Benjamin
LOISON
MP*
1/3

28/03/20

Informatique :

I.A :

```
SELECT COUNT( * ) FROM vol WHERE jour = '2016-05-02'
AND heure <= '12:00'
```

I.B :

```
SELECT id_vol FROM vol JOIN aeroport ON id_aero =
depart WHERE ville = 'Paris' AND jour = '2016-05-02'
```

I.C :

Cette requête liste tous les numéros de vols de la France vers la France ayant un décollage souhaité pour le 2 mai 2016.

I.D :

```
SELECT v1.id_vol, v2.id_vol FROM vol AS v1 JOIN vol AS v2 ON
v2.depart = v1.arrivee AND v2.arrivee = v1.depart AND v1.niveau =
v2.niveau AND v1.jour = v2.jour. Pour avoir unicité des conflits, on peut
rajouter AND v1.id_vol < v2.id_vol.
```

II.A.1) :

```
def nb_conflits():
    s = 0
    for i in range(1,len(conflit)):
        for j in range(i):
            if conflit[i][j] != 0:
                s += 1
    return s
```

II.A.2:

La complexité de l'algorithme est quadratique en n.

II.B.1:

```
def nb_vol_par_niveau_relatif(regulation):
    L = [0, 0, 0]
    for r in regulation:
        L[r] += 1
    return L
```

II.B.2a)

```
def cout_regulation(regulation):
    s = 0; n = len(regulation)
    for i in range(n):
        l = 3*i + regulation[i]
        for j in range(i+1,n):
            s += conflit[l][3*j + regulation[j]]
    return s
```

II.B.2.b:

Cette fonction a une complexité quadratique en n.

II.B.2.c :

```
def cout_RFL():
    regulation = [0] * (len(conflit)//3)
    return cout_regulation(regulation)
```

II.B.3:

Il y a 3^n régulations possibles pour n vols.
En réalité n est très grand, le temps de calcul serait trop long, c'est donc inenvisageable.

II.C.1.a:

```
def cout_du_sommet(s, etat_sommet):
    s = 0
    for i in range(len(etat_sommet)):
        if etat_sommet[i] >= 1:
            s += conflit[s][i]
    return s
```

II.C.1.b:

cout_du_sommet est linéaire en n.

II.c.2.a:

```
def sommet_de_cout_min(etat_sommet):
    i, s = -1, -1
    for j in range(len(etat_sommet)):
        if etat_sommet[j]==2:
            t = cout_du_sommet(j, etat_sommet)
            if t<s or i==-1:
                s = t
                i = j
    return i
```

II.c2.b:

La fonction sommet_de_cout_min est quadratique en n.

II.c.3.a:

```
def minimal(l):
    N = len(conflit)

    regulation = [0]*(N//3)
    etat_sommet = [2]*N
    for i in range(N//3):
        s = sommet_de_cout_min(etat_sommet)
        regulation[s//3] = s%3
        etat_sommet[s] = 1
        for j in range(3):
            if j != (s%3):
                etat_sommet[3*(s//3)+j] = 0
    return regulation
```

Benjamin
LOISON
MP*
2/3

II.c 3.b:

La fonction minimal a une complexité en n^3.
Comparé aux 3^n possibilités, cette fonction semble efficace.

II.d:

```
def recuit(regulation):
    T = 1000
    n = len(regulation); regulationTmp : regulation[:]
    while T >= 1:
        h = randint(n)
        L = []
        for i in range(3):
            if i != regulation[h]:
                L.append(i)
        r = L[randint(2)]
        regulationTmp[h] = r
        cout0, cout1 = cout_regulation(regulation), cout_regulation(regulationTmp)
        if cout1 < cout0 or random() <= exp((cout0-cout1)/T):
            regulation[h] = r
        T = T*0.99
```

III.A.1:

Le débit est de 1 bit par µs. Il y a 16 bits de structure donc 112 disponibles pour les données.

III.A.2:

Il y a respectivement 64000 et 10000 valeurs possibles pour l'altitude et la vitesse.
On a $24 + \lceil \log_2 64000 \rceil + \lceil \log_2 10000 \rceil <= 128$
car $\lceil \log_2 64000 \rceil = 16$
et $\lceil \log_2 10000 \rceil = 14$
Donc on en message les données peuvent être envoyées.

III. A.3:

On considère $4 \times 8 = 32$ octets par entiers.
En une seconde il faut 3 200 octets.
Il y a 3600×100 secondes dans 100 heures.
Donc il faut $32 \times 100 \times 36 \times 10000$ octets
Donc 1 152 000 000 octets soit 1,152 to.
Un disque dur est de l'ordre du téraoctet donc cette quantité est plutôt faible.

