IF111 - Algorithmique et structures de données - Correction TD1

Rohan Fossé

2021 - 2022

1 Maximum - Minimum

On souhaite calculer le minimum et le maximum de deux nombres entiers positifs.

1. Dans un premier temps, on n'utilisera pas d'instructions conditionnelles, mais une fonction `abs(int x)`, qui retourne la valeur absolue de l'entier `x`.
   Écrivez un algorithme affichant le maximum et le minimum de deux nombres passés en paramètres.

```
1: function maxmin(a, b)
2:     max ← a + b + abs(a − b)/2
3:     min ← a + b − abs(a − b)/2
4:     afficher("Maximum :, max)
5:     afficher("Minimum :, min)
```

2. Modifiez l'algorithme précédent, en utilisant cette fois les instructions conditionnelles et l'opérateur >(supérieur à).

```
1: function maxmin2(a, b)
2:     if a > b then
3:         max ← a
4:         min ← b
5:     else
6:         max ← b
7:         min ← a
8:     afficher("Maximum :, max)
9:     afficher("Minimum :, min)
```

2 Rendre la monnaie

On se propose d'écrire un algorithme permettant d'obtenir la suite des billets totalisant une somme donnée (dont on suppose qu'elle est un multiple de 10). Les espèces disponibles sont des billets de 50, 20, et 10 euros.

Le principe est de donner le billet de valeur la plus grande possible inférieure ou égale à la somme à rendre et de poursuivre la même stratégie avec la somme restante jusqu'à ce que celle-ci soit nulle.

1. Écrire cet algorithme en utilisant les structures de contrôle suivantes : tant que, si, et si sinon. L'algorithme utilisera une variable nommée qui contiendra la somme restant à rendre au fur et à mesure de la remise de billets au client. On n'utilisera comme opérations arithmétiques que des additions ou des soustractions. Le résultat sera l'affichage de la suite des nombres de billets à rendre.

   Par exemple, si la somme est 180, l'affichage devra être :

```
3 billets de 50 euros
1 billet de 20 euros
1 billet de 10 euros
```

```
1: function RendreMonnaie(s)
2:     n1 ← 0
3:     n2 ← 0
4:     n3 ← 0
5:     while s ≠ 0 do
6:         if s ≥ 50 then
7:             s ← s − 50
8:             n1 ← n1 + 1
9:         else
10:            if s ≥ 20 then
11:                s ← s − 20
12:                n2 ← n2 + 1
13:            else
14:                s ← s − 10
15:                n2 ← n3 + 1
16:    afficher(n1, "billets de 50 euros")
17:    afficher(n2, "billets de 50 euros")
18:    afficher(n3, "billets de 50 euros")
```

   On peut faire remarquer aux étudiants que l'algo ne termine pas si la somme n'est pas un multiple de 10, et changer le test $s \neq 0$ par $s > 0$, auquel cas, au pire on rend plus de monnaie que prévu (le minimum nécessaire). Si on donne une correction, il vaut mieux donner le test $s > 0$.

2. Déterminer le nombre de soustractions effectuées par l'algorithme, on donnera une formule faisant intervenir des divisions entières par les nombres 50, 20, et 10.

On remarque que l'algorithme va commencer par rendre autant de billets de 50 que possible, puis de 20, puis de 10.

Le premier nombre de soustractions est donc le quotient entier de s par 50. Soit $q_1 = \lfloor s/50 \rfloor$, c'est le nombre de billets de 50 rendus et aussi le nombre de soustractions à effectuer pour les billets de 50.

Il reste alors à rendre $s - 50q_1$. Soit q_2 le nombre de billets de 20 à rendre, on a $q_2 = \lfloor (s - 50q_1)/20 \rfloor$.

Enfin, q_3 le nombre de billets de 10 à rendre est égal à $\lfloor (s - 50q_1 - 20q_2)/10 \rfloor$.

Le nombre de soustractions effectuées est $q_1 + q_2 + q_3$. Ca nous fait une belle jambe, il vaut mieux remarquer qu'après avoir soustrait tous les billets de 50, il reste une somme qui est soit 40, soit 30, soit 20, soit 10, et que donc au pire on va rendre q_1 billets de 50 plus deux billets (20+20, 20+10, 10+10, 10 respectivement). Le nombre de soustractions est donc au plus $\lfloor \frac{s}{50} \rfloor + 2$.

On pourra se convaincre que si la somme n'est pas un multiple de 10 et que notre algo a le test $s > 0$, le maximum de soustractions est $\lfloor \frac{s}{50} \rfloor + 3$.

3. Généraliser votre algorithme au cas où les valeurs de billets ou pièces disponibles sont en nombre quelconque et figurent dans un tableau à valeurs décroissantes `val[1]` > `val[2]` > ... > `val[k]`. Ainsi dans l'exemple considéré plus haut, on aurait `k = 3` et `val[] = { 50, 20, 10 }`.

