BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

Décembre 2024 – Bac Blanc

N.S.I.
Numérique et Sciences Informatiques

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé

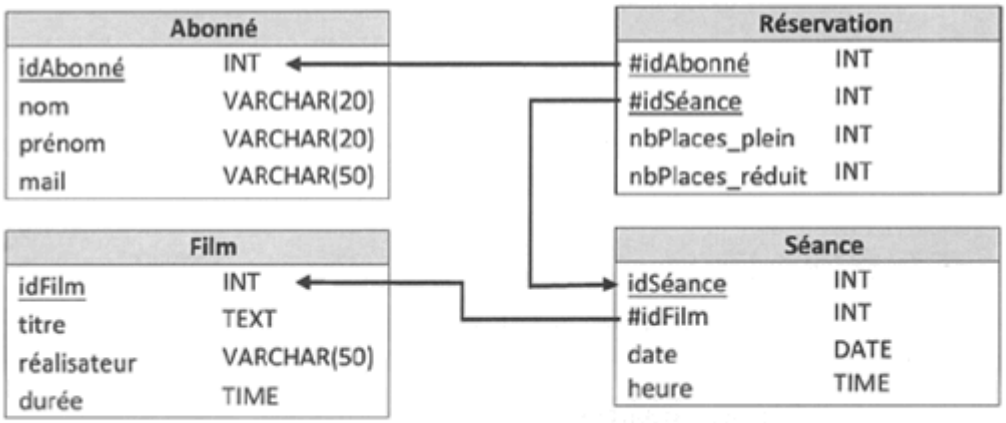
Dès que ce sujet vous est remis, assurez-vous qu'il est complet.
Ce sujet comporte 11 pages numérotées de 1 à 11

Exercice 1 : *(7 points)*

Cet exercice porte sur les bases de données relationnelles et le langage SQL.

Cet exercice utilise les mots du langage SQL suivants : `SELECT`, `FROM`, `WHERE`, `JOIN ON`, `UPDATE`, `SET`, `DELETE`, `COUNT`, `AND`, `OR`.

Une salle de cinéma propose un site Web à ses abonnés afin d'effectuer des réservations de séances en ligne. Deux tarifs sont proposés : plein et réduit (-16 ans, sénior +65 ans, étudiants, ...). Le site est associé à une base de données dont le modèle relationnel contient les quatre relations décrites ci-dessous :



Un attribut souligné correspond à une clé primaire et un attribut précédé du symbole # à une clé étrangère.

Voici un extrait de quelques enregistrements des relations `Film`, `Séance` et `Abonné` :

idFilm	titre	réalisateur	durée
1	Le sens de la famille	Jean-Patrick Benes	90
2	Les croods 2	Joel Crawford	95
8	Black widow	Cate Shortland	134
...			

Extrait de la relation `Film`, les durées sont en minutes.

idSéance	idFilm	date	heure
35	1	2021-10-11	21:00
737	8	2021-10-11	21:00
738	8	2021-10-13	16:15
...			

Extrait de la relation `Séance`.

idAbonné	nom	prénom	mail
1	Henry	Jean	jean.henry@envoi.fr
2	Jacquin	Morgane	jacquin.morgane@mail.com
13	Dupont	Charles	charles.dupont@envoi.fr
...			

Extrait de la relation `Abonné`.

1. (a) Définir le rôle d'une clé primaire.
   (b) Définir le rôle d'une clé étrangère.
   (c) Déterminer, en justifiant, si un abonné peut réserver plusieurs fois une même séance.
   (d) M. Charles Dupont réserve trois places au tarif plein et deux places au tarif réduit pour assister à la projection du film "Black widow" le 11 octobre 2021 à 21:00. À l'aide des extraits des relations donnés précédemment, recopier et compléter l'enregistrement correspondant dans la relation `Réservation` ci-dessous :

idAbonné	idSéance	nbPlaces_plein	nbPlaces_réduit

Relation `Reservation`.

