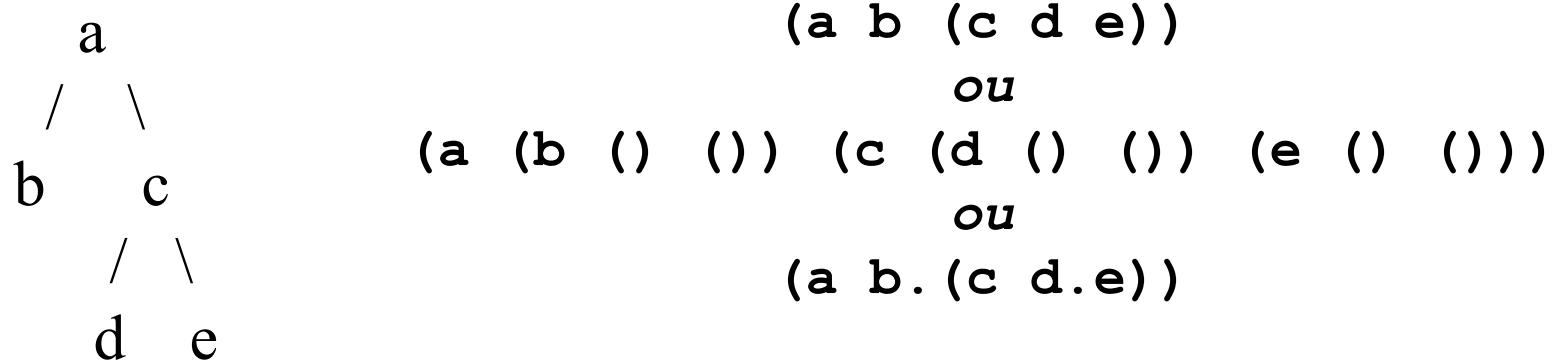


Les arbres

- ▶ Un arbre binaire peut être représentée avec des listes imbriquées



Est-ce un arbre?

```
(define arbre?
  (lambda (t)
    (cond
      ((not (list? t)) #f)
      ((null? t) #t)
      ((not (= (length t) 3)) #f)
      ((not (arbre? (cadr t))) #f)
      ((not (arbre? (caddr t))) #f)
      (else #t)
    ) ) )  
  
(arbre? '(73 31 (5 () ()) () (101 (83 () (97 () ()))))())())
(arbre? '(73 31 (5 () ()) () (101 (83 () (97 () ()))))())())
(arbre? '(73 31 (5 () ()) () (101 (83 () (97 ()))))))()
```

Parcours inordre

```
(define inordre
  (lambda (t)

    (define do-inordre
      (lambda (t)
        (if
          (null? t)
          ' ()
          (append (inordre (cadr t))
                  (cons (car t) (do-inordre (caddr t)))))

        ) ) )
      (if
        (not (arbre? t))
        (list 'erreur-arbre-invalidé t)
        (do-inordre t)
      ) ) )

(inordre '(73 (31 (5 () ()) ()) (101 (83 () (97 () ()) ()))))
```

La recherche dans un arbre

```
(define cherche
  (lambda (x t)

(define do-cherche
  (lambda (x t)
    (cond
      ((null? t) #f)
      ((equal? x (car t)) #t)
      ((precedes? x (car t)) (do-cherche x (cadr t)))
      ((precedes? (car t) x) (do-cherche x (caddr t)))
      (else #f)
    ) ) )

(if
  (not (arbre? t))
  (list 'erreur-arbre-invalide t)
  (do-cherche x t)
) ) )

(define precedes? (lambda (x y) (< x y)))

(cherche 83 '(73 (31 (5 () ()) ()) (101 (83 () (97 () ()))) ())))
(cherche 84 '(73 (31 (5 () ()) ()) (101 (83 () (97 () ()))) ()))
```

Insertion dans une liste

```
(define (arbre-insere arbre valeur)
  (cond ((null? arbre) (list valeur '() '()))
        ((< valeur (car arbre))
         (list (car arbre)
               (arbre-insere (cadr arbre) valeur)
               (caddr arbre)))
        (else (list (car arbre) (cadr arbre)
                    (arbre-insere (caddr arbre) valeur))))))
```

Détruire un nœud de l'arbre

```
(define detruire
  (lambda (x t)
    (if
      (not (arbre? t))
      (list 'erreur-arbre-invalide t)
      (do-detruire x t)
    ) ) )
```

Détruire la racine d'un arbre

```
(define do-detruire
  (lambda (x t)
    (cond
      ((null? t) '())
      ((and (equal? x (car t)) (null? (cadr t))) (caddr t))
      ((and (equal? x (car t)) (null? (caddr t))) (cadr t))
      ((equal? x (car t))
       (let ((r (removemax (cadr t))))
         (list (cdr r) (car r) (caddr t)))
       ))
      ((precedes? x (car t)) (list (car t)
                                      (do-detruire x (cadr t))
                                      (caddr t)))
      ((precedes? (car t) x) (list (car t) (cadr t)
                                    (do-detruire x (caddr t))))
      (else t)
    ) ) )
```

