Ce document a été mis en ligne par l'organisme FormaV®.

Toute reproduction, représentation ou diffusion, même partielle, sans autorisation préalable, est strictement interdite.

Pour consulter tous nos sujets d'annales et corrigés du Bac Général Maths + NSI, veuillez visiter :
bac-maths-nsi.fr/annales-sujets-et-corriges

Polynésie - juin 2024 - sujet 2 (corrigé)

Exercice 1 (Algorithmique, graphes, piles et files)
Partie A

1. On obtient le chemin Mp – Ar – Ax – Nc pour un coût de 332.
2. On a les deux chemins suivants : Mp – Ar – Ax – Nc et Mp – Ar – Mr – Nc

Partie B

3. On a l'implémentation suivante :

```
G = {Mr: [Av, Ar, Ax, To, Nc],
     Av: [Mr, Ni, Ax],
     Ni: [Av, Ar, Mp],
     Ar: [Mr, Ni, Mp, Ax],
     Mp: [Ar, Ni],
     Ax: [Av, Ar, Mr, To, Nc, Di],
     To: [Mr, Nc, Ax],
     Nc: [Mr, To, Ax, Di],
     Di: [Nc, Ax]}
```

4. - LIFO signifie Last In First Out, soit « dernier entré, premier sorti »
   - FIFO signifie First In First Out, soit « premier entré, premier sorti »
5. Une file est désignée par l'acronyme FIFO.
6. L'appel `parcours(G, 'Av')` renvoie la liste `['Av', 'Mr', 'Ni', 'Ax', 'Ar', 'To', 'Nc', 'Mp', 'Di']`, soit l'ensemble des sommets du graphe `G`.
7. La fonction `parcours` réalise un parcours en largeur. Il s'agit donc de la proposition A.
8. La fonction suivante convient :

```
def distance(graphe, sommet):
    f = creerFile()
    enfiler(f, sommet)
    d = 0
    visite = {sommet: d}
    while not estVide(f):
        s = defiler(f)
        d += 1
        for v in graphe[s]:
            if not (v in visite):
                visite[v] = d
                enfiler(f, v)
    return visite
```

9. On obtient le résultat suivant :

```
{'Av': 0,
 'Mr': 1, 'Ni': 1, 'Ax': 1,
 'Ar': 2, 'To': 2, 'Nc': 2, 'Mp': 2, 'Di': 2}
```

10. Le code suivant convient :

```
def parcours2(G, s):
    p = creerPile()
    empiler(p, s)
    visite = []
    while not estVide(p):
        x = depiler(p)
        if x not in visite:
            visite.append(x)
            for v in G[x]:
                empiler(p, v)
    return visite
```

11. L'appel `parcours2(G, 'Av')` renvoie la liste `['Av', 'Ax', 'Di', 'Nc', 'To', 'Mr', 'Ar', 'Mp', 'Ni']` obtenue à l'aide d'un parcours en profondeur.

Exercice 2 (Arbres binaires de recherche, POO et récursivité)
Partie A

1. - Le nœud initial est appelé *racine*
   - Un nœud qui n'a pas de fils est appelé *feuille*
   - Un arbre binaire est un arbre dans lequel chaque nœud a *au maximum* deux fils
   - Un arbre binaire de recherche est un arbre binaire dans lequel tout nœud est associé à une clé qui est :
     — supérieure à chaque clé de tous les nœuds de son*sous-arbre gauche*
     — inférieure à chaque clé de tous les nœuds de son *sous-arbre droit*
2. On a les clés suivantes : $1-0-2-3-4-5-6$
3. On a les clés suivantes : $0-1-2-6-5-4-3$
4. On a les clés suivantes : $0-1-2-3-4-5-6$
5. Le code suivant convient :

```
arbre_1 = ABR()
arbre_2 = ABR()
arbre_3 = ABR()
for cle_a_inserer in [1, 0, 2, 3, 4, 5, 6]:
    arbre_1.inserer(cle_a_inserer)
for cle_a_inserer in [3, 2, 1, 0, 4, 5, 6]:
    arbre_2.inserer(cle_a_inserer)
for cle_a_inserer in [3, 1, 0, 2, 5, 4, 6]:
    arbre_3.inserer(cle_a_inserer)
```

6. - `arbre_1` a pour hauteur 5
   - `arbre_2` a pour hauteur 3
   - `arbre_3` a pour hauteur 2
7. Le code suivant convient :

```
def est_present(self, cle_a_rechercher):
    if self.est_vide() :
        return False
    elif cle_a_rechercher == self.cle() :
        return True
    elif cle_a_rechercher < self.cle() :
        return self.sag().est_present(cle_a_rechercher)
    else :
        return self.sad().est_present(cle_a_rechercher)
```

8. La complexité de la recherche d'un élément dans un ABR est de l'ordre de la hauteur. L'appel qui nécessitera donc le moins d'appels récursifs est `arbre_3.est_presente(7)`.

