ROUSSEAU Antonin
KREROWICZ Robin

Compte-rendu
Le compte est bon

S3D

fonction calculer(opérateur,n1,nb2) :

Cette fonction renvoie le résultat de l'opération entre n1 et n2 ou :

- -1 si le resultat est negatif (pas pris en compte dans cet énoncé)
- -1 si la division n'est pas possible (reste différent de 0)

recherche d'une solution - fonction compteEstBon(tab, attendu) :
le principe comme décrit dans le sujet est de tester toute les combinaisons d'opérations entre les nombres contenu dans le tableau on choisit à chaque fois deux nombres et une opérations puis si l'opération est intéressante on insère le résultat dans le tableau diminué d'une case, bien sûr on retire les opérandes utilisées car on ne peut pas les utiliser plusieurs fois , puis on relance compte est bon sur ce nouveau tableau .

Cas du résultat le plus proche

Tant que la méthode ne trouve pas le résultat attendu, on stocke le résultat le plus proche dans l'attribut plusproche. Les résultats sont comparés par valeur absolue de leur différence par rapport à l'attendu.
Si la méthode ne trouve pas de successions d'opérations permettant de trouver l'attendu, on relance la méthode compteEstBon() avec le tableau d'entiers donnés en argument, et le résultat le plus proche à trouver comme résultat attendu. La méthode nous indiquera donc la succession d'opérations pour arriver au résultat le plus proche.

sauvegarde et affichage de la solution

Lorsque l'algorithme trouve une suite d'opération qui donne le résultat attendu il effectue un backtracking et concatène le résultat dans la variable affichage
Les différentes étapes pour arriver à la solution sont séparées par "EOL" (*End of Line*) et on utilise la méthode split("EOL") pour obtenir un tableau de toutes les étapes menant à la solution, dans le bon ordre.
Pour cela le tableau est affiché, étape par étape dans l'ordre inverse de la concaténation..

Algorithme formel :

compteEstBon.algo

```
fonction compteEstBon(t:tableau entier, nb:entier,attendu:entier):booleen
debut
    pour i de 0 a n faire
        si t[i] = attendu alors
            retourne  vrai
        fsi
        pour j de i+1 a n faire
            pour k de nombre a nombre faire
                res <- calculer(k,t[i],t[j])
                si res > 0 alors
                    //on compare la valeur absolu de leur difference
                    si |attendu-plusproche| > |attendu-res| alors
                        plusproche <- res
                    fsi
                    tempi <- t[i]
                    tempj <- t[j]

                    t[i] <- res
                    t[j] <- t[nb-1]

                    si compteEstBon(t,nb-1,attendu) alors
                        affichage <- concat(affichage,concat(tempi,concat(" ",concat(operateurs[k],concat(" ",concat(tempj,concat("=",concat(res,"EOL")))))))
                        retourne vrai;
                    fsi
                    t[i]=tempi;
                    t[j]=tempj;
                fsi
            fpour
        fpour
    fpour
    retourne faux
fin
```

```
fonction calculer( op:entier, nb1:entier, nb2:entier):entier
debut
    si op=0 alors
        retourne nb1+nb2
    sinon
        si op=1 alors
            si nb1>nb2 alors
                retourne nb1-nb2
            sinon
                retourne -1
            fsi
        sinon
            si op=2 alors
                retourne nb1*nb2
            sinon
                si op=3 alors
                    si nb1%nb2==0 alors
                        retourne nb1/nb2
                    fsi
                sinon

                fsi
            fsi
        fsi
    fsi
fin
```

```
lexique:
    i,j,k: entier // incrementeur
    nb: entier // taille du tableau courant
    attendu: entier // resultat attendu
    t: tableau entier // tableau qui contient les operandes
    operateurs: tableau entier // tableau qui contient les operateurs sous forme de caractere
    affichage: chaine // chaine qui contient les calculs à faire
```

Optimisation :

Dans notre première version nous triions le tableau dans l'ordre décroissant pour éliminer le plus de calcul inutile par exemple soustraire un nombre plus grand que le premier ce qui donne un nombre négatif et est interdit dans les règles du jeu .
Cette version utilisait un nouveau tableau a chaque appel diminué d'une case et nécessitait donc un parcours complet de tableau pour le remplir , un autre pour le trier puis le renverser.

