LIF3

Présentation de l'UE

Modalités de Contrôle des Connaissances

PRÉSENTATION DE L'UE LIF3

- Responsable de l'UE
  - Nathalie Guin – bâtiment Nautibus (automne 2015)

- Site Web de l'UE
  - http://liris.cnrs.fr/nathalie.guin/LIF3/
  - Planning (salles, horaires)
  - Supports des CMs, TDs et TPs
  - MCC
  - Corrigés des TPs
  - Lien vers la plateforme d'entraînement : ASKER

Modalités de contrôle des connaissances

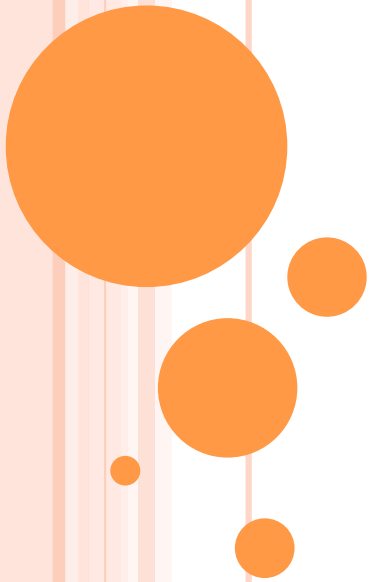
- Cette UE est en CCI
  - Pas de seconde session
- Le contrôle continu est constitué de plusieurs épreuves :
  - interrogations écrites en début de TD (coeff. 0,20)
  - TP noté en conditions d'examen (TP 6, coeff. 0,25)
  - TP projet (TP 8 et 9, coeff. 0,15)
  - épreuve écrite en amphi en fin de semestre (coeff. 0,4)
- Harmonisation des notes en fin de semestre

CONTRÔLES EN TD

- Moins de 10mn
  - Arriver à l'heure !
- Peuvent porter sur
  - Des notions vues en cours (et pas encore en TD)
    - Revoir et apprendre son cours
    - S'entrainer sur ASKER pour vérifier qu'on a compris
  - Des exercices qui étaient à préparer
    - Préparer les exercices indiqués comme tels sur les sujets de TD
  - Des exercices déjà faits en TD (ou des variantes)
    - Revoir les TD précédents, savoir refaire les exercices

ASKER : DES EXERCICES D'ENTRAINEMENT

- Exercices d'auto-évaluation
- Permettent de faire des exercices sur le cours et de préparer les tests en TD
- On peut demander plusieurs versions du même exercice
- Lien et explications sur la page web de l'UE



Algorithmique et programmation fonctionnelle et récursive

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 ← 2
  - BX4 ← A+2*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 ← A*3

  FinSi

De quoi ai-je besoin pour écrire l'algorithme ? (3)

- La conditionnelle (2)

  Si Condition Alors

    Instruction(s)

  Sinon

    Instruction(s)

  FinSi

- 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 ← a*i

    i ← 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 ← premier(L)
    i ← 2
    TantQue i <= longueur(L) Faire
        Si ième(L,i) < min Alors
            min ← ième(L,i)
        FinSi
        i ← 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 ← premier(L)
    i ← 2
    TantQue i <= longueur(L) Faire
        Si ième(L,i) < min Alors
            min ← ième(L,i)
        FinSi
            i ← 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)

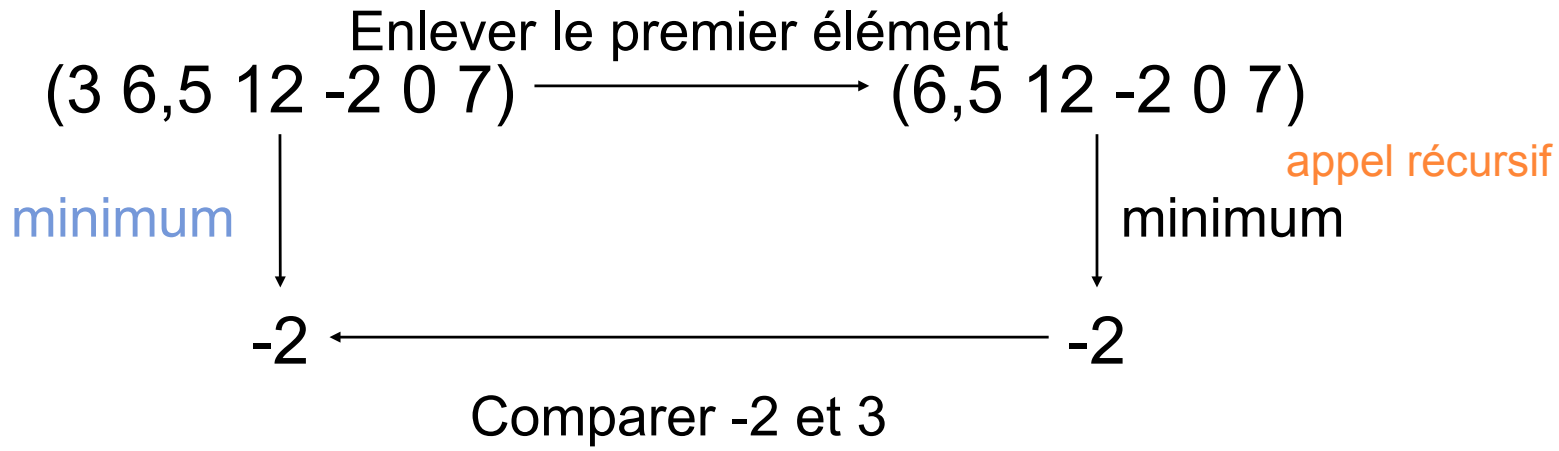
MÉTHODE RÉCURSIVE (2)

