BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION AVRIL 2024

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.

(12 pages)

Le sujet est composé de trois exercices indépendants.

EXERCICE 1 (6 points)

Cet exercice porte sur la notion de listes, la récursivité et la programmation dynamique.

Pour extraire de l'eau dans des zones de terrain instable, on souhaite forer un conduit dans le sol pour réaliser un puits tout en préservant l'intégrité du terrain. Pour représenter cette situation, on va considérer qu'en forant à partir d'une position en surface, on s'enfonce dans le sol en allant à gauche ou à droite à chaque niveau, jusqu'à atteindre le niveau de la nappe phréatique.

Le sol pourra donc être représenté par une pyramide d'entiers où chaque entier est le *score de confiance* qu'on a dans le forage de la zone correspondante. Une telle pyramide est présentée sur la figure 1, à gauche, les flèches indiquant les différents déplacements possibles d'une zone à une autre au cours du forage.

Un conduit doit partir du sommet de la pyramide et descendre jusqu'au niveau le plus bas, où se situe l'eau, en suivant des déplacements élémentaires, c'est-à-dire en choisissant à chaque niveau de descendre sur la gauche ou sur la droite. Le score de confiance d'un conduit est la somme des nombres rencontrés le long de ce conduit. Le conduit gris représenté à droite sur la figure 1 a pour score de confiance $4+2+5+1+3=15$.

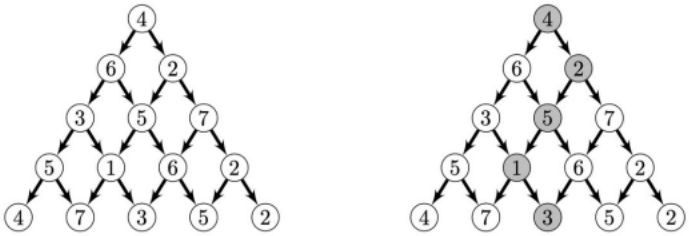


Figure 1.

On va utiliser un ordinateur pour chercher à résoudre ce problème. Pour cela, on représente chaque niveau par la liste des nombres de ce niveau et une pyramide par une liste de niveaux.

La pyramide ci-dessus est donc représentée par la liste de listes

```
ex1 = [[4],[6,2],[3,5,7],[5,1,6,2],[4,7,3,5,2]].
```

1. Dessiner la pyramide représentée par la liste de listes

   ```
   ex2 = [[3],[1,2],[4,5,9],[3,6,2,1]].
   ```

2. Déterminer un conduit de score de confiance maximal dans la pyramide `ex2` et donner son score.

On souhaite déterminer le score de confiance maximal pouvant être atteint pour une pyramide quelconque. Une première idée consiste à énumérer tous les conduits et à calculer leur score pour déterminer les meilleurs.

3. Énumérer les conduits dans la pyramide de trois niveaux représentée sur la figure 2.

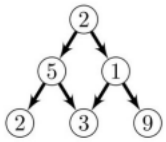


Figure 2.

Afin de compter le nombre de conduits pour une pyramide de `n` niveaux, on remarque qu'un conduit est uniquement représenté par une séquence de `n` déplacements `gauche` ou `droite`.

4. En considérant un codage binaire d'un tel conduit, où `gauche` est représenté par `0` et `droite` par `1`, déterminer le nombre de conduits dans une pyramide de `n` niveaux.

5. Justifier que la solution qui consiste à tester tous les conduits possibles pour calculer le score de confiance maximal d'une pyramide n'est pas raisonnable.

On dira dans la suite qu'un conduit est maximal si son score de confiance est maximal. Afin de pouvoir calculer efficacement le score maximal, on peut analyser la structure des conduits maximaux.

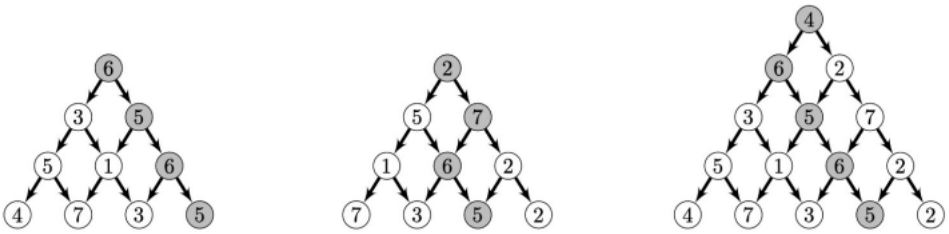


Figure 3.

