TP1 - Sous-séquences maximales

Sébastien Delecraz Éloi Perdereau

30 septembre 2013

Résumé

Ce rapport propose une étude de différents algorithmes pour résoudre le problème du calcul de la sous-séquence de somme maximale dans un tableau d'entier. Quatre algorithmes de complexité différentes sont présentés ici. On trouvera donc pour chacun leur pseudo-code, une étude de leur complexité et des tests de performance. Enfin nous confronterons les différents résultats obtenus et l'analyse théorique réalisée.

1 Analyse théorique

1.1 Algorithme naïf

Algorithme 1 : Naïf

Entrées : T, n
Sorties : Sous-séquence maximale
début
  $S_{max} \leftarrow -\infty$
  pour tous les $1 \leq k \leq n$ **faire**
    pour tous les $k \leq l \leq n$ **faire**
      $S \leftarrow 0$
      pour tous les $k \leq j \leq l$ **faire**
        $S \leftarrow S + T[j]$
      fin
      si $S > S_{max}$ **alors**
        $S_{max} \leftarrow S$
      fin
    fin
  fin
  retourner S_{max}
fin

Complexité

$$O(n^3)$$

1.2 Algorithme moins naïf

Algorithme 2 : Moins naïf

Entrées : T, n
Sorties : Sous-séquence maximale
début
  $S_{max} \leftarrow -\infty$
  pour tous les $1 \leq k \leq n$ **faire**
    $S \leftarrow 0$
    pour tous les $k \leq l \leq n$ **faire**
      $S \leftarrow S + T[l]$
      si $S > S_{max}$ **alors**
        $S_{max} \leftarrow S$
      fin
    fin
  fin
  retourner S_{max}
fin

Complexité

$$O(n^2)$$

1.3 Algorithme diviser pour régner

Complexité

$$O(n \log n)$$

1.4 Algorithme incrémental

Complexité

$$O(n)$$

Algorithme 3 : Diviser pour régner

```
Entrées : T, k, l
Sorties : Sous-séquence maximale
début
|  si l − k = 1 alors retourner T[k];
|  si l − k = 2 alors retourner max{T[k], T[k+1], T[k]+T[k+1]};
|  j ← (l−k)/2
|  S1 ← Diviser_pour_regner(T, k, j)
|  S2 ← Diviser_pour_regner(T, j+1, l)
|  S3 ← Stmp ← T[j]
|  pour i variant de j-1 à k descendant faire
|  |  Stmp ← Stmp + T[i]
|  |  si Stmp > S3 alors
|  |  |  S3 ← Stmp
|  |  fin
|  fin
|  S4 ← Stmp ← T[j]
|  pour i variant de j+1 à l-1 montant faire
|  |  Stmp ← Stmp + T[i]
|  |  si Stmp > S4 alors
|  |  |  S4 ← Stmp
|  |  fin
|  fin
|  S0 ← S3 + S4 − T[j]
|  retourner max{S0, S1, S2}
fin
```

Algorithme 4 : Incrémental

Entrées : T, n

Sorties : Sous-séquence maximale

```
début
    S_max ← T[1]
    S_1 ← S_2 ← 0
    pour tous les 2 ≤ i ≤ n faire
        S_1 ← S_1 + T[i]
        S_2 ← S_2 + T[i]
        si S_2 < 0 alors
            S_2 ← 0
        fin
        si S_1 > 0 alors
            S_max ← S_max + S_1
            S_1 ← S_2 ← 0
        fin
        si S_2 > S_max alors
            S_max ← S_2
            S_1 ← S_2 ← 0
        fin
    fin
    retourner S_max
fin
```