Épreuve d'informatique de l'X - 2004 - PSI

Question 1 : Pour $s = 0$, on a $E = \langle 0, 1, 2, 3, ..., (n-1)\rangle$ et pour $s = 1$ avec $c_i \sim c_j$ on obtient deux solutions qui sont $E = \langle 0, 1, \ldots, i, \ldots, i, \ldots, (n-1)\rangle$ et $E = \langle 0, 1, \ldots, j, \ldots, j, \ldots, (n-1)\rangle$ où les i et les j sont en position i et j dans les tableaux précédents.

Question 2 :

```
def representant(E, i):
    ami = E[i]
    while (E[ami] != ami):
        ami = E[ami]
    return(ami)
```

Question 3 :

```
def sontAmis(E, i, j):
    return(representant(E, i) == representant(E, j));
```

Question 4 : On choisit comme représentant de la fusion des cercles d'amis de i et de j, le représentant du cercle d'amis de i. On remarque que si i et j appartiennent au même cercle d'amis, il n'y a pas de modification du tableau E. La fonction « `représentant` » s'exécutant en temps linéaire par rapport à n (au plus un parcours dans le tableau E), il en est de même pour la fonction « `ajout` » (on a au plus deux parcours du tableau E).

En conclusion, on a un temps d'exécution linéaire.

```
def ajout(E, i, j):
    E[representant(E, j)] = representant(E, i)
```

Question 5 : Si à l'arrivée le tableau E vérifie $E[i] = representant(E, i)$ pour tout i de $[\![0, n-1]\!]$, la fonction « `représentant` » s'exécutera en temps constant. Ce sera le cas si on ajoute les $c_i \sim c_{i+1}$ par ordre croissant pour i variant de 0 à $n-2$ pour lequel on obtient alors $E = \langle 0, 0, 0, ..., 0\rangle$. Par contre l'ajout des $c_i \sim c_{i+1}$ par ordre décroissant, pour i variant de $n-2$ à 0, nous donne $E = \langle 0, 0, 1, 2, \ldots, n-2\rangle$ qui entraîne le fait que la fonction « `representant` » s'exécute en temps linéaire.
Une solution pour garantir un meilleur temps d'exécution est de prendre en compte les relations $c_i \sim c_j$ à condition de les avoir rangé par ordre « croissant » avant. Dans la relation $c_i \sim c_j$, on impose $i < j$ et on dira que $c_{i_k} \sim c_{j_k}$ est inférieur à $c_{i_k} \sim c_{j_k}$ si et seulement si $(i_k = i_{k+1}$ et $j_k <= j_{k+1})$ ou $i_k < i_{k+1}$.

Question 6 :

```
def couleurNO(T, i, j):
    if i == 0 or j == 0:
        return(0)
    else :
        return(T[i-1,j-1])

def couleurN(T, i, j):
    if i==0:
        return(0)
    else :
        return(T[i-1,j])

def couleurO(T, i, j):
    if j==0 :
        return(0)
    else :
        return(T[i,j-1])
```

Question 7 :

```
nc = 0
C = 20
E = C*[0]
for i in range(C):
    E[i] = i # on initialise E

def marquage(T):
    global nc, R, E, C
    for i in range(R):
        for j in range(R):
            K = T[i,j]
            if K != 0 :
                no = couleurNO(T, i, j)
                if no != 0 :
                    T[i,j] = no
                else :
                    o = couleurO(T, i, j)
                    n = couleurN(T, i, j)
                    if n == 0 and o != 0 :
                        T[i,j] = o
                    elif n != 0 and (o == 0 or o == n) :
                        T[i,j] = n
                    elif n != 0 and o != 0 :
                        ajout(E, n, o)
                        T[i,j] = n
                    else:
                        nc = nc + 1
                        T[i,j] = nc
```

En considérant que le nombre de couleurs utilisées est du même ordre de grandeur que le nombre d'objets qui a priori est faible devant le nombre de pixels, les fonctions « `couleurO` », « `couleurN` », « `couleurNO` », la fonction `marquage` va s'exécuter en temps quadratique.

Question 8 :

```
def diffusion(T):
    R = len(T)
    for j in range(R):
        for i in range(R):
            if T[i, j] != 0 :
                T[i, j] = E[representant(E, T[i, j])]
```

La fonction « `representant` » s'exécutant, dans le pire des cas en temps linéaire par rapport à C, la double boucle de la fonction « `diffusion` » va s'exécuter en $O(C.R^2)$

```
def colorier(T):
    marquage(T)
    diffusion(T)
```