisão


- ▶ A função monádica \downarrow (*split*, *cisão*) gera, a partir do argumento, um novo *array* aninhado, com *rank* uma unidade menor do que o argumento
- ▶ Caso o argumento seja escalar, o retorno terá *rank* zero
- ▶ Se o argumento tem forma $d_1 d_2 \dots d_N$, então o retorno terá forma $d_1 d_2 \dots d_{N-1}$, cujos elemento de profundidade 1 tem forma d_N

```

  A ← 2 3 ρ 2 3 5 7 9 11
    ρ A
2 3
  ↓A
2 3 5 7 9 11
  ρ ↓A
2
  ρρ A ♦ ρρ ↓A
2
1

```

Novo símbolo

Símbolo	Aridade	Descrição
 (<i>split</i>)	monádico	Rearranja os elementos do <i>array</i> em um novo <i>array</i> com <i>rank</i> uma unidade menor
Unicode	TAB	APL
U+2193	v <tab>	APL + u

Vetores inclusos

- ▶ Aplicar a cisão em um vetor \mathbf{v} (*array* de dimensão 1) resultará em um *array* de dimensão zero cujo conteúdo é próprio \mathbf{v}

```

      ↓ 2 3 5 7 11
2 3 5 7 11
  ρ ↓ 2 3 5 7 11

    ρρ ↓ 2 3 5 7 11
0
```

Vetores inclusos

- ▶ Aplicar a cisão em um vetor \mathbf{v} (*array* de dimensão 1) resultará em um *array* de dimensão zero cujo conteúdo é próprio \mathbf{v}

```

      ↓ 2 3 5 7 11
2 3 5 7 11
  ρ ↓ 2 3 5 7 11

    ρρ ↓ 2 3 5 7 11
0
```

- ▶ Neste caso, o retorno de $\downarrow \mathbf{v}$ tem a mesma forma do vetor θ

Vetores inclusos

- ▶ Aplicar a cisão em um vetor \mathbf{v} (*array* de dimensão 1) resultará em um *array* de dimensão zero cujo conteúdo é próprio \mathbf{v}

```

      ↓ 2 3 5 7 11
2 3 5 7 11
  ρ ↓ 2 3 5 7 11

    ρρ ↓ 2 3 5 7 11
0
```

- ▶ Neste caso, o retorno de $\downarrow \mathbf{v}$ tem a mesma forma do vetor θ
- ▶ Este retorno é denominado vetor incluso (*enclosed vector*) ou escalar aninhado (*nested scalar*)

Escalares aninhados

- ▶ Aplicar uma cisão em um escalar simples (profundidade zero) produz o próprio escalar

$$\begin{array}{c} \rho \rho \downarrow 2 \diamond \equiv \downarrow 2 \\ 0 \\ 0 \end{array}$$

Escalares aninhados

- ▶ Aplicar uma cisão em um escalar simples (profundidade zero) produz o próprio escalar

$$\begin{array}{c} \rho \rho \downarrow 2 \diamond \equiv \downarrow 2 \\ 0 \\ 0 \end{array}$$

- ▶ Aplicar uma cisão em um escalar aninhado aumenta sua profundidade em uma unidade

$$\begin{array}{c} \equiv 2 \ 3 \ 5 \ 7 \ 11 \diamond \equiv \downarrow\downarrow 2 \ 3 \ 5 \ 7 \ 11 \\ 1 \\ 3 \end{array}$$

Escalares aninhados

- ▶ Aplicar uma cisão em um escalar simples (profundidade zero) produz o próprio escalar

$$\begin{array}{c} \rho \rho \downarrow 2 \diamond \equiv \downarrow 2 \\ 0 \\ 0 \end{array}$$

- ▶ Aplicar uma cisão em um escalar aninhado aumenta sua profundidade em uma unidade

$$\begin{array}{c} \equiv 2 \ 3 \ 5 \ 7 \ 11 \diamond \equiv \downarrow\downarrow 2 \ 3 \ 5 \ 7 \ 11 \\ 1 \\ 3 \end{array}$$

- ▶ Aplicar uma mistura em um escalar aninhado reduz sua profundidade em uma unidade

$$\begin{array}{c} \equiv \uparrow\downarrow\downarrow 2 \ 3 \ 5 \ 7 \ 11 \\ 2 \end{array}$$