- **Première observation :** si on a des conduits maximaux `cm1` et `cm2` (représentés en gris dans la figure 3) pour les deux pyramides obtenues en enlevant le sommet de `ex1`, on obtient un conduit maximal en ajoutant le sommet 4 devant le conduit de plus grand score parmi `cm1` et `cm2`. Ici le score de `cm1` est $6+5+6+5=22$ et le score de `cm2` est $2+7+6+5=20$ donc le conduit maximal dans `ex1` est celui obtenu à partir de `cm1` et dessiné à droite dans la figure 3.

- **Deuxième observation :** si la pyramide n'a qu'un seul niveau, il n'y a que le sommet, dans ce cas, il n'y a pas de choix à faire, le seul conduit possible est celui qui contient le sommet et le nombre de ce sommet est le score maximal que l'on peut obtenir.

Avec ces deux observations, on peut calculer le score maximal possible pour un conduit dans une pyramide `p` par récurrence. Posons `score_max(i,j)` le score maximal possible depuis le nombre d'indice `j` du niveau `i`, c'est-à-dire dans la petite pyramide issue de ce nombre. On a alors les relations suivantes :

- `score_max(len(p)-1,j,p) = p[len(p)-1][j]` ;
- `score_max(i,j,p) = p[i][j] + max(score_max(i+1,j,p),score_max(i+1,j+1,p))`.

Le score maximal possible pour `p` toute entière sera alors `score_max(0,0,p)`.

6. Écrire la fonction récursive `score_max` qui implémente les règles précédentes.

Si on suit à la lettre la définition de `score_max`, on obtient une résolution dont le coût est prohibitif à cause de la redondance des calculs. Par exemple `score_max(3,1,p)` va être calculé pour chaque appel à `score_max(2,0,p)` et `score_max(2,1,p)`. Pour éviter cette redondance, on décide de mettre en place une approche par programmation dynamique. Pour cela, on va construire une pyramide `s` dont le nombre à l'indice `j` du niveau `i` correspond à `score_max(i,j,p)`, c'est-à-dire au score maximal pour un conduit à partir du nombre correspondant dans `p`.

7. Écrire une fonction `pyramide_nulle` qui prend en paramètre un entier `n` et construit une pyramide remplie de `0` à `n` niveaux.

8. Compléter la fonction `prog_dyn` ci-dessous qui prend en paramètre une pyramide `p`, et qui renvoie le score maximal pour un conduit dans `p`. Pour cela, on construit une pyramide `s` remplie de `0` de la même taille et la remplit avec les valeurs de `score_max` en commençant par le dernier niveau et en appliquant petit à petit les relations données ci-dessus.

```
1  def prog_dyn(p):
2      n = len(p)
3      s = ...
4      # remplissage du dernier niveau
5      for j in ...
6          s[n-1][j] = ...
7      # remplissage des autres niveaux
8      for i in ...
9          for j in ...
10             s[i][j] = ...
11     # renvoie du score maximal
12     return s[0][0]
```

9. Montrer que le coût d'exécution de cette fonction est quadratique en `n` pour une pyramide à `n` niveaux.

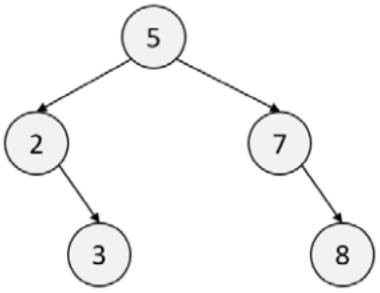
10. Expliquer comment adapter la fonction `score_max` pour éviter la redondance des calculs afin d'obtenir également un coût quadratique, tout en gardant une approche récursive.

EXERCICE 2 (6 points)

Cet exercice traite principalement du thème « algorithmique, langages et programmation » et en particulier les arbres binaires de recherche. La première partie aborde les arbres en mode débranché via l'application d'un algorithme sur un exemple. La suivante porte sur la programmation orientée objet. La dernière partie fait le lien avec les algorithmes de tri.

Partie A : Étude d'un exemple

Considérons l'arbre binaire de recherche ci-dessous.



1. Indiquer quelle valeur a le nœud racine et quels sont les fils de ce nœud.
2. Indiquer quels sont les nœuds de la branche qui se termine par la feuille qui a pour valeur 3.
3. Dessiner l'arbre obtenu après l'ajout de la valeur 6.

