Corrigé X-ENS 2022

Informatique tronc commun 2ème année

1)

```
def contraire(dir) :
    if dir == H :
        return B
    if dir == B :
        return H
    if dir == D :
        return G
    if dir == G :
        return D
```

```
def est_sans_rebroussement(g) :
    dernier = B
    for elem in g :
        if dernier == contraire(elem) :
            return False
        dernier = elem
    return True
```

2)

```
def est_une_vallee(g) :
    descente = True
    for elem in g :
        if elem == G or elem == B and not descente :
            return False
        elif elem == H :
            descente = False
    return True
```

3)

```
def voisin(x,y,d) :
    if d == D :
        return (x+1,y)
    elif d == G :
        return (x-1,y)
    elif d == H :
        return (x,y-1)
    else :
        return (x,y+1)
```

4)

```
def liste_des_points(g) :
    rep = [(0,0)]
    for elem in g :
        (x,y) = rep[len(rep)-1]
        rep.append(voisin(x,y,elem))
    return rep
```

5)

```
def est_simple(g) :
    points = liste_des_points(g)
    x_deb, y_deb = points[0]
    x_fin, y_fin = points[-1]
    if x_fin == x_deb :
        return False
    points_vu = [points[0],points[-1]]
    for i in range (0,len(points) - 1) :
        pos_x, pos_y = points[i]
        if pos_x == x_deb and pos_y <= y_deb :
            return False
        elif pos_x == x_fin and pos_y <= y_fin :
            return False
        elif points[i] in points_vu :
            return False
        points_vu.append(points[i])
    return True
```

On notera n = len(g). La fonction **liste_des_points(g)** est de complexité $O(n)$.
À l'intérieur de la boucle de **est_simple(g)**, il n'y a qu'une opération qui n'est pas en $O(1)$, c'est points[i] in points_vu, qui se déroule en la longueur de points_vu. Comme la longueur de points_vu va de 2 à n, la complexité de la boucle est en $O(n^2)$. C'est la complexité de **est_simple(g)**.

6)

```
def fond(v) :
    points = liste_des_points(v)
    x_max, y_max = 0, 0
    for coord in points :
        pos_x, pos_y = coord
        if pos_y > y_max :
            y_max = pos_y
            x_max = pos_x
    return (x_max,y_max)
```

7)

```
def plateaux(v) :
    points = liste_des_points(v)
    x_debplat, y_debplat = 0, 0
    rep = []
    for coord in points :
        pos_x, pos_y = coord
        if y_debplat != pos_y and pos_x != x_debplat :
            rep.append((x_debplat,pos_x,y_debplat)
            x_debplat, y_debplat = pos_x, pos_y
        else :
            y_debplat = pos_y
    fin_x, fin_y = points[-1]
    if x_debplat != fin_x :
        rep.append((x_debplat,fin_x,y_debplat)
    return rep
```

La fonction **plateaux(v)** est bien de complexité linéaire, car ce que l'on fait dans la boucle est en $O(n)$ en posant $n = len(v)$ et la fonction **liste_des_points(v)** est aussi en $O(n)$.

8)

Dans la fonction qui suit, **decomposition_en_rectangle**, on va d'abord chercher le plateau du fond. Ensuite, on regarde au fur et à mesure quelle plateau en allant vers la droite ou la gauche est le plus bas. En fonction de la différence entre la hauteur à laquelle finissait le rectangle précédent et la hauteur du nouveau plateau étudiée, on ajoute un rectangle dont la hauteur est la différence. La largeur du rectangle est en fait la largeur de tous les plateaux qui sont en dessous de la hauteur du rectangle, comme nous nous trouvons dont un vallée.

La longueur de la fonction est dû au fait qu'il faut prendre en compte le cas où le plateau de droite est le plus bas, le cas où le plateau de gauche est le plus bas, et les cas où les plateaux à droite où à gauche sont épuisés. La complexité est cependant linéaire, chacune des boucles est dans le pire des cas réalisés n fois avec n la longueur de v, et les opérations présentes dans les boucles sont en O(n).

Fonction début page suivante

9)

```
def hauteur_de_l_eau(t,v) :
    rectangles = decomposition_en_rectangles(v)
    h = 0.0
    t_actuel = 0.0
    for rect in rectangles :
        l_rect,h_rect = rect
        if h_rect != -1 :
            if l_rect*h_rect + t_actuel > t :
                return h + (t - t_actuel)/l_rect
            else :
                t_actuel = t_actuel + h_rect * l_rect
                h = h + h_rect
        else :
            return h + (t - t_actuel)/l_rect
```

```python
def decomposition_en_rectangles(v) :
    platx = plateaux(v)
    x_fond, y_fond = fond(v)
    rep = []

    # Recherche du plateau le plus bas
    for i in range(len(platx)) :
        plat = platx[i]
        x_deb, x_fin, y_plat = plat
        if y_plat = y_fond :
            # La largeur du rectangle le plus bas est la
            # longueur du plateau au fond de la vallée.
            i_fond = i
            y_rect = y_plat
            largeur_rect = x_fin - x_deb
            break
    # Fin recherche
    i_g = i_fond - 1
    i_d = i_fond + 1

    while i_g >= 0 and i_d <= len(platx) - 1 :
        # Il y a encore des plateaux à gauche ou à droite
        plat_g = platx[i_g]
        plat_d = platx[i_d]
        gxdeb, gxfin, gyplat = plat_g
        dxdeb, dxfin, dyplat = plat_d
        if dyplat > gyplat :
            if dyplat < y_fond :
                rep.append((largeur_rect,y_rect - dyplat)
            largeur_rect = largeur_rect + (dx_fin - dx_deb)
            i_d = i_d + 1
        else :
            ff gyplat < y_fond :
                rep.append((largeur_rect,y_rect - gyplat)
            largeur_rect = largeur_rect + (gx_fin - gx_deb)
            i_g = i_g - 1
    if i_g < 0 :
        # Il n'y a plus de plateau à gauche
        while i_d <= len(platx) - 1 :
            plat_d = platx[i_d]
            dxdeb, dxfin, dyplat = plat_d
            if dyplat < y_fond :
                rep.append((largeur_rect,y_rect - dyplat)
            largeur_rect = largeur_rect + (dx_fin - dx_deb)
            i_d = i_d + 1
    else :
        # Il n'y a plus de plateau à droite
        while i_g >= 0 :
            plat_g = platx[i_g]
            gxdeb, gxfin, gyplat = plat_d
            if gyplat < y_fond :
                rep.append((largeur_rect,y_rect - dyplat)
            largeur_rect = largeur_rect + (dx_fin - dx_deb)
            i_g = i_g - 1
    return rep
```