Devoir d'informatique 5 – Corrigé

D.Malka – MPSI 2020-2021 – Lycée Jeanne d'Albret

11.06.2021

Exercice 1 – Base de données universitaire

1. Requête affichant les noms et prénoms des étudiants salariés.

```
SELECT nom, prenom
FROM  etudiants
WHERE salarie=1
```

2. Requête affichant les numéros étudiant, les noms et prénoms des étudiants de L1.

```
SELECT e.ne, e.nom, e.prenom
FROM  etudiants e
JOIN formation f ON f.id_formation=e.id_formation
WHERE f.type="L1"
```

3. Requête affichant les noms et prénoms des étudiants inscrits en M1 et ayant suivi le module « Informatique ».

```
SELECT e.nom, e.prenom
FROM etudiants e
JOIN formation f ON f.id_formation==e.id_formation
JOIN module m ON m.id_formation=f.id_formation
WHERE f.type="M1" AND m.libelle_m="Informatique"
```

4. Requête affichant le libellé et le nombre d'étudiants dans chaque formation.

```
SELECT f.libelle_f, COUNT (ne)
FROM formation f
JOIN etudiants e on e.id_formation=f.formation
GROUP BY id_formation
```

5. Requête affichant la liste (numéro d'étudiants, noms, prénoms) ainsi que les moyennes générales de tous les étudiants ayant validé leur année en 2015, quelle que soit la formation.

```
SELECT e.ne, e.nom, e.prenom, AVG(note)
FROM etudiants e
JOIN module m ON m.id_formation=e.id_formation
JOIN resultat r ON r.id_module=m.module
WHERE r.annee=2015
GROUP BY e.ne
HAVING AVG(note)>=10
```

Exercice 2 – Gestion du trafic aérien

Plan de vol

1. Requête SQL qui fournit le nombre de vols qui doivent décoller dans la journée du 2 mai 2016 avant midi.

```
1 SELECT COUNT(*)
2 FROM vol v
3 WHERE
4   v.jour = '2016-05-02' AND
5   v.heure<'12:00';
```

2. Requête SQL qui fournit la liste des numéros de vols au départ d'un aéroport desservant Paris le 2 mai 2016.

```sql
SELECT v.id_vol
FROM vol v
  JOIN aeroport a ON v.arrivee=a.id_aero
WHERE
  v.arrivee='Paris'
  v.jour = '2016-05-02';
```

3. La requête

```sql
SELECT id_vol
FROM vol
  JOIN aeroport AS d ON d.id_aero = depart
  JOIN aeroport AS a ON a.id_aero = arrivee
WHERE
  d.pays = 'France' AND
  a.pays = 'France' AND
  jour = '2016-05-02';
```

renvoie la liste des vols intérieurs à la France le 2 mai 2016.

4. Requête SQL qui fournit la liste des couples (Id1 , Id2) des identifiants des vols dans cette situation.

L'idée est de réaliser une jointure de la table `vol` avec elle même afin de pouvoir comparer le départ et l'arrivée des différents vols. On ne conserve que les n-uplets correspondant à des risques de conflit. La condition `v1.id_vol>v2.id_vol` sert à éliminer les doublons, (Id_1, Id_2) et (Id_2, Id_1) représentant le même conflit.

```sql
SELECT v1.id_vol, v2.id_vol
FROM vol v1
  JOIN vol v2 ON v1.depart=v2.arrivee AND
            v1.arrivee=v2.depart AND
            v1.niveau=v2.niveau AND
            v1.jour=v2.jour AND
WHERE
  v1.id_vol>v2.id_vol;
```

Allocation des niveaux de vol

Implantation du problème

5. Fonction `nb_conflits()` sans paramètre qui renvoie le nombre de conflits potentiels, c'est-à-dire le nombre d'arêtes de valuation non nulle du graphe.

```python
def nb_conflits():
    nb=0
    n=len(conflit)
    for i in range(n):
        for j in range(i+1,n):#par symetrie on ne parcourt que le triangle superieur de la
            matrice
            if conflit[i][j]!=0:
                nb+=1
    return nb
```

6. Complexité : parcours de l'ordre d'environ $\frac{1}{2}.(3n)^2$ arêtes du graphes. On prenant l'addition comme coût élémentaire, on trouve une complexité quadratique : $O(n^2)$.

