

Plan	Parcours de listes ○○○○○ ○○○○○ ○○○○○○○○○○○	Modification de listes ○○○ ○○
------	---	-------------------------------------

Les listes (2)

Nour-Eddine Oussous, Éric Wegrzynowski

Licence ST-A, USTL - API2

2 novembre 2009

Les listes (2)	Licence ST-A, USTL - API2
----------------	---------------------------

Plan	Parcours de listes ●○○○○○ ○○○○○ ○○○○○○○○○○○	Modification de listes ○○○ ○○
Longueur d'une liste		

Spécification

Spécification

$$\begin{aligned} \text{longueur} : \text{Liste}(E) &\longrightarrow \mathbb{N} \\ \ell &\longmapsto \text{nbre d'élts de } \ell \end{aligned}$$

Exemples

$$\begin{aligned} \text{longueur}() &= 0 \\ \text{longueur}((3, 1, 4, 1, 5, 9, 2)) &= 7 \end{aligned}$$

Les listes (2)	Licence ST-A, USTL - API2
----------------	---------------------------

Plan	Parcours de listes ○○○○○ ○○○○○ ○○○○○○○○○○○	Modification de listes ○○○ ○○
------	---	-------------------------------------

Parcours de listes

Longueur d'une liste
Accès à un élément par son rang
Recherche dans une liste

Modification de listes

Ajout d'un élément en fin de liste
Insertion d'un élément à un rang donné

Les listes (2)	Licence ST-A, USTL - API2
----------------	---------------------------

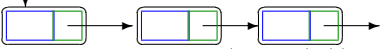
Plan	Parcours de listes ○●○○○○ ○○○○○ ○○○○○○○○○○○	Modification de listes ○○○ ○○
Longueur d'une liste		

Algorithme récursif

$L = \square$

$\text{longueur}(L) =$
► 0 si L est vide

$L = \square$



$1 + \text{longueur}(\text{reste}(L))$

► $1 + \text{longueur}(\text{reste}(L))$
sinon

Les listes (2)	Licence ST-A, USTL - API2
----------------	---------------------------

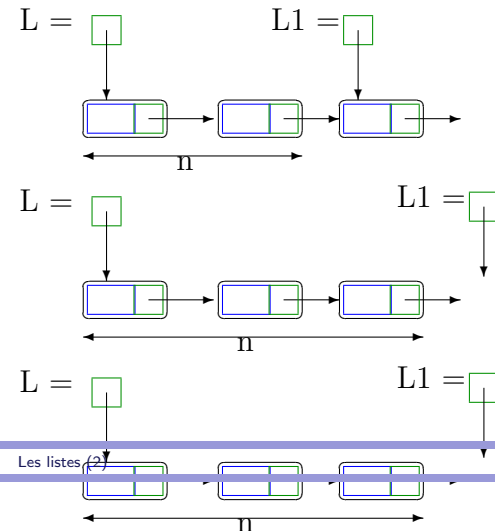
Algorithme récursif en Pascal

```
// longueur(l) = nombre d'éléments de l
function longueur(l : LISTE) : CARDINAL;
begin
  if estListeVide(l) then
    longueur := 0
  else
    longueur := 1+longueur(reste(l));
  end {longueur};
```

Algorithme itératif en Pascal

```
// longueur(l) = nombre d'éléments de l
function longueur(l : LISTE) : CARDINAL;
var
  n : CARDINAL;
  l1 : LISTE;
begin
  n := 0;
  l1 := l;
  { n = nbre d'élts de l déjà parcourus }
  while not(estListeVide(l1)) do begin
    n := n+1;
    l1 := reste(l1);
    { n = nbre d'élts de l déjà parcourus }
  end {while};
  longueur := n;
end {longueur};
```

Algorithme itératif



{n = nbre d'élts de L déjà comptés
L1 = liste des élts de L restant à compter }

n := n+1
L1 := reste(L1)

{n = nbre d'élts de L déjà comptés
L1 = liste des élts de L restant à compter }

À faire tant que L1 non vide

Coût du calcul de la longueur

$c(\ell)$ = nombre d'additions, ou d'appels récursifs, ou d'étapes du TantQue.

$$c(()) = 0$$

$$c(< x; \ell >) = 1 + c(\ell)$$

Conclusion

$$c(\ell) = \text{longueur}(\ell)$$

algorithme linéaire en fonction de la longueur de la liste

Rang d'un élément dans une liste

Définition

Le **rang** d'un élément (ou d'une cellule) d'une liste est la position de cet élément dans la liste. Les positions sont numérotées à partir de 0.

