BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 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.

Il comporte 12 pages et 3 exercices indépendants que vous pouvez traiter dans l'ordre que vous souhaitez.

Exercice 1 - *listes, dictionnaires et programmation de base en Python* 8 points

Pour son évaluation de fin d'année, l'institut d'Enseignement Néo-moderne (EN) a décidé d'adopter le principe du QCM. Chaque évaluation prend la forme d'une liste de questions numérotées de 0 à 19. Pour chaque question, 5 réponses sont proposées. Les réponses sont numérotées de 1 à 5. Exactement une réponse est correcte par question et chaque candidat coche exactement une réponse par question. Pour chaque évaluation, on dispose de la correction sous forme d'une liste `corr` contenant pour chaque question, la bonne réponse ; c'est-à-dire telle que `corr[i]` est la bonne réponse à la question `i`. Par exemple, on présente ci-dessous la correction de l'épreuve 0 :

```
corr0 = [4, 2, 1, 4, 3, 5, 3, 3, 2, 1, 1, 3, 3, 5, 4, 4, 5, 1, 3, 3].
```

Cette liste indique que pour l'épreuve 0, la bonne réponse à la question 0 est 4, et que la bonne réponse à la question 19 est 3.

Avant de mettre une note, on souhaite corriger les copies question par question ; c'est-à-dire associer à chaque copie, une liste de booléens de longueur 20, indiquant pour chaque question, si la réponse donnée est la bonne. Le candidat Tom Matt a rendu la copie suivante pour l'épreuve 0 :

```
copTM = [4, 1, 5, 4, 3, 3, 1, 4, 5, 3, 5, 1, 5, 5, 5, 1, 3, 3, 3, 3].
```

La liste de booléens correspondante est alors :

```
corrTM = [True, False, False, True, True, False, False, False,
False, False, False, False, False, True, False, False, False, False,
True, True].
```

1. Écrire en Python une fonction `corrige` qui prend en paramètre `cop` et `corr`, deux listes d'entiers entre 1 et 5 et qui renvoie la liste des booléens associée à la copie `cop` selon la correction `corr`.

   Par exemple, `corrige(copTM, corr0)` renvoie `corrTM`.

La note attribuée à une copie est simplement le nombre de bonnes réponses. Si on dispose de la liste de booléens associée à une copie selon la correction, il suffit donc de compter le nombre de `True` dans la liste. Tom Matt obtient ainsi 6/20 à l'épreuve 0. On remarque que la construction de cette liste de booléens n'est pas nécessaire pour calculer la note d'une copie.

2. Écrire en Python une fonction `note` qui prend en paramètre `cop` et `corr`, deux listes d'entiers entre 1 et 5 et qui renvoie la note attribuée à la copie `cop` selon la correction `corr`, sans construire de liste auxiliaire.

   Par exemple, `note(copTM, corr0)` renvoie 6.

L'institut EN souhaite automatiser totalement la correction de ses copies. Pour cela, il a besoin d'une fonction pour corriger des paquets de plusieurs copies. Un paquet de copies est donné sous la forme d'un dictionnaire dont les clés sont les noms des candidats et les valeurs sont les listes représentant les copies de ces candidats. On peut considérer un paquet `p1` de copies où l'on retrouve la copie de Tom Matt :

```
p1 = {('Tom', 'Matt'): [4, 1, 5, 4, 3, 3, 1, 4, 5, 3, 5, 1, 5, 5, 5,
1, 3, 3, 3, 3], ('Lambert', 'Ginne'): [2, 4, 2, 2, 1, 2, 4, 2, 2, 5,
1, 2, 5, 5, 3, 1, 1, 1, 4, 4], ('Carl', 'Roth'): [5, 4, 4, 2, 1, 4,
5, 1, 5, 2, 2, 3, 2, 3, 3, 5, 2, 2, 3, 4], ('Kurt', 'Jett'): [2, 5,
5, 3, 4, 1, 5, 3, 2, 3, 1, 3, 4, 1, 3, 1, 3, 2, 4, 4], ('Ayet',
'Finzerb'): [4, 3, 5, 3, 2, 1, 2, 1, 2, 4, 5, 5, 1, 4, 1, 5, 4, 2,
3, 4]}.
```

3. Écrire en Python une fonction `notes_paquet` qui prend en paramètre un paquet de copies `p` et une correction `corr` et qui renvoie un dictionnaire dont les clés sont les noms des candidats du paquet `p` et les valeurs sont leurs notes selon la correction `corr`.