Partie B : Implémentation en Python

Voici un extrait d'une implémentation en Python d'une classe modélisant un arbre binaire de recherche.

```
class ABR:
    """Implémentation d'un arbre binaire de recherche (ABR)"""
    def __init__(self, valeur=None):
        self.valeur = valeur
        self.fg = None
        self.fd = None

    def estVide(self):
        return self.valeur == None

    def insererElement(self, e):
        if self.estVide():
            self.valeur = e
        else:
            if e < self.valeur:
                if self.fg:
```

```
                self.fg.insererElement(e)
            else:
                self.fg = ABR(e)
        if e > self.valeur:
            if self.fd:
                self.fd.insererElement(e)
            else:
                self.fd = ABR(e)
```

1. Expliquer le rôle de la fonction `__init__`.

2. Dans cette implémentation, expliquer ce qui se passe si on ajoute un élément déjà présent dans l'arbre.

3. Recopier et compléter les lignes de code surlignées ci-dessous permettant de créer l'arbre de la partie A.

Partie C : Tri par arbre binaire de recherche

On souhaite trier un ensemble de valeurs entières distinctes grâce à un arbre binaire de recherche. Pour cela, on ajoute un à un les éléments de l'ensemble dans un arbre initialement vide. Il ne reste plus qu'à parcourir l'arbre afin de lire et de stocker dans un tableau résultat les valeurs dans l'ordre croissant.

1. Donner le nom du parcours qui permet de visiter les valeurs d'un arbre binaire de recherche dans l'ordre croissant.

2. Comparer la complexité de cette méthode de tri avec celle du tri par insertion ou du tri par sélection.

EXERCICE 3 (8 points)

Cet exercice porte sur la programmation Python (dictionnaire), la programmation orientée objet, les bases de données relationnelles et les requêtes SQL.

Cet exercice est composé de 3 parties indépendantes.

On veut créer une application permettant de stocker et de traiter des informations sur des livres de science-fiction. On désire stocker les informations suivantes :

- l'identifiant du livre (`id`) ;
- le titre (`titre`) ;
- le nom de l'auteur (`nom_auteur`) ;
- l'année de première publication (`ann_pub`) ;
- une note sur 10 (`note`).

Voici un extrait des informations que l'on cherche à stocker :

Livres de science-fiction				
id	titre	auteur	ann_pub	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
14	Fondation	Asimov	1951	9
4	Ubik	K.Dick	1953	9
8	Blade Runner	K.Dick	1968	8
7	Les Robots	Asimov	1950	10
15	Ravage	Barjavel	1943	6
17	Chroniques martiennes	Bradbury	1950	7
9	Dragon déchu	Hamilton	2003	8
10	Fahrenheit 451	Bradbury	1953	8

Partie A

Dans cette première partie, on utilise un dictionnaire Python. On considère le programme suivant :

```
1 dico_livres = {
    'id' : [1, 2, 14, 4, 5, 8, 7, 15, 9, 10],
    'titre' : ['1984', 'Dune', 'Fondation', 'Ubik', 'Blade Runner',
        'Les Robots', 'Ravage', 'Chroniques martiennes',
        'Dragon déchu', 'Fahrenheit 451'],
    'auteur' : ['Orwell', 'Herbert', 'Asimov',
        'K.Dick', 'K.Dick', 'Asimov', 'Barjavel',
        'Bradbury', 'Hamilton', 'Bradbury'],
    'ann_pub' : [1949, 1965, 1951, 1953,1968,
        1950, 1943, 1950, 2003, 1953],
    'note' : [10, 8, 9, 9, 8, 10, 6, 7, 8, 8]
  }

2 a = dico_livres['note']
3 b = dico_livres['titre'][2]
```

1. Déterminer les valeurs des variables `a` et `b` après l'exécution de ce programme.

La fonction `titre_livre` prend en paramètre un dictionnaire (de même structure que `dico_livres`) et un identifiant, et renvoie le titre du livre qui correspond à cet identifiant. Dans le cas où l'identifiant passé en paramètre n'est pas présent dans le dictionnaire, la fonction renvoie `None`.

```
1 def titre_livre(dico, id_livre):
2     for i in range(len(dico['id'])):
3         if dico['id'][i] == ... :
4             return dico['titre'][...]
5     return ...
```

2. Recopier et compléter les lignes 3, 4 et 5 de la fonction `titre_livre`.
3. Écrire une fonction `note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la note maximale.
4. Écrire une fonction `livres_note` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et une note `n`, et qui renvoie la liste des titres des livres ayant obtenu la note `n` (on rappelle que `t.append(a)` permet de rajouter l'élément `a` à la fin de la liste `t`).
5. Écrire une fonction `livre_note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la liste des titres des livres ayant obtenu la meilleure note sous la forme d'une liste Python.

