



LIF3

1

Plan « Réussite en Licence »

Présentation de l'UE

Modalités de Contrôle des Connaissances

PLAN « RÉUSSITE EN LICENCE »

- Référent Pédagogique
 - suit votre progression
 - répond à vos questions, vous conseille sur vos difficultés
 - vous devez répondre à ses courriels et vous rendre aux RDV
- Toutes les UE en Contrôle Continu Intégral (CCI)
 - plusieurs notes pour vous évaluer
 - plus de notion d'examen ni de seconde session d'examen
- 2 séances de TD de révisions
- 2 séances de soutien
 - Sur avis des intervenants de TD-TP
 - Présence obligatoire
- Harmonisation des notes en fin de semestre
- Vous devez respecter le contrat pédagogique
 - En cas de non respect, limitation des IP pour le prochain semestre

PRÉSENTATION DE L'UE LIF3

○ Responsables de l'UE

- Marie Lefevre – bâtiment Nautibus (semestre d'automne 2014)
- Nathalie Guin – bâtiment Nautibus (semestre de printemps 2015)
- Florence Zara – bâtiment Nautibus

○ Site Web de l'UE

- Automne 2014 : <http://liris.cnrs.fr/marie.lefevre/ens/LIF3/>
- Planning (salles, horaires)
- Supports des CMs, TDs et TPs
- MCC
- Corrigés des TPs

MODALITÉS DE CONTRÔLE DES CONNAISSANCES

- Cette UE est en CCI
 - Pas d'examen écrit pendant la « session d'examens »
 - Pas de seconde session
- Le contrôle continu est constitué de plusieurs épreuves :
 - interrogations écrites en début de TD (coeff. 0,20)
 - TP noté en conditions d'examen (TP 6, coeff. 0,25)
 - TP projet (TP 7 et 9, coeff. 0,15)
 - épreuve écrite en amphi en fin de semestre (coeff. 0,4)



ALGORITHMIQUE ET PROGRAMMATION FONCTIONNELLE ET RÉCURSIVE

5

Algorithme itératif / récursif

Programmation impérative / fonctionnelle

QU'EST-CE QU'UN ALGORITHME ?

- On souhaite résoudre le problème :
Trouver le minimum d'une liste de nombres :
par exemple (3 6,5 12 -2 0 7)
- Expliquer à une machine comment trouver le minimum de n'importe quelle liste de nombres
 - Langage commun entre la machine et nous : Scheme

DÉFINITION D'UN ALGORITHME

- Un algorithme est une méthode
 - Suffisamment générale pour pouvoir traiter toute une classe de problèmes
 - Combinant des opérations suffisamment simples pour être effectuées par une machine

COMPLEXITÉ D'UN ALGORITHME

○ Il faut :

- que la machine trouve le plus vite possible
 - Complexité en temps
- qu'elle trouve en utilisant aussi peu de place mémoire que possible
 - Complexité en espace

RAPPEL : LA MACHINE N'EST PAS INTELLIGENTE

- L'exécution d'un algorithme ne doit normalement pas impliquer des décisions subjectives, ni faire appel à l'utilisation de l'intuition ou de la créativité
- Exemple : une recette de cuisine n'est pas un algorithme si elle contient des notions vagues comme « ajouter du sel »

MON PREMIER ALGORITHME

- Déterminer le minimum d'une liste de nombres
 - par exemple (3 6,5 12 -2 0 7)

MÉTHODE ITÉRATIVE

- Je parcours la liste de gauche à droite, en retenant à chaque pas le minimum provisoire

(3 6,5 12 -2 0 7)

- Entre 3 et 6,5, c'est 3
- Entre 3 et 12, c'est 3
- Entre 3 et -2, c'est -2
- Entre -2 et 0, c'est -2
- etc.

DE QUOI AI-JE BESOIN POUR ÉCRIRE L'ALGORITHME ? (1)

- La séquence

Début

Instruction(s)

Fin

- L'affectation

Variable ← Expression

- $A \leftarrow 2$

- $BX4 \leftarrow A + 2 * \text{racine}(15)$

DE QUOI AI-JE BESOIN POUR ÉCRIRE L'ALGORITHME ? (2)

- La conditionnelle (1)

Si Condition Alors

Instruction(s)

FinSi

- Exemple :

Si $(A > 27)$ Alors

$B \leftarrow A * 3$

FinSi

DE QUOI AI-JE BESOIN POUR ÉCRIRE L'ALGORITHME ? (3)

- o La conditionnelle (2)

Si Condition Alors

Instruction(s)

Sinon

Instruction(s)

FinSi

- o Exemple :

