CONCOURS INTERNE DE TECHNICIEN DE POLICE TECHNIQUE ET SCIENTIFIQUE DE LA POLICE NATIONALE

SESSION 2015

INFORMATIQUE – DEVELOPPEMENT LOGICIEL

Épreuve écrite de connaissance
se rapportant à la spécialité choisie

Durée de l'épreuve : 3 heures – Coefficient : 2

Il vous appartient de vous assurer que le sujet en votre possession comporte la totalité des pages (7 pages).

Il vous est demandé de répondre avec clarté à chaque question, sur votre feuille de composition (coin gommé).

Ce sujet comporte quatre exercices valant 5 points chacun.

Modalités particulières :

- Il n'y a pas d'annexe à rendre. Le seul document à rendre est donc la (ou les) feuille(s) de composition.
- Les documents papiers et les matériels électroniques (calculatrices, téléphones, ordinateurs) sont interdits.

Sous peine d'annulation de leur épreuve, les candidats ne devront faire apparaître **aucun signe ou mention** pouvant permettre l'identification des copies et intercalaires.

Exercice 1

Question 1.1 : Qu'est ce qu'une signature électronique ? A quoi peut-elle servir ?

Question 1.2 : Comment peut-on assurer la confidentialité des données lors d'une transmission via un réseau ?

Question 1.3 : Qu'est-ce qu'un certificat ? A quoi sert-il ?

Question 1.4 : Est-il utile d'installer un antivirus sur un ordinateur qui ne dispose pas de connexion Internet ? Pourquoi ?

Question 1.5 : Citez des outils utilisés pour analyser les données échangées sur un réseau local.

Question 1.6 : Quels matériels et/ou logiciels doivent être installés pour filtrer le trafic entrant et sortant d'un réseau d'entreprise ?

Question 1.7 : Qu'est-ce que le *phishing* (ou hameçonnage en français) ?

Question 1.8 : Un ami vous demande de le conseiller sur le choix d'un mot de passe de messagerie électronique. Que lui répondez-vous ?

Question 1.9 : Citez deux protocoles de transfert sécurisé de fichiers.

Question 1.10 : Qu'entend-t'on par "casser une clé de chiffrement" ? Comment peut-on y parvenir ?

Exercice 2

Dans cet exercice, vous pourrez écrire des algorithmes ou des programmes. Si vous écrivez des programmes, indiquez en début d'exercice le langage de programmation que vous utilisez.

Dans cet exercice, on s'intéresse à la réalisation de certains composants d'un jeu d'échec. Le jeu d'échec se joue à deux joueurs, sur un plateau de 8x8 cases. Chaque joueur possède initialement les pièces suivantes : un roi, une reine, deux fous, deux cavaliers, deux tours, et huit pions. Chaque pièce a des règles de déplacement différentes.

Question 2.1 : Écrivez la classe "Piece", qui contient le numéro du joueur à qui appartient cette pièce, ainsi que la position de la pièce. La classe "Piece" contient une méthode "deplacer", qui prend en paramètre une nouvelle position et l'échiquier (de la classe "Echiquier"), et qui renvoie "vrai" si le déplacement vers cette nouvelle position est autorisé, et "faux" sinon. La classe "Piece" contient aussi une méthode "getJoueur" qui retourne le numéro du joueur propriétaire de la pièce.

Question 2.2 : Qu'est-ce qu'une classe abstraite ? Pourquoi la classe "Piece" peut-elle être déclarée abstraite ?

Question 2.3 : Le roi et la tour sont deux exemples de pièces. Quel est le nom du mécanisme de programmation orientée objets qui établit ce lien ?

Question 2.4 : Écrivez la classe "Roi". Le roi peut se déplacer sur les huit cases voisines, à l'exception des cases contenant une pièce appartenant au joueur. On supposera qu'il existe une méthode "getPiece" dans la classe "Echiquier", qui retourne la pièce se trouvant dans une case donnée, si elle existe.

Question 2.5 : Écrivez la classe "Tour". La tour peut se déplacer de manière horizontale ou verticale, sur un nombre quelconque de cases. La tour ne peut pas passer au-dessus d'une autre pièce. La tour doit s'arrêter sur une case vide, ou sur une case contenant une pièce appartenant à l'autre joueur.

Question 2.6 : Est-ce qu'une méthode qui prend en paramètre un objet de la classe "Piece" peut prendre à la place un objet de la classe "Roi" ? Est-ce qu'une méthode qui prend en paramètre un objet de la classe "Roi" peut prendre à la place un objet de la classe "Piece" ? Quel est le nom du mécanisme de programmation orientée objets qui autorise ou empêche cela ?

