Devoir d'informatique : Gendarmes et Voleurs

1 Introduction

Question 1 *Pour la liste GV2 et k=2, proposer les différentes solutions possibles sous forme de liste de couple d'indices.*

Correction
`[[(0,2),(1,3),(4,5)],[(0,2),(3,4)],[(1,2),(3,4)],[(1,2),(4,5)],[(2,4),(1,3)]]`

Question 2 *Écrire une fonction* `associe(tab:list,i:int,j:int)->bool:` *qui teste si l'élément à l'indice* `i` *de tab est un gendarme* `'G'` *et si l'élément à l'indice* `j` *est un voleur* `'V'`*, si c'est le cas les éléments aux indices* `i` *et* `j` *sont passés à* `None` *et la fonction renvoie* `True`*. Sinon, rien n'est modifié et la fonction renvoie* `False`*. les indices* `i` *et* `j` *sont pris tel que i>j ou j>i.*

Correction
```
def associe(tab, i, j):
    if tab[i]=='G':
        if tab[j]=='V':
            tab[i]=None
            tab[j]=None
            return True
    return False
```

Question 3 *Supposons que la liste* `GV` *de taille n possède* `m` *'G' avec m<n. Pour une distance donnée k>0, quel serait dans le pire des cas le nombre d'appel à la fonction* `associe(tab:list,i:int,j:int)` *si on veut toutes les solutions possibles.*

Correction
Pour un gendarme, il y a au plus $2 \times k$ possibilités d'attraper un voleur. Au rang suivant, $4 \times k^2$, etc. Donc pour m gendarmes, $2^m \times k^m$, ce qui est rapidement ingérable.

2 Algorithme glouton

2.1 Le plus proche voleur

Question 4 *Commenter les différentes lignes de 1 à 12 de la fonction* `lePlusProche(tab:list,k:int)->int:` *qui prend en argument la liste* `tab` *d'éléments* `'G'` *et* `'V'` *et* `k` *la distance maximale entre 'G' et 'V' (k>0). Est-ce que* `tab` *est modifiée?*

Correction

Cette fonction renvoie le nombre de voleurs attrapés. `tab` est modifiée mais pas renvoyée.

```
def lePlusProche(tab,k):
    res=0 # initialisation du résultat a 0
    for i in range(len(tab)): # parcours de toute la liste
        rep=False # initialisation du test a False
        if tab[i]=='G': # test de la valeur de tab a G
            j=1 # j est initialisé a 1
            while j<k+1 and rep==False:
      # tant que j n'est pas égal a k et que le test est faux
                if (i-j>=0 and associe(tab,i,i-j)) or (i+j<len(tab) and associe(tab,i,i+j)):
      # on teste a gauche en veillant a conserver i-j>=0, si tab[i-j] est un voleur,
      # rep=True et res est incrémenté
      # sinon, on teste a droite sans être out of range, si tab[i+j] est un voleur,
      # rep=True et res est incrémenté
      # sinon, j est incrémenté
                    rep=True
                    res+=1
                j=j+1
    return res
```

Question 5 *Rédiger la signature de la fonction.*

Correction

"'Fonction qui teste les éléments non identiques de tab à une distance (différence d'indices) inférieure ou égale à k. Si deux éléments différents sont proches, ces éléments sont modifiés (None). La fonction compte le nombre de paires associées. entrées :

- tab : list, liste des éléments 'G' et 'V'
- k : int, la différence maximale des indices

sortie :

- res : int, nombre de paires possibles

"'

Question 6 *Que renvoient* `lePlusProche(GV1,1)` *et* `lePlusProche(GV2,2)` *? Conclure. Qu'affiche print (GV1) après l'appel à* `lePlusProche(GV1,1)` *?*

Correction

```
# >>> lePlusProche(GV1,1)
# 2
# >>> lePlusProche(GV2,2)
# 2
#>>> GV1
#[None, None, None, None, 'V']
```

lePlusProche(GV2,2) ne donne pas la solution optimale, c'est un optimal local.

Question 7 *Donner la définition du variant de boucle. Quel est le variant de la boucle* `while` *de la fonction* `lePlusProche(tab,k)` *? Justifier votre réponse.*

Correction

Un variant de boucle permet de prouver la terminaison d'une boucle conditionnelle. Un variant de boucle est une **quantité entière positive** à l'entrée de chaque itération de la boucle et qui **diminue strictement à chaque itération**.

la variant de boucle est $v_j = k+1-j$ imposé >0. En entrée de boucle, j=1, k>0 donc $v_1 = k > 0$. A chaque tour de boucle $v_j = v_{j-1} - 1$, suite strictement décroissante.

2.2 Le plus éloigné voleur possible

Règle : pour chaque gendarme, en commençant le traitement de la liste par la gauche, lui associer le voleur le plus éloigné possible.

