

Droulers Nicolas

Duval Valentin

Lamour Antoine

## Compte-rendu du projet d'Algorithmique et programmation 4

L'intégralité des fichiers se trouve dans le lien suivant : <https://github.com/m4ddabs/TPALGO>.

Pour l'utiliser correctement, voici quelques informations. Les fichiers du programme principal (méthode aboutie) se trouvent dans la branche « legacy ». Les fichiers du programme secondaire (méthode simple) se trouvent dans la branche « master ». Ces fichiers utilisent la bibliothèque `rapport_deformations{.c, .h}`. Dans chacune des deux branches est présent un programme « main » permettant de tester ces programmes.

Nous avons commencé par créer un algorithme répondant à la demande de la façon la plus simple possible. Le paquet simulé n'étant pas trié, on parcourt pour chaque position la totalité du paquet. On mesure par la suite la valeur absolue de la différence entre cette valeur et chacune des autres positions du paquet. A chaque fois qu'elle sera inférieure ou égale à 100, on incrémente de 1 le nombre de voisins pour la position étudiée. Si le nombre de voisins après avoir parcouru tout le paquet est supérieur ou égal à 100, alors on pourra déclarer une alerte.

Cette solution est simple et fonctionne, mais elle n'est cependant pas du tout optimisée. En effet, la compilation est tout d'abord très longue (85 secondes cf photo ci-dessous) du fait que l'algorithme parcourt tout le paquet pour chaque position. De plus, ce programme ne gère pas les doublons (une même position peut subir plusieurs déformations donc elle pourrait avoir plusieurs alertes, or chaque position ne peut avoir qu'une seule alerte). Nous avons donc créé un programme afin de vérifier si une position se situe déjà dans le tableau d'alertes ou non :

**Algorithme** vérification : (position, ref alerte, ref taille\_tab\_alerte, nb\_deform\_locales) → () **selon**

$i \leftarrow 0$

**Tant que**  $i < \text{val } \text{taille\_tab\_alerte}$  **répéter** :

(pos, nbdeform)  $\leftarrow$  ref alerte [i]

**Si** position = pos **alors** (« position déjà présente ») .

.

val taille\_tab\_alerte = taille\_tab\_alerte + 1

(pos, nbdeform)  $\leftarrow$  ref alerte [i]

val pos  $\leftarrow$  position

val nbdeform  $\leftarrow$  nb\_deform\_locales

.

L'algorithme suivant se charge d'identifier les alertes :

**Algorithme** chercheAlerte (paquet, taillepaquet)  $\rightarrow$  (alertes) **selon** :

alertes  $\leftarrow$  0

i  $\leftarrow$  1

k  $\leftarrow$  1

nmbdeformlocales  $\leftarrow$  0

premierealerte  $\leftarrow$  0

premierepos  $\leftarrow$  0

taillealertes  $\leftarrow$  0

**Tant que** premierealerte = 0 **et** i != taillepak **répéter**:

**Tant que** k<=taillepak **répéter**:

**Si** |paquet(k)-paquet(i)| <= 100 **alors**

            nmbdeformlocales  $\leftarrow$  nmbdeformlocales + 1

        .

        k  $\leftarrow$  k+1

    .

**Si** nmbdeformlocales >= 100 **alors**

        alertes  $\leftarrow$  tab 1 ((position,nmb\_deformation\_locales))

        alertes(1)  $\leftarrow$  (paquet(i),nmbdeformlocales)

        nmbdeformlocales  $\leftarrow$  0

        premierespos  $\leftarrow$  i + 1

        premierealerte  $\leftarrow$  1

    .

**Sinon** nmbdeformlocales  $\leftarrow$  0, i  $\leftarrow$  i+1 .

  .

**Si** alertes != 0 **alors**

    i  $\leftarrow$  premierespos

    k  $\leftarrow$  1

**Tant que** i <=taillepak **répéter** :

**Tant que** k<=taillepak **répéter** :

**Si** |paquet(k)-paquet(i)| <= 100 **alors**

$\text{nmbdeformlocales} \leftarrow \text{nmbdeformlocales} + 1$

**Si**  $\text{nmbdeformlocales} \geq 100$  **alors**

$\text{vérification}(\text{paquet}(i), \text{ref alertes}, \text{ref taillealertes}, \text{nmbdeformlocales})$

$\text{nmbdeformlocales} \leftarrow 0$

**Sinon**  $\text{nmbdeformlocales} \leftarrow 0$ .

L'algorithme de vérification a une complexité linéaire de par la présence de l'unique boucle tant que.  
L'algorithme pour trouver les alertes a une complexité quadratique  $n^2$ .

On peut voir que ce programme détecte 31196 alertes.