Inclusão

- A função monádica \hookleftarrow (*enclose*) gera um escalar a partir de qualquer *array*, incluindo-o em um vetor aninhado (escalar incluso)

```

      ← 2 2 ρ 2 3 5 7
2 3
5 7
      ρρ ← 2 2 ρ 2 3 5 7
0

```

Inclusão

- ▶ A função monádica \hookleftarrow (*enclose*) gera um escalar a partir de qualquer *array*, incluindo-o em um vetor aninhado (escalar incluso)

```

      ↵ 2 2 ρ 2 3 5 7
2 3
5 7
      ρρ ↵ 2 2 ρ 2 3 5 7
0

```

- ▶ A inclusão é usada principalmente em funções escalares, ou que tratam os escalares de forma diferenciada

```

      2 3 5 + 7 11 13
9 14 18
      2 3 5 + ↵ 7 11 13
9 13 15    10 14 16    12 16 18

```

Novo símbolo

Símbolo	Aridade	Descrição
⊂ (<i>enclose</i>)	monádico	Retorna um escalar incluso que contém o <i>array</i> passado como parâmetro

Unicode	TAB	APL
U+2282	c c <tab>	APL + z

Extração

- ▶ A função monádica \triangleright (*disclose*, *first*) faz o trabalho inverso da inclusão

Extração

- ▶ A função monádica \triangleright (*disclose, first*) faz o trabalho inverso da inclusão
- ▶ Quando aplicada em um vetor aninhado, ela desfaz a inclusão e retorna o vetor

```

       $\triangleright c$  2 3 5 7
2 3 5 7
       $\rho\rho$   $\triangleright c$  2 3 5 7
1
```

Extração

- ▶ A função monádica \triangleright (*disclose*, *first*) faz o trabalho inverso da inclusão
- ▶ Quando aplicada em um vetor aninhado, ela desfaz a inclusão e retorna o vetor

```

       $\triangleright c$  2 3 5 7
2 3 5 7
       $\rho\rho$   $\triangleright c$  2 3 5 7
1

```


- ▶ Se for aplicada em um *array*, ela extrai o primeiro elemento do nível de profundidade 1

```

       $\triangleright$  2 3 5 7
2
       $\triangleright$  ((2 3 5) 7 ((11 13) (17 19 23)))
2 3 5

```

Novo símbolo

Símbolo	Aridade	Descrição
 (<i>disclose, first</i>)	monádico	Retorna o primeiro elemento no nível de profundidade 1
Unicode	TAB	APL
U+2283)) <tab>	APL + x

Ranqueamento ascendente

- ▶ APL não disponibiliza uma primitiva para a ordenação dos elementos de um *array*

Ranqueamento ascendente

- ▶ APL não disponibiliza uma primitiva para a ordenação dos elementos de um *array*
- ▶ Para ordenar um vetor é preciso recorrer as funções de ranqueamento

Ranqueamento ascendente

- ▶ APL não disponibiliza uma primitiva para a ordenação dos elementos de um *array*
- ▶ Para ordenar um vetor é preciso recorrer as funções de ranqueamento
- ▶ A função monádica \uparrow (*grade up*) ranqueia um vetor ascendentemente

Ranqueamento ascendente

- ▶ APL não disponibiliza uma primitiva para a ordenação dos elementos de um *array*
- ▶ Para ordenar um vetor é preciso recorrer as funções de ranqueamento
- ▶ A função monádica \uparrow (*grade up*) ranqueia um vetor ascendentemente
- ▶ Ela retorna um vetor de índices, cujo i -ésimo elemento indica o índice do i -ésimo menor elemento do argumento

$\mathbf{a} \leftarrow 6 \ 9 \ 3 \ 1 \ 4 \ 7 \ 1 \ 8 \ 0$
 $\uparrow \mathbf{a}$
 9 4 7 3 5 1 6 8 2

\mathbf{a} dígitos da expansão decimal de $\ln 2$

Ranqueamento ascendente

- ▶ APL não disponibiliza uma primitiva para a ordenação dos elementos de um *array*
- ▶ Para ordenar um vetor é preciso recorrer as funções de ranqueamento
- ▶ A função monádica \uparrow (*grade up*) ranqueia um vetor ascendentemente
- ▶ Ela retorna um vetor de índices, cujo i -ésimo elemento indica o índice do i -ésimo menor elemento do argumento


$a \leftarrow 6 \ 9 \ 3 \ 1 \ 4 \ 7 \ 1 \ 8 \ 0$
 $\uparrow a$
 $9 \ 4 \ 7 \ 3 \ 5 \ 1 \ 6 \ 8 \ 2$

