

Complexité des algorithmes [cx] Exercices de cours

Karine Zampieri, Stéphane Rivière, Béatrice Amerein-Soltner

Unisciel  algoprogram  Version 21 mai 2018

Table des matières

1	Appréhender le cours	2
2	Appliquer le cours	2
2.1	Recherche du maximum	2
2.2	Schéma de Hörner	4
2.3	Tours de Hanoi	5

C++ - 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` défini comme suit :



Définitions C++

```
const int TMAX = ...;
using ITableau = int[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)

Solution Profil C++

```
int maximumTab(const ITableau& t, int n)
```



Validez votre fonction avec la solution.

Solution C++ @[UtilsTBOpers.cpp]

```
/**
 * Maximum d'un ITableau
 * @param[in] t - un ITableau
 * @param[in] n - nombre de valeurs dans [1..TMAX[
 * @return le maximum des n valeurs de t
 */
int maximumTab(const ITableau& t, int n)
{
    int vmax = t[0];
    for (int j = 1; j < n; ++j)
    {
        if (vmax < t[j])
        {
            vmax = t[j];
        }
    }
}
```

```
}  
return vmax;  
}
```



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 :

$$\begin{aligned} P(x) &= a_0 + a_1x + \dots + a_nx^n \\ &= a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-1} + x(a_n)) \dots)) \end{aligned}$$



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 C++ @[evalHorner.cpp]

```
double evalHorner(const double p[], int n, double x)
{
    double rs = 0.0;
    for (int j = n; j >= 0; --j)
    {
        rs = p[j] + rs * x;
    }
    return rs;
}
```



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 )
|   | déplacer ( 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 :

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

Par sommation des équations membre à membre :

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



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.