Devoir surveillé n°1

Durée : 45 minutes

L'utilisation d'une calculatrice ou de tout document est interdite.
Toute sortie anticipée est interdite.

***Rappel** : si* `a` *et* `b` *désignent deux nombres entiers en Python, alors l'instruction* `min(a,b)` *renvoie le minimum des entiers* `a` *et* `b`. *Par exemple,* `min(-4, 5)` *renvoie* `-4`.

1. Introduction au problème des coupes de somme minimale

Dans tout le sujet, on s'intéresse aux **listes non vide dont les éléments sont des entiers relatifs**.

Pour une telle liste $\texttt{t} = [t_0, ..., t_{n-1}]$, on appelle **coupe** de `t` toute sous-liste non vide d'éléments consécutifs de `t`. Toute coupe d'une liste `t` est donc une liste de la forme $\texttt{t}[i : j] = [t_i, ..., t_{j-1}]$ où $0 \leqslant i < j \leqslant n$.

À toute coupe $\texttt{t}[i : j]$, on associe la somme $s_{i,j} = \sum_{k=i}^{j-1} t_k$ de ses éléments. Le but du problème est de construire un algorithme efficace pour déterminer la valeur minimale de sommes des coupes d'une liste `t`.

Par exemple, la valeur minimale des sommes des coupes de la liste

```
t=[4, -1, -9, -10, 9, -2, 8, -9]
```

est -20*, atteinte pour la coupe* `t[1:4]`.

1. Quelle est la valeur minimale des sommes des coupes de la liste

```
t=[4, -4, 1, -1, -9, 8, -3]
```

?

Pour quelle coupe cette somme est-elle atteinte ?

2. Écrire une fonction `somme` prenant en argument une liste `t`, ainsi que deux entiers i et j, et qui renvoie la somme $s_{i,j}$ de la coupe $\texttt{t}[i : j]$. *On ne vérifiera pas dans le code de la fonction que les indices i et j sont valides : on supposera qu'ils vérifieront la propriété $0 \leqslant i < j \leqslant n$, où n est la longueur de la liste* `t`.

2. Résolution par force brute

Le principe est de calculer la somme de toutes les coupes $\texttt{t}[i : j]$ et d'en déterminer le minimum.

1. Parmi les quatre fonctions ci-après, nommer l'unique fonction qui renvoie la valeur minimale des sommes des coupes de la liste passée en argument. *On ne demande pas de justifier la réponse.*

```
def somme_min_brute1(t):
    n = len(t)
    s_min = 0
    for i in range(n-1):
        for j in range(i+1,n+1):
            s = somme(t,i,j)
            if s < s_min:
                s_min = s
    return s_min
```

```
def somme_min_brute2(t):
    n = len(t)
    s_min = t[n-1]
    for i in range(n):
        for j in range(i,n):
            s = somme(t,i,j)
            if s < s_min:
                s_min = s
    return s_min
```

```
def somme_min_brute3(t):
    n = len(t)
    s_min = t[0]
    for i in range(n):
        for j in range(i+1,n+1):
            s = somme(t,i,j)
            if s < s_min:
                s_min = s
    return s_min
```

```
def somme_min_brute4(t):
    n = len(t)
    s_min = t[0]
    for i in range(n-1):
        for j in range(i+1,n):
            s = somme(t,i,j)
            if s < s_min:
                s_min = s
    return s_min
```

2. En déduire le code d'une fonction `coupe_min` qui prend en argument une liste d'entiers `t` et qui renvoie les indices i et j d'une coupe $\texttt{t}[i : j]$ de somme minimale.

3. Résolution par les suffixes

La méthode par force brute répond au problème mais recalcule plusieurs fois les mêmes sommes. On se propose ici de résoudre le problème pour toutes les coupes qui commencent par t_i, puis d'en déduire la solution.

1. Sans utiliser la fonction `somme`, écrire une fonction `min_coupe` prenant en argument une liste d'entiers `t` et un entier i, et qui renvoie la valeur minimale des sommes des coupes de `t` **qui commencent par** t_i, c'est-à-dire de la forme $\texttt{t}[i : j] = [t_i, ..., t_{j-1}]$.

2. Recopier sur la copie et compléter le code de la fonction ci-dessous de manière à ce qu'elle renvoie la valeur minimale des sommes des coupes de la liste passée en argument.

```
def somme_min_coupe(t):
    n = len(t)
    s_min = ...
    for i in range(...):
        s = min_coupe(..., ...)
        if ... :
            ...
    return s_min
```

4. Résolution par programmation dynamique

Pour une liste `t`$= [t_0, ..., t_{n-1}]$ et un entier naturel $j \in [\![1, n]\!]$, on note :

- x_j la valeur minimale des sommes des coupes de la sous-liste `t`$[0 : j] = [t_0, ..., t_{j-1}]$;
- y_j la valeur minimale des sommes des coupes de la sous-liste `t`$[0 : j] = [t_0, ..., t_{j-1}]$ **finissant par** t_{j-1} (c'est-à-dire d'une coupe de la forme `t`$[i : j] = [t_i, ..., t_{j-1}]$).

On peut montrer que, pour toute liste `t`$= [t_0, ..., t_{n-1}]$ et tout $j \in [\![1, n-1]\!]$,

$$y_{j+1} = \min(y_j + t_j, t_j) \text{ et } x_{j+1} = \min(x_j, y_{j+1}).$$

1. Pour une liste `t`$= [t_0, ..., t_{n-1}]$, que vaut y_1 ? et x_1 ? *On justifiera rapidement la réponse.*

2. Le code de la fonction ci-dessous contient plusieurs erreurs. Corriger sur la copie le code de la fonction ci-dessous de manière à ce qu'elle renvoie la valeur minimale des sommes des coupes de la liste passée en argument.

```
def somme_min_dyn(t):
  x = 0
  y = 0
  for j in range(1,n+1):
    x = min(x,y)
    y = min(y + j, x)
  return y
```

Corrigé

1. Introduction au problème des coupes de somme minimale

1. Le valeur minimale de sommes des coupes de `t` est -13 atteinte pour la coupe `t[1:5]`.

2.

```
def somme(t,i,j):
  S = 0
  for k in range(i,j):
    S += t[k]
  return S
```

2. Résolution par force brute

1. La fonction correcte est `somme_min_brute3`.

2.

```
def somme_coupe_min(t):
  n = len(t)
  s_min = t[0]
  i_min = 0
  j_min = 1
  for i in range(n):
    for j in range(i+1,n+1):
      s = somme(t,i,j)
      if s < s_min:
        s_min = s
        i_min = i
        j_min = j
  return i_min, j_min
```

3. Résolution par les suffixes

1.

```
def min_coupe(t,i):
  n = len(t)
  s_min = t[i]
  s = s_min
  for j in range(i+1,n):
    s += t[j]
    if s < s_min:
      s_min = s
  return s_min
```

2.

```
def somme_min_coupe(t):
  n = len(t)
  s_min = t[0]
  for i in range(n):
    s = min_coupe(t, i)
    if s < s_min:
      s_min = s
  return s_min
```

4. Résolution par programmation dynamique

1. Il n'y a qu'une sous-liste se terminant par t_0 : il s'agit $[t_0]$ donc $y_1 = t_0$ et de même $x_1 = t_0$.

2.

```
def somme_min_dyn(t):
  x = t[0]
  y = t[0]
  for j in range(1,len(t)):
    y = min(y + t[j], t[j])
    x = min(x,y)
  return x
```