A dígitos da expansão decimal de $\ln 2$

- ▶ A ordenação de um vetor pode ser obtida por meio da expressão $a[\uparrow a]$

$a[\uparrow a]$
 $0 \ 1 \ 1 \ 3 \ 4 \ 6 \ 7 \ 8 \ 9$

Novo símbolo

Símbolo	Aridade	Descrição
 (<i>grade up</i>)	monádico	Ranqueia o argumento de forma ascendente
Unicode	TAB	APL
U+234B	A <tab>	APL + Shift + 4

Novo símbolo

Símbolo	Aridade	Descrição
$[\]$ (<i>square brackets</i>)	-	$v[u]$ retorna os elementos do vetor v que ocupam os índices indicado pelo vetor u

Unicode	TAB	APL
U+005[BD]	-	-

Ordenação lexicográfica

- A ordenação apresentada ordena os elementos de um *array* segundo a ordem lexicográfica

A ← 5 3 ρ a

A

6 9 3

1 4 7

1 8 0

6 9 3

1 4 7

⌕ A

2 5 3 1 4

A Ordem lexicográfica das linhas

Ordenação lexicográfica

- ▶ A ordenação apresentada ordena os elementos de um *array* segundo a ordem lexicográfica

A ← 5 3 ρ a
A

6 9 3

1 4 7

1 8 0

6 9 3

1 4 7

⌈A

2 5 3 1 4

A Ordem lexicográfica das linhas

- ▶ A função diádica \lceil (*index*) permite, em conjunto com uma inclusão, ordenar um *array* de forma arbitrária


sortA ← {(c⌈ω)⌈ω}

A Ordenação ascendente (dfn)

sortB ← (c⌈)⌈-

A Ordenação ascendente (trem)

Novo símbolo

Símbolo	Aridade	Descrição
 (<i>index</i>)	diádico	Retorna os índices do argumento à direita indicados pelo escalar à esquerda
Unicode	TAB	APL
U+2337	[<tab>	APL + Shift + l

Ranqueamento descendente

- ▶ A função monádica Ψ (*grade down*) retorna o ranqueamento descendente de seu argumento

```
a ← 6 9 3 1 4 7 1 8 0
Ψ a
2 8 6 1 5 3 4 7 9
```

Ranqueamento descendente

- ▶ A função monádica Ψ (*grade down*) retorna o ranqueamento descendente de seu argumento

```

a ← 6 9 3 1 4 7 1 8 0
Ψa
2 8 6 1 5 3 4 7 9

```


- ▶ Strings são ordenadas de acordo com os valores dos caracteres na tabela Unicode

```

sortDesc ← (<Ψ)[]
sortDesc 'abacaxi' 'abobora' 'abacate'
abobora abacaxi abacate

```

Novo símbolo

Símbolo	Aridade	Descrição
 (<i>grade down</i>)	monádico	Ranqueia o argumento de forma descendente

Unicode	TAB	APL
U+2352	V <tab>	APL + Shift + 3

Aplicações do ranqueamento

- ▶ É possível obter o índice do menor ou do maior elemento de um *array* usando os *atops* $\triangleright \Delta$ e $\triangleright \Psi$, respectivamente

```

a ← 6 9 3 1 4 7 1 8 0
 $\triangleright \Delta$  a
9
 $\triangleright \Psi$  a
2

```

Aplicações do ranqueamento

- É possível obter o índice do menor ou do maior elemento de um *array* usando os *atops* $\triangleright \Phi$ e $\triangleright \Psi$, respectivamente

```

a ← 6 9 3 1 4 7 1 8 0
 $\triangleright \Phi$  a
9
 $\triangleright \Psi$  a
2
```

- O duplo ranqueamento permite obter as posições que cada elemento do *array* ocupará após a ordenação

```

 $\Phi$  a
9 4 7 3 5 1 6 8 2
 $\Phi \Phi$  a
6 9 4 2 5 7 3 8 1
```

Ranqueamentos diádicos

- ▶ As funções de ranqueamento também tem formas diádicas

Ranqueamentos diádicos

- ▶ As funções de ranqueamento também tem formas diádicas
- ▶ O parâmetro à esquerda indicará o alfabeto α que será utilizado como critério de ordenação

