Université de Monastir

Cours: Conception et Analyse d'algorithmes

Chapitre 1: La récursivité (Parie 2)

Réalisé par:
Dr. Sakka Rouis Taoufik

1

I. La dérécursivation

➢Au niveau de l'utilisateur: la récursivité permet d'écrire des algorithmes simples et élégants.

➢Au niveau de la machine: le compilateur élimine toujours la récursivité ("Dérécursiver") en construisant un programme itératif. Ceci permet d'économiser l'espace de la pile d'exécution.

➔ Dérécursiver: c'est transformer un algorithme récursif en un algorithme itératif équivalent.

2

II. Élimination de la récursivité terminale simple

Rappel: un algorithme est dit récursif terminal s'il ne contient aucun traitement après un appel récursif.

Procédure **ALGO (X) // Liste des paramètres**
Si (**condition) alors // condition portant sur X**
<Traitement 1> // traitement de base de l'algorithme (dépendant de X)
ALGO (F(X)) // F(X) représente la transformation des paramètres
// Rien
Sinon
<Traitement 2> // Traitement de terminaison (dépendant de X)
Finsi

3

II. Élimination de la récursivité terminale simple

Algorithme Récursif	Algorithme Itératif
Proc **Algo-Rec (…, resultat)** Début **Si (condition) alors** {traitement de réduction du prob } Algo-Rec (…, F(resultat)) {Attention : pas d'autres instructions ici sinon la sol n'est pas terminale } **Sinon** {traitement de la condition d'arrêt } **Finsi** **Fin proc.**	Proc **ALgo-Iter(…,resultat)** Début **Tant que (condition) faire** {traitement de réduction du prob } resultat← **F**(resultat) **FinTQ** {traitement de la condition d'arrêt } **Fin proc.**

4

II. Élimination de la récursivité terminale simple

Algorithme Récursif	Algorithme Itératif
Fonction Facto1 (n: entier, resultat: entier): entier {précond: initialement resultat = 1} Début **Si (n>1) alors** retourne Facto1 (n-1, n* resultat) **Sinon** retourne resultat Fin Fn	Fonction Facto2 (n: entier, resultat: entier): entier {précond: initialement resultat = 1} Début **Tant que (n>1) faire** resultat←n * resultat n←n-1 **FTQ** retourne resultat Fin Fn.

II. Élimination de la récursivité terminale simple

Exercice 1:

1) Ecrire une fonction récursive permettant de calculer le PGCD de deux nombres naturels non nul (a, b) en utilisant l'algorithme d'Euclide.

2) Dérécursiver cette fonction.

$$PGCD(a,b) = \begin{cases} a & \text{si } a=b \\ PGCD(a-b, b) & \text{si } a>b \\ PGCD(a, b-a) & \text{si } b>a \end{cases}$$

II. Élimination de la récursivité terminale simple

Solution de l'exercice 1:

II. Élimination de la récursivité terminale simple

Exercice 2:

1) Ecrire une fonction récursive (basée sur l'algorithme d'Euclide) permettant de vérifier si a est un diviseur de b.

2) Dérécursiver cette fonction.

II. Élimination de la récursivité terminale simple

Solution de l'exercice 2:

Algorithme Récursif	Algorithme Itératif

```
Fonction Diviseur (a,b) : Bool
Si (a <=0) Alors
   Retourner(Faux)
Sinon
   Si (a>=b) Retourner (a=b)
   Sinon
   Retourner (Diviseur (a,b-a))
   Fin Si
Fin Si
Fin
```

9

III. Élimination de la récursivité non terminale simple

➢Rappel: Dans un algorithme récursif non terminal, l'appel récursif est suivi d'un traitement.
➔ Traitement des instructions situées après l'appel récursif.
➔ Traitement de la phase de la remontée (Phase de retour arrière :calculs)

Procédure **ALGO (X) // Liste des paramètres**
Si (**condition) alors // condition portant sur X**
<Traitement 1> // traitement de base de l'algorithme (dépendant de X)
ALGO (F(X)) // F(X) représente la transformation des paramètres
<Traitement 2>
Sinon
<Traitement 3> // Traitement de terminaison (dépendant de X)
Finsi

10

III. Élimination de la récursivité non terminale simple

- Il faut donc sauvegarder le contexte de l'appel récursif (typiquement les paramètres de l'appel engendrant l'appel récursif) sur une pile.