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

- 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

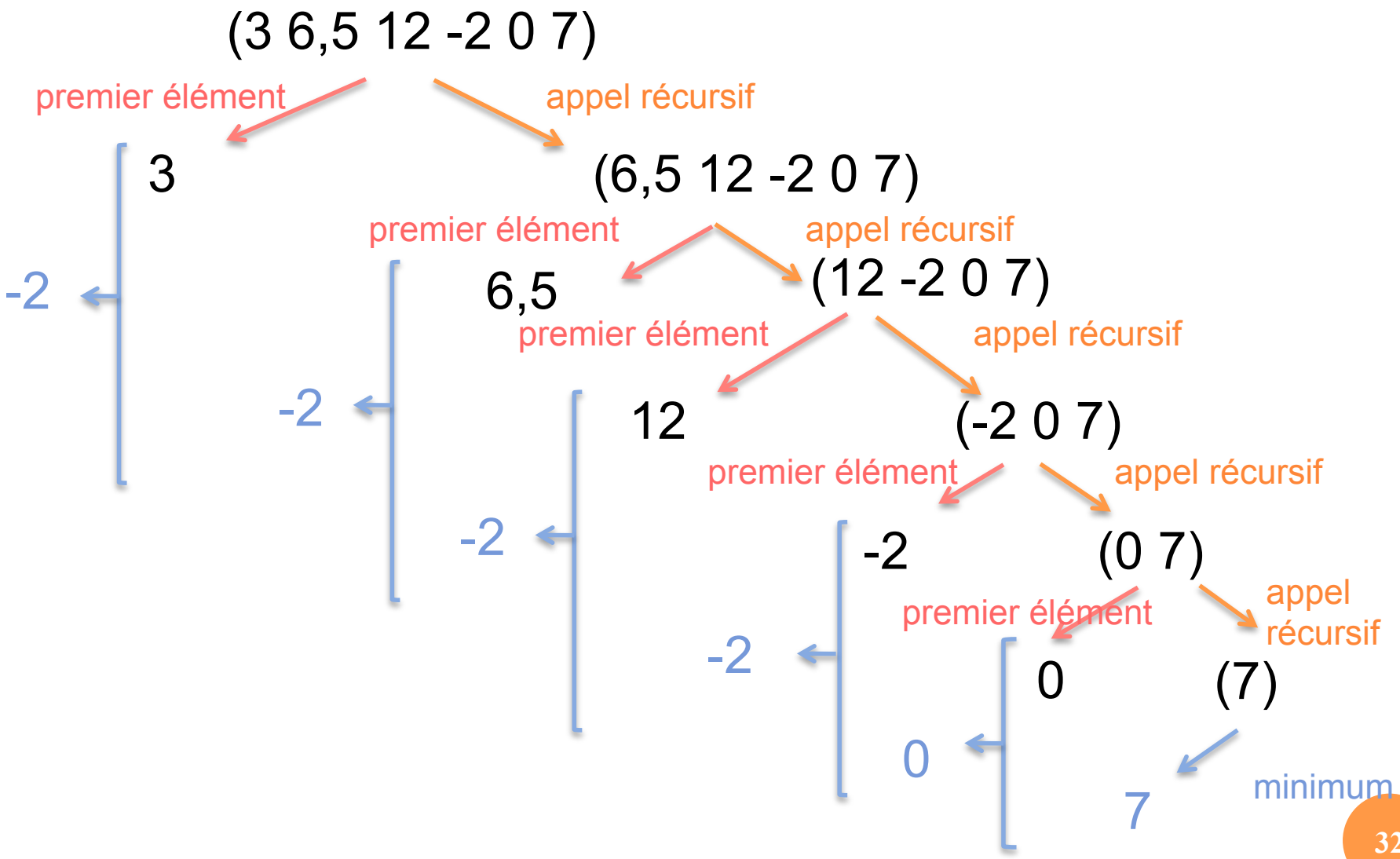
Définition de la fonction minimum **(L)**

```
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)
  = ...
  = 1+1+…+C(1)
  = 1+1+…+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*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	Impérative	Itérative
Langage C	Séquence (faire les choses l'une après l'autre)	Boucle (répéter)
LIF3 : Programmation	Fonctionnelle	Récursive
Langage Scheme	Appliquer une fonction à des arguments pour obtenir un résultat Composer les fonctions pour enchaîner les traitements	La répétition est assurée par l'enchaînement des appels récursifs Le test de la boucle est remplacé par le cas d'arrêt