   Par exemple, `notes_paquet(p1, corr0)` renvoie `{('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3}`.

   La fonction `notes_paquet` peut faire appel à la fonction `note` demandée en question 2, même si cette fonction n'a pas été écrite.

Pour éviter les problèmes d'identification des candidats qui porteraient les mêmes noms et prénoms, un employé de l'institut EN propose de prendre en compte les prénoms secondaires des candidats dans les clés des dictionnaires manipulés.

4. Expliquer si on peut utiliser des listes de noms plutôt qu'un couple comme clés du dictionnaire.

5. Proposer une autre solution pour éviter les problèmes d'identification des candidats portant les mêmes prénoms et noms. Cette proposition devra prendre en compte la sensibilité des données et être argumentée succinctement.

Un ingénieur de l'institut EN a démissionné en laissant une fonction Python énigmatique sur son poste. Le directeur est convaincu qu'elle sera très utile, mais encore faut-il comprendre à quoi elle sert.

Voici la fonction en question :

```
1  def enigme(notes) :
2      a = None
3      b = None
4      c = None
5      d = {}
6      for nom in notes :
7          tmp = c
8          if a == None or notes[nom] > a[1] :
9             c = b
10            b = a
11            a = (nom, notes[nom])
12         elif b == None or notes[nom] > b[1] :
13            c = b
14            b = (nom, notes[nom])
15         elif c == None or notes[nom] > c[1] :
16            c = (nom, notes[nom])
17         else :
18             d[nom] = notes[nom]
19         if tmp != c and tmp != None :
20             d[tmp[0]] = tmp[1]
21     return (a, b, c, d)
```

6. Calculer ce que renvoie la fonction `enigme` pour le dictionnaire `{('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3}`.

7. En déduire ce que calcule la fonction `enigme` lorsqu'on l'applique à un dictionnaire dont les clés sont les noms des candidats et les valeurs sont leurs notes.

8. Expliquer ce que la fonction `enigme` renvoie s'il y a strictement moins de 3 entrées dans le dictionnaire passées en paramètre.

9. Écrire en Python une fonction `classement` prenant en paramètre un dictionnaire dont les clés sont les noms des candidats et les valeurs sont leurs notes et qui, en utilisant la fonction `enigme`, renvoie la liste des couples ((prénom, nom), note) des candidats classés par notes décroissantes.

   Par exemple, `classement({('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3})` renvoie `[(('Tom', 'Matt'), 6), (('Lambert', 'Ginne'), 4), (('Kurt', 'Jett'), 4), (('Ayet', 'Finzerb'), 3), (('Carl', 'Roth'), 2)]`.

Le professeur Paul Tager a élaboré une évaluation particulièrement innovante de son côté. Toutes les questions dépendent des précédentes. Il est donc assuré que dès qu'un candidat s'est trompé à une question, alors toutes les réponses suivantes sont

également fausses. M. Tager a malheureusement égaré ses notes, mais il a gardé les listes de booléens associées. Grâce à la forme particulière de son évaluation, on sait que ces listes sont de la forme

```
[True, True, …, True, False, False, …, False].
```

Pour recalculer ses notes, il a écrit les deux fonctions Python suivantes (dont la seconde est incomplète) :

```
def renote_express(copcorr) :
    c = 0
    while copcorr[c] :
        c = c + 1
    return c
```

```
def renote_express2(copcorr) :
    gauche = 0
    droite = len(copcorr)
    while droite - gauche > 1 :
        milieu = (gauche + droite)//2
        if copcorr[milieu] :
            ...
        else :
            ...
     if copcorr[gauche] :
        return ...
    else :
        return ...
```

10. Compléter le code de la fonction Python `renote_express2` pour qu'elle calcule la même chose que `renote_express`.

11. Déterminer les coûts en temps de `renote_express` et `renote_express2` en fonction de la longueur n de la liste de booléens passée en paramètre.

12. Expliquer comment adapter `renote_express2` pour obtenir une fonction qui corrige très rapidement une copie pour les futures évaluations de M. Tager s'il garde la même spécificité pour ses énoncés. Cette fonction ne devra pas construire la liste de booléens correspondant à la copie corrigée, mais directement calculer la note.

Exercice 2 - *graphe* 6 points

La société CarteMap développe une application de cartographie-GPS qui permettra aux automobilistes de définir un itinéraire et d'être guidés sur cet itinéraire. Dans le cadre du développement d'un prototype, la société CarteMap décide d'utiliser une carte fictive simplifiée comportant uniquement 7 villes : A, B, C, D, E, F et G et 9 routes (toutes les routes sont considérées à double sens).

