

BACCALAURÉAT GÉNÉRAL

CORRECTION DE L'ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2024

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Durée de l'épreuve : 3 heures 30

Le sujet est composé de trois exercices indépendants.

Exercice 1		6 points	
Questions	Contenu et notions	Capacités exigibles / Niveau	Éléments de réponses et commentaires
1	Graphes : plus court chemin	N1	Mp - Ar - Ax - Nc, chemin de 332 km
2	Graphe : plus courts chemins (nombre minimum de sommets)	N1	Mp - Ar - Mr - Nc et Mp - Ar - Ax - Nc
3	Modéliser : implémentation d'un graphe à l'aide d'un dictionnaire	N1	<pre> G = {'Av' : ['Mr', 'Ni', 'Ax'], 'Mr' : ['Av', 'Ar', 'Ax', 'To', 'Nc'], 'Ni' : ['Av', 'Ar', 'Mp'], 'Ar' : ['Mr', 'Ni', 'Mp', 'Ax'], 'Mp' : ['Ni', 'Ar'], 'Ax' : ['Av', 'Ar', 'To', 'Mr', 'Nc', 'Di'], 'To' : ['Mr', 'Nc', 'Ax'], 'Nc' : ['Mr', 'To', 'Ax', 'Di'], 'Di' : ['Ax', 'Nc']} </pre>
4	Structures de données : files et piles	N1	LIFO : Last In First Out FIFO : First In First Out
5	Structures de données : files	N1	Une file est désignée par l'acronyme FIFO
6	Graphe : parcours en largeur	N2	Le résultat dépend du dictionnaire écrit par le candidat à la question 3. Tout résultat cohérent est donc accepté. Avec le dictionnaire de ce corrigé, le résultat est : ['Av', 'Mr', 'Ni', 'Ax', 'Ar', 'To', 'Nc', 'Mp', 'Di']
7	Parcours d'un graphe	N1	Proposition A : parcours en largeur

Exercice 1		6 points	
8	Modifier une fonction existante	N3	<pre> def distance(graphe, sommet): f = creerFile() enfiler(f, sommet) visite = [sommet] dist = {} dist[sommet] = 0 while not estVide(f): s = defiler(f) for v in graphe[s]: if not (v in visite): dist[v] = dist[s] + 1 visite.append(v) enfiler(f, v) return dist </pre>
9	Structures de données : dictionnaires (distance entre 2 sommets)	N2	<pre> {'Av' : 0, 'Mr' : 1, 'Ni' : 1, 'Ax' : 1, 'Ar' : 2, 'To' : 2, 'Nc' : 2, 'Mp' : 2, 'Di' : 2} </pre>
10	Graphe : parcours en profondeur - Traduction d'un algorithme	N2	<pre> def parcours2(G, s): p = creerPile() empiler(p, s) visite = [s] while not estVide(p): x = depile(p) if x not in visite: visite.append(x) for v in G[x]: empiler(p, v) return visite </pre>
11	Graphe : parcours en profondeur	N2	Le résultat dépend de l'ordre d'énumération des voisins. Un résultat possible est : ['Av', 'Ni', 'Mp', 'Ar', 'Ax', 'To', 'Mr', 'Nc', 'Di']. Toute liste de sommets donnant un parcours en profondeur du graphe est acceptée.

Exercice 2		6 points	
<i>Questions</i>	<i>Contenu et notions</i>	<i>Capacités exigibles / Niveau</i>	<i>Éléments de réponses et commentaires</i>
1	Arbres : structures hiérarchiques.	N1	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 supérieure à chaque clé de tous les nœuds de son sous-arbre gauche et inférieure à chaque clé de tous les nœuds de son sous-arbre droit .
2	Algorithmes sur les arbres binaires et sur les arbres binaires de recherche	N1	1, 0, 2, 3, 4, 5, 6
3	Algorithmes sur les arbres binaires et sur les arbres binaires de recherche	N1	0, 1, 2, 6, 5, 4, 3
4	Algorithmes sur les arbres binaires et sur les arbres binaires de recherche	N1	0, 1, 2, 3, 4, 5, 6
5	Vocabulaire de la programmation objet : classes, attributs, méthodes, objets.	N2	<pre> arbre_no1 = ABR() arbre_no2 = ABR() arbre_no3 = ABR() for cle_a_inserer in [1, 0, 2, 3, 4, 5, 6] :# ou tout ordre correct arbre_no1.inserer(cle_a_inserer) for cle_a_inserer in [3, 2, 1, 0, 4, 5, 6]: # ou tout ordre correct arbre_no2.inserer(cle_a_inserer) for cle_a_inserer in [3, 1, 5, 0, 2, 4, 6]: # ou tout ordre correct arbre_no3.inserer(cle_a_inserer) </pre>

