



CONCOURS D'ENTRÉE

CYCLE INGENIEUR

OPTION : INFORMATIQUE

Samedi 15 Avril 2017

Durée : 2 Heures

A) Piles et garage ...

1) Est-ce que les séquences suivantes sont valides ?

a) Oui !

b) Non : La quatrième sortie intervient sur un garage (pile) vide, en effet v1, v2 et v3 sont déjà sorties.

2) La règle:

Une séquence formée de $E1$ de $E2$ et de D est dite *admissible* si elle contient autant de D que de $E1$ et de $E2$ cumulés et si toutes les actions qui lui correspondent peuvent être accomplies dans l'ordre indiqué par la séquence : on ne peut dépiler que s'il reste au moins un élément. A la fin, la pile (garage) doit être vide.

B) Dichotomie : « Chemin » de recherche ...

Les séquences b) et c) sont impossibles, en effet :

b) 31 - 62 - 90 - 72 - 61 - 66

On ne peut pas passer par 61 inférieur à 62 déjà envisagé en tant que valeur inférieure à celle recherchée.

c) 36 - 70 - 53 - 50 - 61 - 66

On ne peut pas passer par 50 inférieur à 53 déjà envisagé en tant que valeur inférieure à celle recherchée.

C) Codage RLE simplifié ...

1) La fonction `decodeRLE` décompresse une liste compressée en RLE.

```
def decodeRLE(L):
    R = []
    for (n, val) in L:
        for i in range(n):
            R.append(val)
    return R
```

Une autre version:

```
def decodeRLE2(L):
    R = []
    (i, n) = (0, len(L))
    while i < n:
        (nb, val) = L[i]
        while nb > 0:
            R.append(val)
            nb -= 1
        i += 1
    return R
```

2) La fonction encodeRLE compresse une liste en utilisant l'algorithme RLE.

```
def encodeRLE(L):
    R = []
    if L != []:
        (nb, val) = (1, L[0])
        for i in range(1, len(L)):
            if L[i] == val:
                nb += 1
            else:
                R.append((nb, val))
                (nb, val) = (1, L[i])
        R.append((nb, val))
    return R
```

D) Des matrices ...

1) Minimum des maximums.

La fonction posMinimax(M) retourne **la position de la valeur minimale** parmi les maximums de chaque ligne de la matrice d'entiers M .

```
def posMaxList(L):
    ''' maximum position of list L, not empty '''
    p = 0
    for i in range(1, len(L)):
        if L[i] > L[p]:
            p = i
    return p

def posMinimax(M):
    (min_i, min_j) = (0, posMaxList(M[0]))
    for i in range(1, len(M)):
        max_j = posMaxList(M[i])
        if M[i][max_j] < M[min_i][min_j]:
            (min_i, min_j) = (i, max_j)
    return (min_i, min_j)
```

#-----

```
def posMinimax2(M):
    mini = maxint
    (min_i, min_j) = (0, 0)
    (l, c) = (len(M), len(M[0]))
    for i in range(l):
        max_j = 0
        for j in range(1, c):
            if M[i][j] > M[i][max_j]:
                max_j = j
        if M[i][max_j] < mini:
            mini = M[i][max_j]
            (min_i, min_j) = (i, max_j)
    return (min_i, min_j)
```

- 2) La fonction `symmetric(M)` vérifie si la matrice M est symétrique selon un axe vertical (symétrie horizontale).

```
def symmetric(M):
    (l, c) = (len(M), len(M[0]))
    cdiv2 = c // 2
    i = 0
    stop = False
    while i < l and not stop:
        j = 0
        while j < cdiv2 and M[i][j] == M[i][c-j-1]:
            j += 1
        stop = (j < cdiv2)
        i += 1
    return not stop
```