Complexité

INFORMATIQUE COMMUNE - TP nº 2.2 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ calculer la complexité d'un algorithme simple
- ☞ donner les complexités dans le pire des cas des tris comparatifs génériques
- ☞ expliquer la différence entre le tri rapide et le tri fusion

A Complexité d'algorithmes simples

A1. Calculer la complexité de l'algorithme 1. Y-a-il un pire et un meilleur des cas ?

Algorithme 1 Produit scalaire

Fonction PRODUIT_SCALAIRE(x, y) ▷ x et y sont des vecteurs à n éléments
    $s \leftarrow 0$
    pour $i = 0$ à $n - 1$ **répéter**
        $s \leftarrow s + x_i y_i$ ▷ coût ?
    renvoyer s

A2. Calculer la complexité de l'algorithme 2 dans le meilleur et dans le pire des cas.

Algorithme 2 Palindrome

Fonction PALINDROME(w)
    $n \leftarrow$ la taille de la chaîne de caractères w
    $i \leftarrow 0$
    $j \leftarrow n - 1$
    tant que $i < j$ **répéter**
        si $w[i] = w[j]$ **alors**
            $i \leftarrow i + 1$
            $j \leftarrow j - 1$
        sinon
            renvoyer Faux
    renvoyer Vrai

A3. Calculer la complexité de l'algorithme 3. Y-a-t-il un pire et un meilleur des cas ? On fait l'hypothèse que la fonction *power*(a,b) est de complexité logarithmique $O(1)$. Cette hypothèse vous semble-t-elle raisonnable ?

Algorithme 3 Évaluation simple d'un polynôme

Fonction EVAL_POLYNÔME(p, v)
    $d \leftarrow$ degré de p
    $r \leftarrow$ p[0]
    pour $i = 1$ à d **répéter**
        $r \leftarrow r +$ p[i] $\times$POWER(v, i)
    renvoyer r

A4. Utiliser la méthode de Horner pour écrire un autre algorithme pour évaluer un polynôme. Quelle complexité pouvez-vous obtenir ? Est-ce plus rapide ?

B Tri fusion

Algorithme 4 Tri fusion

Fonction TRI_FUSION(t, g, d)
    si g < d **alors**
        m ← (g+d)//2 ▷ on découpe au milieu
        TRI_FUSION(t, g, m)
        TRI_FUSION(t, m+1, d)
        FUSION(t, g, m, d)
▷ Sinon on n'arrête !

B1. Programmer le tri fusion en Python.

B2. Quelle est la complexité de cet algorithme ? Y-a-t-il un pire et un meilleur cas ?

B3. Vérifier, en mesurant le temps d'exécution, la justesse de votre calcul précédent.

Algorithme 5 Fusion de sous-tableaux triés

```
Fonction FUSION(t, g, m, d)
    ng ← m - g +1
    nd ← d - m
    G, D ← deux tableaux de taille ng et nd
    pour k de 0 à ng répéter
        G[k] ← t[g + k]
    pour k de 0 à nd répéter
        D[k] ← t[m + 1 + k ]
    i ← 0
    j ← 0
    k ← g
    tant que i < ng et j <nd répéter
        si G[i] ⩽ D[j] alors
            t[k] ← G[i]
            i ← i + 1
        sinon
            t[k] ← D[j]
            j ← j + 1
        k ← k + 1
    tant que i < ng répéter
        t[k] ← G[i]
        i ← i + 1
        k ← k + 1
    tant que j < nd répéter
        t[k] ← D[j]
        j ← j + 1
        k ← k + 1
```

C Tri rapide

Algorithme 6 Tri rapide

```
Fonction TRI_RAPIDE(t, p, d)
    si p < d alors
        i_pivot ← PARTITION(t, p, d)
        TRI_RAPIDE(t, p, i_pivot -1)
        TRI_RAPIDE(t, i_pivot+1, d)
                                        ▷ Sinon on n'arrête!
```

Algorithme 7 Partition en deux sous-tableaux

```
Fonction PARTITION(t, p, d)
    i_pivot ← un nombre au hasard entre p et d inclus
    pivot ← t[i_pivot]
    ÉCHANGER(t,d,i_pivot)          ▷ On met le pivot à la fin du tableau
    i = p - 1                      ▷ i va pointer sur le dernier élément du premier tableau
    pour j = p à d -1 répéter
        si t[j] ⩽ pivot alors
            i ← i + 1
            ÉCHANGER(t,i,j)        ▷ On échange les places de t[i] et t[j]
    t[d] ← t[i+1]                  ▷ t[i+1] appartient au tableau de droite
    t[i+1] ← pivot                 ▷ Le pivot est entre les deux tableaux
    renvoyer i + 1                 ▷ La place du pivot!
```

C1. Programmer le tri rapide en Python.

C2. Quelle est la complexité de cet algorithme? Y-a-t-il un pire et un meilleur cas?

C3. Vérifier, en mesurant le temps d'exécution, la justesse de vos calculs précédents.