Ranqueamentos diádicos

- ▶ As funções de ranqueamento também tem formas diádicas
- ▶ O parâmetro à esquerda indicará o alfabeto α que será utilizado como critério de ordenação
- ▶ No exemplo a seguir os caracteres ímpares devem anteceder os caracteres pares na ordenação

```

n ← '693147180'
o ← '1357902468'  ⧻ n
o
4 7 3 6 2 9 5 1 8
n[o]
113790468

```

Ranqueamentos diádicos

- ▶ As funções de ranqueamento também tem formas diádicas
- ▶ O parâmetro à esquerda indicará o alfabeto α que será utilizado como critério de ordenação
- ▶ No exemplo a seguir os caracteres ímpares devem anteceder os caracteres pares na ordenação


```

n ← '693147180'
o ← '1357902468' ⧻ n
o
4 7 3 6 2 9 5 1 8
n[o]
113790468


```

- ▶ Caracteres que não forem indicado no alfabeto serão considerados equivalentes, ocupando a posição logo após o último caractere indicado

Novo símbolo

Símbolo	Aridade	Descrição
 (<i>grade up</i>)	diádico	Ranqueia o argumento de forma ascendente, de acordo com o alfabeto indicado
Unicode	TAB	APL
U+2352	V <tab>	APL + Shift + 4

Novo símbolo

Símbolo	Aridade	Descrição
 (<i>grade down</i>)	diádico	Ranqueia o argumento descendentemente, de acordo com o alfabeto indicado
Unicode	TAB	APL
U+2352	V <tab>	APL + Shift + 3

Reduções

- ▶ Em APL, um operador recebe um ou dois operandos (geralmente funções) como argumentos e retorna uma função (monádica ou diádica)

Reduções

- ▶ Em APL, um operador recebe um ou dois operandos (geralmente funções) como argumentos e retorna uma função (monádica ou diádica)
- ▶ O operador */* (*reduce*) é monádico e retorna uma função ambivalente (que pode ser usada de forma monádica ou diádica)

Reduções

- ▶ Em APL, um operador recebe um ou dois operandos (geralmente funções) como argumentos e retorna uma função (monádica ou diádica)
- ▶ O operador */* (*reduce*) é monádico e retorna uma função ambivalente (que pode ser usada de forma monádica ou diádica)
- ▶ O nome diz respeito ao fato de que a função resultante reduz o *rank* do argumento em 1 unidade

Reduções

- ▶ Em APL, um operador recebe um ou dois operandos (geralmente funções) como argumentos e retorna uma função (monádica ou diádica)
- ▶ O operador `/` (*reduce*) é monádico e retorna uma função ambivalente (que pode ser usada de forma monádica ou diádica)
- ▶ O nome diz respeito ao fato de que a função resultante reduz o *rank* do argumento em 1 unidade
- ▶ A redução de um vetor é direta: `F/a b c d e ...` equivale a `c a F b F c F d F e F ...`, onde o resultado tem a mesma forma do argumento, exceto o último eixo

```

+ / 2 3 5 7 11
28
+ / (2 3 5) (7 11 13)
9 14 18

```

Novo símbolo

Símbolo	Aridade	Descrição
\diagup (<i>reduce</i>)	operador	$F \diagup v_1 \ v_2 \ v_3 \ \dots \equiv c \ v_1 \ F \ v_2 \ F \ v_3 \ F \ \dots$
Unicode	TAB	APL
U+002F	-	-

Características da redução

- ▶ A ordem de precedência das funções em APL (associativa à direita) afeta o comportamento da redução:

`- / 2 3 5 7 11`

⌈ Soma alternada

8

`⊗ / 2 3 5`
0.5508745883

⌈ ⊗ é a função logaritmo

Características da redução

- ▶ A ordem de precedência das funções em APL (associativa à direita) afeta o comportamento da redução:

```

      - / 2 3 5 7 11      A Soma alternada
8
      * / 2 3 5          A * é a função logaritmo
0.5508745883

```

- ▶ Em *arrays* com *rank* maior do que um, a redução se aplica sempre na última dimensão

```

      A ← 2 3 ρ 16 ⋄ A
1 2 3
4 5 6
      × / A
6 120
      ρ A ⋄ ρ × / A
2 3
2

```

Novo símbolo

Símbolo	Aridade	Descrição
\otimes (<i>logarithm</i>)	diádico	Computa $\log_{\alpha} \omega$
Unicode	TAB	APL
U+235F	* O <tab>	APL + Shift + *

Novo símbolo

Símbolo	Aridade	Descrição
\otimes (<i>logarithm</i>)	monádico	Computa $\ln \omega$