Question 8 *A partir de la fonction* `lePlusProche(tab,k)`*, écrire une fonction* `lePlusEloigne(tab:list,k:int)->int:` *qui prend en argument la liste* `tab` *d'éléments 'G' et 'V' et* `k` *la distance maximale entre 'G' et 'V' (k>0). Cette fonction renvoie le nombre de voleurs attrapés. Mettre en évidence les lignes de code différentes entre les deux fonctions.*

Correction

```
def lePlusEloigne(tab,k):
    res=0
    for i in range(len(tab)):
        rep=False
        if tab[i]=='G':
            j=k
            while j>0 and rep==False:
                if (i-j>=0 and associe(tab,i,i-j)) or (i+j<len(tab) and associe(tab,i,i+j)):
                    rep=True
                    res+=1
                j=j-1
    return res
```

Question 9 *Que renvoient* `lePlusEloigne(GV2,2)` *et* `lePlusEloigne(GV3,3)` *? Conclure.*

Correction

```
# >>> lePlusEloigne(GV2,2)
# 3
# >>> lePlusEloigne(GV3,3)
# 2
```

lePlusEloigne(GV3,3) ne donne pas la solution optimale, c'est un optimal local.

3 Approches itérative et récursive

Afin de résoudre le problème posé, nous allons séparer les gendarmes 'G' et les voleurs 'V' dans deux listes que nous pourrons ensuite comparer avec la distance k.

Question 10 *Écrire une fonction* `indicesGV(tab:list)->tuple`*, qui prend en argument la liste des 'G' et des 'V' et renvoie deux listes,* `indices_G` *la liste des indices des éléments 'G' et* `indices_V` *la liste des indices des éléments 'V'. Par exemple,* `indicesGV(['G', 'V', 'V', 'G', 'V'])` *renvoie* `([0, 3], [1, 2, 4])`.

Correction

Une proposition parmi d'autres solutions

```
def indicesGV(tab:list)->tuple:
    '''tuple représente deux listes
    '''
    n=len(tab)
    indices_G=[]
    indices_V=[]
    for i in range(n):
        if tab[i]=='G':
            indices_G.append(i)
        else:
            indices_V.append(i)
    return indices_G,indices_V
```

On peut parcourir les deux listes `indices_G` et `indices_V` de gauche à droite. Si le premier indice de `indices_G`

soit `indices_G [0]` est assez "proche" du premier indice de `indices_V` soit `indices_V [0]`, alors le voleur est attrapé, sinon soit le gendarme est trop loin (à droite) et on passe au voleur suivant soit le voleur est trop loin (à droite) et on passe au gendarme suivant.

Exemple : On reprend les deux listes créées `indices_G=[0, 3]` et `indices_V=[1, 2, 4]` et k=1;
- `indices_G [0]`=0 et `indices_V [0]`=1, ils sont assez proches, le gendarme peut attraper le voleur;
- On passe aux indices suivants et couplesGV=1;
- `indices_G [1]`=3 et `indices_V [1]`=2, ils sont assez proches, le gendarme peut attraper le voleur et couplesGV=2;
- Toute la liste `indices_G` a été lue, la fonction renvoie le résultat : 2

Question 11 *Écrire une fonction* `gendarmeVoleurIte(ind_G :list ,ind_V :list ,k:int)->int`, *qui prend en argument la liste des indices de 'G' et la liste des indices de 'V' obtenues avec la fonction* `indicesGV(tab:list)` *ainsi que k la distance maximale et qui renvoie le nombre de voleurs attrapés.*

Correction

Une proposition parmi d'autres solutions

```
def gendarmeVoleurIte(ind_G:list,ind_V:list,k:int):
    v = 0
    g = 0
    res = 0
    while v < len(ind_V) and g < len(ind_G):

        # peut être attrapé
        if (abs(ind_V[v] - ind_G[g]) <= k):
            res += 1
            v += 1
            g += 1

        # on cherche un voleur plus près
        elif ind_V[v] < ind_G[g]:
            v += 1
        else:
            g += 1
    return res
```

Question 12 *Écrire une fonction* `gendarmeVoleurRec(ind_G :list ,ind_V :list ,k:int)->int`, *qui prend en argument la liste des indices de 'G' et la liste des indices de 'V' obtenues avec la fonction* `indicesGV(tab:list)` *ainsi que k la distance maximale et qui renvoie le nombre de voleurs attrapés par une méthode récursive. Préciser les arguments qui varient dans la récursion.*

Correction

Une proposition parmi d'autres solutions

```
def gendarmeVoleurRec(ind_G:list,ind_V:list,k):
    res=0
    if len(ind_G)==0 or len(ind_V)==0:
        return res
    if abs(ind_G[0]-ind_V[0])<=k:
        res=gendarmeVoleurRec(ind_G[1:],ind_V[1:],k)+1
        return res
    elif ind_G[0]>ind_V[0]:
        res=gendarmeVoleurRec(ind_G[:],ind_V[1:],k)
    else:
        res=gendarmeVoleurRec(ind_G[1:],ind_V[:],k)
```