Régulation

7. Fonction `nb_vol_par_niveau_relatif(regulation)` qui prend en paramètre une régulation (liste de n entiers) et qui renvoie une liste de 3 entiers $[a, b, c]$ dans laquelle a est le nombre de vols à leurs niveaux RFL, b le nombre de vols au-dessus de leurs niveaux RFL et c le nombre de vols au-dessous de leurs niveaux RFL.

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

8. Fonction `cout_regulation(regulation)` qui prend en paramètre une liste représentant une régulation et qui renvoie le coût de celle-ci.

```python
def cout_regulation(regulation):
    c=0
    for k,r in enumerate(regulation):
        s1=3*k+r
        for q,R in enumerate(regulation):
            s2=3*q+R
            c+=conflit[s1][s2]
    return c//2 #matrice symetrique => conflits comptes deux fois
```

9. Complexité : $O(n^2)$.
10. Fonction `cout_RFL()` qui renvoie le coût de la régulation pour laquelle chaque vol est à son RFL.

```python
def cout_RFL():
    n=len(conflit)//3
    all_RFL=[0 for i in range(n)]
    return cout_regulation(all_RFL)
```

11. Il existe 3^n régulations de vols. L'évaluation du coût d'une régulation se fait en $O(n^2)$. Une fonction qui rechercherait la régulation de coût minimal par évaluation de toutes les régulations possibles aurait une complexité $O(3^n.n^2)$ absolument rédhibitoire !

L'algorithme Minimal

12. Fonction `cout_du_sommet(s, etat_sommet)` qui prend en paramètres un numéro de sommet s (n'ayant pas été supprimé) ainsi que la liste `etat_sommet` et qui renvoie le coût du sommet s dans le graphe défini par la variable globale `conflit` et le paramètre `etat_sommet`.

```python
def cout_du_sommet(s,etat_sommet):
    c=0
    for j in range(len(etat_sommet)):
        if etat_sommet[j]!=0:
            c+=conflit[s][j]
    return c
```

13. Complexité de la fonction `cout_du_sommet` : $O(n)$.
14. Fonction `sommet_de_cout_min(etat_sommet)` qui, parmi les sommets qui n'ont pas encore été choisis ou supprimés, renvoie le numéro du sommet de coût minimal.

```python
def maj(s_min,etat_sommet):
    k=(s_min//3)*3;#k: indice principal
    for r in [k,k+1,k+2]:# 0,1,2: indice relatif au vol k
        etat_sommet[r]=0
    etat_sommet[s_min]=1

def sommet_de_cout_min(etat_sommet):
    s_min=0;c_min=cout_du_sommet(s_min,etat_sommet)
    for s in range(1,len(etat_sommet)):
        if etat_sommet[s]==2:#on parcourt les sommets non traites
            c=cout_du_sommet(s,etat_sommet);
            if c<c_min:
                c_min=c;s_min=s
    maj(s_min,etat_sommet) #mise a jour du graphe via etat_sommet
    return s_min#inutile mais demande par l'enonce
```

15. La fonction `sommet_de_cout_min` appelle environ n fois la fonction `cout_du_sommet` de complexité $O(n)$ donc sa complexité est $O(n^2)$.
16. Fonction `minimal()` qui renvoie la régulation résultant de l'application de l'algorithme Minimal.

```
1 def minimal():
2     non_traite=2#sommet ni choisi, ni supprime
3     etat_sommet=[non_traite for i in range(len(conflit))];
4     while non_traite in etat_sommet:
5         sommet_de_cout_min(etat_sommet)
6     opt_reg=[i%3 for i,s in enumerate(etat_sommet) if s==1]#s%3 indice relatif donc niveau
          de vol
7     return opt_reg
```

17. La fonction `minimal()` appelle environ $3n$ fois la fonction `sommet_de_cout_min` de complexité $O(n^2)$ donc sa complexité est $O(n^3)$ donc polynomiale, ce qui est acceptable contrairement à la complexité de la détermination de la régulation optimale par force brute évaluée précédemment. Il n'est pas démontré que l'algorithme utilisé renvoie la régulation de coût minimal mais on préfère une bonne solution non optimale obtenue en un temps raisonnable qu'une solution optimale obtenue en un temps démesuré.