Feuille de Travaux Pratiques

Projet : Etude et Comparaison d'Algorithmes

Janvier-Février 2016

Pour ce projet, **6 séances (1h20 chacune)** de TP seront consacrées, et le projet sera à effectuer en binômes (ou seul/e si vous le désirez). Les trinômes sont interdits.

La programmation se fera dans le langage de votre choix.

A l'issue des 6 séances, vous rendrez l'ensemble de votre projet sous la forme d'une archive <NOM1-NOM2>.zip. La décompression de cette archive doit produire un répertoire <NOM1-NOM2> contenant tous les éléments de votre travail (rapport, sources, exécutable, jeux d'essai le cas échéant), ainsi qu'un fichier texte README.txt contenant les instructions de compilation et d'exécution.
Assurez-vous que vos programmes compilent et fonctionnent sous Linux dans les salles machine du CIE.

Cette archive est à **déposer sur Madoc**. La date limite de restitution est fixée au **Jeudi 25 février 2016 à 22h**. Aucun retard ne sera accepté.

Par ailleurs, un malus sera appliqué lorsqu'une ou plusieurs des consignes ci-dessus n'auront pas été respectées.

IMPORTANT : *dans ce projet, la programmation ne présente pas de grande difficulté technique. La très grande majorité de la note sera donc donnée en fonction :*
- *des qualité et quantité des tests effectués*
- *de la qualité de l'analyse faite à partir de ces tests*
- *de la qualité du rapport*

1 Introduction : le problème MaxSomme

On se donne un tableau T de taille n, dont les cases sont indicées de 1 à n. T contient des entiers, positifs ou négatifs, mais jamais 0. D'autre part, T contient au moins un entier positif et ne contient pas deux fois le même élément.

Le problème qu'on se pose est le suivant : trouver les deux indices i et j, $1 \leq i \leq j \leq n$, tels que la somme $S_{i,j} = \Sigma_{p=i}^{j} T[p]$ des éléments de T entre $T[i]$ et $T[j]$ (inclus) soit *la plus grande possible*.

Par exemple, si $T = [21, -31, \underline{39, 26, -50, 28, 17}, -53, -32, 42]$, alors la plus grande somme $S_{i,j}$ est $S_{3,7} = 60$.

Dans la suite, on appellera **MaxSomme** ce problème. Le but de ce projet est d'étudier quatre algorithmes qui résolvent **MaxSomme**. Plus précisément, pour chacun de ces algorithmes, il est demandé de l'implémenter, de calculer sa complexité d'une part de manière théorique, d'autre part de manière expérimentale à partir d'une série de tests, et enfin de comparer les résultats obtenus.

2 Travail demandé

Pour chacun des quatre algorithmes décrits dans le Section 3, il est demandé de répondre aux questions suivantes :

1. Implémenter l'algorithme, et le modifier de telle manière qu'en plus de la somme $S_{i,j}$ recherchée, il affiche à l'écran les indices i et j.
2. Déterminer sa complexité de manière théorique, en donnant les détails des calculs dans le rapport. On attend ici une complexité sous la forme d'un $O()$ ou d'un $\Theta()$, en étant bien entendu le plus précis possible.
3. Exécuter l'algorithme sur des tableaux T de plus en plus grands, et faire des tests de la manière suivante : pour chaque valeur de n, générer *au hasard* 10 tableaux de taille n, et relever le temps moyen d'exécution de l'algorithme sur ces 10 tableaux.
   Remarque : le temps nécessaire à la *génération* des tableaux ne doit pas être compté.
   Remplir le tableau ci-dessous, qui indique les temps d'exécution moyens de chacun de vos tests. Ces temps d'exécution doivent systématiquement être donnés en *secondes (s)*, avec trois chiffres significatifs. Si, pour une taille n de tableau donnée, le temps d'exécution moyen dépasse 60 secondes, indiquer "PAR" (Pas Assez Rapide) ; en cas de dépassement mémoire indiquer "DM". Tant qu'on n'est pas dans le cas PAR ou DM, augmenter la valeur de n (suivant le principe décrit dans le tableau ci-dessous), et continuer les tests. Dans le cas contraire, on arrête les tests pour cet algorithme.