2. (a) Parmi les trois requêtes SQL suivantes, recopier celle qui permet d'afficher le titre et le réalisateur des films de moins de 120 minutes.

```
SELECT titre, réalisateur
FROM Film
WHERE durée < 120 ;
```

```
SELECT Film
FROM titre, réalisateur
WHERE durée < 120 ;
```

```
SELECT titre
FROM Film
WHERE durée < 120 ;
```

   (b) En SQL, la fonction `COUNT()` permet de compter le nombre d'enregistrements dans une table. Exprimer en langage naturel la requête SQL suivante :

```
SELECT COUNT(*)
FROM Séance
WHERE date="2021-10-22" OR date="2021-10-23" ;
```

   On remarquera que les dates apparaissent sous la forme `"aaaa-mm-jj"`.
   Par exemple, le 11 octobre 2021 apparaît sous la forme `"2021-10-11"`.

3. Écrire en SQL les requêtes permettant d'effectuer les tâches suivantes :
   (a) Afficher le nom et le prénom de tous les abonnés.
   (b) Afficher le titre et la durée des films projetés le 12 octobre 2021 à 21:00.
   On remarquera que les heures apparaissent sous la forme `"hh:mm"`.

4. (a) Écrire une requête en SQL permettant de modifier la durée du film `Jungle Cruise` (initialement enregistré avec 90) à 127.
   (b) On souhaite écrire une requête SQL permettant de supprimer la séance dont l'attribut `idSéance` vaut 135.
   Déterminer la contrainte d'intégrité que pourrait violer cette requête.
   (c) Écrire la requête précédente en SQL.

Exercice 2 : *(6 points)*

Cet exercice traite du thème «programmation», et principalement de la récursivité.

On rappelle qu'une chaîne de caractères peut être représentée en Python par un texte entre guillemets `""` et que :

- la fonction `len` renvoie la longueur de la chaîne de caractères passée en paramètre ;
- si une variable `ch` désigne une chaîne de caractères, alors `ch[0]` renvoie son premier caractère, `ch[1]` le deuxième, etc. ;
- l'opérateur `+` permet de concaténer deux chaînes de caractères.

Exemples :

```
>>> texte = "bricot"
>>> len(texte)
6
>>> texte[0]
"b"
>>> texte[1]
"r"
>>> "a" + texte
"abricot"
```

On s'intéresse dans cet exercice à la construction de chaînes de caractères suivant certaines règles de construction.

Règle A : une chaîne est construite suivant la règle A dans les deux cas suivants:

- soit elle est égale à `"a"` ;
- soit elle est de la forme `"a"+chaine+"a"`, où `chaine` est une chaîne de caractères construite suivant la règle A.

Règle B : une chaîne est construite suivant la règle B dans les deux cas suivants :

- soit elle est de la forme `"b"+chaine+"b"`, où `chaine` est une chaîne de caractères construite suivant la règle A ;
- soit elle est de la forme `"b"+chaine+"b"`, où `chaine` est une chaîne de caractères construite suivant la règle B.

On a reproduit ci-dessous l'aide de la fonction `choice` du module `random`.

```
>>>from random import choice
>>>help(choice)
Help on method choice in module random:
choice(seq) method of random.Random instance
    Choose a random element from a non-empty sequence.
```

La fonction `A()` ci-dessous renvoie une chaîne de caractères construite suivant la règle A, en choisissant aléatoirement entre les deux cas de figure de cette règle.

```
def A():
    if choice([True, False]):
      return "a"
    else:
      return "a" + A() + "a"
```

1. a. Cette fonction est-elle récursive ? Justifier.

   b. La fonction `choice([True, False])` peut renvoyer `False` un très grand nombre de fois consécutives. Expliquer pourquoi ce cas de figure amènerait à une erreur d'exécution.

Dans la suite, on considère une deuxième version de la fonction `A`. À présent, la fonction prend en paramètre un entier `n` tel que, si la valeur de `n` est négative ou nulle, la fonction renvoie `"a"`. Si la valeur de `n` est strictement positive, elle renvoie une chaîne de caractères construite suivant la règle A avec un `n` décrémenté de 1, en choisissant aléatoirement entre les deux cas de figure de cette règle.

```
def A(n):
    if ... or choice([True, False]) :
      return "a"
    else:
      return "a" + ... + "a"
```

2. a. Recopier sur la copie et compléter aux emplacements des points de suspension `...` le code de cette nouvelle fonction `A`.

   b. Justifier le fait qu'un appel de la forme `A(n)` avec `n` un nombre entier positif inférieur à 50, termine toujours.