Unicode	TAB	APL
U+235F	* O <tab>	APL + Shift + *

Redução nas demais dimensões

- ▶ O operador $\times \text{ / } A$ (*reduce first*) produz uma redução em relação à primeira dimensão

$\times \text{ / } A$
4 10 18

Redução nas demais dimensões

- ▶ O operador $\times \text{ / } A$ (*reduce first*) produz uma redução em relação à primeira dimensão

$\times \text{ / } A$
4 10 18

- ▶ Para produzir uma redução em relação a k -ésima dimensão, use a notação $f \text{ / } [k]$

$A \leftarrow 2 \ 3 \ 5 \ 9 \ 19 \ 10 \ A$
1 2 3 4 5
6 7 8 9 1
2 3 4 5 6

A A tem duas matrizes com 5 linhas e 3 colunas cada

7 8 9 1 2
3 4 5 6 7
8 9 1 2 3

$+ \text{ / } [2] \ A$
9 12 15 18 12
18 21 15 9 12

A A redução tem 2 linhas com colunas, cada elemento
A corresponde a soma das linhas das matrizes de A

Redução nas demais dimensões

- ▶ O operador $\text{f}/$ (*reduce first*) produz uma redução em relação à primeira dimensão

$\times \text{f}/ A$
4 10 18

- ▶ Para produzir uma redução em relação a k -ésima dimensão, use a notação $\text{f}/[k]$

$A \leftarrow 2 \ 3 \ 5 \ \rho \ 19 \ \diamond A$
1 2 3 4 5
6 7 8 9 1
2 3 4 5 6

A A tem duas matrizes com 5 linhas e 3 colunas cada

7 8 9 1 2
3 4 5 6 7
8 9 1 2 3

$+/[2] A$
9 12 15 18 12
18 21 15 9 12

A A redução tem 2 linhas com colunas, cada elemento
A corresponde a soma das linhas das matrizes de A

- ▶ A notação $\text{f}/[1]$ equivale a $\text{f}/$

Novo símbolo

Símbolo	Aridade	Descrição
∇ (<i>reduce first</i>)	operador	Produz um redução em relação a primeira dimensão

Unicode	TAB	APL
U+233F	/ - <tab>	APL + ;

Reduções diádicas

- ▶ Na forma diádica, $L \text{ f } R$ é uma redução que usa uma janela de tamanho L em R

Reduções diádicas

- ▶ Na forma diádica, $L \text{ f } / R$ é uma redução que usa uma janela de tamanho L em R
- ▶ Esta redução por janela não altera o *rank* do argumento

```
      2 +/ 2 3 5 7 11
5 8 12 18
      3 +/ 2 3 5 7 11
10 15 23
```

Reduções diádicas

- ▶ Na forma diádica, $L \text{ f } / R$ é uma redução que usa uma janela de tamanho L em R
- ▶ Esta redução por janela não altera o *rank* do argumento

```

      2 +/ 2 3 5 7 11
5 8 12 18
      3 +/ 2 3 5 7 11
10 15 23

```

- ▶ Se L é negativo, a janela é invertida

```

      2 -/ 2 3 5 7 11
-1 -2 -2 -4
      -2 -/ 2 3 5 7 11
1 2 2 4

```

Funções lógicas e reduções

- ▶ A função `all` retorna 1 se todos os booleanos do argumento são iguais a 1

```
all ← ^/  
all 1 0 0 1 0 1  
0
```

Funções lógicas e reduções

- ▶ A função **all** retorna 1 se todos os booleanos do argumento são iguais a 1

```
all ← ^/  
all 1 0 0 1 0 1  
0
```

- ▶ A função **any** retorna 1 se ao menos um booleano do argumento é igual a 1

```
any ← v/  
any 1 0 0 1 0 1  
1
```

Funções lógicas e reduções

- ▶ A função `all` retorna 1 se todos os booleanos do argumento são iguais a 1

```
all ← ^/  
all 1 0 0 1 0 1  
0
```

- ▶ A função `any` retorna 1 se ao menos um booleano do argumento é igual a 1

```
any ← v/  
any 1 0 0 1 0 1  
1
```

- ▶ De fato, a função diádica `^` computa o mínimo múltiplo comum de seus argumentos e equivale a conjunção lógica para valores booleanos

Funções lógicas e reduções

- ▶ A função `all` retorna 1 se todos os booleanos do argumento são iguais a 1

```
all ← ^/  
all 1 0 0 1 0 1  
0
```

