Exercice 1 (6 points)

Correction

1	0.25	1	Structures de données, interface et implémentation - POO - Graphes	`balise12 = Balise(12, ['vert', 'noir'])`
2	0.25	1	Structures de données, interface et implémentation - POO - Graphes	`balise9.voisines = [balise8, balise11, balise12]`
3	0.5	1	Structures de données, interface	`[2, 5, 6]`

			et implémentation - POO - Graphes - Tableau donné en compréhension	
4	0.5	1	Structures de données, interface et implémentation - POO - Graphes - Tableau donné en compréhension	`['noir', 'vert']`

5	1	2	Algorithmes sur les graphes,	`4 while balise.num_balise != balise_fin.num_balise:`
				`5 for b in balise.voisines:`
				`6 if (couleur in b.couleurs_balise) and (b not in chemin):`

			paradigmes de programmation	7 balise = b 8 chemin.append(balise) Attribuer 0.25 par ligne correctement complétée.
6	0.5	2	Algorithmes sur les graphes	1 – 2 – 4 – 5 – 10 – 7 – 6 – 3 – 11 – 9 – 8 – 12
7	0.25	1	p-uplets, tableau indexé	`balise4.voisines = [(balise2, 13), (balise5, 21), (balise6, 15)]`
8	0.5	2	Constructions élémentaires, paradigmes de programmation	5 (numéro de la balise la plus proche de la balise numéro 10)
9	0.5	2	Algorithmes sur les graphes - Algorithmes gloutons	1 – 2 – 4 – 6 – 11 – 9 – 12

10	1	3	Algorithmes sur les graphes - Algorithmes gloutons	`while balise_fin not in chemin:`
				`    prochaine = mystere(balise)`
				`    if prochaine != None:`
				`        prochaine.visitee = True`
				`        chemin.append(prochaine)`
				`        balise = prochaine`
				`    else:`
				`        return None`
				`return [b.num_balise for b in chemin]`
				
				Attribuer 0.25 point pour l'appel à la fonction `mystere` et 0.25 point si la fonction renvoie bien la valeur `None` dans le cas où l'itinéraire ne peut pas être déterminé.

11	0.25	1	Algorithmes gloutons	**Proposition A** : algorithme glouton
12	0.5	1	Algorithmes gloutons	Inconvénient (0.25) : un tel algorithme ne fournit pas toujours la solution optimale, voire peut ne pas fournir de solution. Avantage (0.25) : l'exécution de tels algorithmes se fait en des temps raisonnables car ils construisent une solution assez simple, sans effectuer trop de calculs.

Exercice 2 (6 points)

Correction

1	0.25	1	langages_programmation	6

2	0.5	1	langages_programmation	`1 def plateau_init(n, m, cartes):`
				`2  shuffle(cartes)`
				`3  plateau = []`
				`4  for i in range(n):`
				`5  plateau.append([cartes[i+j*n] for j in range(m)])`
				`6  return plateau`

3	1	2	langages_programmation	`1 def cartes_voisines(n, m, i, j):`
				`2  voisines = []`
				`3  for i2 in range(i-1, i+2):`
				`4  for j2 in range(j-1, j+2):`
				`5  if (i2, j2) != (i, j) and \`
				`6  i2 in range(n) and \`
				`7  j2 in range(m):`
				`8  voisines.append( (i2, j2) )`
				`9  return voisines`

4	0.75	2	langages_programmation	e1 = 30 e2 n'est pas une chaine (cartes non voisines) e3 n'est pas une chaine (on sort du tableau)

5	1.25	2	langages_programmation	`def chaine_evalue(plateau, chemin):`
				`    s = 0`
				`    for i, j in chemin:`
				`        s += plateau[i][j]`
				`    return s`

6	0.5	2	langages_programmation	Seul le parcours en largeur permet d'obtenir la chaine la plus courte possible.

