PUISSANCE ALPHA

Concours Ingénieurs Bac+5

ANNALES
Samedi 27 avril 2024

Bac général :
ÉPREUVE DE SCIENCES APPLIQUÉES
Durée : 1H

☑ Vous devez **traiter 1 seule matière et y choisir 6 exercices**
☑ Vous devez **traiter la matière présentée au Bac** (et indiquée sur votre étiquette)

Concernant les candidats présentant deux de ces matières au Bac, vous devez **n'en choisir qu'une à traiter.**

☑ Vous devez **traiter 6 exercices (au choix)** de la seule et unique matière que vous avez choisie.

- Exercices 1 à 7 **: EDS Numérique et Sciences Informatiques**
- Exercices 8 à 14 **: EDS Science de l'Ingénieur**
- Exercices 15 à 24 **: EDS Sciences de la Vie et de la Terre / Éco-biologie**
- Exercices 25 à 31 **: EDS Physique-chimie**
- Exercices 32 à 38 **: Tronc commun de sciences (réservé à ceux qui n'ont AUCUNE de ces EDS en terminale)**

Si vous traitez plus de 6 exercices de la matière, **seuls les 6 premiers seront corrigés**.
Si vous sélectionnez plusieurs exercices de différentes matières, seules les réponses aux exercices de la 1ère matière seront comptabilisées.

- Un exercice comporte **4 affirmations** repérées par les lettres **a, b, c, d.**
- Vous devez indiquer pour chacune d'elles si elle est **vraie (V) ou fausse (F).**
- ***Un exercice est considéré comme traité dès qu'une réponse à une des 4 affirmations est donnée.***

- Une réponse exacte rapporte 1 point.
- Une réponse inexacte entraîne le retrait de 0.5 point.
- Une réponse annulée ou l'abstention de réponse ne rapporte ni ne retire aucun point.

L'attention des candidats est attirée sur le fait que, dans le type d'exercices proposés, une lecture attentive des énoncés est absolument nécessaire, le vocabulaire employé et les questions posées étant très précis.

L'usage de la calculatrice ou de tout appareil électronique est interdit.

PARTIE NSI

Choisir 6 exercices
entre les exercices 1 et 7

Exercice n°1 : Arbres Binaires

Dans cette question, la méthode `vide` renvoie: `True` si l'arbre est vide, `False` sinon, les méthodes `g` et `d`: les sous-arbre gauche et sous-arbre droit, la méthode `clef`: la valeur de la racine de l'arbre. On considère qu'un arbre vide est un arbre binaire de recherche.

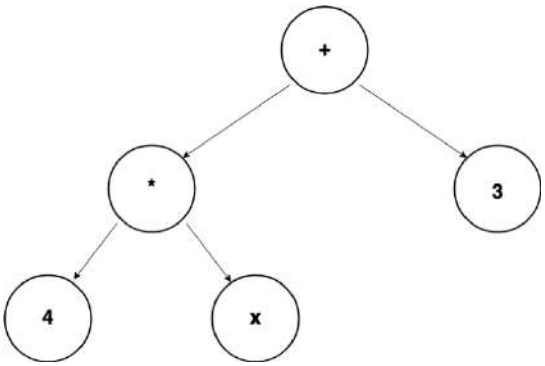
a) *La fonction suivante renvoie `True` si un arbre binaire, dont les clefs sont des nombres, est un arbre binaire de recherche. Dans le cas contraire, elle renvoie `False`.*

```
def est_abr(a):
    if a.vide() or (a.g().vide() and a.d().vide()):
        return True
    if (not a.g().vide()) and a.clef() < a.g().clef():
        return False
    if (not a.d().vide()) and a.clef() > a.d().clef():
        return False
    if not(a.g().vide() or a.d().vide()) and a.g().clef() > a.d().clef():
        return False
    return est_abr(a.g()) or est_abr(a.d())
```

b) *Il est possible à partir d'un parcours préfixe d'un arbre binaire de déterminer sa hauteur avec une complexité en temps logarithmique (O(ln(n)) avec n le nombre de noeuds de l'arbre) dans le pire des cas.*

c) *Il est possible à partir d'un parcours en largeur d'un arbre binaire de déterminer sa hauteur avec une complexité en temps linéaire (O(n) avec n le nombre de nœuds de l'arbre) dans le pire des cas.*

d) *Lors du parcours postfixe (parfois appelé suffixe) de l'arbre ci-dessous, l'ordre de visite des nœuds est 4 * x + 3.*