Voici une description de cette carte :

- A est relié à B par une route de 4 km de long ;
- A est relié à E par une route de 4 km de long ;
- B est relié à F par une route de 7 km de long ;
- B est relié à G par une route de 5 km de long ;
- C est relié à E par une route de 8 km de long ;
- C est relié à D par une route de 4 km de long ;
- D est relié à E par une route de 6 km de long ;
- D est relié à F par une route de 8 km de long ;
- F est relié à G par une route de 3 km de long.

1. Représenter ces villes et ces routes sur sa copie en utilisant un graphe pondéré, nommé G1.
2. Déterminer le chemin le plus court possible entre les villes A et D.
3. Définir la matrice d'adjacence du graphe G1 (en prenant les sommets dans l'ordre alphabétique).

Dans la suite de l'exercice, on ne tiendra plus compte de la distance entre les différentes villes et le graphe, non pondéré et représenté ci-dessous, sera utilisé :

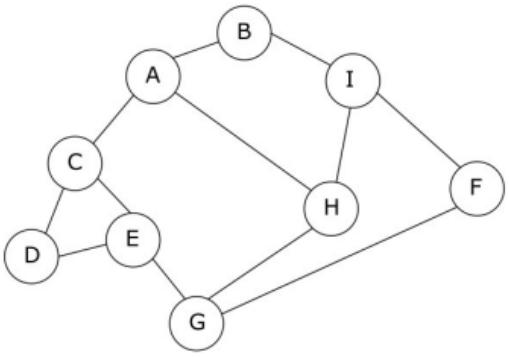


Figure 1. Graphe G2

Chaque sommet est une ville, chaque arête est une route qui relie deux villes.

4. Proposer une implémentation en Python du graphe G2 à l'aide d'un dictionnaire.
5. Proposer un parcours en largeur du graphe G2 en partant de A.

La société CarteMap décide d'implémenter la recherche des itinéraires permettant de traverser le moins de villes possible. Par exemple, dans le cas du graphe G2, pour aller de A à E, l'itinéraire A-C-E permet de traverser une seule ville (la ville C), alors que l'itinéraire A-H-G-E oblige l'automobiliste à traverser 2 villes (H et G).

Le programme Python suivant a donc été développé (programme p1) :

```
1  tab_itineraires=[]
2  def cherche_itineraires(G, start, end, chaine=[]):
3          chaine = chaine + [start]
4          if start == end:
5              return chaine
6          for u in G[start]:
7              if u not in chaine:
8                  nchemin = cherche_itineraires(G, u, end, chaine)
9                  if len(nchemin) != 0:
10                     tab_itineraires.append(nchemin)
11         return []
12
13 def itineraires_court(G,dep,arr):
14     cherche_itineraires(G, dep, arr)
15     tab_court = ...
16     mini = float('inf')
17     for v in tab_itineraires:
18         if len(v) <= ... :
19             mini = ...
20     for v in tab_itineraires:
21         if len(v) == mini:
22             tab_court.append(...)
23     return tab_court
```

La fonction `itineraires_court` prend en paramètre un graphe `G`, un sommet de départ `dep` et un sommet d'arrivée `arr`. Cette fonction renvoie une liste Python contenant tous les itinéraires pour aller de `dep` à `arr` en passant par le moins de villes possible.

Exemple (avec le graphe G2) :

```
itineraires_court(G2, 'A', 'F')
>>> [['A', 'B', 'I', 'F'], ['A', 'H', 'G', 'F'], ['A', 'H', 'I',
'F']]
```

On rappelle les points suivants :

- la méthode `append` ajoute un élément à une liste Python ; par exemple, `tab.append(el)` permet d'ajouter l'élément `el` à la liste Python `tab` ;
- en python, l'expression `['a'] + ['b']` vaut `['a', 'b']` ;
- en python `float('inf')` correspond à l'infini.

6. Expliquer pourquoi la fonction `cherche_itineraires` peut être qualifiée de fonction récursive.
7. Expliquer le rôle de la fonction `cherche_itineraires` dans le programme p1.
8. Compléter la fonction `itineraires_court`.

