Cours Algorithmes avancés
Master 2 Réseaux et sécurité informatique

Réalisé par :Mme Merzougui
Département d'informatique
Université Badji Mokhtar-Annaba
Année 2020/2021

Organisation

- **Cours théoriques** :Les Lundis de 10:30 à 11:30 à la salle conférence.
- **Travaux dirigés** : Les Lundis de 11:30 à 12:30 à la salle conférence.
- **Travail pratique** : Implémentation de quelques méthodes étudiées en cours.
- **Réalisation** : C, C++, JAVA
- **Evaluation :** TP (50%) Examen écrit (50%).

Objectifs de cours

- Avoir des outils de programmation performante pour construire des algorithmes optimisés ;
- Utilisation des structures des données pour minimiser les espaces mémoires utilisée pour le stockage des données des programmes .

Plan de Cours

- Chapitre 1: Rappel sur la récursivité
- Chapitre 2: Arbres binaires de recherche
- Chapitre 3 :Arbres équilibrés
- Chapitre 4:Arbre Bicolore
- Chapitre 5: Méthodes de hachage

Chapitre I :
Rappel sur la récursivité

Chapitre I :
Rappel sur la récursivité

- **Principe**
- **Utilisation**
- **Exemples**

Chapitre I : Rappel sur la récursivité

Définition

- Moyen simple et élégant pour résoudre certain problème.
- Tout objet est dit récursif s'il se définit à partir de lui- même
- On appelle récursive toute fonction ou procédure qui s'appelle elle-même

Chapitre I : Rappel sur la récursivité

Principes de la récursivité

Décomposer le problème en un problème plus simple ⇒ réduire la taille du problème considéré.

- Pour la récurions sur des entiers : la taille du problème est dénie par un entier, on réduit la valeur de cet entier à chaque appel récursif.

- Pour la récurions sur les tableaux : Soit on considère la taille du tableau, on réduit la taille du tableau considéré à chaque appel récursif I Ou bien on utilise un ou des indices qui varient à chaque appel pour tendre vers la condition d'arrêt (dépendant des valeurs des indices).

Chapitre I : Rappel sur la récursivité

Une fonction récursive doit comporter :

- Un cas d'arrêt dans lequel aucun autre appel n'est effectué .
- Un cas général dans lequel un ou plusieurs autres appels sont effectués La chaîne d'appel doit conduire au critère d'arrêt
- Optionnellement, des cas impossibles ou incorrects à traiter par des exceptions

Chapitre I : Rappel sur la récursivité

```
void p (…) {
        if (fin) {
                        …….
                    }
else {
… p (…)
        }
}
```

Pas d'appel récursif (partie alors)

La procédure p s'appelle elle- même une ou plusieurs fois (partie « sinon »)

Chapitre I : Rappel sur la récursivité

Utilisation

- Récursivité et Récurrence Deux notions très proche : mathématiques : récurrence informatique : récursivité
- De nombreuses définitions mathématiques sont récursives :

Définition (Peano)

0 est un entier naturel. Tout entier n a un successeur unique Sn (= n + 1) ; Tout entier sauf 0 est le successeur d'un unique entier ; Pour tout énoncé P(n) si P(0) est vrai et si pour tout n, P(n) implique P(Sn) alors l'énoncé
∀n : P(n) est vrai.

Chapitre I : Rappel sur la récursivité

Récursivité terminale

- On dit qu'un fonction est récursive terminale, si tout appel récursif est de la forme ***return f(...)***
- Un algorithme récursif simple est *terminal* lorsque l'appel récursif est la dernière chose effectuée. Généralement ce genre d'algorithme peut facilement être transformé en une boucle.

Exemple:

si N = 0 retourner 1
sinon retourner N x Fact(N-1)

Chapitre I : Rappel sur la récursivité

Récursivité multiple

- Un algorithme récursif est *multiple* si l'un des cas qu'il distingue se résout avec plusieurs appels récursifs.
- un algorithme à récursivité multiple ne peut pas être terminal.

Exemple (tours de Hanoï)

```
si n>0
aux← le piquet différent de d et a
hanoi(n−1,d,aux)
deplacer_dique(d,a)
hanoi(n−1,aux,a)
fin si
```

Chapitre I : Rappel sur la récursivité

Récursivité Mutuelle

- Deux algorithmes sont *mutuellement récursifs* si l'un fait appel à l'autre et l'autre à l'un. On peut étendre cette définition à un nombre quelconque d'algorithmes ..

Exemple

Predicat de parité
Entrée : n∈
Sortie : *Vrai* si n est pair, *Faux* sinon
si n=0
Renvoyer *Vrai*
sinon
Renvoyer impair(n−1)
fin si

Predicat d'imparité
Entrée : n∈N
Sortie : *Vrai* si n est impair, *Faux* sinon
si n=0
Renvoyer *faux*
sinon
Renvoyer pair(n−1)
fin si

Chapitre I : Rappel sur la récursivité

Calcule la factorielle d'un nombre entier

Algorithme Fact

Entrée : un entier positif N
Sortie : factorielle de N
  si N = 0 retourner 1 //condition d'arret
  sinon retourner N x Fact(N-1) // appel récursive

Chapitre I : Rappel sur la récursivité

Comment ça marche ?

```
Appel à fact(4)
  4*fact(3) = ?
      Appel à fact(3)
          3*fact(2) = ?
              Appel à fact(2)
                  2*fact(1) = ?
                      Appel à fact(1)
                        1*fact(0) = ?
                            Appel à fact(0)
                         Retour de la valeur 1
                        1*1
                       Retour de la valeur 1
                   2*1
              Retour de la valeur 2
            3*2
       Retour de la valeur 6
     4*6
  Retour de la valeur 24
```

Chapitre I : Rappel sur la récursivité

Constatations

- L'écriture sous forme récursive est toujours plus simple que l'écriture sous forme itérative

Une question

- Une même fonction est-elle plus efficace sous forme récursive ou sous forme itérative ? (Ou, sous une autre forme, y a-t-il un choix optimal généralisable ?)
- **La réponse** : est non

Chapitre I : Rappel sur la récursivité

- **En revanche**

  La plupart des traitements itératifs simples sont facilement traduisibles sous forme récursive (exemple du for)

- L'inverse est faux : Il arrive même qu'un problème ait une solution récursive triviale alors qu'il est très difficile d'en trouver une solution itérative