Vectores

José A. Alonso y María J. Hidalgo

Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Listas y vectores

- **Listas:**
  - **Ventaja: son extensibles.**
  - **Inconveniente: el acceso a los elementos es secuencial.**
- **Vectores:**
  - **Ventaja: el acceso a los elementos es directo.**
  - **Inconveniente: el tamaño es fijo.**

Procedimientos relativos a vectores

- **Reconocedor:**

```
(define v #(a b c))  =>  #<unspecified>
v                    =>  #(a b c)
(list? v)            =>  #f
(vector? v)          =>  #t
```

- **Constructores:**

```
(make-vector 3)     =>  #(#<unspecified> #<unspecified> #<unspecified>)
(make-vector 3 'a)  =>  #(a a a)
(vector 'a 'b 'c)   =>  #(a b c)
```

- **Selector:**

```
(define v (vector 'a 'b 'c))  =>  #<unspecified>
v                             =>  #(a b c)
(vector-ref v 1)              =>  b
```

Procedimientos relativos a vectores

- **Longitud:**

```
(vector-length #(a b c))  =>  3
```

- **Comparación de listas, cadenas y vectores**

	Listas	Cadenas	Vectores
Reconocedor	(list? l)	(string? c)	(vector? v)
Constructor	(cons x ())	(make-string n)	(make-vector n)
		(make-string n x)	(make-vector n x)
	(list x1 ... xn)	(string x1 ... xn)	(vector x1 ... xn)
Selector	(car l)		
	(cdr l)		
	(list-ref l n)	(string-ref c n)	(vector-ref v n)
Tamaño	(length l)	(string-length c)	(vector-length v)

Procedimientos relativos a vectores

- **Transformaciones entre vectores y listas**

```
(list->vector '(a b c))  =>  #(a b c)
(vector->list #(a b c))  =>  (a b c)
```

- **Mutador**

```
(define v #(a b c))   =>  #<unspecified>
v                     =>  #(a b c)
(vector-set! v 1 'x)  =>  #<unspecified>
v                     =>  #(a x c)
```

Definición de procedimientos con vectores

- **Ejemplo:** `genera-vector`

```
;;; ((genera-vector (lambda (i) (* i i))) 4)  =>  #(0 1 4 9)
;;; ((genera-vector (lambda (i) (- i))) 4)    =>  #(0 -1 -2 -3)
(define genera-vector
  (lambda (proc)
    (lambda (n)
      (letrec
          ((v (make-vector n))
           (bucle
            (lambda (i)
              (cond ((< i n)
                     (vector-set! v i (proc i))
                     (bucle (+ i 1)))))))
        (bucle 0)
        v))))
```

Definición de procedimientos con vectores

- **El procedimiento** `bucle`

```
;;; > (bucle 0 9 3 (lambda (i) (display i) (newline)))
;;; 0
;;; 3
;;; 6
;;; #<unspecified>
(define bucle
  (lambda (principio fin incremento procedimiento)
    (letrec ((aux
              (lambda (i)
                (cond ((< i fin)
                       (procedimiento i)
                       (aux (+ i incremento)))))))
      (aux principio))))
```

Definición de procedimientos con vectores

- **Ejemplo:** `genera-vector` **con** `bucle`

```
;;; ((genera-vector (lambda (i) (* i i))) 4)  =>  #(0 1 4 9)
;;; ((genera-vector (lambda (i) (- i))) 4)    =>  #(0 -1 -2 -3)
(define genera-vector
  (lambda (proc)
    (lambda (n)
      (let ((v (make-vector n)))
        (bucle 0 n 1
               (lambda (i) (vector-set! v i (proc i))))
        v))))
```

Definición de procedimientos con vectores

- **Procedimiento** `vector-inverso`

```
;;; (define v #(a (b c) d))  =>  #<unspecified>
;;; v                        =>  #(a (b c) d)
;;; (vector-inverso-1 v)     =>  #(d (b c) a)
;;; v                        =>  #(a (b c) d)
(define vector-inverso-1
  (lambda (v)
    (list->vector (reverse (vector->list v)))))

;;; (define v #(a (b c) d))  =>  #<unspecified>
;;; v                        =>  #(a (b c) d)
;;; (vector-inverso-2 v)     =>  #(d (b c) a)
;;; v                        =>  #(a (b c) d)
(define vector-inverso-2
  (lambda (v)
    (let ((n (vector-length v)))
      ((genera-vector (lambda (i) (vector-ref v (- n i 1)))) n))))
```

Definición de procedimientos con vectores

```
;;; (define v #(a b c d))   =>  #<unspecified>
;;; ((intercambia! v) 1 3)  =>  #<unspecified>
;;; v                       =>  #(a d c b)
(define intercambia!
  (lambda (v)
    (lambda (i j)
      (let ((aux (vector-ref v i)))
        (vector-set! v i (vector-ref v j))
        (vector-set! v j aux)))))

;;; (define v #(a (b c) d))  =>  #<unspecified>
;;; (vector-inverso! v)      =>  #(d (b c) a)
;;; v                        =>  #(d (b c) a)
(define vector-inverso!
  (lambda (v)
    (let ((n (- (vector-length v) 1)))
      (bucle 0 n 1
             (lambda (i)
               ((intercambia! v) i (- n i)))))
    v))
```

Definición de procedimientos con vectores

- **Procedimiento** `vector-map`

```
;;; (vector-map-1 sqrt #(9 4 25))  =>  #(3.0 2.0 5.0)
(define vector-map-1
  (lambda (proc v)
    ((genera-vector (lambda (i) (proc (vector-ref v i))))
     (vector-length v))))

;;; ((vector-map-2 +) #(7 4 5) #(2 4))  =>  #(9 8)
(define vector-map-2
  (lambda (proc)
    (lambda (v1 v2)
      ((genera-vector (lambda (i)
                        (proc (vector-ref v1 i)
                              (vector-ref v2 i))))
       (min (vector-length v1)
            (vector-length v2))))))
```

Definición de procedimientos con vectores

- **Operaciones con vectores**

```
;;; (multiplica-escalar-y-vector 2 #(1 5 7))  =>  (2 10 14)
(define multiplica-escalar-y-vector
  (lambda (n v)
    (vector-map (lambda (x) (* n x)) v)))

;;; (suma-vectores #(1 3 5) #(2 4 6))  =>  #(3 7 11)
(define suma-vectores (vector-map-2 +))

;;; (multiplica-vectores #(1 3 5) #(2 1 0))  =>  #(2 3 0)
(define multiplica-vectores (vector-map-2 *))
```

Definición de procedimientos con vectores

```
;;; (suma-elementos #(1 3 5))  =>  9
(define suma-elementos
  (lambda (v)
    (let ((aux (vector 0)))
      (bucle 0 (vector-length v) 1
             (lambda (i)
               (vector-set! aux 0 (+ (vector-ref aux 0)
                                     (vector-ref v i)))))
      (vector-ref aux 0))))

;;; (producto-escalar #(1 3 5) #(2 1 0))  =>  5
(define producto-escalar
  (lambda (v1 v2)
    (suma-elementos (multiplica-vectores v1 v2))))

;;; (norma #(3 4))  =>  5.0
(define norma
  (lambda (v)
    (sqrt (producto-escalar v v))))
```