Les ingénieurs sont confrontés à un problème lors du test du programme p1. Voici les résultats obtenus en testant dans la console la fonction `itineraires_court` deux fois de suite (sans exécuter le programme entre les deux appels à la fonction `itineraires_court`) :

```
exécution du programme p1

itineraires_court(G2, 'A', 'E')
>>> [['A', 'C', 'E']]

itineraires_court(G2, 'A', 'F')
>>> [['A', 'C', 'E']]
```

alors que dans le cas où le programme p1 est de nouveau exécuté entre les 2 appels à la fonction `itineraires_court`, on obtient des résultats corrects :

```
exécution du programme p1

itineraires_court(G2, 'A', 'E')
>>> [['A', 'C', 'E']]

exécution du programme p1

itineraires_court(G2, 'A', 'F')
>>> [['A', 'B', 'I', 'F'], ['A', 'H', 'G', 'F'], ['A', 'H', 'I',
'F']]
```

9. Donner une explication au problème décrit ci-dessus. Vous pourrez vous appuyer sur les tests donnés précédemment.

La société CarteMap décide d'ajouter à son logiciel de cartographie des données sur les différentes villes, notamment des données classiques : nom, département, nombre d'habitants, superficie, …, mais également d'autres renseignements pratiques, comme par exemple, des informations sur les infrastructures sportives proposées par les différentes municipalités.

Exercice 3 - *arbre binaire de recherche, programmation orientée objet et récursivité* 6 points

Le code Morse doit son nom à Samuel Morse, l'un des inventeurs du télégraphe.
Il a été conçu pour transférer rapidement des messages en utilisant une série de points et de tirets.
Pour cet exercice, les points seront représentés par le caractère "o" et les tirets par le caractère "-".

Chaque caractère du message que l'on veut transmettre est constitué d'une série de 1 à 5 points ou tirets. Le code a été conçu en tenant compte de la fréquence de chaque caractère dans la langue anglaise, de sorte que les caractères les plus fréquents, tels que E et T, ne comportent qu'un seul point ou tiret (E = "o", T = "-"), tandis que les caractères moins fréquents peuvent comporter 4 à 5 points ou tirets (par exemple, Q = "- - o -" et J = "o - - -").

Pour connaître le code morse de chaque caractère, on peut utiliser l'arbre binaire ci-dessous. En partant de la racine de l'arbre, la succession des branches reliant le nœud racine au caractère recherché nous donne le code morse de ce caractère en considérant que :

- une branche gauche correspond à un point ("o") ;
- une branche droite correspond à un tiret ("-").

Par exemple, le code morse de la lettre P est "**o - - o**" comme expliqué sur ce schéma :

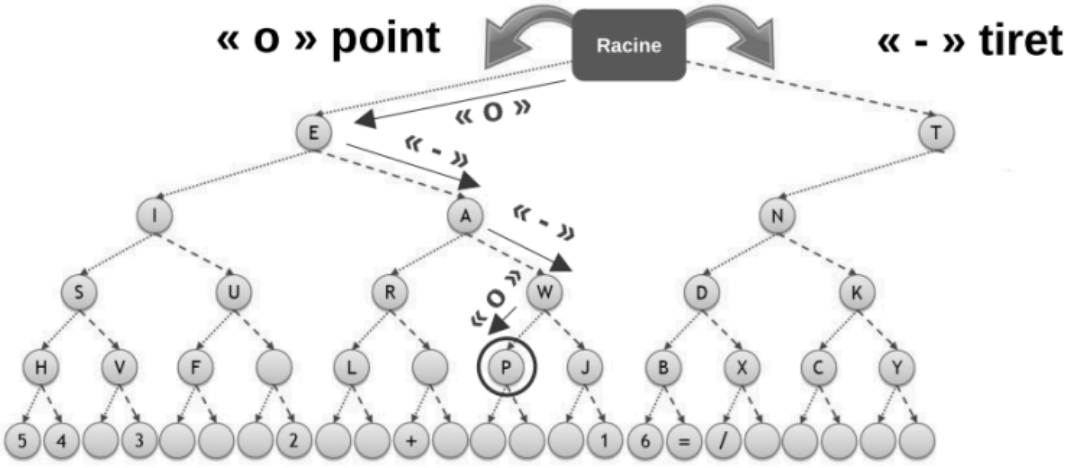


Figure 1 : Extrait de l'arbre binaire du code Morse

1. Déterminer le code morse du message "NSI", à l'aide de la figure de l'arbre binaire, en laissant un espace entre le code de chaque lettre.

