

Rapport TP1 ACT

Gaspar Henniaux - Marwane Ouaret

1

1.1

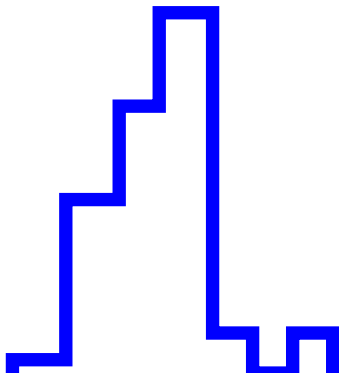
- On a une ligne oblique quand on passe du point (2, 5) au point (4, 4) ce n'est donc pas une ligne de toit.
- De même entre les points (2, 0) et (1, 4).
- Pour la troisième polyligne on a une ligne de toit car tous les traits sont verticaux ou horizontaux.
- Les couples (6, 7) et (5, 0) forment un trait oblique.
- Pour la cinquième polyligne on a pas de ligne de toit car tous les traits sont verticaux ou horizontaux mais on note tout de même qu'il y a un pic entre les points (4, 8) et (4, 7), il n'y a pas de "plafond".

1.2

pour une liste de couples, (c_0, \dots, c_n) soit un couple c_x et c_{x+1} pair alors ces 2 couples sont de formats (A, B) (A, C) (inversement si impair).

1.3

Soit une liste de couples, (c_0, \dots, c_n) Pour passer de l'écriture brute à l'écriture compacte il suffit de supprimer chaque couple de numéro pair



PROF

2

```
N : nombre d'immeuble
L : liste de triplet de la forme (g, h, d)
H : h max
D : d max

initialiser la matrice M selon H et D à false
pour i allant de 0 à N :
```

```

    pour j allant de 0 à L[i][1]:
        pour k allant de L[i][0] à L[i][2]:
            M[j, k] -> True
dessiner la ligne (complexité :  $O(H*D)$ )

```

complexité : $O(NHD)$

désavantages : on passe des "pixels" à true plusieurs fois,

3

```

Li : ligne de toit
N : nombre d'immeuble
L : liste de triplet de la forme (g, h, d)
H : h max
D : d max

Li = []
Pour i=0 à N                                O(N)
    Li = ajouter_immeuble(Li, L[i])          O(N)

ajouter_immeuble(Li, [g,h,d] ){
    rep = []
    test = 0
    i = 0

    Si Li vide
        alors ajouter (g,h)(d,0) à rep

    sinon
        Pour chaque segment (x1, y1) (x2,y2) dans Li

            si g >= x2 ou d <= x1
                alors si i == 0
                    alors ajouter (x1, y1) (x2,y2) à rep
                sinon ajouter (x2,y2) à rep
                Si test == 0
                    test = 1
            sinon
                test = 2

            si x1 <= g et x2 >= d
                si h > y1
                    si i == 0
                        alors ajouter (x1, y1) (g,h) (d, y1) (x2,
y2) à rep
                    sinon

```

```

        alors ajouter (g,h) (d, y1) (x2, y2) à rep
    sinon
        si i == 0
            ajouter (x1,y1) (x2,y2) à rep
        sinon
            ajouter (x2,y2) à rep

    sinon si x1 => g et x2 <= d
        si h < y1
            si i == 0
                ajouter (g, h) (x1, y1) (x2 , h) (d, 0) à
rep
            sinon
                retirer dernier element de rep
                ajouter (g, h) (x1, y1) (x2 , h) (d, 0) à
rep
        sinon
            si i == 0
                ajouter (g,h)(d,0) à rep
            sinon
                retirer dernier element de rep
                ajouter (g,h)(d,0) à rep

    sinon si x1 <= g et x2 <= d
        si h > y1
            si i == 0
                alors ajouter (x1, y1) (g,h) (d, 0) à rep
            sinon
                ajouter (g,h) (d, 0) à rep
        sinon
            si i == 0
                ajouter (x1,y1) (x2,h) (d, 0) à rep
            sinon
                ajouter (x2,h) (d, 0) à rep

    sinon si x1 >= g et x2 >= d
        si h > y1
            alors ajouter (g, h) (d,y1) (x2, y2) à rep
        sinon
            ajouter (g,h) (x1,y1) (x2, y2) à rep
    i++

Si test = 1
    ajouter (g,h)(d,0) à rep

Supprimer doublon cote à cote dans rep ainsi que les couples de
format (x ,y1) (x, y2) et garder celui dont la valeur y est la plus
grande0(n)

return rep