Question 2.7 : Écrivez la classe "Reine". La reine peut se déplacer comme un fou ou comme une tour. Vous réutiliserez les fonctions de déplacement existantes (sans les reprogrammer), en supposant que la classe "Fou" existe.

Question 2.8 : Dans le cas de la classe "Reine", quel est l'intérêt de ne pas reprogrammer les fonctions de déplacement du fou ou de la tour ?

Exercice 3

Soit le schéma relationnel suivant, qui permet de modéliser les révisions mécaniques de véhicules :

MODELES (IdModele, Marque, Modele, Année)

Table qui liste les modèles de véhicules.

VEHICULES (Immatriculation, DatePreCirculation, Km, #IdModele)

Table qui liste les véhicules avec leur date de 1ère mise en circulation et leur kilométrage

PIECES (RefPiece, LibPiece, PrixPiece)

Table qui liste les pièces pouvant intervenir dans l'entretien d'un véhicule.

ENTRETIEN (#IdModele, #RefPiece, km, temps, CoutMO)

Table qui liste les entretiens à effectuer en fonction des modèles : pièce à changer, kilométrage max (km), durée de vie (temps) de la pièce, coût de la main d'œuvre (coutMO).

CARNET (#Immatriculation, #RefPiece, DateChangement, km)

Table qui liste les interventions réalisés sur les véhicules (km représente le kilométrage du véhicule au moment de l'intervention).

Afin d'exploiter les données stockées, faire les requêtes SQL suivantes.

Question 3.1 : Affichez le carnet d'entretien des véhicules PEUGEOT mis en circulation en janvier 2015, en précisant le prix des pièces changées et le kilométrage lors du changement. Le résultat doit être trié sur ce dernier champ.

Question 3.2 : Affichez, pour chaque modèle, le kilométrage maximum entre deux révisions.

Question 3.3 : Affichez le modèle qui demande le moins d'entretien (celui pour lequel il y a le moins de pièces à changer)

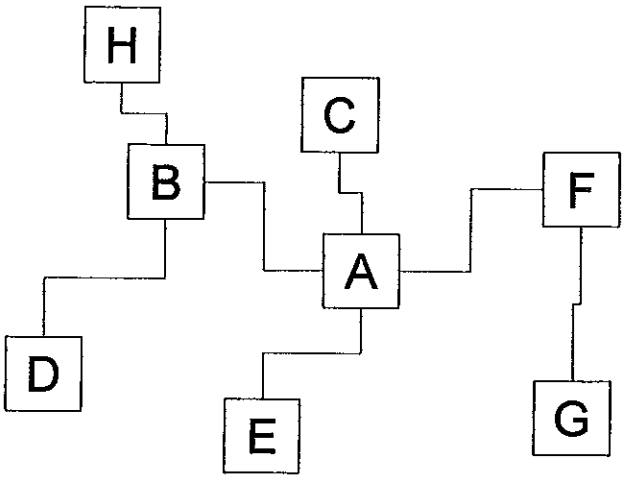
Question 3.4 : Accordez à l'utilisateur 'Tech' le droit de lecture sur la table ENTRETIEN.

Question 3.5 : Accordez à l'utilisateur 'Chef' les droits de lecture et de modification sur la table ENTRETIEN.

Exercice 4

Vous pouvez écrire les réponses aux questions de cet exercice dans le langage de programmation de votre choix ou en pseudo-code. Les questions peuvent être traitées séparément.

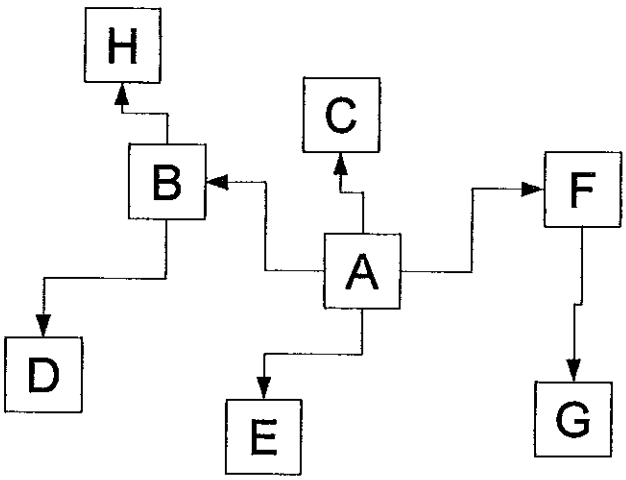
Soit le graphe n°1 suivant. Deux nœuds sont voisins s'ils sont reliés directement. Ex : A est voisin de B mais pas de G.