On donne ci-après le code de la fonction récursive `B` qui prend en paramètre un entier `n` et qui renvoie une chaîne de caractères construite suivant la règle B.

```
def B(n):
    if n <= 0 or choice([True, False]):
      return "b" + A(n-1) + "b"
    else:
      return "b" + B(n-1) + "b"
```

On admet que :

- les appels `A(-1)` et `A(0)` renvoient la chaîne `"a"`;
- l'appel `A(1)` renvoie la chaîne `"a"` ou la chaîne `"aaa"`;
- l'appel `A(2)` renvoie la chaîne `"a"`, la chaîne `"aaa"` ou la chaîne `"aaaaa"`.

3. Donner toutes les chaînes possibles renvoyées par les appels `B(0)`, `B(1)` et `B(2)`.

On suppose maintenant qu'on dispose d'une fonction `raccourcir` qui prend comme paramètre une chaîne de caractères de longueur supérieure ou égale à 2, et renvoie la chaîne de caractères obtenue à partir de la chaîne initiale en lui ôtant le premier et le dernier caractère.
Par exemple :

```
>>> raccourcir("abricot")
"brico"
>>> raccourcir("ab")
""
```

4. a. Recopier sur la copie et compléter les points de suspension ... du code de la fonction `regleA` ci-dessous pour qu'elle renvoie `True` si la chaîne passée en paramètre est construite suivant la règle A, et `False` sinon.

```
def regleA(chaine):
  n = len(chaine)
  if n >= 2:
     return chaine[0] == "a" and chaine[n-1] == "a" and
           regleA(...)
  else:
      return chaine == ...
```

b. Écrire le code d'une fonction `regleB`, prenant en paramètre une chaîne de caractères et renvoyant `True` si la chaîne est construite suivant la règle B, et `False` sinon.

Exercice 3 : *(7 points)*

En informatique, chaque pixel d'une image numérique est affiché sur un écran standard par synthèse additive du rouge, du vert et du bleu : c'est le système colorimétrique RVB. Chacune des trois composantes RVB d'un pixel est stockée sur un octet.

Pour représenter une image, un fichier au format *Windows bitmap* (BMP) contient un entête, suivi des valeurs des composantes RVB de chacun des pixels, écrites les unes à la suite des autres.

1. On considère une image de hauteur 1000 pixels et de largeur 1500 pixels.
   Combien de mégaoctets faut-il pour représenter l'ensemble des pixels de cette image en omettant l'entête ?

2. En Python, les images sont généralement représentées à l'aide de tableaux de tableaux.
   - les composantes RVB d'un pixel sont stockées dans un `tuple` de trois entiers compris entre 0 et 255 ;
   - pour chaque ligne de l'image, les composantes RVB des pixels sont stockées dans un tableau ;
   - l'image est alors représentée par un tableau `img` contenant tous les tableaux précédents ; par convention, l'image vide est représentée par le tableau contenant un tableau vide `[[]]`.

   Ainsi, si l'image `img` a une hauteur de 1000 pixels et largeur de 1500 pixels,

```
>>> len(img)
1000
>>> len(img[0])
1500
```

   La position de chaque pixel dans une image `img` est repérée par un couple d'entiers `(i, j)` qui sont respectivement les indices de ligne et de colonne du pixel dans `img`.

   Encadrer `i` et `j` lorsque l'image `img` a pour hauteur `n` et largeur `m`.

Lorsque `i` et `j` vérifient ces encadrements, on dira qu'ils sont **compatibles** avec la taille de l'image.

On s'intéresse dans la suite de cet exercice à l'algorithme de recadrage intelligent (*seam carving*), qui permet de réduire la largeur d'une image tout en conservant ses principales caractéristiques graphiques.