Détruire un nœud de l'arbre

```
(define removemax
  (lambda (t)
    (cond
      ((null? (caddr t)) (cons (cadr t) (car t)))
      (else
        (let ((r (removemax (caddr t))))
          (cons (list (car t) (cadr t) (car r))
                (cdr r)))
        )))
  ) ) )
```

Décompte du nombre d'éléments dans un arbre

- *Représentation sans les ()*

```
(define (nsymbols tree)
  (if (list? tree)
      (+ (nsymbols (car tree))
          (nsymbols (cdr tree)))
      (if (symbol? tree) 1 0) ) )
```

```
> (nsymbols '+ (* a b c))  
5
```

Avec récursivité terminale

```
(define (nsymbols tree) (nsymbolsb tree 0))

(define (nsymbolsb tree n)
  (if (list? tree)
      (nsymbolsb (cdr tree)
                 (nsymbolsb (car tree) n))
      (+ n (if (symbol? tree) 1 0)) ))
```

Tic Tac Toe

1	2	3
4	5	6
7	8	9

X		
	O	

```
(define start  `((1 2 3) (4 5 6) (7 8 9)
                  (1 4 7) (2 5 8) (3 6 9) (1 5 9) (3 5 7)))
```

X joue:

```
((x 2 3) (4 5 6) (7 8 9) (x 4 7) (2 5 8) (3 6 9) (x 5 9) (3 5 7))
```

O joue:

```
((x 2 3) (4 5 6) (7 o 9) (x 4 7) (2 5 o) (3 6 9) (x 5 9) (3 5 7))
```

Tic Tac Toe

La substitution:

```
(define subst
  (lambda (new old l)
    (cond ((null? l) (quote ()))
          ((atom? (car l))
           (cond ((eq? (car l) old)
                  (cons new
                        (subst new old (cdr l)))))
                 (else (cons (car l)
                             (subst new old (cdr l)))))))
          (else (cons (subst new old (car l))
                      (subst new old (cdr l)))))))
```

Tic Tac Toe

Egalité de tous les éléments d'une liste?

```
(define (all-equal? list)
  (cond ((null? list) '())
        ((null? (cdr list)) (car list))
        ((equal? (car list) (cadr list))
         (all-equal? (cdr list)))
        (else #f)))
```

Tic Tac Toe

```
(define (play board player position)
  (subst player position board))
```

```
(define (winner board)
  (map all-equal? board))
```

```
> (winner '((X 2 3) (4 X X) (7 O O) (X X X) (2 X O) (3 X O) (X X O) (3 X 7)))
(#f #f #f X #f #f #f #f)
```

Tic Tac Toe

```
(define (number-of-member x list)
  (cond ((null? list) 0)
        ((equal? x (car list))
         (+ 1 (number-of-member x (cdr list))))
        (else (number-of-member x (cdr list)))))
```

Tic Tac Toe

X		
	X	X
	O	O

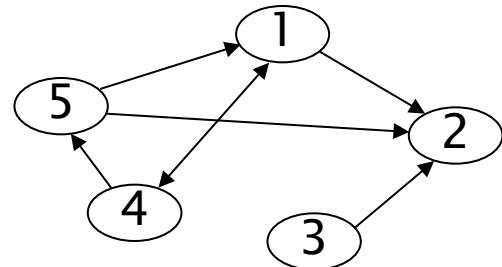
```
(map (lambda (list)
  (number-of-member 'X list))
  '(( (x 2 3) (4 x x) (7 o o)
    (x 4 7) (2 x o) (3 x o)
    (x x o) (3 x 7)) )
  (1 2 0 1 1 1 2 1))
```

Tours de Hanoi

```
(define (dohanoi n to from using)
  (if (> n 0)
      (begin
        (dohanoi (- n 1) using from to)
        (display "move ")
        (display from)
        (display " --> ")
        (display to)
        (newline)
        (dohanoi (- n 1) to using from) )
    ) )
  (define (hanoi n)
    (dohanoi n 3 1 2) )
```

Représentation des graphes

- ▶ Avec une liste d'adjacence



```
(define Graph '((1 (2 4)) (2 ()) (3 (2)) (4 (1 5)) (5 (1 2)))
```

```
(define valeur car)  
(define enfants cadr)  
(define premier-enfant caadr)
```

Obtenir un noeud du graphe

```
(define (noeud x graphe)
  (cond ((null? graphe) '())
        ((equal? x (valeur (car graphe))) (car graphe))
        (else (noeud x (cdr graphe)) ))))
```

Parcours d'un graphe

▶ En profondeur

```
(define (parcours x graphe)
  (reverse (parcours-liste-noeuds (list x) graphe '())))
```

▶ Parcours d'une liste de noeuds étant donné une liste de noeuds déjà visités

```
(define (parcours-liste-noeuds liste graphe visites)
  (cond ((null? liste) visites)
        ((member (car liste) visites) ; déjà visite?
         (parcours-liste-noeuds
           (cdr liste) graphe visites))
        (else (parcours-liste-noeuds (
          append (enfants (noeud (car liste) graphe))
          (cdr liste))
          graphe (cons (car liste) visites))))
```