```

```
}
```

La fonction ajouter immeuble étant $O(n)$ car dans le pire des cas à au dernier appel de cette fonction la boucle fera autant d'itérations qu'il y a d'immeubles,

La suppression de doublon étant $O(n)$ alors la fonction ajouter immeuble est $O(n + n)$ soit $O(2n) \rightarrow O(n)$

Cette fonction étant appelé N fois pour insérer tous les immeubles alors le programme est $O(n*n)$, soit $O(n^2)$

4

```
fonction merge_roof_line(l1, l2)
    i1 = 0
    i2 = 0
    h1 = 0
    h2 = 0
    d = 0
    hMax = 0
    merged = liste vide

    tant que i1 est inférieur à longueur de l1 et i2 est inférieur à
    longueur de l2 :
        si l1[i1][0] < l2[i2][0] :
            d = l1[i1][0]
            h1 = l1[i1][1]
            hMax = maximum entre h1 et h2
            i1 += 1
        sinon :
            si l1[i1][0] > l2[i2][0] :
                d = l2[i2][0]
                h2 = l2[i2][1]
                hMax = maximum entre h1 et h2
                i2 += 1
            sinon :
                d = l1[i1][0]
                h1 = l1[i1][1]
                h2 = l2[i2][1]
                hMax = maximum entre h1 et h2
                i1 += 1
                i2 += 1

    si merged est vide ou hMax est différent du dernier élément de
    merged :
```

ajouter (d, hMax) à merged

ajouter le reste de l1[i1:] à merged

ajouter le reste de l2[i2:] à merged

retourner merged

voici le code en python :

```
def merge_roof_line(l1, l2):
    i1 = 0
    i2 = 0
    h1 = 0
    h2 = 0
    d = 0
    hMax = 0
    merged = []

    while i1 < len(l1) and i2 < len(l2):
        if l1[i1][0] < l2[i2][0]:
            d = l1[i1][0]
            h1 = l1[i1][1]
            hMax = max(h1, h2)
            i1 += 1
        else:
            if l1[i1][0] > l2[i2][0]:
                d = l2[i2][0]
                h2 = l2[i2][1]
                hMax = max(h1, h2)
                i2 += 1
            else:
                d = l1[i1][0]
                h1 = l1[i1][1]
                h2 = l2[i2][1]
                hMax = max(h1, h2)
                i1 += 1
                i2 += 1

        if len(merged) == 0 or hMax != merged[-1][1]:
            merged.append((d, hMax))

    merged += l1[i1:]
    merged += l2[i2:]

    return merged
```

```

fonction divide_roof_line(liste l)
    si la longueur de l est égale à 1 :
        retourner l[0]
    sinon :
        partie_gauche = divide_roof_line(l[:longueur(l)//2])
        partie_droite = divide_roof_line(l[longueur(l)//2:])
        retourner merge_roof_line(partie_gauche, partie_droite)

fonction building_to_roof_line(liste l)
    roofs = liste vide

    pour i de 0 à longueur(l) - 1 :
        roof = liste vide
        ajouter (l[i][0], l[i][1]) à roof
        ajouter (l[i][2], 0) à roof
        ajouter roof à roofs

    retourner divide_roof_line(roofs)

```

voici le code en python :

```

def divide_roof_line(l):
    if len(l) == 1:
        return l[0]
    else:
        return merge_roof_line(divide_roof_line(l[:len(l)//2]),
                                divide_roof_line(l[len(l)//2:]))

def building_to_roof_line(l):
    roofs = []
    for i in range(len(l)):
        roof = []
        roof.append((l[i][0], l[i][1]))
        roof.append((l[i][2], 0))
        roofs.append(roof)
    return divide_roof_line(roofs)

```