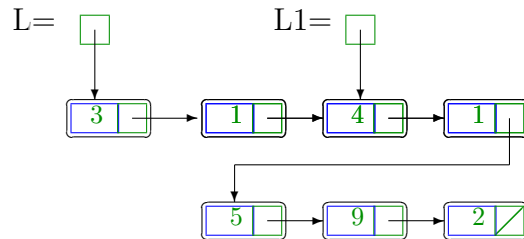
Exemples

Dans la liste (3,1,4,1,5,9,2)

- ▶ l'élément de rang 0 est 3,
- ▶ l'élément de rang 1 est 1,
- ▶ l'élément de rang 2 est 4, ...

Exemple

$L1 := \text{acces}(2, (3, 1, 4, 1, 5, 9, 2))$



Aucune nouvelle cellule créée !

Spécification

Spécification

$$\begin{aligned} \text{acces} : \mathbb{N} \times \text{Liste}(E) &\longrightarrow \text{Liste}(E) \\ k, \ell &\longmapsto \ell' \end{aligned}$$

ℓ' = sous-liste de ℓ débutant par l'élément de rang k .

CU : ℓ non vide et $k < \text{longueur}(\ell)$.

Exemples

$$\begin{aligned} \text{acces}(2, (3, 1, 4, 1, 5, 9, 2)) &= (4, 1, 5, 9, 2) \\ \text{acces}(7, (3, 1, 4, 1, 5, 9, 2)) &= \text{Non défini} \end{aligned}$$

Algorithme récursif en Pascal

```

// donne la liste débutant au k-ième élément de l
// les éléments étant numérotés à partir de 0
// CU : 0 ≤ k < long(l)
function acces(k : CARDINAL; l : LISTE) : LISTE;
begin
  if k=0 then
    acces := l
  else
    acces := acces(k-1, reste(l));
end {acces};

```

Plan	Parcours de listes ○○○○○ ○○○○● ○○○○○○○○○○○	Modification de listes ○○ ○
Accès à un élément par son rang		

Algorithme itératif en Pascal

```
// donne la liste débutant au k-ième élément de l
// les éléments étant numérotés à partir de 0
// CU : 0 ≤ k < long(l)
function acces(k : CARDINAL; l : LISTE) : LISTE;
var
  l1 : LISTE;
  i : CARDINAL;
begin
  l1 := l;
  i := 0;
  { tete de l1 = élément de rang i de l }
  while i < k do begin
    l1 := reste(l1);
    i := i + 1;
    { tete de l1 = élément de rang i de l }
  end {while};
  acces := l1;
end {acces};
```

Plan	Parcours de listes ○○○○○ ○○○○○ ●○○○○○○○○○○○	Modification de listes ○○ ○
Recherche dans une liste		

Différentes recherches dans une liste

Étant donné un élément $e \in E$ et une liste $\ell \in \text{Liste}(E)$, on peut s'intéresser à

- ▶ la première occurrence de e dans ℓ ,
- ▶ la dernière occurrence de e dans ℓ ,
- ▶ toutes les occurrences de e dans ℓ .

Plan	Parcours de listes ○○○○○ ○○○○○ ○○○○○○○○○○○	Modification de listes ○○ ○
Accès à un élément par son rang		

Coût de l'accès

$c(k, \ell)$ = nombre d'appels récurifs, ou d'étapes du TantQue.

$$\begin{aligned} c(0, \ell) &= 0 \\ c(k, \ell) &= 1 + c(k-1, \text{reste}(\ell)) \end{aligned}$$

Conclusion

$$c(k, \ell) = k$$

le coût dépend uniquement de k .
Algorithme linéaire en fonction de k

Plan	Parcours de listes ○○○○○ ○○○○○ ●○○○○○○○○○○○	Modification de listes ○○ ○
Recherche dans une liste		

Différentes issues d'une recherche

Une recherche d'occurrence d'un élément e dans une liste ℓ peut aboutir à

- ▶ un succès lorsque e est dans ℓ ,
- ▶ un échec lorsque e n'est pas dans ℓ

Première occurrence

Spécification

$$\begin{aligned} \text{premiereOccur} : E \times \text{Liste}(E) &\longrightarrow \text{Liste}(E) \\ e, \ell &\longmapsto \ell' \end{aligned}$$

ℓ' = la sous-liste de ℓ débutant à la première occurrence de e si $e \in \ell$, $\ell' = ()$ sinon

Exemples

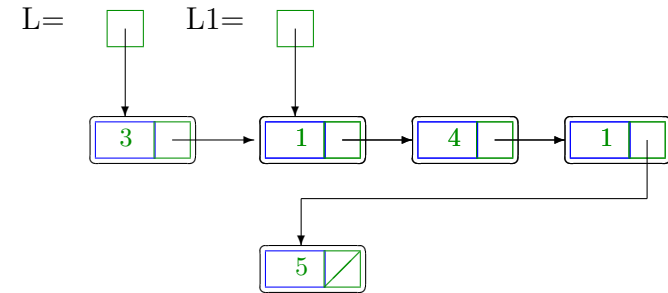
$$\begin{aligned} \text{premiereOccur}(1, (3, 1, 4, 1, 5)) &= (1, 4, 1, 5) \\ \text{premiereOccur}(6, (3, 1, 4, 1, 5)) &= () \end{aligned}$$