Partie B

9. Un arbre est partiellement équilibré si la hauteur de son sous-arbre gauche et la hauteur de son sous-arbre droit diffèrent d'au plus 1.
10. - L'arbre 1 n'est pas partiellement équilibré car son sous-arbre gauche a pour hauteur 0 et son sous-arbre droit a pour hauteur 4.
    - L'arbre 2 est partiellement équilibré car son sous-arbre gauche et son sous-arbre droit ont tous les deux pour hauteur 2.
    - L'arbre 3 est partiellement équilibré car son sous-arbre gauche et son sous-arbre droit ont tous les deux pour hauteur 1.
11. L'arbre 3 est équilibré car il est complet). L'arbre 2 n'est pas équilibré car, pour son sous-arbre gauche, le sous-arbre gauche a pour hauteur 1 et le sous-arbre droit a pour hauteur -1.
12. La méthode suivante convient :

```
def est_equilibre(self);
    return self.est_partiellement_equilibre()\
           and self.sag().est_partiellement_equilibre()\
           and self.sad().est_partiellement_equilibre()
```

Exercice 3 (Protocoles réseaux, bases de données, SQL, algorithmique et programmation Python)

Partie A : le réseau informatique d'un hôpital

1. On a les deux chemins suivants :
   - service neurologie - R4 - R8 - R1 - R2 - imagerie
   - service neurologie - R4 - R8 - R7 - R2 - imagerie
2. On a le tableau suivant :

Noeud R2	
Destination	Coût
R1	1
R3	3
R4	3
R5	2
R6	3
R7	1
R8	2
R9	3

3. Puisque 10 Gbit/s = 10^{10} bit/s, le coût d'une communication FTTH et 0,01
4. On a le chemin suivant : service neurologie - R8 - R9 - R1 - R2 - imagerie pour un coût de 0,22

Partie B : le dossier médical d'un patient

5. La requête renvoie les noms et prénoms des patients dont les numéros de sécurité sociale commencent par `1` (donc les patients hommes)
6. La requête suivante convient :

```
SELECT num_SS FROM hospitalisation WHERE service = 'orthopédique'
```

7. La requête suivante convient :

```
SELECT type, date FROM examen
JOIN patient ON examen.num_SS = patient.num_SS
WHERE nom = 'Baujean' AND prenom = 'Emma'
```

8. La requête suivante convient :

```
SELECT patient.nom, patient.prenom FROM patient
JOIN consultation ON patient.num_SS = consultation.num_SS
JOIN medecin ON consultation.id_medecin = medecin.id_medecin
WHERE medecin.nom = 'ARNOS' AND medecin.prenom = 'Pierre'
```

Partie C : la sécurité des mots de passe d'un médecin

9. Le code suivant convient :

```
def mdp_fort(mdp):
    if len(mdp) < 12:
        return False
    majuscules = 0
    chiffres = 0
    symboles = 0
    for caractere in mdp :
        if caractere.isupper():
            majuscules+=1
        if caractere.isdigit():
            chiffres+=1
        if caractere in liste_symboles:
            symboles+=1
    if majuscules < 2 or chiffres < 2 or symboles < 2:
        return False
    return True
```

10. Le code suivant convient :

```
def creation_mdp(n, nbr_m, nbr_c, nbr_s):
    mdp = ''
    caracteres = 'abcdefghijklmnopqrstuvwxyz' + \
                 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789' + \
                 '#@!?%<>=$+-*/&'
    majuscules = 0
    chiffres = 0
    symboles = 0
    while len(mdp) < n or majuscules < nbr_m or chiffres < nbr_c or symboles < nbr_s:
        # la variable 'c' contient un caractère
        # choisi aléatoirement dans la variable 'caracteres'
        c = choice(caracteres)
        if c in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':
            majuscules += 1
        if c in '0123456789':
            chiffres += 1
        if c in '#@!?%<>=$+-*/&':
            symboles += 1
        mdp = mdp + c
    return mdp
```

11. Le code suivant convient :

```
def recherche_mot(mdp):
    mot = transforme(mdp)
    trouve = []
    i = 0
    while i < len(mot):
        if mot[i].isdigit(): # si le caractère est un chiffre
            i = i + 1
        elif mot[i] in liste_symboles:
            i = i + 1
        else:
            # si le caractère est une lettre, on prend les
            # lettres qui la suivent jusqu'au moment où
            # on trouve un chiffre ou un symbole
            chaine = ''
            while mot[i].isalpha():
                chaine = chaine + mot[i]
                i = i + 1
            trouve.append(chaine)
    return trouve
```

12. La fonction suivante convient :

```
def mdp_extra_fort(mdp):
    liste_mots_mdp = recherche_mot(mdp)
    nb_mots = 0
    for mot in liste_mots_mdp:
        if len(mot) >= 4 and mot in dicoFR:
            nb_mots += 1
    return nb_mots == 0
```

Copyright © 2027 FormaV®. Tous droits réservés.

FormaV® est une marque déposée auprès de l'INPI sous le numéro 5227897.

Ce document a été élaboré par FormaV® avec le plus grand soin afin d'accompagner chaque apprenant vers la réussite de ses examens. Son contenu (textes, graphiques, méthodologies, tableaux, exercices, concepts, mises en forme) constitue une œuvre protégée par le droit d'auteur.

Toute copie, partage, reproduction, diffusion ou mise à disposition, même partielle, gratuite ou payante, est strictement interdite sans accord préalable et écrit de FormaV®, conformément aux articles L.111-1 et suivants du Code de la propriété intellectuelle. Dans une logique anti-plagiat, FormaV® se réserve le droit de vérifier toute utilisation illicite, y compris sur les plateformes en ligne ou sites tiers.

En utilisant ce document, vous vous engagez à respecter ces règles et à préserver l'intégrité du travail fourni. La consultation de ce document est strictement personnelle.

Merci de respecter le travail accompli afin de permettre la création continue de ressources pédagogiques fiables et accessibles.