Dans cette partie, on utilise le paradigme orientée objet (POO). On propose deux classes : `Livre` et `Bibliotheque`.

```
1  class Livre:
2      def __init__(self, id_livre, titre, auteur, ann_pub, note):
3          self.id = id_livre
4          self.titre = titre
5          self.auteur = auteur
6          self.ann_pub = ann_pub
7          self.note = note
8      def get_id(self):
9          return self.id
10     def get_titre(self):
11         return self.titre
12     def get_auteur(self):
13         return self.auteur
14     def get_ann_pub(self):
15         return self.ann_pub
16
17 class Bibliotheque:
18     def __init__(self):
19         self.liste_livre = []
20     def ajout_livre(self, livre):
21        self.liste_livre.append(livre)
22     def titre_livre(self, id_livre):
23         for livre in self.liste_livre :
24             if ... == id_livre :
25                 return ...
26         return ...
```

6. Citer un attribut et une méthode de la classe `Livre`.
7. Écrire la méthode `get_note` de la classe `Livre`. Cette méthode devra renvoyer la note d'un livre.
8. Écrire le programme permettant d'ajouter le livre Blade Runner à la fin de la "bibliothèque" en utilisant la classe `Livre` et la classe `Bibliotheque` (voir le tableau en début d'exercice).

9. Recopier et compléter la méthode `titre_livre` de la classe `Bibliotheque`. Cette méthode prend en paramètre l'identifiant d'un livre et renvoie le titre du livre si l'identifiant existe, ou `None` si l'identifiant n'existe pas.

Partie C

On utilise maintenant une base de données relationnelle. Les commandes nécessaires ont été exécutées afin de créer une table `livres`. Cette table `livres` contient toutes les données sur les livres. On obtient donc la table suivante :

livres				
id	titre	auteur	ann_pub	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
14	Fondation	Asimov	1951	9
4	Ubik	K.Dick	1953	9
8	Blade Runner	K.Dick	1968	8
7	Les Robots	Asimov	1950	10
15	Ravage	Barjavel	1943	6
17	Chroniques martiennes	Bradbury	1950	7
9	Dragon déchu	Hamilton	2003	8
10	Fahrenheit 451	Bradbury	1953	8

L'attribut `id` est la clé primaire pour la table `livres`.

10. Expliquer pourquoi l'attribut `auteur` ne peut pas être choisi comme clé primaire.
11. Donner le résultat renvoyé par la requête SQL suivante :

```
SELECT titre
FROM livres
WHERE auteur = 'K.Dick';
```

12. Écrire une requête SQL permettant d'obtenir les titres des livres écrits par Asimov publiés après 1950.
13. Écrire une requête SQL permettant de modifier la note du livre Ubik en la passant de 9/10 à 10/10.

On souhaite proposer plus d'informations sur les auteurs des livres. Pour cela, on crée une deuxième table `auteurs` avec les attributs suivants :

- `id` de type INT ;
- `nom` de type TEXT ;
- `prenom` de type TEXT ;
- `annee_naissance` de type INT (année de naissance).

auteurs			
id	nom	prenom	annee_naissance
1	Orwell	George	1903
2	Herbert	Franck	1920
3	Asimov	Isaac	1920
4	K.Dick	Philip	1928
5	Bradbury	Ray	1920
6	Barjavel	René	1911
7	Hamilton	Peter	1960

La table `livres` est aussi modifiée comme suit :

livres				
id	titre	id_auteur	ann_pub	note
1	1984	1	1949	10
2	Dune	2	1965	8
14	Fondation	3	1951	9
4	Ubik	4	1953	9
8	Blade Runner	4	1968	8
7	Les Robots	3	1950	10
15	Ravage	6	1943	6
17	Chroniques martiennes	5	1950	7
9	Dragon déchu	7	2003	8
10	Fahrenheit 451	5	1953	8

14. Expliquer l'intérêt d'utiliser deux tables (`livres` et `auteurs`) au lieu de regrouper toutes les informations dans une seule table.
15. Expliquer le rôle de l'attribut `id_auteur` de la table `livres`.
16. Écrire une requête SQL qui renvoie le nom et le prénom des auteurs des livres publiés après 1960.

17. Décrire par une phrase en français le résultat de la requête SQL suivante :

```
SELECT titre
FROM livres
JOIN auteurs ON id_auteur = auteurs.id
WHERE ann_pub - annee_naissance < 30;
```

Un élève décide de créer une application d'annuaire pour sa classe. On pourra retrouver, grâce à cette application, différentes informations sur les élèves de la classe : nom, prénom, date de naissance, numéro de téléphone, adresse email, etc.

18. Expliquer en quoi la réalisation de ce projet pourrait être problématique.