Algorithme récursif en Pascal

```
// premiereOccur(e,l) =
//   la liste débutant à la première occurrence
//   de e dans l si elle existe
//   la liste vide sinon
function premiereOccur(const e : ELEMENT;
                      const l : LISTE) : LISTE;
begin
  if estListeVide(l) then
    premiereOccur := LISTEVIDE
  else
    if tete(l)=e then
      premiereOccur := l
    else
      premiereOccur := premiereOccur(e,reste(l));
end {premiereOccur};
```

Exemple

$L1 := \text{premiereOccur}(1, (3, 1, 4, 1, 5))$



Aucune nouvelle cellule créée !

Coût de la recherche de la première occurrence

$c(e, \ell)$ = nombre d'appels récurifs (ou d'accès à la tête d'une liste).

- dépend de e et de ℓ
- pour une recherche qui échoue $c(e, \ell) = \text{longueur}(\ell)$
- pour une recherche avec succès
 - dans le meilleur des cas $c(e, \ell) = 1$
 - dans le pire des cas $c(e, \ell) = \text{longueur}(\ell)$

Conclusion

Coût de la recherche compris entre 0 (liste vide) et $\text{longueur}(\ell)$
Algorithme linéaire (dans le pire des cas) en fonction de la longueur.

Dernière occurrence

Spécification

$$\begin{aligned} \text{derniereOccur} : E \times \text{Liste}(E) &\longrightarrow \text{Liste}(E) \\ e, \ell &\longmapsto \ell' \end{aligned}$$

ℓ' = la sous-liste de ℓ débutant à la dernière occurrence de e si $e \in \ell$, $\ell' = ()$ sinon

Exemples

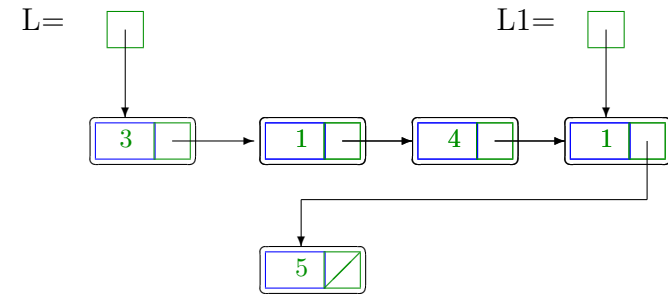
$$\begin{aligned} \text{derniereOccur}(1, (3, 1, 4, 1, 5)) &= (1, 5) \\ \text{derniereOccur}(6, (3, 1, 4, 1, 5)) &= () \end{aligned}$$

Algorithme récursif en Pascal

```
// derniereOccur(e,l) =
//   la liste débutant à la dernière occurrence
//   de e dans l si elle existe
//   la liste vide sinon
function derniereOccur(const e : ELEMENT;
                      const l : LISTE) : LISTE;
var
  l1 : LISTE;
begin
  if estListeVide(l) then
    derniereOccur := LISTEVIDE;
  else begin
    l1 := derniereOccur(e, reste(l));
    if estListeVide(l1) and (tete(l) = e) then
      derniereOccur := l
    else
      derniereOccur := l1;
    end {if};
  end {derniereOccur};
```

Exemple

$L1 := \text{derniereOccur}(1, (3, 1, 4, 1, 5))$



Aucune nouvelle cellule créée !

Coût de la recherche de la dernière occurrence

$c(e, \ell)$ = nombre d'appels récursifs (ou d'accès à la tête d'une liste).

- ▶ dans tous les cas $c(e, \ell) = \text{longueur}(\ell)$
- ▶ ne dépend donc pas de e

Conclusion

Coût de la recherche égal à $\text{longueur}(\ell)$
Algorithme linéaire en fonction de la longueur.

Plan	Parcours de listes ○○○○○ ○○○○○ ○○○○○○○○○●○○	Modification de listes ○○○ ○○
Recherche dans une liste		

Dernier élément

Spécification

$$\begin{array}{ccc} \text{dernier} : \text{Liste}(E) & \longrightarrow & \text{Liste}(E) \\ \ell & \longmapsto & \ell' \end{array}$$

ℓ' = la sous-liste de ℓ débutant au dernier élément de ℓ si ℓ n'est pas vide, $\ell' = ()$ sinon

Exemples

$$\begin{array}{l} \text{dernier}() = () \\ \text{dernier}((3,1,4,1,5)) = (5) \end{array}$$

Les listes (2)	Licence ST-A, USTL - API2
----------------	---------------------------

Plan	Parcours de listes ○○○○○ ○○○○○ ○○○○○○○○○○○●	Modification de listes ○○○ ○○
Recherche dans une liste		

Coût de la recherche du dernier élément

$c(\ell)$ = nombre d'appels récurifs

- ▶ $c(\ell) = 0$ si ℓ est vide
- ▶ dans tous les autres cas $c(\ell) = \text{longueur}(\ell) - 1$

Conclusion

Coût de la recherche égal à $\text{longueur}(\ell) - 1$
Algorithme linéaire en fonction de la longueur.

