DM2 : Sous séquence de poids maximal

On définit la constante : #define $NMAX$ 100000
et le type : typedef int tab[NMAX] ;
Étant donné un tableau t de type tab et N un entier $1 \leq N \leq NMAX$, on veut déterminer une sous-séquence $t[d, f]$ de $t[0, N-1]$ dont le poids (i.e la somme des valeurs) est maximal. On note max_res ce poids maximal :

$$max\_res = \sum_{k=d}^{f} t[k] = Max\{0 \leq deb \leq fin < N, \quad \sum_{k=deb}^{fin} t[k]\}$$

Par exemple :

- $[\mathbf{6}, -\mathbf{1}, \mathbf{3}, -10, \mathbf{8}]$ 2 sous-séquences de somme maximale 8 : $[6, -1, 3]$ et $[8]$
- $[2, -1, 0, -2, 3, -4, \mathbf{8}, -\mathbf{1}, \mathbf{2}]$ 1 sous-séquence de somme maximale : $8 - 1 + 2 = 9$

Pour faciliter l'écriture des prédicats, pour tout couple deb, fin tel que : $0 \leq deb \leq fin < N$ on note :

$$S(t, deb, fin) = \sum_{k=deb}^{fin} t[k]$$

1 Solutions proposées

1. Solution naïve 1 ; $O(N^3)$:

   Pour toutes les sous-séquences possibles : $[deb, fin]$ avec $0 \leq deb \leq fin < N$, on calcule leurs poids $poids(deb, fin) = \sum_{k=deb}^{fin} t[k]$ et on conserve la solution de poids maximal max_res. L'algorithme prend la forme suivante :

```
Pour deb allant de 0 à N-1
   Pour fin allant de deb à N-1
      Calcul de poids(deb,fin) : 3 ème boucle For
      Mise à jour éventuelle de max_res avec poids(deb,fin)
         (si poids(deb,fin) est le plus grand jamais rencontré)
```

2. Solution naïve 2 : $O(N^2)$:
   Adapter l'algo précédent pour calculer le poids au fur et à mesure sans faire appel à la 3ème boucle For

3. Programmation dynamique : $O(N)$:

Pour tout indice i tel que $0 \leq i < N$, on note $res[i]$ le poids de la sous-séquence la plus lourde **se terminant exactement à la case d'indice** i.

Une sous-séquence maximale se termine nécessairement à un indice i du tableau $0 \leq i < N$; le résultat cherché noté max_res est donc :

$$max\_res = max\{0 \leq i < N, res[i]\}$$

On établit une relation de récurrence sur la suite d'entiers $res[i], 0 \leq i < N$ en constatant que :

- si $res[i-1] > 0$ on complète la plus lourde sous-séquence se terminant à la case $i-1$ en y ajoutant la valeur $t[i]$ et on obtient la plus lourde sous-séquence se terminant à la case i
- sinon la plus lourde sous-séquence se terminant à l'indice i est le singleton $t[i]$

L'algorithme consiste en une seule boucle :

```
Initialiser res[0], max_res et i
/* INV : ∃D ∃F (0 ≤ D ≤ F < i ≤ N ∧ S(t, D, F) = max_res)
                          ∧
        ∀Deb ∀Fin (0 ≤ Deb ≤ Fin < i ≤ N → max_res ≥ S(t, Deb, Fin))
                          ∧
                       (i ≤ N)
*/
Tant que i ≠ N
    /* INV ∧ i ≠ N */
    Calculer res[i]
    Mettre à jour le maximum courant max_res
    Incrémenter i
    /* INV */
/* INV ∧ i = N */
```

2 Travail demandé :

1. Spécification : donner la précondition et la postcondition du problème : utiliser la notation $S(deb, fin)$. *Ces assertions sont communes aux 3 fonctions* $f1$, $f2$ *et* $f3$
2. Ecrire une fonction $f1$ qui met en oeuvre la solution naïve 1 pour déterminer max_res
3. Ecrire une fonction $f2$ qui met en oeuvre la solution naïve 2 pour déterminer max_res
4. Programmation dynamique :
   - Ecrire à la main le tableau res correspondant à $t = -1 \quad 3 \quad -3 \quad 1 \quad 6$.
     Que vaut ici max_res ?
   - Ecrire une fonction $f3$ qui met en oeuvre la programmation dynamique pour déterminer max_res

Contraintes :

- Votre développement doit passer par une boucle while correspondant à la méthode et l'invariant donnés ci-dessus.

- Commenter la fonction avec les diverses assertions devant être vérifiées pour prouver la boucle.
- Ces assertions devront ensuite être intégrées sous forme d'assert. Comme dans le TP6, il est nécessaire de créer 2 fonctions auxiliaires associées aux 2 premiers conjoints de l'invariant. *Ces fonctions peuvent s'inspirer fortement de f1 et f2*
- Proposer une variante et vérifier la terminaison. (Introduire une variable v et utiliser assert comme dans le TP6...)

5. Mesurer et comparer le temps d'exécution de ces 3 fonctions pour des listes aléatoires d'entiers pris dans $[-10, 10]$ et contenant $1000, 10000, 100000$ éléments.

   A titre d'exemple, j'ai obtenu :
   - $N = 10000$: $f1 : 611, 9s$ $f2 : 0, 25s$ $f3 : 0, 0001s$
   - $N = 100000$: $f2 : 24, 9s$ $f3 : 0, 0009s$
6. Modifier les fonctions $f1, f2$ et $f3$ pour qu'elles permettent aussi de déterminer la position de la première sous-séquence de poids maximal (c'est à dire telle que deb et fin sont le plus petits possible).

   Par exemple, pour $t = [\mathbf{6}, \mathbf{-1}, \mathbf{3}, 0, -14, \mathbf{8}]$ il ya 3 sous-séquences solutions : $[6, -1, 3]$, $[6, -1, 3, 0]$ et $[8]$. La première est $[6, -1, 3]$.

3 Compléments de langage C

- Pour générer un nombre aléatoire $x \in [-10, 10]$:
```
#include <time.h>
#include <stdlib.h>
srand(time(NULL)); /*A PLACER UNE SEULE FOIS EN DEBUT DE MAIN */
int x=rand()%21-10;
```

- Pour mesurer le temps écoulé :
```
#include <time.h>
clock_t debut=clock(); /* départ chrono */

Appel de la fonction fi à chronométrer

clock_t fin=clock(); /* fin chrono */
printf("temps CPU : %.2f secondes \ n", (double) (fin-debut)/CLOCKS_PER_SEC);
```

4 Barème

6 questions : $1 + 2 + 4 + 7 + 3 + 3$

◇ ◇ ◇◇ ◇ ◇◇