III. B.1:

$\forall t \geq t_0$, on a $\overrightarrow{OG}(t) = \overrightarrow{OG}(t_0) + (t - t_0)\vec{V}$

III. B.2:
On note $\overrightarrow{OG}(t_0) = (x_0, y_0, z_0)$ et $\vec{V} = (v_x, v_y, v_z)$ dans la base étudiée
On a $OG^2(t) = (x_0 + (t-t_0)v_x)^2 + (y_0 + (t-t_0)v_y) + (z_0 + (t-t_0)v_z)^2$
Donc $\frac{dOG^2(t)}{dt} = 2((x v_x + y v_y + z v_z) + (v_x^2 + v_y^2 + v_z^2)(t - t_0))$

Donc on a $t_c = t_0 - \frac{x v_x + y v_y + z v_z}{v_x^2 + v_y^2 + v_z^2}$

III.B.3:

Avoir $\overrightarrow{O_0O}(t_0) \cdot \vec{V} \geq 0$ revient ici à avoir $x_0 v_x + y_0 v_y + z_0 v_z \geq 0$
Donc en regardant la dérivée de $\overrightarrow{OO}(t)$ par rapport au temps, si cette condition est satisfaite, la distance entre les deux avions ne peut que croître.

III.B.4:

```
def calculer_CPA(intrus):
    id, x, y, z, vx, vy, vz, t0 = intrus; q = x*vx + y*vy + z*vz
    if q >= 0:
        return None
    tCPA = t0 - q / ((vx**2) + (vy**2) + (vz**2))
    xCPA, yCPA, zCPA = x + vx*(tCPA - t0), y + vy*(tCPA - t0)
                       , z + vz*(tCPA - t0)
    dCPA = sqrt(xCPA**2 + yCPA**2 + zCPA**2)
    return tCPA, dCPA, zCPA
```

Remarquons que l'on peut simplifier tCPA et xCPA, yCPA et zCPA en t0 n'annule.

III.C.1:

```
def mettre_a_jour_CPAs(CPAs, id, nv_CPA, intrus_max,
                       suivi_max):
    h = len(CPAs)
    estSuivi = False; CPAi = -1
    for i in range(h):
        CPA = CPAs[i]
        if CPA[0] == id:
            CPAi = i
            estSuivi = True
            break
    if estSuivi : ... (nv_CPA == None or nv_CPA[0] >= suivi_max):
        if
```

```
            del (CPAs[CPAi])
            CPAi = -1
        else:
            CPAs[CPAi][1:] = nv_CPA
    else :
        if n < intrus_mosc:
            CPAs.append(nv_CPA)
            CPAi = n-1
        else if CPAs[-1][1] > nv_CPA[1]:
            CPAs[-1] = [id, nv_CPA[0], nv_CPA[1], nv_CPA[2]]
            CPAi = n-1
    if CPAi != -1:
        return CPAi
```

II.C.2:

```
def replacer(ligne, CPAs):
    n = len(CPAs)
    CPA = CPAs[ligne]
    del(CPAs[ligne])
    i = 0
    while i < n and CPAs[i][1] < CPA[1]:
        i += 1
    CPAs.insert(ligne, CPA)
```

III.C3:

```
def enregistrer_CPA(intrus, CPAs, intrus_mosc, suivi_mosc):
    CPA = calculer_CPA(intrus)
    ligne = mettre_a_jour_CPAs(CPAs, intrus[0], CPA, intrus_mosc, suivi_mosc)
    if ligne != None:
        replacer(ligne, CPAs)
```

Benjamin
LOISON
MP*

III.D.1:

Chaque avion parcourt 250 mètres en une seconde donc en 240 secondes, il parcourt 60 km.
Donc si les deux avions vont à cette vitesse dans le pire cas (comme un face à face), il n'y a que 120 secondes qui s'écoule entre la première détection et le CPA. La valeur de suivi_max semble alors appropriée.

III.D2:

Si l'un monte et l'autre descend, chacun doit se translater de 250 pieds, ce qui est fait en 10 secondes

III.D.3:

~~Si la boucle d'éxécution du TCAS s'éxécute bien, l'alarme est enclenché avec les 25 CPA~~

On a de part et d'autre 15 secondes pour commencer les manoeuvres, ce qui est correct.

III.D.4:

La fonction TCAS a maximum 1000/30 ≈ 33 ms pour éxécuter une fois sa boucle.

III.D.5:

La faible quantité d'intrus à traiter (30) laisse présager que l'émission est le plus chronophage.
Cette dernière prend 128 * (60/300 000) × 10^6 ≈ 650 µs

Or $0,65 < 33$ donc les spécifications TCAS sont respectées.