- Les piles sont des structures de stockage qui fonctionnent sur le principe LIFO " Last In First Out ": le dernier entré est le premier sorti »

- Le principe est de simuler la pile d'appels des processus pour éliminer la récursivité non terminale.

Chapitre 1: La récursivité

III. Élimination de la récursivité non terminale simple

Algorithme Récursif	Algorithme Itératif
Procédure **ALGO-Rec (n)** Début **Si (condition) alors** <Traitement 1> ALGO-Rec (F(n)) <Traitement 2> **Sinon** {traitement de la condition d'arrêt } <Traitement 3> **Finsi** Fin	Procédure **ALGO-Iter (n)** **Var** Res:… Début **créer_pile ( )** **Tant que (condition) faire** **Empiler (n)** <Traitement 1> n←F(n) **FinTQ** <Traitement 3> **Tant que (Non PileVide ( )) faire** {calcul du résultat en fonction du résultat et du dernier élément de la pile} Res← **Calcul_Dépend (Res , dernier ( )) ;** **dépiler ( )** <Traitement 2> **FinTQ** Fin

III. Élimination de la récursivité non terminale simple

Algorithme Récursif	Algorithme Itératif
Fonction Facto (n : Entier) : Entier Var Res: entier Début **Si** (n > 1) **Alors** Res← n * Facto (n-1) **Sinon** Res ← 1 **FinSi** retourne Res Fin Fn.	Fonction Facto (n : Entier) : Entier Var Res: entier Début **créer_pile ( )** **Tant que (n>1) faire** **empiler (n)** n ← n-1 **FinTQ** Res←1 **Tant que (Non Pilevide ( )) faire** Res←Res ***dernier ( )** **dépiler ( )** **FinTQ** retourne Res ; Fin Fn.

III. Élimination de la récursivité non terminale simple

Exercice 3:

1) Proposer une procédure récursive non terminale qui permet d'afficher les éléments d'un tableau d'entier de droite à gauche.

2) Dérécursiver cette procédure.

Exercice 4:

1) Proposer une procédure récursive non terminale qui permet d'afficher les éléments d'un tableau d'entier de gauche à droite.

2) Dérécursiver cette procédure.

III. Élimination de la récursivité non terminale simple

Solution de l'exercice 3:

Algorithme Récursif	Algorithme Itératif
Proc AfficherDroiteGauche (T : Tab entier, n: entier, i: entier) Début **Si (i<=n) alors** AfficherDroiteGauche (T, n, i+1) Ecrire (**T[i]**) **Fin Si** **Fin** **{Remarque:** <Traitement 1>= **Vide** <Traitement 2>= **Ecrire (T[i])** <Traitement 3>= **vide }**	Proc AfficherDroiteGauche (T : Tab entier, n: entier, i: entier) Début **créer_pile ( )** **Tant que (i<=n) faire** **empiler (T[i])** i ← i+1 **FinTQ** **Tant que (Non PileVide ( )) faire** Écrire **(dernier ( ))** **dépiler ( )** **FinTQ** Fin Proc.

III. Élimination de la récursivité non terminale simple

Solution de l'exercice 4:

Algorithme Récursif	Algorithme Itératif
Proc Afficher (T : Tab entier, n: entier) Début **Si (n>0) alors** Afficher (T, n-1) Ecrire (**T[n]**) **Fin Si** **Fin** **{Remarque:** <Traitement 1>= **Vide** <Traitement 2>= **Ecrire (T[i])** <Traitement 3>= **vide }**	Proc Afficher (T : Tab entier, n: entier) Début **créer_pile ( )** **Tant que (n>0) faire** **empiler (n)** n←n-1 **FinTQ** **Tant que (Non PileVide ( )) faire** Écrire **(T[dernier ( )])** **dépiler ( )** **FinTQ** Fin Proc.

IV. La dérécursivation….conclusion

Les programmes itératifs sont souvent plus efficaces,

mais les programmes récursifs sont plus faciles à écrire.

Les compilateurs savent, la plupart du temps, reconnaître les appels récursifs terminaux, et ceux-ci n'engendrent pas de surcoût par rapport à la version itérative du même programme.

Il est toujours possible de dérécursiver un algorithme récursif.