TP/TD D'INFORMATIQUE N°8
Révisions

Exercice 1 : Matrices, SQL (CCP Maths 2023)

Dans cet exercice d'informatique commune, on se propose d'écrire des algorithmes dans le but de faire du calcul matriciel, en particulier sur des matrices d'adjacence de graphe.

Notation

Les matrices sont carrées et représentées par des listes dont les éléments correspondent aux lignes de la matrice. Par exemple, la matrice $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ est représentée par la liste `[[1, 2], [3, 4]]`.

Dans la suite, pour définir la matrice d'adjacence $A \in \mathcal{M}_n(\mathbb{R})$ d'un graphe ayant n sommets, on numérote ses sommets de 0 à $n-1$.

1. Écrire une fonction `produit(A,B)` prenant en arguments deux matrices carrées A et B de mêmes dimensions et qui renvoie AB le produit de la matrice A et de la matrice B.
2. Écrire une fonction `oriente(A)` prenant en argument la matrice d'adjacence A d'un graphe et qui retourne `True` si le graphe est orienté et `False` sinon.
3. On admet que le nombre de chemins de longueur p reliant i à j dans un graphe de matrice d'adjacence A est égal au coefficient d'indice (i, j) de la matrice A^p.
   Écrire une fonction `distance(A,i,j)` où A est la matrice d'adjacence d'un graphe et qui renvoie le nombre minimal d'arêtes que l'on doit parcourir pour atteindre le sommet j depuis le sommet i (on suppose qu'un tel chemin existe).

On considère deux tables CLIENTS et PARTENAIRES. La première contient des informations sur les clients et la deuxième permet d'identifier qui sont les partenaires des clients.
La table CLIENTS contient les attributs suivants :

- `id` : identifiant d'un individu (entier), clé primaire ;
- `nom` (chaîne de caractères) ;
- `prenom` (chaîne de caractères) ;
- `ville` (chaîne de caractères) ;
- `email` (chaîne de caractères).

La table PARTENAIRES contient les attributs suivants :

- `id` : identifiant de suivi (entier), clé primaire ;
- `id_client` : identifiant du client représenté par l'attribut `id` dans la table CLIENTS ;
- `partenaire` : nom du partenaire (chaîne de caractères).

4. Écrire une requête SQL permettant d'extraire les identifiants de tous les clients provenant de la ville de ≪Toulouse≫.
5. Écrire une requête SQL permettant d'extraire les emails de tous les clients ayant ≪SCEI≫ comme partenaire.

Exercice 2 : Représentation de nombres

1. Donner la représentation de 38 en binaire sur 8 bits
2. Écrire une fonction `binaire` prenant en entrée k et n et renvoyant la liste correspondant à l'écriture de k sur n bits.
3. Donner la représentation de -38 en complément à deux sur 8 bits

Exercice 3 : Calcul d'attracteur dans un jeu d'accessibilité

On rappelle l'algorithme de calcul des attracteurs pour résoudre un jeu d'accessibilité :

```
fonction attracteur(G, S1, F1):
  soit strat une stratégie vide
  soit A un ensemble vide de sommets
  soit ds(u) le degré sortant de chaque sommet u
  soit Gt le graphe transposé de G
  fonction parcours(v):  (parcourt les éléments de A)
    si v n'est pas dans A:
      on ajoute v à A
      pour chaque successeur u de v dans Gt:
        ds(u) <- ds(u) - 1
        si u est dans S1:
          on ajoute le coup u -> v à strat
          parcours(u)
        sinon, si ds(u) = 0:
          parcours(u)
  pour chaque v dans F1:
    parcours(v)
  on renvoie A et strat
```

1. Écrire une fonction `transpose` prenant en entrée un graphe représenté par listes d'adjacences, et renvoyant son graphe transposé.
2. Implémenter la fonction `attracteur`. Le graphe sera représenté par listes d'adjacences, les ensembles `S1`, `F1`, `A` sous forme de listes de booléens (ie `S1[u]` vaut `True` ssi u est dans S_1), `strat` par une liste de sommets (`strat[u]` vaut `v` si la stratégie demande de jouer v depuis u, ou `None` si la stratégie n'est pas définie sur u).
3. Déterminer la complexité de cette fonction.
4. Tester cette fonction grâce à la fonction `partie_morpion` donnée dans le fichier `attracteur.py` disponible sur cahier de prepa.

Exercice 4 : Programmation dynamique

On considère une grille de m lignes et n colonnes remplie d'entiers positifs, représentée par une liste de listes `G` où `G[i][j]` est la valeur de la case (i, j). On souhaite trouver le coût minimal d'un chemin allant de la case $(0, 0)$ à la case $(m-1, n-1)$, en ne se déplaçant que vers la droite ou vers le bas. Le coût d'un chemin est la somme des valeurs des cases traversées.

1. On note $d(i, j)$ le coût minimal pour atteindre la case (i, j) depuis $(0, 0)$. Donner la valeur de $d(0, 0)$ et établir la relation de récurrence vérifiée par $d(i, j)$.
2. Écrire une fonction `cout_min` qui prend en entrée la grille `G` et renvoie $d(m-1, n-1)$ par programmation dynamique ascendante.
3. Modifier la fonction pour qu'elle renvoie également le chemin optimal sous forme d'une liste de cases parcourues.