Théodore JENOUDET TG1

Devoir maison type Bac

Exercice 1 :

1) Le résultat de la requête « SELECT salle, marque_ordi FROM Ordinateur » sera : '012,HP', '114,Lenovo', '223,Dell', '223, Dell', '223,Dell'.
   La requête « SELECT nom_ordi, salle FROM Ordinateur WHERE video=True » renverra : 'Gen-24,012', 'Tech-62,114', 'Gen-132,223'.

2) La requête « SELECT * FROM Ordinateur WHERE anne >= 2017 ORDER BY anne » renverra tous les attributs des ordinateurs correspondant aux années supérieurs ou égales à 2017.

3) a. L'attribut « salle » ne peut pas être une clé primaire puisque la valeur « 223 » apparait plusieurs fois dans la relation Ordinateur or une clé primaire est unique et il n'est pas possible d'avoir plusieurs fois la même valeur.

   b. Les clé primaire sont soulignée et les clés étrangères sont notées en gras.
   Imprimante (<u>nom imprimante</u> : String, marque_imp : String, modele_imp : String, **salle** : String, **<u>nom_ordi</u>** : String).

4) a. La requête « INSERT INTO Videoprojecteur VALUES (315, 'NEC', 'ME402X', False) » va insérer dans la relation Videoprojecteur le vidéo projecteur installé dans la salle 315 de marque NEC de modèle ME402X et qui n'est pas relié à un TNI.
   b. « SELECT Ordinateur.salle, Ordinateur.nom_ordi, Videoprojecteur.marque_video FROM Ordinateur JOIN Videoprojecteur ON Ordinateur.salle = Videoprojecteur.salle WHERE tni=True » Cette requête SQL va récupérer les attributs salle, nom_ordi et marque_video de tous les ordinateurs connectés à un vidéo projecteur connecté à un TNI.

Exercice 2 :

1) a. Le chemin allant de la case (0,0) à la case (2,3) du tableau comprend obligatoirement 3 déplacements vers la droite et 2 déplacements vers le bas.
   b. La longueur de tous les chemins allant de (0,0) à (2,3) est nécessairement égale à 6 puisqu'il faut forcément 3 déplacements vers la droite et 2 déplacements vers le bas, en comptant la première case (la case dans laquelle ont se trouve, la case (0,0) cela nous donne 3+2+1 ce qui nous fait bien 6
2) Le chemin qui permet d'obtenir la somme maximale est (0,0), (1,0), (2,0), (2,1), (2,2), (2,3) et la somme maximale est de 16.

3) a. T'=

4	5	6	**9**
6	**6**	8	10
9	10	**14**	16

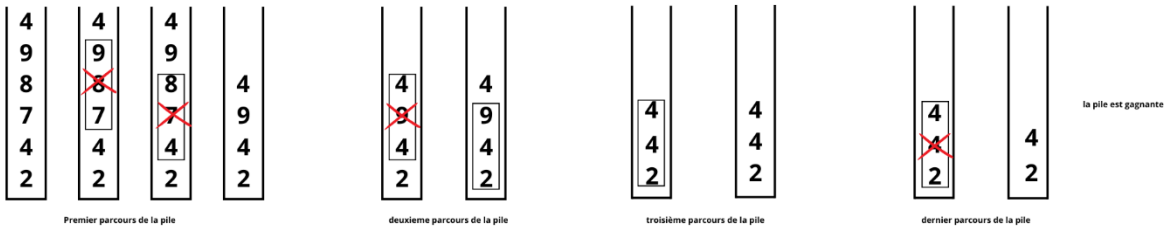
b. Pour tout j≠0 T'[0][j] = T[0][j] + T'[0][j-1]. Cette équation est vérifiée puisque T'[0][j] est égale à la somme maximale pour arriver à cette case, de ce fait on ajoute la case T[0][j] à la case précédente soit [0][j-1] qui contient pour T' la somme maximale pour arriver a cette case ce qui nous donneras donc la somme maximale pour arriver à la case [0][j-1] + l'entier contenu dans la case [0][j].

4) Pour tout i≠0 et j≠0 on a T'[i][j] = T[i][j] + max(T'[i-1][j], T'[i][j-1]) Cette équation est vérifiée puisque on ajoute la case [i][j] avec la case qui contient la plus grande somme maximale (dans le tableau T') entre T'[i-1][j] et T'[i][j-1] ce qui revient à faire le chemin pour arriver à la case [i][j] et à ajouter la somme maximale des entiers pour arriver à la case [i-1][j], donc à droite de [i][j] ou la somme maximale des entiers pour arriver à la case [i][j-1], donc en bas de [i][j].

5)
```
def somme_max(t,i,j):
      if i or j == 0 :
         return False
   return t[i][j] + max(somme_max(t,i-1,j), somme_max(t,i,j-1))
```

Exercice 3 :

1. a.



b. C'est la pile A qui est gagnante puisqu'il va rester 5 et 1 dans la pile.

2)
```
def reduire_triplet_au_sommet(p):
    a=depiler(p)
    b=depiler(p)
    c=sommet(p)
    if a%2 != c%2 :
        empiler(p,b)
        empiler(p,a)
    empiler(p,a)
```

3) a. La taille minimale que doit avoir une pile pour être réductible est de 3 éléments

b.
```
def parcourir_pile_en_reduisant(p):
    q = creer_pile_vide()
    while taille(p)>= 2 :
        reduire_triplet_au_sommet(p)
        e = depiler(p)
        empiler(q,e)
    while not est_vide(q):
        reduire_triplet_au_sommet(q)
        empiler(p,e)
        return p
```

4)
```
def jouer(p):
    q = parcourir_pile_en_reduisant(p)
    if p == q :
        return p
    else :
        return jouer(q)
```

```
def rendu(somme_a_rendre):
    n1=0
    n2=0
    n3=0
    li=[]
    monnaie = [5,2,1]
    indice = 0
    while somme_a_rendre > 0 :
        monnaie_test=monnaie[indice]
        if monnaie_test > somme_a_rendre :
            indice += 1
        else :
            if monnaie_test == 5 :
                n1 += 1
            if monnaie_test == 2 :
                n2 += 1
            if monnaie_test == 1 :
                n3 += 1


            somme_a_rendre -= monnaie_test
    li=[n1,n2,n3]
    return li


class AdresseIP:
    def __init__(self, adresse):
        self.adresse = adresse
    def liste_octet(self):
        return [int(i) for i in self.adresse.split(".")]
    def est_reservee(self):
        return self.adresse == "192.168.0.0" or self.adresse ==
"192.168.0.255"
    def adresse_suivante(self):
        if AdresseIP.liste_octet(self)[3] < 254:
            octet_nouveau = AdresseIP.liste_octet(self)[3] + 1
            return (AdresseIP('192.168.0.' + str(octet_nouveau)))
        else:
            return False

adresse1=AdresseIP("192.168.0.1")
adresse2=AdresseIP("192.168.0.2")
adresse3=AdresseIP("192.168.0.0")
```