Si ((A<10) et
(B > racine(A*5))) Alors

B ← A*3

A ← A+B

Sinon

A ← A+2

B ← A*B

FinSi

DE QUOI AI-JE BESOIN POUR ÉCRIRE L'ALGORITHME ? (4)

La **condition** est :

- soit une condition élémentaire
- soit une condition complexe, c'est à dire une conjonction, disjonction, ou négation de conditions élémentaires et/ou complexes

DE QUOI AI-JE BESOIN POUR ÉCRIRE L'ALGORITHME ? (5)

- L'itérative, ou boucle
TantQue Condition Faire
Instruction(s)
FinTantQue
- Exemple :
TantQue $i < 10$ Faire
 $a \leftarrow a * i$
 $i \leftarrow i + 1$
FinTantQue

OPÉRATEURS BOOLÉENS

- Pour écrire une conjonction, disjonction, ou négation de conditions, on a besoin des opérateurs booléens **ET, OU, NON**
- Une variable **booléenne** est une variable dont les valeurs peuvent être **vrai** ou **faux**
- Un opérateur booléen est un opérateur dont les arguments sont des variables booléennes et dont le résultat est booléen

L'OPÉRATEUR ET

X	Y	X ET Y
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

L'OPÉRATEUR OU

X	Y	X OU Y
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

L'OPÉRATEUR NON

X	NON X
Vrai	Faux
Faux	Vrai

ALGORITHME ITÉRATIF

○ Soient

- L : la liste
- *premier*, *longueur*, *ième* et *écrire* : des primitives (i.e. algorithmes déjà définis)

Définition de minimum(L)

Début

min \leftarrow premier(L)

i \leftarrow 2

TantQue i \leq longueur(L) Faire

 Si *ième*(L,i) < min Alors

 min \leftarrow *ième*(L,i)

 FinSi

 i \leftarrow i+1

FinTantQue

Écrire(« Le minimum est »)

Écrire(min)

Fin

VOCABULAIRE

- *minimum* et *écrire* sont des **procédures**,
i.e. modifient l'environnement
- *premier*, *longueur* et *ième* sont des **fonctions**,
i.e. retournent une valeur,
mais ne modifient pas l'environnement
- L est le **paramètre** de la procédure *minimum*
- i est **l'indice** ou le **compteur** de la boucle

COMPLEXITÉ EN TEMPS DE L'ALGORITHME ITÉRATIF DU MINIMUM

Définition de minimum(L)

Début

min \leftarrow premier(L)

i \leftarrow 2

TantQue i \leq longueur(L) Faire

 Si $i^{\text{ème}}(L,i) < \text{min}$ Alors

 min $\leftarrow i^{\text{ème}}(L,i)$

 FinSi

 i \leftarrow i+1

FinTantQue

Écrire(« Le minimum est »)

Écrire(min)

Fin

Soit n la longueur de la liste L

1 affectation (initialisation)

1 affectation (initialisation)

n comparaisons

n-1 comparaisons

m affectations

n-1 affectation (incrémentation)

1 écriture

1 écriture

LE NOMBRE m DÉPEND DE LA LISTE

- **Meilleur cas** : $m=0$ si le premier nombre de la liste est le minimum
- **Pire cas** : $m=n-1$ si les nombres de la liste sont rangés en ordre décroissant
- **Cas moyen** : $m=(n-1)/2$ s'ils respectent une distribution parfaitement aléatoire

COMPLEXITÉ EN ESPACE DE L'ALGORITHME

- Un mot pour stocker le minimum provisoire (min)
- Un mot pour savoir où on en est dans la liste (i)

MÉTHODE RÉCURSIVE (1)

- Pour trouver le minimum de la liste (3 6,5 12 -2 0 7)
- On suppose le problème résolu pour la liste privée de son premier élément i.e. (6,5 12 -2 0 7)
- Soit min le minimum de cette sous-liste, ici -2
- Le minimum de la liste complète s'obtient par comparaison entre le premier élément de la liste (ici 3) et min (ici -2)

