CM 3

Deux parties sont abordées. La conception par une approche « diviser pour régner » et les algorithmes avec retour arrière.

LECTURES :

Chapitre 5 IAAD : introduction et un exemple pour avoir compris le concept de diviser pour régner. (IAAD= Introduction to Algorithl Analysis and Design Levitin)

Chapitre 12.1 IAAD : backtracking

Supports de cours (revoir cours 1ère année sur la récursivité pour le backtrack)

QUESTIONS, REFLEXIONS (DIVISER)

Soit l'algorithme suivant qui calcule le maximum d'un tableau

```
Methode Maxtab(T: tableau d'element;
                                left,right: indice ): element
    Si left = right alors
        retourner T[left]
    Sinon
        m←(left+right)/2
        u ←Maxtab(T,left,m)
        v ← Maxtab(T,m+1,right)
        retouner max (u,v)
    fsi
```

Quelle est la complexité de cet algorithme.

Proposer un algorithme qui calcule la hauteur d'un arbre /qui recherche une valeur dans un arbre (sans hypothèse particulière) en utilisant un principe simulaire.

RÉFLEXIONS (RETOUR ARRIÈRE)

Cavalier sur équichier

On pose un cavalier sur un échiquier. Comment trouver une suite de mouvements tel que le cavalier parcourt toutes les cases de l'échiquier ?

Proposer un algorithmes utilisant le retour arrière pour résoudre ce problème. Ce qui nous intéresse est l'écriture de l'algorithme de recherche de solution (et donc on supposera l'existence de fonctions qui permettent l'écriture de cet algorithme comme par exemple une fonction qui étant donnée une position retourne la liste des positions accessibles selon le mouvement d'un cavalier).

Formalisez le problème sous la forme d'un algorithme récursif :

a) Paramètres de l'algorithme récursif
b) Cas trivial (triviaux)
c) Cas récursif
d) 1er appel

TD3

DIVISER POUR REGNER

Question d'exam

Il est attendu une justification claire de vos réponses.

Ecriture d'un nombre écrit en base 10 dans une base b

On vous donne deux algorithmes réalisant le traitement évoqué ci-dessus.

a) Premier algorithme itératif (n et base sont deux entiers donnés) ***(3 points)***

```
String res ="";
while (n > 0) {
    res=  n % base +res;
    n = n / base;
}
```

complexité constante

Comment calcule-t-on la complexité d'un algorithme utilisant une itération ?

$\sum_i C(1)$

Comment ce principe s'applique ici ?

Quelle est donc la complexité de la version itérative du changement de base ?

b) Deuxième algorithme ***(3 points)***

```java
public static String convert(int nb, int base) {
    if (nb < base) {
        return "" + nb;
    } else {
        return convert(nb / base, base) + nb % base;
    }
}
```

Quel principe général / théorème applique-t-on pour calculer la complexité de ce type d'algorithme (type = diviser pour régner) ?

Théorème maître pour calculer la complexité du algorithme : $T(n) = aT(\frac{n}{b}) + O(n^d)$

- a >= 1, b> 1
- On divise un problème de taille ***n*** en a sous problèmes chacun de taille ***n/b***, le coût de diviser et combiner le problème est de ***f(n) ($O(n^d)$)***

Comment ce principe s'applique ici ?

Quelle est donc la complexité de la version diviser pour régner du changement de base

c) Conclusion ***(1 point)***

Commentaires sur les complexités respectives, explications, …?

Pouvez-vous montrer expérimentalement que la classe de complexité trouvée est correcte ?

RETOUR ARRIERE

Balance

Reprendre le problème de la balance vu en cours, écrivez son algorithme, codez-le

Transformez le problème initial en un problème d'optimisation : on veut minimiser la différence de poids entre les deux plateaux. Affichez la solution/affichez toutes les solutions.

Codez votre solution et montrez son bon fonctionnement.

Balance généralisée

Une ville dispose de k alimentations en électricité. Elle est constituée de N quartiers (N> k) chacun (noté i) demandant une puissance électrique maximale p_i. Comment répartir les quartiers sur chacune des alimentations en minimisant les déséquilibres ?

Vous vous inspirerez de l'exemple de la balance à plateaux en considérant k plateaux (k > 2).

Codez votre solution et montrez son bon fonctionnement.

Problème du rendu de monnaie

Reprendre le problème de rendu de monnaie et proposer une solution sous la forme d'un algorithme de retour arrière (cf préparation du cours).

Codez votre solution et montrez son bon fonctionnement.

Proposez une solution qui minimise (ou maximise) le nombre de pièces utilisées.

AUTRES PROPOSITIONS

Faire l'algo et le coder

Plus proches points dans un plan selon diviser pour régner

Voir 5.5 dans Levitin

Version backtrack de send+more = money