Après des test nous sommes parvenu à une version n'utilisant que le tableau de base mais en passant en paramètre un attribut qui représente la taille du tableau de manière artificielle comme lors d'un exercice réalisé en cours, ainsi même si le nombre d'appel récursif est plus important (même si en moyenne il ne l'est pas) la complexité de notre fonction reste faible .

Notre programme n'utilise que des types primitifs pour stocker les opérandes et n'alloue que deux nouvelles variables à l'exécution à la place d'un nouveau tableau , celles-ci permettent d'effectuer la récursivité et si elle ne donne rien le tableau est restitué dans l'état de départ à la prochaine itération de la boucle grâce à ces deux variables temporaires.

La suite d'opération est stocké dans un StringBuffer pour accélérer le backtracking car un String est immuable en effet lors de la concaténation de String java est obligé de recréer un autre String alors qu'un StringBuffer peut être modifié sans être recréé avec la méthode append , mais cette optimisation n'est pas la plus importante car cet étape n'est réalisé que 5 fois (une fois par ligne d'opération) .

Pour constater les optimisations, nous incrémentons une variable a chaque appel et nous mesurons le temps en milliseconde entre l'appel et le retour de compteEstBon() dans le Main.

Nous avons comparé différents programmes issus de différents binômes au CharlyLab grâce à un script python qui lance chaque .class donné avec les mêmes paramètres et cela un grande nombre de fois , sur la même machine pour pouvoir comparer les temps d'exécution.
Nous avons remarqué que sur certains cas nous effectuons plus d'appels récursif mais que l'exécution était 10 fois plus rapide avec notre programme.
Après quelques discussions nous nous sommes rendu compte que les autres binômes avaient une approche différente et qu'ils utilisaient des ArrayList et d'autre type d'objet tout en les triant ce qui ralentit considérablement le programme .
Et qu'ils stockaient à chaque appel la suite d'opération alors qu'on ne sait pas s' ils vont aboutir .

Après quelques modifications le programme m'a permis de mesurer le temps d'exécution et le nombre d'appels moyen de la fonction récursive ainsi après avoir effectué un test sur 10 000 échantillons.
Nous pouvons constater qu'en moyenne l'algorithme effectue 133 000 appels de la fonction compteEstBon et que sur une machine puissante en moyenne la solution est trouvé en moins de 5ms .

```
PS C:\Users\Antonin\Documents\GitHub\compteEstBon\test> python launcher.py 10
moyenne en ms 3.3
moyenne d'appels 76845.1
PS C:\Users\Antonin\Documents\GitHub\compteEstBon\test> python launcher.py 100
moyenne en ms 4.85
moyenne d'appels 142313.41
PS C:\Users\Antonin\Documents\GitHub\compteEstBon\test> python launcher.py 1000
moyenne en ms 4.85
moyenne d'appels 145611.958
PS C:\Users\Antonin\Documents\GitHub\compteEstBon\test> python launcher.py 10000
moyenne en ms 4.7962
moyenne d'appels 133098.9742
PS C:\Users\Antonin\Documents\GitHub\compteEstBon\test>
```

Nous avons testé d'effectuer les opérations dans un ordre différent : + * - / mais cela n'a pas modifié considérablement le nombre d'appels et il est même un peu plus élevé

```
moyenne en ms 4.87
moyenne d'appels 140000.934
```

Exemples du sujet :

```
PS C:\Users\Antonin\Documents\GitHub\compteEstBon> java CompteEstBon 9 1 6 8 2 3 845
Le compte est bon !!
Calcul :
9 * 6 = 54
54 - 1 = 53
53 * 2 = 106
106 * 8 = 848
848 - 3 = 845

====STATISTIQUES====
nombre d'appels de la fonction compteEstBon : 101967
Temps ecoule en ms: 12
```

```
PS C:\Users\Antonin\Documents\GitHub\compteEstBon> java CompteEstBon 1 7 1 2 25 100 591
Pas de solution exacte
La valeur la plus proche est : 592
Calcul :
1 + 7 = 8
100 + 1 = 101
101 - 25 = 76
76 - 2 = 74
8 * 74 = 592

====STATISTIQUES====
nombre d'appels de la fonction compteEstBon : 426672
Temps ecoule en ms: 20
```

Ici le pire cas , on ne trouve pas de solution exacte et donc on doit relancer la valeur la plus proche à cause de notre approche qui ne stocke pas à chaque fois les opérations.