- ▶ A função `any` retorna 1 se ao menos um booleano do argumento é igual a 1

```
any ← v/  
any 1 0 0 1 0 1  
1
```

- ▶ De fato, a função diádica \wedge computa o mínimo múltiplo comum de seus argumentos e equivale a conjunção lógica para valores booleanos
- ▶ A função diádica \vee computa o maior divisor comum de seus argumentos, assumindo que $\text{gcd}(0, 0) = 0$, de forma que equivale a disjunção lógica para valores booleanos

Novo símbolo

Símbolo	Aridade	Descrição
\wedge (lcm)	diádico	Computa o mínimo múltiplo comum entre α e ω

Unicode	TAB	APL
U+2227	$\wedge \wedge$ <tab>	APL + 0

Novo símbolo

Símbolo	Aridade	Descrição
\vee (gcd)	diádico	Computa o maior divisor comum entre α e ω

Unicode	TAB	APL
U+2228	\vee \vee <tab>	APL + 9

Fatorial

- ▶ É possível computar o fatorial de um inteiro positivo n de, no mínimo, três formas distintas em APL

Fatorial

- ▶ É possível computar o fatorial de um inteiro positivo n de, no mínimo, três formas distintas em APL
- ▶ A primeira delas é recorrer a função monádica `!`, a segunda é implementar uma *dfn* recursiva e a terceira é usar uma redução

Fatorial

- ▶ É possível computar o fatorial de um inteiro positivo n de, no mínimo, três formas distintas em APL
- ▶ A primeira delas é recorrer a função monádica `!`, a segunda é implementar uma *dfn* recursiva e a terceira é usar uma redução
- ▶ A função `cmpx` do *workspace dfns* pode ser usada para comparar a performance destas três implementações

```

CY 'dfns'
f1 ← !
f2 ← x / 1
f3 ← {ω ≤ 1 : ω ◊ ω × ∇ω - 1}
cmpx 'f1 100' 'f2 100' 'f3 100'
f1 100 → 1.8E-7 | 0%
f2 100 → 3.0E-7 | +69%
f3 100 → 2.7E-5 | +15339%

```

Varredura

- ▶ O operador \backslash (*scan*) gera uma função monádica que age nos prefixos das últimas dimensões de seu parâmetro

Varredura

- ▶ O operador \backslash (*scan*) gera uma função monádica que age nos prefixos das últimas dimensões de seu parâmetro
- ▶ A expressão $f \backslash a \ b \ c \ d \ \dots$ equivale a $(f/a) \ (f/a \ b) \ (f/a \ b \ c) \ \dots$

```

      +\2 3 5 7 11      A somas acumuladas
2 5 10 17 28
      A ← 2 3 p 16 ◊ A
1 2 3
4 5 6
      ×\A
1 2 6
4 20 120

```


Varredura

- ▶ O operador \backslash (*scan*) gera uma função monádica que age nos prefixos das últimas dimensões de seu parâmetro
- ▶ A expressão $f \backslash a \ b \ c \ d \ \dots$ equivale a $(f/a) \ (f/a \ b) \ (f/a \ b \ c) \ \dots$


```

      +\ 2 3 5 7 11      A somas acumuladas
2 5 10 17 28
      A ← 2 3 p 16 ◊ A
1 2 3
4 5 6
      ×\ A
1 2 6
4 20 120

```


- ▶ O operador ∇ gera uma varredura na primeira dimensão

Novo símbolo

Símbolo	Aridade	Descrição
 (<i>scan</i>)	operador	Gera uma varredura na última dimensão

Unicode	TAB	APL
U+005C	-	-

Novo símbolo

Símbolo	Aridade	Descrição
 (<i>scan first</i>)	operador	Gera uma varredura na primeira dimensão
Unicode	TAB	APL
U+2340	\ - <tab>	APL + .

Referências

1. APL Wiki. [Defined function \(traditional\)](#), acesso em 27/09/2021.
2. APL Wiki. [Dfns workspace](#), acesso em 01/10/2021.
3. Dyalog. [Try APL – Interactive lessons](#), acesso em 23/09/2021.
4. **IVERSON**, Kenneth E. *A Programming Language*, John Wiley and Sons, 1962.
5. Unicode Character Table. [Página principal](#), acesso em 27/09/2021.
6. Xah Lee. [Unicode APL Symbols](#), acesso em 23/09/2021.