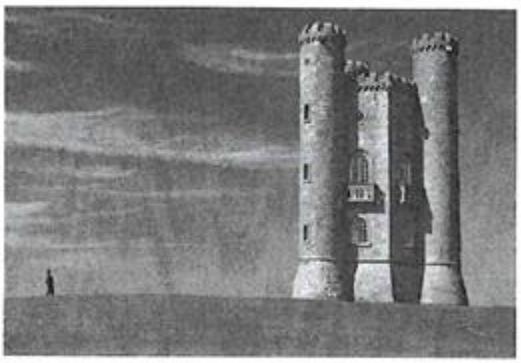


Image dont on souhaite réduire la largeur

Découpe de l'image
une partie du château n'est plus visible

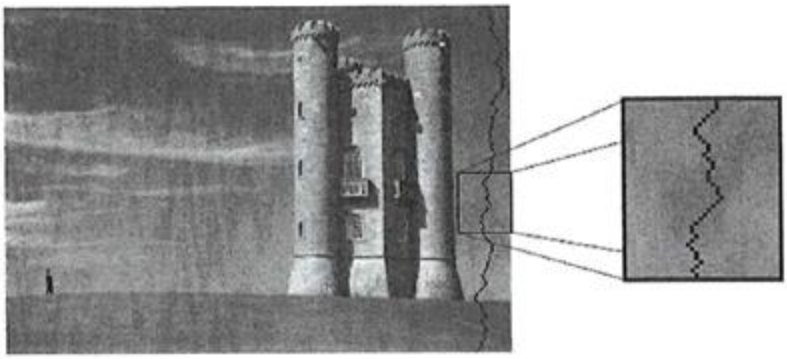
Redimensionnement
le château est déformé

Seam carving
les aspects des principaux éléments graphiques sont conservés

Source des illustrations : https://en.wikipedia.org/wiki/Seam_carving

L'algorithme de recadrage intelligent s'appuie sur la suppression de pixels « bien choisis » dans l'image. Pour déterminer ces pixels, il recherche ce que l'on appellera la « couture de moindre énergie ».

Une couture est une suite de pixels adjacents allant du haut au bas de l'image. On associe à chaque pixel de l'image un nombre positif appelé énergie du pixel. L'énergie d'une couture est alors la somme des énergies des pixels qui la composent.



Une couture et un détail de cette couture.

L'algorithme de recadrage intelligent d'une image de dimensions `n` × `m` (hauteur `n`, largeur `m`) détermine alors la couture de plus basse énergie et supprime de l'image tous les pixels présents dans la couture. L'image résultante est de dimensions `n` × `(m - 1)` (hauteur `n`, largeur `m - 1`). On recommence le procédé jusqu'à obtenir la largeur souhaitée.

3. On suppose que l'on dispose d'une fonction `energie(img, i, j)`, qui prend en paramètres une image `img`, un indice de ligne `i`, un indice de colonne `j` et renvoie l'énergie du pixel à la position `(i, j)` de l'image.
   On décide de calculer au préalable les énergies des différents pixels de l'image. La fonction `calcule_tab_energie` prend comme paramètre une image `img` et renvoie le tableau de tableaux des énergies correspondantes. Ainsi `tab_energie[i][j]` contiendra à la fin de l'exécution l'énergie du pixel à la position `(i, j)`.

```
def calcule_tab_energie(img):
    n = len(img)
    m = len(img[0])
    tab_energie = []
    for i in range(n):
        ligne = []
        for j in range(m):
            e = energie(img, i, j)
            ligne.append(e)
        tab_energie.append(ligne)
    return tab_energie
```

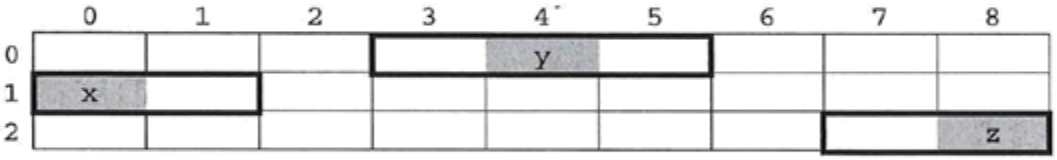
   a. Quel est le type de la variable `tab_energie` renvoyée par la fonction `calcule_tab_energie(img)` ?

   b. Exprimer en fonction de `n` et de `m` le nombre d'appels à la fonction `energie`.