Il y a plusieurs solutions plus ou moins élégantes ; en gros, soit on fait la boucle externe sur s, soit on la fait sur l'indice qui avance dans le tableau. Voici une solution.

```
Rendre-Monnaie-General(s, val, k)
   {  pour j de 1 à k
      {i ← 0
   tant que (s ≥ val[i])
      s ← s − val[i]
         i ← i + 1;
      fin }ant que afficher(i, "billets  de", val[i] )
      }
}
```

4. Montrer qu'il existe des valeurs de billets et une somme à rendre pour lesquels cet algorithme ne donne pas le nombre minimum de billets ou de pièces à rendre.

Essayer avec les billets 200, 120, et 10, et une somme à rendre de 240. L'algo donnera 5 billets à rendre (200 + 10 + 10 + 10 + 10) alors qu'on pourrait rendre (120 + 120).

3 Manipulation de tableaux

1. Écrire un algorithme pour trouver l'élément minimum et l'élément maximum d'un tableau de n cases. Déterminer le nombre d'affectations effectuées par votre algorithme.

```
Max_Min_tab(tableau, n)
    {max ← tableau[0];
    min ← tableau[0];
    pour i de 1 à n − 1 faire
        {si (max < tableau[i])
            alors max ← tableau[i];
        si (min > tableau[i])
            alors min ← tableau[i];
        }
    afficher("Maximum : ", max);
    afficher("Minimum : ", min);
    }
```

2. Écrire un algorithme qui permet de renverser un tableau *tab*. Le tableau *tab* sera passé en paramètre avec sa taille n. Déterminer le nombre de comparaisons effectuées par votre algorithme.

```
Renverser(tab, n)
    {pour i de 0 à n/2  faire
        {tmp ← tab[i]
        tab[i] ← tab[n − i + 1];
        tab[n − i] ← tmp;
    retourner tab;
    }
```

3. Un tableau est un palindrome si c'est identique en le lisant de gauche à droite (indices dans l'ordre 1,2,... n) et de droite à gauche (indices dans l'ordre n, n-1,...1). Écrire un algorithme qui teste si le tableau fourni est un palindrome. Donner la complexité en temps de votre algorithme dans le pire de cas.

Solution

Palindrome(T)

```
1. {
2.     n ← longueur(A);
3.     pour i de 0 à ⌊n/2⌋faire
4.         si T[i] ≠ T[n − i + 1] alors
5.         retourner faux;
6.         fin si;
7.     fin pour;
8.     retourner true;
9. }
```

4. Écrire un algorithme qui vérifié s'il est possible de permuter les éléments d'un tableau donné en entrée de manière à obtenir un tableau palindrome. L'algorithme retournera *vrai* si c'est possible, *faux* autrement. On suppose que chaque élément du tableau est une lettre de l'alphabet français. Vous avez à disposition (pas nécessaire de l'utiliser) une fonction

CalculPositionAlphabet(c) qui prend en entrée un caractère et retourne l'entier correspondant à sa position dans l'alphabet.

> Plusieurs solutions sont possibles. On peut considerer...

5. Écrire un algorithme pour trouver l'élément minimum et l'élément maximum d'un tableau de n cases. On suppose que tous les éléments du tableau sont différents. L'algorithme doit effectuer au plus $3\lfloor \frac{n}{2} \rfloor$ comparaisons. L'algorithme *peut* utiliser un tableau auxiliaire de taille n.

> On compare les éléments en paires et on stocke les plus petits sur une moitié du tableau aux et le plus grand dans l'autre moitié. Après on cherche le min dans le sous-tableau de 0 a $\frac{n}{2}$ et le max dans le soustableau de $\frac{n}{2}+1$ a $n-1$.
> $\lfloor \frac{n}{2} \rfloor$ pour diviser $2 \cdot (\lfloor \frac{n}{2} \rfloor - 1)$ pour trouver le max et le min + 2 pour comparer avec l'élément du milieu.
>
> Max_Min_tab_rapide(T)
> {$n \leftarrow longuer(T)$;
> **entier** Taux[1..n] $\leftarrow$ [0,. . .,0];
> pour i de 0 à $n/2$ faire
> si ($T[i] < T[n-i+1]$)
> alors $Taux[i] \leftarrow T[i]$; $Taux[n-i+1] \leftarrow T[n-i+1]$;
> sinon $Taux[i] \leftarrow T[n-i+1]$; $Taux[n-i+1] \leftarrow T[i]$;
> fin si
> $max \leftarrow Taux[n]$; $min \leftarrow Taux[1]$;
> pour i de 0 à $(n/2)+1$ faire
> {si ($Taux[i] < min$) $alors$ $min \leftarrow Taux[i]$
> pour i de $(n/2)+1$ à n faire
> {si ($Taux[i] > max$) $alors$ $max \leftarrow Taux[i]$
> afficher("Maximum : ", max);
> afficher("Minimum : ", min);
> }

6. (bonus) Écrire un algorithme récursif qui teste si le tableau fourni est un palindrome.