Exercice n°2 : Réseaux et sécurité

a) *La seule différence entre les protocoles de routage OSPF et RIP est la métrique utilisée.*
b) *Si vous utilisez la méthode POST pour envoyer des données personnelles à un serveur Web via HTTPS, ces données ne seront pas visibles dans l'URL de la requête. Cela signifie qu'un attaquant ne pourra pas les voir, même s'il arrive à décrypter la requête.*
c) *Les commutateurs/switch utilisent un protocole de routage pour échanger des informations de routage entre eux.*
d) *Lors d'un échange avec un serveur Web via HTTPS, un attaquant qui intercepte les requêtes ne pourra pas voir le contenu des requêtes car celui-ci est chiffré. Mais il pourra se faire passer pour le serveur Web et envoyer de fausses informations aux clients, sans que celui-ci ne puisse s'en rendre compte.*

Exercice n°3 : Système d'exploitation, processus

On s'intéresse à la suite de commandes entrée dans un terminal suivante :

```
$ pwd
/home/camille/web/static/
$ ls
style.css logo.png
$ mkdir img
$ mv logo.png img/
$ cp img/logo.png ../logo-saved.png
$ ls
```

a) *Le dernier `ls` dans la suite de commande ci-dessus affiche: `style.css logo-saved.png`.*
b) *A l'exception du processus init, un processus ne peut être lancé que par un autre processus.*
c) *L'UID d'un processus est un code unique attribué par un système d'exploitation Unix à un processus pour l'identifier.*
d) *Les systèmes d'exploitation de la famille UNIX proposent pour chaque fichier quatre types de droits modifiables: droit de changer les droits, droit de lecture, droit d'écriture, droit d'exécution.*

Exercice n°4 : Graphes

Dans cette partie, la méthode `voisin` renvoie la liste des voisins d'un sommet donné. La méthode `sommets`, renvoie la liste des sommets du graphe.
La distance entre deux nœuds d'un graphe est la longueur d'un plus court chemin entre ces deux nœuds. La longueur d'un chemin est sa longueur en nombre d'arêtes.

a) *La fonction suivante permet de déterminer la distance entre les deux sommets `depart` et `arrivee` d'un graphe non orienté connexe `G`.*

```
def distance(depart, arrivee, G):
    d = 0
    vus = {}
    pile = [depart]
    while len(pile) > 0:
        s = pile.pop()
        if s == arrivee:
            return d
        vus[s] = True
        d = d + 1
        for v in G.voisins(s):
            if v not in vus:
                pile.append(v)
```

Le degré d'un sommet dans un graphe non orienté est le nombre d'arêtes reliant ce sommet.

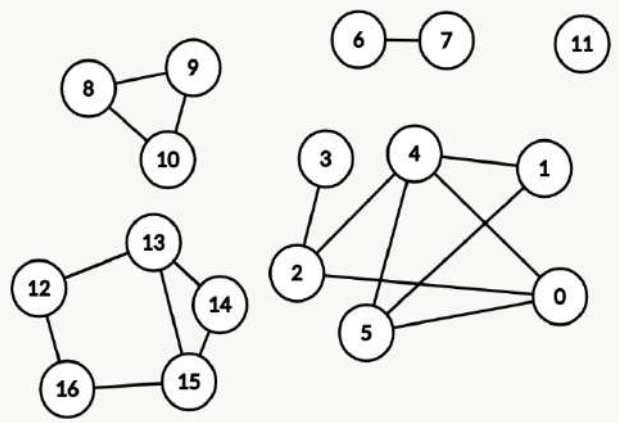
b) *Le degré du sommet d'indice i d'un graphe non orienté, est la somme des éléments de la iéme colonne et de la iéme ligne de sa matrice d'adjacence.*

c) *Pour le graphe ci-dessous la fonction `mystere` renvoie 5.*

```
def mystere(G):
    vus = {}

    def rec(s):
        vus[s] = True
        for v in G.voisins(s):
            if v not in vus:
                rec(s)

    total = 0
    for s in G.sommets():
        if s not in vus:
            rec(s)
            total = total + 1
    return total
```



Un graphe non orienté a pour matrice d'adjacence :

[[0, 1, 0, 1, 0],
[1, 0, 0, 0, 0],
[0, 0, 0, 1, 0],
[1, 0, 1, 0, 1],
[0, 0, 0, 1, 0]]

d) *Ce graphe est acyclique.*

Exercice n° 5 : Base de données

On considère une base de données `Voyage` dont le schéma relationnel est le suivant :

- `Voyageur (idVoyageur, nom, prenom)`
- `Sejour (idSejour, #idVoyageur, #codeResidence, debut, duree)`
- `Residence (code, nbrPlace, adresse)`
- `Activite (#codeResidence, codeActivite, description)`

Dans ce schéma, les clés primaires sont soulignées et les clés étrangères sont précédées du symbole #.