graphe n°1

Partie 4.1

Pour cette partie, nous allons doter le graphe n°1 d'une orientation, et nommer le graphe ainsi créé graphe n°2.



graphe n°2

Lorsqu'une flèche va d'un nœud à un autre, le nœud d'arrivée de la flèche est appelé un fils. B est un fils de A, et G est un fils de F. Réciproquement, A est le père de B. Un nœud n'a qu'un seul père. Chaque nœud contient une liste de ses fils, ordonnée par ordre alphabétique.

Le but est d'écrire un algorithme passant par tous les nœuds du graphe.

Nous allons utiliser l'algorithme de parcours du graphe suivant, appelé parcours en profondeur :

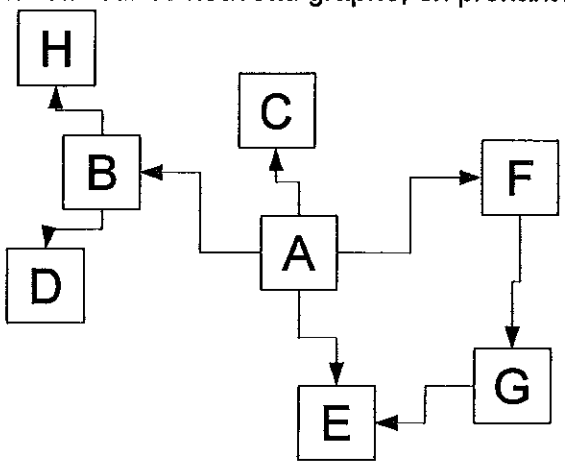
- On part du nœud racine, c'est à dire celui qui n'a aucun père.
- On explore ensuite au maximum chacun des chemins possibles. Par exemple, si un noeud a deux fils, fils1 et fils2, on explore d'abord fils1, puis tous les fils de fils1, avant de passer à fils2 et ainsi de suite.
- Lorsque l'on explore un nœud, on affiche son nom.

Question 4.1.1 : Quel sera l'affichage lors du parcours en profondeur du graphe n°2, en partant de la racine A ?

Question 4.1.2 : Ecrivez la fonction parcoursProfondeurA qui prend en argument un nœud et permet de parcourir le graphe à partir de ce nœud.

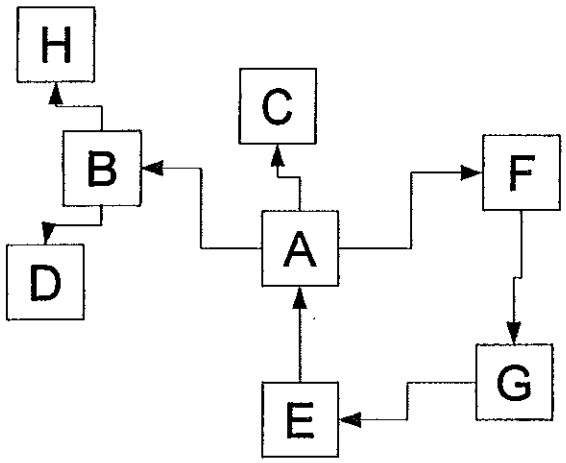
Question 4.1.3 : Si l'on donne comme point de départ le nœud B, qu'affichera la fonction parcoursProfondeurA ?

Question 4.1.4 : Supposons qu'un nœud peut avoir plusieurs pères, et que l'on modifie le graphe en ajoutant un lien entre G et E. Le graphe résultant est le graphe n°3. Quel sera le résultat de la fonction parcoursProfondeurA sur ce nouveau graphe, en prenant A comme point de départ ?



graphe n°3

Question 4.1.5 : Quel sera le résultat de la fonction parcoursProfondeurA sur le graphe dans lequel le lien entre A et E est inversé, toujours en prenant A comme point de départ ? Le graphe résultant est le graphe n°4.



