Centrale Informatique PSI 2016 — Corrigé

Ce corrigé est proposé par Cyril Ravat (Professeur en CPGE) ; il a été relu par Jean-Julien Fleck (Professeur en CPGE) et Benjamin Monmege (Enseignant-chercheur à l'université).

Ce sujet d'informatique s'intéresse aux méthodes de prévention des collisions aériennes. Il est découpé en trois parties indépendantes de longueurs très différentes.

- La première partie traite des plans de vol et de la gestion d'une base de données contenant les indications de chaque vol. Toutes les questions portent sur les bases de données ; leur difficulté est progressive. Cette partie donne un bon aperçu des connaissances attendues dans ce domaine.
- La deuxième partie aborde le choix des niveaux de vol. Elle introduit pour cela un graphe pondéré non orienté. Cette notion n'est pas explicitement au programme, car elle ne figure que dans les activités facultatives de deuxième année ; néanmoins, tout le vocabulaire associé (arête, sommet, valuation) est rappelé par l'énoncé et la traduction du graphe sous forme de matrice d'adjacence est entièrement donnée. Remarquons cependant que les candidats en MP option informatique avaient un net avantage sur les autres, puisque cette notion est au programme de l'option.

  L'objet de cette partie est l'écriture de fonctions élémentaires codant un algorithme de minimisation d'un coût global. Les questions de programmation demandent du soin, notamment quand elles utilisent des listes de listes, mais ne posent pas de difficulté conceptuelle majeure. Pour chaque fonction, l'énoncé demande de calculer sa complexité. La partie s'achève avec l'algorithme du recuit simulé (un classique de l'optimisation), qui est entièrement décrit et qu'il faut convertir en fonctions dans le langage Python.
- La troisième partie décrit les grandes lignes du système de prévention automatique de collisions TCAS. Cette partie est dans l'ensemble beaucoup moins claire que la précédente. Le système est composé d'un certain nombre de boîtes noires dont on ne comprend pas toujours l'intérêt. Les questions de la première sous-partie, reliées à la représentation des nombres, sont plutôt simples. La deuxième sous-partie consiste à calculer le temps avant collision et commence par deux questions de mécanique élémentaire. La troisième sous-partie contient à nouveau de la programmation Python. La dernière sous-partie, beaucoup moins intéressante que le reste de l'épreuve, est faite de questions qualitatives, simplistes ou mal posées, sur le temps d'exécution de l'algorithme global. On ne sait pas trop ce qui est évalué dans ces questions.

Ce problème présente des questions d'intérêt inégal. De nombreuses questions de programmation sont cependant intéressantes, notamment dans la deuxième partie. Notons qu'il ne fait appel qu'au programme de première année et que les techniques d'ingénierie numérique ne sont pas abordées.

Indications

Partie I

I.A La fonction comptant un nombre d'enregistrements est `COUNT(*)`.

I.B La sélection exploite des informations présentes dans deux tables : il faut réaliser une jointure.

I.D Il faut joindre deux fois la table `vol` : il est nécessaire de la renommer pour distinguer chaque occurrence (on peut s'inspirer de la question précédente).

Partie II

II.A.1 Il s'agit d'une double itération, donc de deux boucles imbriquées. Attention à ne pas tout compter en double...

II.B.2 La structure de l'algorithme est proche de celle de la question II.A.1, mais avec des compteurs variant sur le nombre de vols. Penser à utiliser les indications de l'énoncé sur les liens entre s, r_k et k.

II.C.1 Rechercher les sommets non supprimés et ajouter le coût correspondant. Que leur état soit égal à 1 ou à 2 n'intervient pas.

II.C.2 Il s'agit d'une recherche de minimum, algorithme simple à connaître. Penser à ne parcourir que les sommets ni supprimés ni déjà choisis.

II.C.3 Il faut se demander si l'on sait combien d'itérations l'algorithme doit effectuer. La boucle doit être précédée de l'initialisation des variables (liste des états et des régulations) ; son contenu est très bien décrit dans l'énoncé.

II.D La modification aléatoire de r_k peut être longue à écrire : penser à une permutation circulaire d'une ou deux cases sur un total de trois peut permettre d'obtenir un code plus simple.

Partie III

III.B.1 Le référentiel d'étude $\mathcal{R}_0$ est défini à la question III.A.3. Il est lié à l'avion propre, qui y est donc immobile.

III.B.2 t_c est l'instant où la norme de $\overrightarrow{OG}$ est minimale.

III.C.1 L'ensemble de l'algorithme est donné dans l'énoncé. Il faut commencer par chercher l'avion intrus dans la liste et effectuer les traitements prévus lorsqu'on le trouve. Si chaque possibilité de traitement se conclue par un `return`, la sortie de la boucle de recherche correspond à un avion absent de la liste.

III.C.2 Supprimer l'avion à traiter de la liste avant de la parcourir permet de simplifier et de systématiser le traitement.

III.C.3 Cette question ne présente aucune difficulté, si ce n'est de relire l'introduction de cette partie (pages 4 et 5) et d'avoir bien compris ce que réalisent les différentes fonctions écrites aux questions précédentes.

I. Plan de vol

I.A L'énoncé spécifie que l'on peut comparer une date et une heure avec une chaîne de caractères équivalente. De plus, l'opérateur d'agrégation permettant de compter le nombre d'enregistrements est `COUNT(*)`. On peut ainsi écrire :

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

I.B Les informations à utiliser sont contenues dans deux tables : on doit afficher des identifiants de vol `id_vol` contenus dans la table `vol` et faire la sélection en particulier sur `ville`, dans la table `aeroport`. Il faut donc réaliser une jointure, c'est-à-dire l'assemblage des lignes correspondantes dans les deux tables. La condition de jointure est par conséquent `depart=id_aero`. On obtient :

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

> Dès que l'on utilise deux tables dans la même requête, il y a un risque de nom identique pour deux colonnes. Ce n'est pas le cas ici, mais si les colonnes `id_vol` et `id_aero` s'appelaient toutes les deux `id`, alors le système ne pourrait pas comprendre la requête si on ne précise pas de quel `id` il s'agit. Il faudrait alors écrire respectivement `vol.id` et `aeroport.id`. Il est par ailleurs possible d'écrire ici `vol.id_vol`, mais cela n'apporte rien.