a) *D'après ce schéma, il est possible qu'une résidence n'ait pas d'activité.*

b) Le schéma n'est pas correct car `codeResidence` est à la fois une clef primaire et une clef étrangère pour la table `Activite`.

c) Il est possible à partir de ce schéma de déterminer si une personne dont on connait le nom et le prénom a pu se voir proposer une activité dont le `codeActivite` est ski.

d) La requête suivante donne la durée des séjours effectués dans une résidence proposant 4 places ou plus, proposant l'activité Surf mais ne proposant pas l'activité Voile.

```
SELECT duree
  FROM Sejour AS S
  JOIN Residence AS R
    ON S.codeResidence = R.code
  JOIN Activite as A
    ON A.codeResidence = R.code
WHERE (R.nbrPlace >= 4 AND
       A.codeActivite = 'Surf' AND A.codeActivite <> 'Voile')
```

Exercice n° 6 : Analyse de programme, débogage

a) Une des fonctionnalités classiques d'un débogueur est de déterminer si pour un programme quelconque donné en entrée, ce programme boucle indéfiniment pour certaines entrées.

b) *Une bonne couverture par des tests unitaires permet de démontrer qu'une fonction renvoie toujours la valeur correspondant à ses spécifications pour toutes les données correspondant à ses préconditions.*

c) *La fonction `appartient_cercle` suivante permet de déterminer correctement si un point se trouve sur le cercle de centre `C` et de rayon `r`.*

```python
import math

def distance(A, B):
    xA, yA = A
    xB, yB = B
    return math.sqrt((xA - xB)**2 + (yA - yB)**2)


def appartient_cercle(C, r, M):
    return distance(C, M) == r
```

d) *La fonction* `recherche_dichotomique` *suivante s'arrête toujours lorsqu'elle prend un tableau d'entiers et un entier en entrée. En effet* `b - a` *correspond à un variant de boucle pour l'algorithme implémenté.*

```
def recherche_dichotomique(t, x):
    assert sorted(t) == t
    a = 0
    b = len(t) - 1
    while a <= b:
        m = (a + b) // 2
        if t[m] == x:
            return m
        if t[m] < x:
            a = m
        else:
            b = m
    return -1
```

Exercice n°7 : Méthodes algorithmiques

Le problème du sous-tableau de somme maximale consiste à trouver, au sein d'un tableau `t`, une suite d'éléments contigus dont la somme est maximale. Autrement dit, on souhaite déterminer la valeur maximale de la somme `t[i] + t[i+1] + ... + t[j-1]` lorsque les indices `i` et `j` varient en respectant la contrainte $0 \leq i < j \leq n$ avec n la taille du tableau.

Par exemple, pour `t = [-2, 1, -3, 4, -1, 2, 1, -5, 4]`, le sous-tableau `[4, -1, 2, 1]` a la plus grande somme : 6.

On admet l'existence d'une fonction `somme_max_inclu`, telle que `somme_max_inclu(t, g, d, k)` renvoie la somme maximale parmi les sous-tableaux contigus de `t` entre les indices `g` et `d` et contenant l'indice `k`.

Par exemple, `somme_max_inclu([-2, 1, -3, 4, -1, 2, 1, -5, 4], 0, 8, 7)` renvoie 5, car le sous-tableau de somme maximal contenant l'élément à l'indice 7 est `[4, -1, 2, 1, -5, 4]`.

a) *La fonction* `somme_coupe_max` *suivante renvoie la somme maximale parmi les sous-tableaux contigus de* `t` *entre les indices* `g` *et* `d` *:*

```
def somme_coupe_max(t, g, d):
    if (g == d):
        return t[g]
    m = (g + d) // 2
    return max(somme_coupe_max(t, g, m-1), somme_coupe_max(t, m+1, d),
               somme_max_inclu(t, g, d, m))
```

Pour $1 \leq i < n$, on pose $x_0 = t[0]$, $y_0 = t[0]$, $x_i = \max(t[i],\ x_{i-1} + t[i])$, et $y_i = \max(y_{i-1},\ x_i)$.

b) *La valeur x_{n-1} correspond à la somme maximale parmi les sous-tableaux contigus de t.*

c) *Il existe un algorithme basé sur la programmation dynamique résolvant le problème avec une complexité en temps linéaire (O(n)) en la taille du tableau*

d) *La fonction suivante résout le problème.*

```
def somme_coupe_maxi(t):
    assert t
    meilleur = t[0]
    max_local = 0
    for i in range(len(t)):
        max_local = max_local + t[i]
        if meilleur < max_local:
            meilleur = max_local
        if max_local < 0:
            max_local = 0
    return meilleur
```