graphe n°4

Partie 4.2

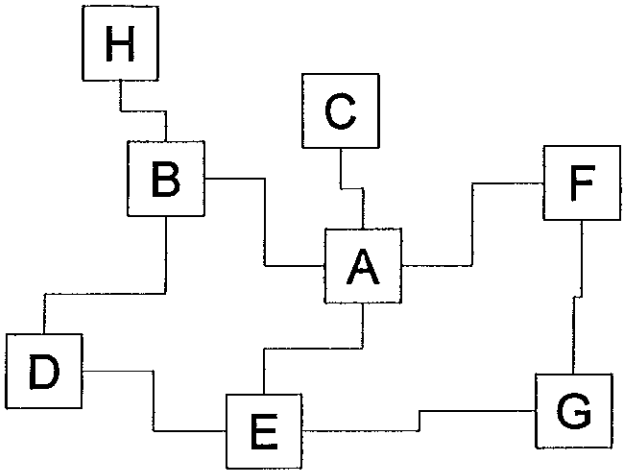
Pour cette partie, nous allons travailler de nouveau sur le graphe n°1. Chaque nœud contient une liste de ses voisins, ordonnée par ordre alphabétique. Par exemple, A contient la liste B, C, E et F, et B contient la liste A, D et H. On cherche à implémenter l'algorithme de parcours en profondeur pour ce graphe non orienté. Le fonctionnement de l'algorithme est alors le suivant :

- On choisit un nœud quelconque comme point de départ.
- On explore ensuite au maximum chacun des chemins possibles. Par exemple, si un nœud a deux voisins, voisin1 et voisin2, on explore d'abord voisin1, puis tous les voisins non encore explorés de voisin1, avant de passer à voisin2 et ainsi de suite.
- Lorsque l'on explore un nœud, on affiche son nom.
- On ne passe jamais deux fois par le même nœud.
- Lorsqu'on doit explorer les voisins d'un nœud, on commence par le premier dans l'ordre alphabétique.

Question 4.2.1 : Quel est le résultat de cet algorithme sur le graphe n°1 en partant du nœud B ?

Question 4.2.2 : Ecrivez la fonction parcoursProfondeurB qui prend en argument un nœud et permet de parcourir le graphe à partir de ce nœud.

Question 4.2.3 : On modifie le graphe en ajoutant un lien entre G et E. Le graphe résultant est le graphe n°5. Quel sera le résultat de la fonction parcoursProfondeurB sur ce nouveau graphe, en prenant B comme point de départ ?



graphe n°5

Partie 4.3

On va cette fois étudier le parcours en largeur du graphe, qui est décrit de la façon suivante :

- On choisit un nœud quelconque comme point de départ.
- Une fois que l'on est arrivé à un nœud, on commence par explorer chacun de ses voisins avant de passer aux voisins de ses voisins. Par exemple, si un nœud a deux voisins, voisin1 et voisin2, on explore d'abord voisin1, puis voisin2 puis tous les voisins non encore explorés de voisin1, avant de passer à ceux de voisin2 et ainsi de suite.
- Lorsque l'on explore un nœud, on affiche son nom.
- On ne passe jamais deux fois par le même nœud.
- Lorsqu'on doit explorer les voisins d'un nœud, on commence par le premier dans l'ordre alphabétique.

Question 4.3.1 : Quel est le résultat de cet algorithme sur le graphe n°1 en partant du nœud B ?

Question 4.3.2 : Ecrivez la fonction parcoursLargeur qui prend en argument un nœud et permet de parcourir le graphe en largeur à partir de ce nœud. Vous avez accès à une file de type FIFO *(First In, First Out)* à travers trois fonctions :

- AjoutFile(Nœud n) qui ajoute un nœud à la fin d'une file.
- PremierFile() qui renvoit le nœud au début de la file et l'enlève de la file. Par exemple, si la file est [A,B,C] elle deviendra [B,C] après un appel de PremierFile.
- FileVide() qui renvoit vrai si la file est vide.

Partie 4.4

L'algorithme de parcours en largeur décrit à la question précédente permet de déterminer la distance d'un nœud à un autre, car il commence par visiter les nœuds proches du premier nœud

passé en argument. La distance d'un nœud à un autre est le nombre de connexions entre ces deux nœuds. Par exemple A et B sont à une distance de 1, mais B et F sont à une distance de 2.

Question 4.4 : Ecrivez la fonction distance(Nœud n1) qui se base sur l'algorithme de parcours en largeur pour calculer la distance entre n1 et tous les nœuds du graphe. Vous pourrez vous aider des fonctions suivantes en plus de celles de la partie 3.

- valoriserDistance(Nœud n, entier d) qui valorise à d la distance entre n1 et le nœud n.
- recupereDistance(Nœud n), qui récupère la distance entre n1 et n. Cette fonction renvoie -1 si vous n'avez pas encore rentré cette distance.