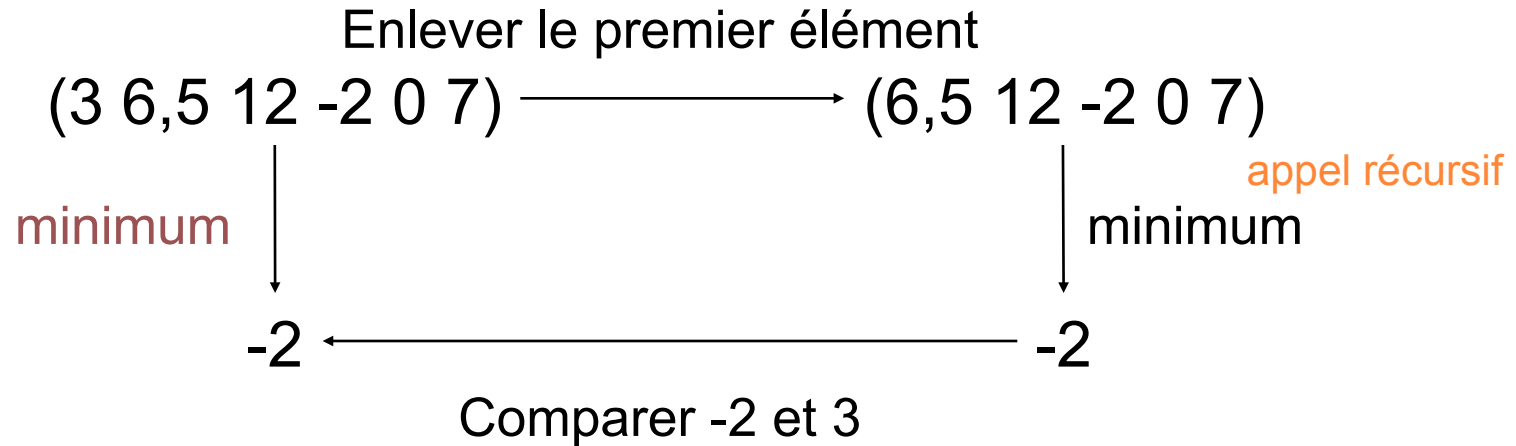
MÉTHODE RÉCURSIVE (2)

- Comment résout-on le problème pour la sous-liste ?
En faisant le même raisonnement.
- C'est la **récurtivité**.

- Quand s'arrête-t-on ?
Quand on ne peut plus parler de sous-liste, i.e. quand la liste a un seul élément.
C'est alors le minimum.

ILLUSTRATION DE LA MÉTHODE

Sur quoi faire l'appel récursif ?



Comment passer du résultat de l'appel récursif au résultat qu'on cherche ?

ALGORITHME RÉCURSIF

- Soient *vide?*, *reste* et *premier* des fonctions primitives

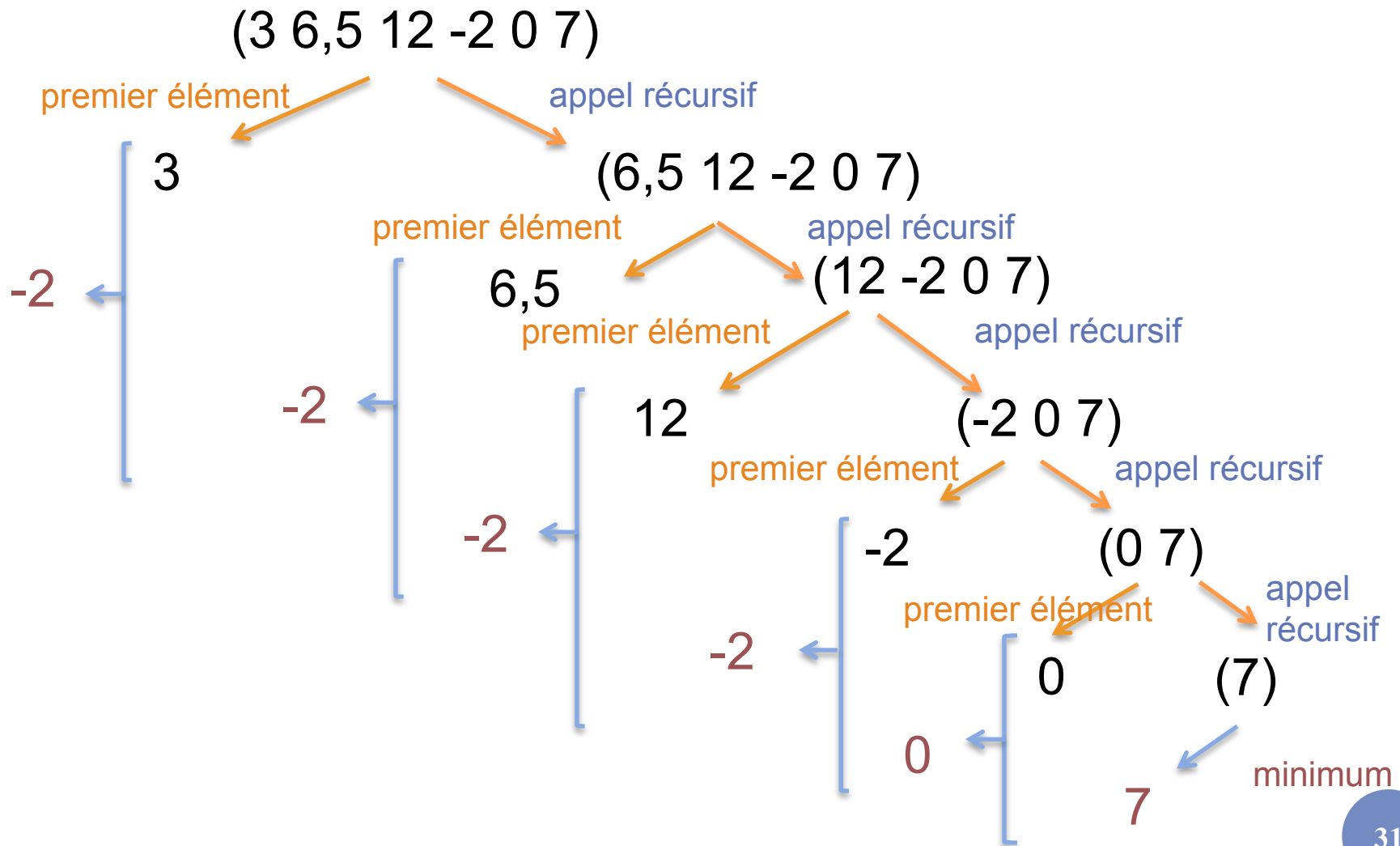
Définition de la fonction **minimum** (L)

```
Si vide?(reste(L)) Alors
    retourne premier(L)
Sinon
    Si premier(L) < minimum(reste(L)) Alors
        retourne premier(L)
    Sinon
        retourne minimum(reste(L))
    FinSi
FinSi
```