2. Représenter le sous-arbre binaire pour les lettres M, G, O, Z et Q à l'aide de l'extrait de la table du code morse international :

G : - - o **M :** - - **O :** - - - **Q :** - - o - **Z :** - - o o

On donne, la déclaration de la classe et un extrait de la définition de l'arbre binaire :

```
1.  class Noeud:
2.    def __init__(self, valeur, gauche=None, droite=None):
3.      self.valeur = valeur
4.      self.gauche = gauche
5.      self.droite = droite
6.
7.  arbre = Noeud("Racine")
8.  arbre.gauche = Noeud("E")
9.  arbre.droite = Noeud("T")
10. arbre.gauche.gauche = Noeud("I")
11. arbre.gauche.droite = Noeud("A")
12. arbre.droite.gauche = Noeud("N")
13. arbre.droite.droite = Noeud("M")
```

3. Écrire les instructions à placer en ligne 14 et 15 permettant de créer les nœuds pour les lettres K et S.

4. La fonction `est_present(n,car)` permet de tester si le caractère `car` est présent ou non dans l'arbre `n` de type `Noeud`.

```
1.   def est_present(n, car) :
2.       if n == ..... :
3.           return False
4.       elif n.valeur == ..... :
5.           return True
6.       else :
7.           return est_present(n.droite, car) or .....
```

a. Recopier le code et compléter les lignes 2, 4 et 7 de la fonction `est_present`.

b. La fonction `est_present` est-elle récursive ? Justifier votre réponse.

c. Déterminer quel type de parcours utilise la fonction `est_present`.

5. La fonction `code_morse(n, car)` permet de traduire un caractère `car` **présent** dans l'arbre `n` et renvoie son code morse sous forme d'une chaîne de caractères.

```
8.   def code_morse(n, car):
9.       if n.valeur == car :
10.          return .....
```

```
11.      elif est_present(.....) :
12.          return "-" + code_morse(n.droite, car)
13.      else :
14.          return .....
```

a. Recopier et compléter les ..... des lignes 10, 11 et 14 de la fonction `code_morse`.

b. Écrire une fonction `morse_message` qui reçoit un arbre de code morse et un message sous forme d'une chaîne de caractères et renvoie le message codé où chaque lettre est séparée par un trait vertical. Par exemple :

```
>>> morse_message(arbre, 'PYTHON')
>>>  o--o|-o--|-|oooo|---|-o|
```

CORRIGE

Exercice 4 - *listes, dictionnaires et programmation de base en Python* 8 points

1.
```
def corrige(cop, corr):
    c = []
    for i in range(len(corr)):
        c.append(cop[i] == corr[i])
    return c
```
2.
```
def note(cop, corr):
    note = 0
    for i in range(len(corr)):
        if cop[i] == corr[i]:
            note = note + 1
    return note
```

3.
```
def note_paquet(p, corr):
    d = {}
    for k,v in p.items():
        d[k] = note(v, corr)
    return d
```

4. Les clés d'un dictionnaire ne doivent pas être des valeurs qui peuvent être modifiées (valeurs mutables). Les listes Python étant mutables, il n'est pas possible d'utiliser une liste Python comme clé d'un dictionnaire.

5. Il est possible d'attribuer un identifiant (id) unique à chaque candidat et d'utiliser cet identifiant comme clé.

6. La fonction renvoie :
```
((('Tom', 'Matt'), 6), (('Lambert',
'Ginne'), 4), (('Kurt', 'Jett'), 4),
 {('Carl', 'Roth'): 2, ('Ayet', 'Fin
zerb'): 3})
```

7. Cette fonction renvoie un tuple contenant :
   - les 3 meilleurs candidats sous forme de 3 tuples (classés dans un ordre décroissant de note)
   - un dictionnaire contenant les autres candidats
8. si par exemple, on a seulement 2 candidats, la fonction renvoie un tuple contenant :
   - les 2 candidats sous forme de 2 tuples (classés dans un ordre décroissant)
   - None
   - un dictionnaire vide
9.
```
def classement(d):
    t = []
    while len(d) != 0:
        l = enigme(d)
        a = l[0]
        b = l[1]
        c = l[2]
        d = l[3]
        if a != None:
            t.append(a)
        if b != None:
            t.append(b)
        if c != None:
            t.append(c)
    return t
```

10.
```
def renote_express2(copcorr) :
    gauche = 0
    droite = len(copcorr)
    while droite - gauche > 1 :
        milieu = (gauche + droite)//2
        if copcorr[milieu] :
            gauche = milieu
        else :
            droite = milieu
    if copcorr[gauche] :
        return droite
    else :
        return gauche
```

11. Le coût en temps de `renote_express` est en O(n) (linéaire), alors que le coût en temps de `renote_express2` est en O(log(n)) (logarithmique).