Deuxièmement, nous avons cherché à optimiser au maximum le programme, afin qu'il soit le plus rapide possible. Ainsi, nous nous sommes tout d'abord dit qu'il allait falloir trier le paquet. La méthode utilisée ici est le « tri rapide ». Il fonctionne à partir d'un pivot afin de ranger toutes les valeurs dans l'ordre croissant. On utilise 2 fonctions, récupérées d'internet :

**Algorithme** permuter :  $(\text{ref } a, \text{ref } b) \rightarrow ()$  **selon**

$\text{tmp} \leftarrow 0$

$\text{tmp} \leftarrow a$

$\text{val } a \leftarrow b$

$\text{val } b \leftarrow \text{tmp}$

•

**Algorithme** TriRapide : (tab[ ], first, last) → () **selon**

pivot ← 0, i ← 0, j ← 0

**Si** (first < last) **alors** :

pivot ← first

i ← first

j ← last

**Tant que** i < j **répéter**

**Tant que** tab[i] ≤ tab[pivot] **et** i < last **répéter** i ← i+1 .

**Tant que** tab[j] > tab[pivot] **répéter** j ← j-1 .

**Si** (i < j) **alors** permuter( (tab+i),(tab+j) ) .

permuter( (tab+pivot),(tab+j) )

TriRapid(tab, first, j-1)

TriRapid(tab, j+1, last)

•

Après utilisation de ces fonctions, on obtient donc un paquet de positions triées dans l'ordre croissant. Nous nous sommes ensuite intéressés à la façon par laquelle nous allions résoudre le problème efficacement. Nous avons abouti sur l'idée suivante.

Tout d'abord, nous distinguerons indice dans le tableau paquet[ ] et valeur de la position à cet indice. Pour chaque indice, on prend la valeur de la position +100 et -100. Dans cet intervalle, on compte le nombre de positions, ce nombre sera le nombre de voisins de la position étudiée. Illustrons cela avec un exemple :

Indice dans le tableau	20	21	22	23	24	25
Valeur de la position	72	121	187	193	218	296

Ici, si on étudie l'indice 22 (position 187) et que l'on applique la méthode détaillée ci-dessus, le nombre de voisins sera de 3. A gauche, l'indice 21 uniquement sera comptabilisé et à droite, les indices 23-24 seront comptabilisés. Les indices 20 et 25 ne seront pas pris en compte car 72 et 296 n'appartiennent pas à l'intervalle [187-100 ; 187+100].

La fonction de recherche des alertes est la suivante :

**Algorithme** repérer\_alertes : (paquet[ ], ref tab\_alerte, ref taille\_tab\_alerte) → ref alerte **selon**

i ← 1, j ← 1, voisins ← 0

**Tant que  $i \leq 100000$  répéter**

$j \leftarrow i$

**Tant que  $(\text{paquet}[i]-100) \leq \text{paquet}[j]$  et  $j > 0$  répéter**

$\text{voisins} \leftarrow \text{voisins} + 1, j \leftarrow j - 1$  .

$j \leftarrow i + 1$

**Tant que  $\text{paquet}[j] \leq (\text{paquet}[i] + 100)$  et  $j < 100000$  répéter**

$\text{voisins} \leftarrow \text{voisins} + 1, j \leftarrow j + 1$  .

**Si  $\text{voisins} \geq 100$  alors**

**Si val  $\text{taille\_tab\_alerte} = 0$  alors**

$\text{val } \text{taille\_tab\_alerte} \leftarrow \text{taille\_tab\_alerte} + 1$

$(\text{pos}, \text{nbdeform}) \leftarrow \text{ref } \text{tab\_alerte}[1]$

$\text{val } \text{pos} \leftarrow \text{paquet}[i], \text{val } \text{nbdeform} \leftarrow \text{voisins}$

.

**Sinon**  $\text{verification}(\text{paquet}[i], \text{alerte}, \text{taille\_tab\_alerte}, \text{voisins})$  .

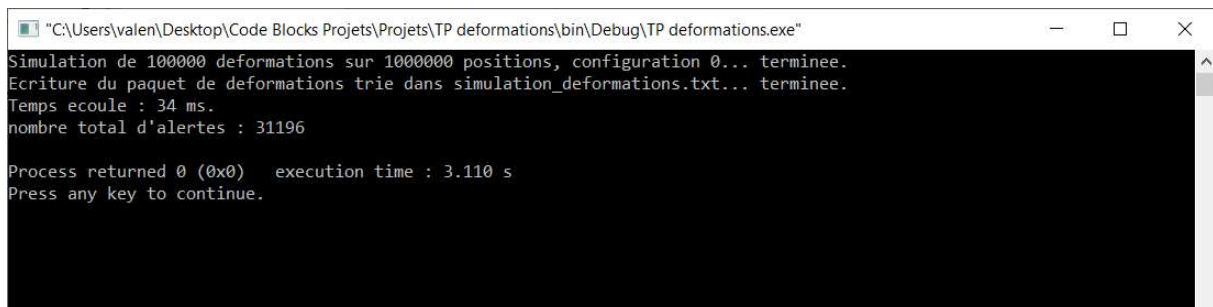
.

.

.

Cet algorithme a une complexité quadratique  $n^2$  : on observe deux boucles tant que imbriquées.

Grâce à cette nouvelle méthode, le programme ne va plus parcourir tout le tableau à chaque fois donc on aura un gain de temps notable. Notons de plus qu'on utilisera le programme vérification pour éliminer les doublons. Après test, cette méthode permet au programme de se terminer en entre 3 et 5 secondes en moyenne, ce qui est très satisfaisant. On note également que le nombre d'alertes trouvé est le même que pour la première méthode.



```
"C:\Users\valen\Desktop\Code Blocks Projets\Projets\TP deformations\bin\Debug\TP deformations.exe"
Simulation de 100000 deformations sur 1000000 positions, configuration 0... terminee.
Ecriture du paquet de deformations trie dans simulation_deformations.txt... terminee.
Temps ecoule : 34 ms.
nombre total d'alertes : 31196
Process returned 0 (0x0) execution time : 3.110 s
Press any key to continue.
```