I.C Cette requête réalise une double jointure entre la table `vol` et deux fois la table `aeroport`, afin d'associer à chaque vol la ville et le pays des deux aéroports concernés. Elle sélectionne ensuite les vols du 2 mai 2016 allant d'un aéroport français à un autre aéroport français.

Cette requête liste les vols nationaux français du 2 mai 2016.

I.D Il est nécessaire d'utiliser deux fois la table `vol`. Comme dans l'exemple donné à la question précédente, il faut renommer les tables. Les conditions de jointure et de sélection sont ici exceptionnellement proches, au point que plusieurs solutions sont possibles en déplaçant les conditions du `ON` vers le `WHERE` ou inversement. Par exemple, en considérant que l'on joint les tables pour les vols ayant lieu le même jour au même niveau, puis que l'on sélectionne les vols ayant les mêmes points de départ et d'arrivée, on a

```
SELECT v1.id_vol AS id1, v2.id_vol AS id2
FROM vol AS v1 JOIN vol AS v2
ON v1.jour=v2.jour AND v1.niveau=v2.niveau
WHERE v1.depart=v2.arrivee AND v1.arrivee=v2.depart
AND v1.id_vol<v2.id_vol
```

> On aurait donc pu aussi joindre les tables à la condition d'aéroports correspondants, et réaliser la sélection sur les égalités de jour et de niveau. On pourrait même positionner les 4 égalités dans la condition de jointure.
>
> La dernière inégalité dans la condition de sélection sert à n'afficher qu'une seule fois chaque couple : sans elle, chaque couple de vols serait affiché deux fois, avec `id1` et `id2` permutant leurs valeurs. Demander cette condition supprime l'une des deux lignes. Les `id_vol` étant des chaînes de caractères, elles sont classiquement ordonnées suivant l'ordre lexicographique (ordre du dictionnaire, prenant en compte chiffres et lettres). On pourrait aussi imaginer

utiliser l'ordre des heures de départ (`AND v1.heure<v2.heure`), mais avec le risque de supprimer les couples de vols partant à la même heure (ou de compter en double ces vols avec `<=`).

II. Allocation des niveaux de vol

II.A.1 L'énoncé demande de déterminer le nombre de conflits, c'est-à-dire le nombre de cases non nulles dans la variable `conflit` : il faut utiliser deux boucles imbriquées, une sur les lignes (éléments de `conflit`) et une sur les colonnes (éléments de chaque élément de `conflit`). La principale difficulté est en réalité de ne parcourir qu'une moitié du tableau, au-dessus ou au-dessous de la première diagonale. Il faut donc que les bornes du compteur de la deuxième boucle dépendent de la valeur du compteur de la première. On écrit ainsi

```
def nb_conflits():
    N = len(conflit)              # N vaut 3n
    resultat = 0                  # Résultat initialisé à zéro
    for i in range(N):            # Autre possibilité : range(1,N)
        for j in range(i+1,N):    # Autre possibilité : range(i)
            if conflit[i][j] > 0:
                resultat = resultat + 1
    return resultat
```

Une autre possibilité est de parcourir l'ensemble de la grille, `i` et `j` allant tous deux de 0 à $N-1$. Bien sûr, le résultat sera le double de celui recherché, il faudra donc penser à le diviser par deux. Une telle réponse ne devrait normalement pas être pénalisée, car la complexité reste du même ordre que celle de la fonction écrite ci-dessus.

Si on préfère rester sur le parcours d'une moitié de tableau, autant faire attention à ne pas prendre en compte la diagonale : c'est pour cela que j commence à $i+1$ (ou s'arrête à $i-1$ dans la variante). La diagonale ne contient cependant que des zéros, les comptabiliser ne change pas le résultat.

II.A.2 Avant la première boucle, il y a deux opérations de complexité indépendante de la taille de `conflit`. À l'intérieur des deux boucles entièrement imbriquées, il n'y a aussi que des opérations de complexité constante. Le calcul de complexité revient donc à un calcul de nombre d'itérations de la deuxième boucle. Lorsque i vaut 0, elle est exécutée $N-1$ fois. Lorsque i vaut 1, elle est exécutée $N-2$ fois, et ainsi de suite : le nombre total d'opérations à effectuer peut s'écrire $k_1 + k_2\,((N-1)+(N-2)+\cdots+1+0) = k_1 + k_2\,(N-1)(N-2)/2$. Comme $N = 3\,n$,

La complexité de la fonction `nb_conflits` est en $O(n^2)$.

On attend pour un calcul de complexité, et plus particulièrement pour le premier calcul de ce genre dans une épreuve, qu'il soit précisément justifié. En particulier, il ne faut pas se contenter de « Il y a deux boucles donc $O(n^2)$. » Le calcul faisant apparaître la somme et sa transformation en produit doit être présent. On pourra en revanche aller plus vite sur les justifications suivantes.

II.B.1 Pour réaliser cette fonction, on peut utiliser une liste `abc`, mise à jour à chaque valeur `r` de `regulation` : si `r` vaut 0, il faut incrémenter `abc[0]`, si `r` vaut 1, il faut incrémenter `abc[1]`, si `r` vaut 2, il faut incrémenter `abc[2]`. On obtient