REMARQUES

- *minimum* est ici une fonction, le mot **retourne** permet de dire quel est son résultat
- *minimum* est **l'appel récursif**
- En programmation fonctionnelle, on n'a plus besoin de la séquence
- En programmation récursive, on n'a plus besoin de la boucle

ILLUSTRATION DE L'ALGORITHME



COMPLEXITÉ EN TEMPS DE L'ALGORITHME RÉCURSIF DU MINIMUM

Définition de la fonction minimum(L)

Si vide?(reste(L)) Alors

 retourne premier(L)

Sinon

 Si premier(L) < minimum(reste(L))

 Alors

 retourne premier(L)

 Sinon

 retourne minimum(reste(L))

 FinSi

FinSi

1 test

1 comparaison
+ le nombre de
comparaisons de
l'appel récursif

COMPLEXITÉ EN TEMPS DE L'ALGORITHME (2)

- Si n est la longueur de la liste
- Si $C(i)$ est le nombre de comparaisons de l'algorithme pour une liste de longueur i
- Alors $C(n) = 1+C(n-1)$
 $= 1+1+C(n-2)$
 $= \dots$
 $= 1+1+\dots+C(1)$
 $= 1+1+\dots+0$
 $= n-1$

RÉSUMÉ SUR LA COMPLEXITÉ

- Choisir ce que l'on va compter
 - unité de comparaison des algorithmes
 - par exemple le nombre de comparaisons
- Ce qui importe est l'ordre de grandeur de la complexité
 - constant, $\log n$, linéaire, $n \cdot \log n$, n^2 , 2^n
- En LIF3 on s'intéressera essentiellement au nombre de fois où l'on parcourt une structure de donnée (liste, arbre)

POUR ÉCRIRE UN ALGORITHME RÉCURSIF

- Il faut choisir
 - Sur quoi on fait l'appel récursif
 - Comment on passe du résultat de l'appel récursif au résultat que l'on cherche
 - Le cas d'arrêt

STRUCTURE TYPE D'UNE FONCTION RÉCURSIVE

Si je suis dans le cas d'arrêt
Alors voilà le résultat
Sinon le résultat est
le résultat de l'application d'une fonction
sur le résultat de l'appel récursif

LES BUGS

- Mon algorithme est faux car son résultat n'est pas défini si la liste est vide ou si elle contient autre chose que des nombres
- Pour éviter les bugs il faut :
 - Définir le problème aussi bien que possible
C'est la **spécification**
 - Tenter de prouver que son programme répond à la spécification
 - Passer des **jeux d'essai**, aussi divers et tordus que possible

POUR RÉSUMER

LIF1 : Programmation

Langage C

Impérative

Séquence
(faire les choses
l'une après
l'autre)

Itérative

Boucle
(répéter)

LIF3 : Programmation

Langage Scheme

Fonctionnelle

Appliquer une fonction
à des arguments
pour obtenir un résultat

Composer les fonctions pour
enchaîner les traitements

Réursive

La répétition est assurée
par l'enchaînement
des appels récursifs

Le test de la boucle est
remplacé par le cas d'arrêt