4. On représente en Python une couture à l'aide d'un tableau de couples d'indices `(i, j)` compatibles avec la taille de l'image. Chaque couple `(i, j)` repère la position dans l'image d'un pixel de la couture.
On rappelle que l'énergie d'une couture est la somme des énergies des pixels qui la composent.

   Écrire une fonction `calcule_energie(couture, tab_energie)`, qui prend en paramètres un tableau `couture` représentant une couture ainsi que le tableau des énergies `tab_energie` renvoyé par la fonction `calcule_tab_energie`. Cette fonction doit renvoyer l'énergie de la couture passée en argument.

5. Écrire la fonction `indices_proches(m, i, j)`, qui prend en paramètres `m` le nombre de colonnes de l'image, `i` un indice de ligne, `j` un indice de colonne, et renvoie parmi les positions des pixels `(i,j-1)`, `(i,j)` et `(i,j+1)` celles qui sont compatibles avec les dimensions de l'image.
On ne tiendra pas compte de l'ordre des éléments de la liste renvoyée.
Par exemple, on a repéré dans l'image de dimension 3×9 ci-dessous trois pixels `x`, `y`, `z`.




```
>>> indices_proches(9, 1, 0) # pixels voisins de x
[(1,0), (1,1)]
>>> indices_proches(9, 0, 4) # pixels voisins de y
[(0,3), (0,4), (0,5)]
>>> indices_proches(9, 2, 8) # pixels voisins de z
[(2,7), (2,8)]
```

6. On appellera couture d'indice `j` une couture dont la position du premier pixel est `(0, j)`. Afin de déterminer une couture de basse énergie parmi toutes les coutures d'indice `j`, on met en œuvre une stratégie de type glouton.
Pour cela, on construit la couture du haut vers le bas en sélectionnant à chaque étape le pixel d'énergie minimale parmi ceux situés immédiatement en bas à gauche, immédiatement en bas ou immédiatement en bas à droite lorsque cela est possible.

Par exemple, on donne ci-dessous un tableau `tab_energie` d'une image de dimension `4 × 3`. Les pixels de la couture d'indice `j = 1` y ont été grisés.

	0	1	2
0	3	4	2
1	4	5	6
2	3	2	1
3	7	8	6

Le premier pixel de la couture a pour position `(0, 1)`
Le 2e pixel sera choisi parmi `[(1,0), (1,1), (1,2)]`
Le 3e pixel sera choisi parmi `[(2,0), (2,1)]`
Le 4e pixel sera choisi parmi `[(3,0), (3,1), (3,2)]`

La couture d'indice 1 est donc : `[(0,1), (1, 0), (2, 1), (3, 2)]`

La fonction `calcule_couture(j, tab_energie)`, ci-dessous a pour paramètres un entier `j` et le tableau des énergies `tab_energie`. Cette fonction doit renvoyer une liste de tuples représentant la couture d'indice `j` construite selon le procédé décrit précédemment.
On pourra utiliser la fonction `indices_proches`, ainsi que la fonction `min_energie(tab_energie, indices_pixels)`, qui prend en paramètres le tableau d'énergie `tab_energie` et la liste de positions de pixels `indices_pixels`. La fonction `min_energie` renvoie le tuple `(i,j)` de la position du pixel de moindre énergie.

Recopier et compléter le code de la fonction `calcule_couture` ci-dessous :

```
def calcule_couture(j, tab_energie) :
    n = len(tab_energie)      # hauteur de l'image
    m = len(tab_energie[0]) # largeur de l'image
    couture = []
    couture.append((0, j))    # premier pixel de la couture
    for i in range(1, n):     # i est l'indice de la ligne du
                              # prochain pixel dans la couture
        ...                   # plusieurs lignes possibles

    return couture
```

7. Le paramètre `j` de la fonction précédente peut prendre toutes les valeurs entières comprises entre `0` (inclus) et `m = len(tab_energie[0])` (exclu).
Écrire une fonction `meilleure_couture(tab_energie)`, qui renvoie une couture d'énergie minimale parmi celles obtenues à l'aide de la fonction `calcule_couture`.
L'algorithme consistera à parcourir tous les indices `j` et ne garde qu'une couture d'indice `j` qui soit d'énergie minimale. Pour cela, on pourra utiliser les fonctions `calcule_couture` et `calcule_energie` définies auparavant.