Complexité des algorithmes [cx]
Exercices de cours

Karine Zampieri, Stéphane Rivière, Béatrice Amerein-Soltner

Unisciel algoprog Version 21 mai 2018

Table des matières

alg - Exercices de cours (Solution)

Mots-Clés Complexité des algorithmes ■
Difficulté ●●○ (1 h) ■

1 Appréhender le cours

2 Appliquer le cours

2.1 Recherche du maximum

Objectif

Cet exercice évalue la complexité en nombre de comparaisons de la recherche du maximum dans un `Tableau` définit comme suit :

Définitions PseudoCode

```
Constante TMAX <- ...
Typedef ITableau = Entier[TMAX]
```

Écrivez une fonction `maximumTab(t,n)` qui calcule et renvoie la plus grande valeur des `n` premières valeurs d'un `ITableau t`.

Solution Paramètres

Entrants : Un `ITableau t` et un entier `n`

Résultat de la fonction : Un entier (type des éléments du tableau)

Validez votre fonction avec la solution.

Solution alg @[UtilsTBOpers.alg]

```
Fonction maximumTab ( DR t : Entier [ TMAX ] ; n : Entier ) : Entier
Variable vmax : Entier
Variable ix : Entier
Début
  |  vmax <- t [ 1 ]
  |  Pour ix <- 2 à n Faire
  |     |  Si ( vmax < t [ ix ] ) Alors
  |     |     |  vmax <- t [ ix ]
  |     |  FinSi
  |  FinPour
  |  Retourner ( vmax )
Fin
```

Quelle est la complexité de votre algorithme en nombre de comparaisons ?

Solution simple

Il y a $n-1$ comparaisons.

Montrez qu'il est optimal.

Solution simple

Tout élément hormis le maximum induit une comparaison, sinon, on ne peut pas savoir qu'il n'est pas le maximum. Il y a $n-1$ tels éléments. Tout algorithme de recherche du maximum dans un tableau quelconque doit donc faire au moins $n-1$ comparaisons.

2.2 Schéma de Hörner

Utilise Complexités en temps ■
Durée estimée 15 min ■

Schéma de Hörner
Le schéma de HÖRNER évalue la valeur d'un polynôme pour une valeur de la variable. Il est basé sur la réécriture :

$$\begin{aligned} P(x) &= a_0 + a_1 x + \ldots + a_n x^n \\ &= a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + x(a_n))\ldots)) \end{aligned}$$

Écrivez une fonction `evalHorner(p,n,x)` qui calcule et renvoie la valeur d'un polynôme `p` (tableau de réels) d'ordre `n` (entier) en `x` (réel).

Validez votre fonction avec la solution.

Solution alg @[evalHorner.alg]

```
Fonction evalHorner ( p : Réel [ 0..NMAX ] ; n : Entier ; x : Réel ) : Réel
Variable rs : Réel
Variable ix : Entier
Début
  |  rs <- 0.0
  |  Pour ix <- n à 0 Pas - 1 Faire
  |     |  rs <- p [ ix ] + ( rs * x )
  |  FinPour
  |  Retourner ( rs )
Fin
```

Quelle est la complexité de votre algorithme en nombre d'opérations ?

Solution simple
L'analyse de complexité du schéma de HÖRNER est exacte et ne dépend pas de la configuration des données. On fait $n+2$ affectations, $n+1$ additions et $n+1$ multiplications. La complexité est donc en $\Theta(n)$.

2.3 Tours de Hanoï

Utilise Complexités en temps ■
Durée estimée 20 min ■

Objectif
L'algorithme des Tours de Hanoi est défini par :
Pour déplacer n disques de la tige A vers la tige C :

1. On déplace les $(n-1)$ plus petits de la tige A vers la tige B.
2. On déplace le plus gros disque de la tige A vers la tige C.
3. On déplace les $(n-1)$ plus petits de la tige B vers la tige C.

L'algorithme s'écrit comme suit :

```
Action hanoi ( n : Entier ; orig , dest , inter : Chaîne )
Début
  |  Si ( n > 0 ) Alors
  |    |  hanoi ( n - 1 , orig , inter , dest )
  |    |  deplacer ( orig , dest )
  |    |  hanoi ( n - 1 , inter , dest , orig )
  |  FinSi
Fin
```

On note $H(n)$ le nombre de passages d'un disque d'un piquet à un autre.
Donnez la relation de récurrence.

Solution simple
On a clairement :

$$\begin{cases} H(0) = 0 \\ H(n) = 2H(n-1) + 1 \end{cases}$$

Résolvez la récurrence.

Solution simple
On a :

$$\begin{aligned} H(n) &= 2H(n-1) + 1 \\ 2H(n-1) &= 2^2 H(n-2) + 2 \\ 2^2 H(n-2) &= 2^3 H(n-3) + 2^2 \\ &\dots \\ 2^{n-1} H(1) &= 2^n H(0) + 2^{n-1} \end{aligned}$$

Par sommation des équations membre à membre :

$$H(n) = 2^n H(0) + 1 + 2 + \ldots + 2^{n-1} = 2^n - 1$$

Déduisez la complexité de la procédure.

Solution simple

Si l'on compte la gestion des appels récursifs et la comparaison à zéro, opérations en $\Theta(1)$, la complexité de l'algorithme, toutes opérations confondues est en $\Theta(2^n)$.

Concluez.

Solution simple

On peut donc dire :

- La complexité est exponentielle donc l'algorithme est impraticable pour les grandes valeurs de n.
- L'algorithme est optimal car on peut montrer qu'il faut au moins $2^n - 1$ passages pour résoudre le problème.
- On aura rapidement des problèmes de mémoire relatifs à la gestion de la récursivité.