Exercice 2		6 points	
6	Algorithmes sur les arbres binaires et sur les arbres binaires de recherche	N1	arbre_no1 a une hauteur de 5, arbre_no2 a une hauteur de 4 et arbre_no3 a une hauteur de 2.
7	Algorithmes sur les arbres binaires et sur les arbres binaires de recherche et récursivité	N2	<pre> 1. def est_present(self, cle_a_rechercher): 2. if self.est_vide() : 3. return False 4. elif cle_a_rechercher == self.cle() : 5. return True 6. elif cle_a_rechercher < self.cle() : 7. return self.sag().est_presente(cle_a_rechercher) 8. else : 9. return self.sad().est_presente(cle_a_rechercher) </pre>
8	Algorithmes sur les arbres binaires et sur les arbres binaires de recherche et récursivité	N2	Pour chaque arbre, 7 n'est pas présent et est supérieur à chaque clé rencontrée. On descend chaque fois dans le sous-arbre droit jusqu'à trouver un sous-arbre vide : • Dans l'arbre no 1, il y a 6 appels. • Dans l'arbre no 2 il y aura 4 appels. • Pour l'arbre no 3, il y aura 3 appels : c'est cette instruction qui nécessite le moins d'appels récursifs.
9	Vocabulaire de la programmation objet : classes, attributs, méthodes, objets et algorithmes sur les arbres	N2	Une instance d'ABR est partiellement équilibré s'il est vide ou si ses deux sous-arbres ont une différence de hauteur inférieure ou égale à 1
10	récursivité	N2	les arbres 2 et 3 ont des sous arbres de même taille : ils sont partiellement équilibrés.
11	récursivité et vocabulaire de la programmation objet	N2	Seul les sous arbres de l'arbre no 3 sont partiellement équilibrés ainsi que leurs sous-arbres. L'arbre no3 est le seul a être équilibré.
12	récursivité et vocabulaire de la programmation objet	N3	<pre> def est_equilibre(self): if self.est_vide() : return True else : return (self.est_partiellement_equilibre() and self.sag().est_equilibre() and self.sad().est_equilibre()) </pre>

Exercice 2 6 points

Exercice 3			
8 points ou 6 points en enlevant la partie A			
Questions	Contenu et notions	Capacités exigibles / Niveau	Éléments de réponses et commentaires
1	utiliser le protocole RIP	N1	R4 - R8 - R1 - R2 R4 - R8 - R7 - R2
2	compléter une table de coût	N1	Voir le tableau 1.
3	calculer coût d'une liaison	N1	$d = 10 * 10^9 = 10^{10}$ bit/s $c = 10^8 / 10^{10} = 10^{-2} = 0,01$
4	utiliser protocole OSPF	N1	R4 - R8 - R9 - R1 - R2
5	comprendre une requête SQL avec LIKE	N1	On affiche le nom et le prénom des patients dont le numéro de Sécurité Sociale commence par 1 (c'est-à-dire les hommes) : Tardus Kylian Montpart Vincent
6	Ecrire une requête SQL avec LIKE	N2	<pre>SELECT num_SS FROM hospitalisation WHERE service='orthopédique' AND date LIKE '%2023';</pre>
7	jointure	N2	<pre>SELECT type, date FROM examen JOIN patient ON examen.num_SS=patient.num_SS WHERE nom='Baujean' AND prenom='Emma';</pre>
8	double jointure	N1	<pre>SELECT patient.nom,patient.prenom FROM medecin JOIN consultation ON medecin.id_medecin=consultation.id_medecin JOIN patient ON patient.mun_SS=consultation.mun_SS WHERE medecin.nom='ARNOS' AND medecin.prenom='Pierre';</pre>
9	tester le type d'un caractère, manipulation de compteur	N2	<pre>2 if len(mdp)<12 : 10 if caractere.isdigit(): 12 if caractere in liste_symboles: 14 if majuscules<2 or chiffres<2 or symboles<2 :</pre>

Exercice 3			8 points ou 6 points en enlevant la partie A
10	tester le type d'un caractère, manipulation de compteur, trouver une condition d'arrêt d'une boucle while	N2	<pre> 9 while len(mdp)<n or majuscules<nbr_m or chiffres<nbr_c or symboles<nbr_s : 13 if c.isupper(): 14 majuscules += 1 15 if c.isdigit(): 16 chiffres += 1 17 if c in liste_symboles : 18 symboles += 1 19 mdp = mdp + c </pre>
11	parcourir une liste avec condition	N2	<pre> 5 while i<len(mot) : 6 if mot[i].isdigit() : <i># si le caractère est un chiffre</i> 15 while mot[i].isalpha() : 16 chaine=chaine+mot[i] 18 trouve.append(chaine) </pre>
12	utilisation des fonctions et variables précédentes	N2	<pre> def mdp_extra_fort(mdp): L_mots=recherche_mot(mdp) for mot in L_mots : if mot in dicoFR and len(mot)>3 : return False return True </pre>

Tableau 1 :

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