ISM MPSI/PCSI, Informatique
Année 2024/2025

Corrigé du concours blanc - Informatique

I Préliminaires

1 . Le couple $([2,3],[1,4])$ est une partition équilibrée de $[1,2,3,4]$. Les couples $([1,2,4],[3,5])$, $([3,4],[1,2,5])$ et $([1,3,4],[2,5])$ sont des partitions équilibrées de $[1,2,3,4,5]$.

2 . On propose :

```
def somme(L):
    res=0
    for x in L:
       res+=x
    return res
```

3 . On propose :

```
def ecart(L1,L2):
    return abs(somme(L1)-somme(L2))
```

4 . On obtient :

```
>>> extract_couple([1,2])
[[[], [1, 2]], [[1], [2]], [[2], [1]], [[1, 2], []]]
>>> extract_couple([1,1])
[[[], [1, 1]], [[1], [1]], [[1], [1]], [[1, 1], []]]
```

5 . Le résultat de l'appel `extract_couple(L)` contient nécessairement les doublons `[L,[]]` et `[[],L]`.

II Partitions équilibrées

6 . On propose :

```
def min_ecart(L):
    L_ecart=extract_couple(L)
    res=float('inf')
    for L1,L2 in L_ecart:
        m=ecart(L1,L2)
        if m<res:
            res=m
    return res
```

7 . On propose :

```
def argmin_ecart(L):
    L_ecart=extract_couple(L)
    mini=min_ecart(L)
    res=[]
    for L1,L2 in L_ecart:
        if ecart(L1,L2)==mini:
            res.append([L1,L2])
    return res
```

8 . On propose :

```
def min_argmin_ecart(L):
    L_ecart=extract_couple(L)
    mini=float('inf')
    for L1,L2 in L_ecart:
        m=ecart(L1,L2)
        if m<mini:
            mini=m
            res=[[L1,L2]]
        elif m==mini:
            res.append([L1,L2])
    return mini,res
```

III Représentation binaire

9 . On propose :

```
def binaire(x,n):
    res=[]
    a=x
    for k in range(n):
        res.append(a%2==1)
        a//=2
    return res
```

10 . On propose :

```
def liste_bin(n):
    res=[]
    for k in range(2**n):
        res.append(binaire(k,n))
    return res
```

11 . On propose :

```
def part(L,ind):
    L1,L2=[],[]
    n=len(L)
    for k in range(n):
        if ind[k]:
            L1.append(L[k])
        else:
            L2.append(L[k])
    return [L1,L2]
```

12 . On propose :

```
def extract_couple(L):
    n=len(L)
    L_ind=liste_bin(n)
    res=[]
    for ind in L_ind:
        res.append(part(L,ind))
    return res
```

13 . On propose :

```
def liste_bin_rec(n):
    if n==0:
        return [[]]
    else:
        aux=liste_bin_rec(n-1)
        return [mot+[False] for mot in aux]+[mot+[True] for mot in aux]
```

IV Partitions équilibrées par indexation

14 . On propose :

```
def somme_ind(L,ind):
    n=len(L)
    S1,S2=0,0
    for k in range(n):
        if ind[k]:
            S1+=L[k]
        else:
            S2+=L[k]
    return S1,S2
```

15 . On propose :

```
def ecart_ind(L,ind):
    S1,S2=somme_ind(L,ind)
    return abs(S1-S2)
```

16 . On propose :

```
def min_ecart_ind(L):
    n=len(L)
    L_ind=liste_bin(n)
    res=float('inf')
    for ind in L_ind:
        m=ecart_ind(L,ind)
        if m<res:
            res=m
    return res
```

17 . On propose :

```
def argmin_ecart_ind(L):
    n=len(L)
    L_ind=liste_bin(n)
    mini=min_ecart_ind(L)
    res=[]
    for ind in L_ind:
        if ecart_ind(L,ind)==mini:
            res.append(part(L,ind))
    return res
```

V Suppression des doublons

18 . On propose :

```
def tri(T):
    if len(T)==0:
        return []
    else:
        pivot=T[0]
        T1,T2=[],[]
        for x in T[1:]:
            if x<pivot:
                T1.append(x)
            else:
                T2.append(x)
        return tri(T1)+[pivot]+tri(T2)
```

19 . On propose :

```
def equal_part(P1,P2):
    flag1=tri(P1[0])==tri(P2[0]) and tri(P1[1])==tri(P2[1])
    flag2=tri(P1[0])==tri(P2[1]) and tri(P1[1])==tri(P2[0])
    return flag1 or flag2
```

20 . On propose :

```
def detect(P,R):
    for T in R:
        if equal_part(T,P):
            return True
    return False
```

21 . On propose :

```
def part_equi(L):
    aux=argmin_ecart_ind(L)
    res=[]
    for x in aux:
        if not detect(x,res):
            res.append(x)
    return res
```