Algorithme			
Valeur de n	Temps d'exécution moyen	Valeur de n	Temps d'exécution moyen
1		100	
2		200	
3		400	
4		600	
5		800	
6		1000	
7		2000	
8		4000	
9		6000	
10		8000	
20		10000	
40		20000	
60		40000	
80		etc.	

4. Sur un graphique, et à partir des résultats précédents (et éventuellement de résultats supplémentaires), tracer la courbe représentant le temps d'exécution en fonction de n (un graphique par algorithme).
5. Déduire de façon expérimentale (c'est-à-dire sur la base des Questions 3. et 4.) la complexité de l'algorithme étudié. On attend ici une complexité (a) d'abord sous la forme d'une fonction précise (la plus proche possible des courbes obtenues), (b) puis sous la forme d'un $O()$ ou d'un $\Theta()$. La méthode qui vous a permis de déterminer la complexité (a) doit être expliquée en détail dans le rapport.
6. Pour chaque algorithme, comparer les complexités expérimentale et théorique obtenues précédemment. Si une différence apparaît entre ces deux complexités, proposer une (ou plusieurs) explication(s) à ce phénomène.

3 Les quatre algorithmes résolvant MaxSomme

Les Algorithmes 1 et 2 sont décrits en français, et les Algorithmes 3 et 4 sont donnés en pseudo-code.

Algorithme 1 : on génère toutes les paires possibles de positions i et j, $1 \leq i \leq j \leq n$, et pour chaque possibilité on calcule la somme $S_{i,j}$. On retourne le $S_{i,j}$ le plus grand.

Algorithme 2 : cet algorithme fonctionne en deux temps :
— on fait un pré-calcul, qui remplira un tableau SOMME, où pour tout $1 \leq i \leq n$, $\texttt{SOMME}[i] = \Sigma_{p=1}^{i} T[p]$
— on génère toutes les paires possibles de positions i et j, $1 \leq i \leq j \leq n$, et pour chaque possibilité, on a :
  — $S_{i,j} = \texttt{SOMME}[j]$ si $i = 1$
  — $S_{i,j} = \texttt{SOMME}[j] - \texttt{SOMME}[i-1]$ sinon
  On retourne le $S_{i,j}$ le plus grand.

Algorithme 3 : on suppose que la fonction $MAX(p, q)$ renvoie le maximum entre p et q. On suppose aussi que la fonction $MAX3(p, q, r)$ renvoie le maximum entre p, q et r. L'algorithme est proposé sous la forme d'une fonction récursive appelée **Calcul3**. L'appel initial à la fonction sera **Calcul3**$(1, n)$.

Algorithme 3 : **Fonction Calcul3(a, b : entiers) :entier**

1. **Si** $(a > b)$ **alors**
2. retourner 0
3. **Fin Si**
4. **Si** $(a = b)$ **alors**
5. **Si** $(T[a] > 0)$ **alors**
6. retourner $T[a]$
7. **Sinon**
8. retourner 0
9. **Fin Si**
10. **Fin Si**
11. $c := \lfloor \frac{a+b}{2} \rfloor$
12. $amax := 0$
13. $somme := 0$
14. $i := c$
15. **Tant que** $i \geq a$ **faire**
16. $somme := somme + T[i]$
17. $amax := MAX(amax, somme)$
18. $i := i - 1$
19. **Fin Tant que**
20. $bmax := 0$
21. $somme := 0$
22. $i := c + 1$
23. **Tant que** $i \leq b$ **faire**
24. $somme := somme + T[i]$
25. $bmax := MAX(bmax, somme)$
26. $i := i + 1$
27. **Fin Tant que**
28. Retourner $MAX3(amax + bmax, Calcul3(a, c), Calcul3(c + 1, b))$

Algorithme 4 : on suppose que la fonction $MAX(p, q)$ renvoie le maximum entre p et q.

Algorithme 4

1. $a := 0$
2. $b := 0$
3. $i := 1$
4. **Tant que** $i \leq n$ **faire**
5. $\quad b := MAX(b + T[i], 0)$
6. $\quad a := MAX(a, b)$
7. $\quad i := i + 1$
8. **Fin Tant que**
9. Retourner a