Les listes (2)	Licence ST-A, USTL - API2
----------------	---------------------------

Plan	Parcours de listes ○○○○○ ○○○○○ ○○○○○○○○○○○●○	Modification de listes ○○○ ○○
Recherche dans une liste		

Algorithme récursif en Pascal

```
// derner(l) =
//   la liste débutant au dernier élément
//   de l si l n'est pas vide
//   la liste vide sinon
function derner(const l : LISTE) : LISTE;
begin
  if estListeVide(l) then
    derner := LISTEVIDE
  else if estListeVide(reste(l)) then
    derner := l
  else
    derner := derner(reste(l));
end {dernier};
```

Les listes (2)	Licence ST-A, USTL - API2
----------------	---------------------------

Plan	Parcours de listes ○○○○○ ○○○○○ ○○○○○○○○○○○●○	Modification de listes ●○○ ○○
Ajout d'un élément en fin de liste		

Spécification

Spécification

$$\begin{array}{ccc} \text{ajouteEnFin} : E \times \text{Liste}(E) & \longrightarrow & \text{Liste}(E) \\ e, \ell & \longmapsto & \ell' \end{array}$$

ℓ' = liste ℓ modifiée par l'ajout d'un nouvel élément e à la fin

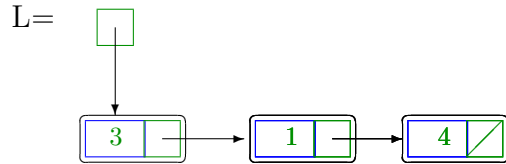
Exemples

$$\begin{array}{l} \text{ajouteEnFin}(1,()) = (1) \\ \text{ajouteEnFin}(9,(3,1,4,1,5)) = (3,1,4,1,5,9) \end{array}$$

Les listes (2)	Licence ST-A, USTL - API2
----------------	---------------------------

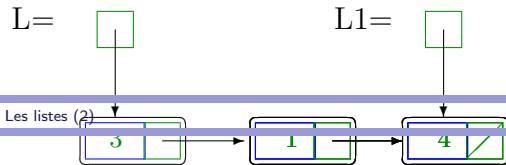
Algorithme

Ajout de 1 en fin de la liste $L=(3,1,4)$.



```
L1 := dernier(L)
L2 :=
ajouteEnTete(1,LISTEVIDE)
modifierReste(L1,L2)
```

Terminé!
Création d'une cellule



Spécification

Spécification

$$\text{inserer} : E \times \text{Liste}(E) \times \mathbb{N} \longrightarrow \text{Liste}(E)$$

$$e, \ell, k \longmapsto \ell'$$

ℓ' = liste ℓ modifiée par l'insertion d'un nouvel élément e au rang k .

CU : $1 \leq k \leq \text{longueur}(\ell)$

Exemples

$\text{inserer}(4, (3, 1, 1, 5), 2) = (3, 1, 4, 1, 5)$
 $\text{inserer}(9, (3, 1, 4, 1, 5), 5) = (3, 1, 4, 1, 5, 9)$
 $\text{inserer}(9, (3, 1, 4, 1, 5), 6) = \text{Non défini !}$

Algorithme en Pascal

```
// ajouteEnFin(e,l) : ajoute un nouvel
// élément e à la fin de l
// la liste l est modifiée
procedure ajouteEnFin(const e : ELEMENT;
                      var l : LISTE);
var
    l1,l2 : LISTE;
begin
    l1 := dernier(l);
    l2 := ajouteEnTete(e,LISTEVIDE);
    if estListeVide(l1) then
        l := l2
    else
        modifierReste(l1,l2);
end {ajouterEnFin};
```

Algorithme en Pascal

```
// inserer(e,l,k): insere un nouvel élément e au
// rang k dans l
// la liste l est modifiée
// CU :  $1 \leq k \leq \text{longueur}(l)$ 
procedure inserer(const e : ELEMENT;
                  const l : LISTE;
                  const k : CARDINAL);
var
    l1,l2 : LISTE;
begin
    l1 := acces(k-1,l);
    l2 := ajouteEnTete(e,reste(l1));
    modifierReste(l1,l2);
end {inserer};
```