7	1	2	langages_programmation	`def explore(plateau, cible):`
				`    n, m = len(plateau), len(plateau[0])`
				`    a_visiter = file_init()`
				`    for i in range(n):`
				`        for j in range(m):`
				`            file_ajoute(a_visiter, [(i,j)])`
				
				`    while not file_est_vide(a_visiter):`
				`        chemin = file_retire(a_visiter)`
				`        if chaine_evalue(plateau, chemin) == cible:`
				`            return chemin`
				`        dernier = chemin[-1]`
				`        i0, j0 = dernier`
				`        for i, j in cartes_voisines(n, m, i0, j0):`
				`            if (i, j) not in chemin:`
				`                file_ajoute(a_visiter,`
				`                    chemin + [ (i,j) ])`
				
				`    # Pas de solutions`
				`    return None`

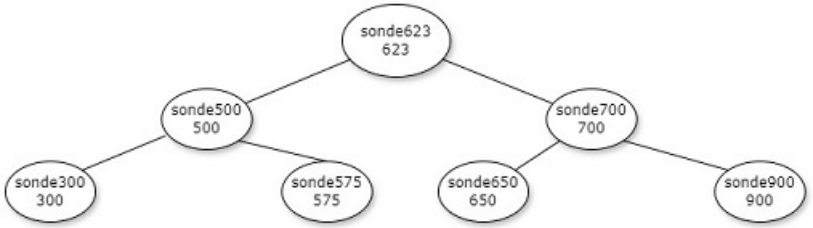
8	0.75	2	langages_programmation	on peut définir une liste `chemins`, à la ligne 11, on remplace le return par un `chemins.append(chemin)`

Exercice 3 (8 points)

Correction

1	0.25	1	clé et valeur d'un dictionnaire	`enregistrement['latitude']`
2	0.5	2	Etude d'un algo	La fonction renvoie `('2024-06-27', '23:36:01')`
3	0.5	1	Fonction len et structure d'une liste et d'un dictionnaire	• `len(frames)` renvoie 4 • `len(frames[1])` renvoie 5
4	1	3	Écrire une fonction	`def detecter_anomalie(enregistrement):` `  return enregistrement['altitude'] > 35000 or enregistrement['altitude']<0`

5	1	3	Écrire une fonction	`def liste_num_serie(frames) :`
				`  liste=[]`
				`  for elt in frames:`
				`  if elt['num_serie'] not in liste:`

				liste.append(elt['num_serie']) return liste
6	1	3	Écrire une fonction	1 def distance_totale(dep): 2 total = 0 3 for i in range(len(dep)-1): 4 total += distance_haversine(dep[i],dep[i+1]) 5 return total
7	0.25	1	Création d'une instance de classe (d'un objet)	sonde623=Sonde(623, 38.38825, 27.09004, '2024-06-27', None, None)
8	0.25	1	insertion d'un noeud dans un ABR	
9	0.5	2	Parcours infixe d'un arbre	Avec un parcours infixe de l'arbre, on obtient la liste des numéros de série triée dans l'ordre croissant. 300->500->575->623->650->700->900

10	0.5	2	recherche dans un arbre	`def rechercher(self, numero):`
				`  if self.est_vide():`
				`  return None`
				`  if numero == self.num_serie:`
				`  return self`
				`  elif numero < self.num_serie:`
				`  return self.gauche.rechercher(numero)`
				`  else:`
				`  return self.droit.rechercher(numero)`

11	0.25	1	définition fonction récursive	La méthode `rechercher` est dite récursive car elle s'appelle elle-même.
12	0.25	1	Définition d'une clé primaire	`id_abonne` est une clé primaire car c'est une donnée unique par abonné. On peut admettre que l'email peut être aussi une clé primaire
13	0.25	1	Définition d'une clé étrangère	`num_serie` est une clé étrangère car elle établit la liaison avec l'attribut `num_serie` de la relation `sondes`. `id_abonne` est une clé étrangère car elle établit la liaison avec l'attribut `id_abonne` de la relation `abonnes`.
14	0.25	1	Analyse d'une requête SQL	'Girard' 'Antoine' 'Détoile' 'Diane'

15	0.5	1	Ecriture d'une requête SQL	`INSERT INTO infosRecuperation VALUES (14,480,24,'2024/07/10',47.230,12.244)`
16	0.75	1	Ecriture d'une requête SQL	`SELECT sondes.num_serie,abonnes.nom,infosRecuperation.date_recup FROM infosRecuperation JOIN Sondes ON infosRecuperation.num_serie=Sondes.num_serie JOIN Abonnes ON infosRecuperation.id_abonne=Abonnes.id_abonne`