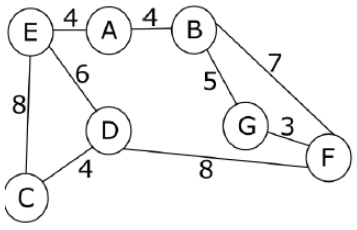
12.

La fonction prendra 2 paramètres `cop` et `corr` et il faudra modifier les lignes 3, 6 et 10 :

```
def renote_express3(cop, corr) :
    gauche = 0
    droite = len(cop)
    while droite - gauche > 1 :
        milieu = (gauche + droite)//2
        if cop[milieu] == corr[milieu] :
            gauche = milieu
        else :
            droite = milieu
    if cop[gauche] :
        return droite
    else :
        return gauche
```

Exercice 5 - *graphe* 6 points

1.



2.

A -> E -> D avec 10 Km

3.

```
0400400
4000075
0004800
0040680
4086000
0708003
0500030
```

4.

```
d = {'A':['B', 'C', 'H'], 'B':['A', 'I'], 'C':['A', 'D', 'E'],
'D':['C', 'E'], 'E': ['C', 'D', 'G'], 'F':['G', 'I'], 'G':['E', 'F',
'H'], 'H':['A', 'G', 'I'], 'I':['B', 'F', 'H']}
```

5.

A-B-C-H-D-E-G-I-F

6. à la ligne 8 la fonction `cherche_itineraires` s'appelle elle-même, la fonction `cherche_itineraires` peut donc être qualifiée de fonction récursive.

7. La fonction `cherche_itineraires` permet de recenser tous les itinéraires (tous les chemins) pour aller de la ville `start` à la ville `end`.

8.
```
def itineraires_court(G,dep,arr):
    cherche_itineraires(G, dep, arr)
    tab_court = []
    mini = float('inf')
    for v in tab_itineraires:
        if len(v) <= mini :
            mini = len(v)
    for v in tab_itineraires:
        if len(v) == mini:
            tab_court.append(v)
    return tab_court
```

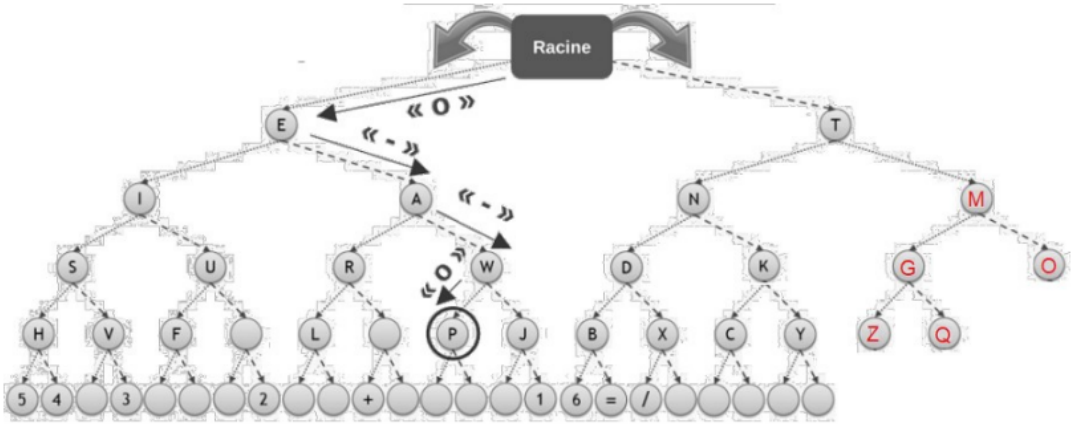
9. Le problème vient de la liste Python `tab_itineraires`. Cette liste est vide au début de l'exécution du programme p1 (ligne 1). Tous les itinéraires possibles pour aller de la ville `start` à la ville `end` sont ajoutés à cette liste grâce à l'appel à la fonction `cherche_itineraires` (ligne 14).
À la suite de l'appel de la fonction `itineraires_court(G2, 'A', 'E')` depuis la console, la liste `tab_itineraires` contient les itinéraires suivants :
`[['A', 'B', 'I', 'F', 'G', 'E'], ['A', 'B', 'I', 'H', 'G', 'E'], ['A', 'C', 'D', 'E'], ['A', 'C', 'E'], ['A', 'H', 'G','E'], ['A', 'H', 'I', 'F', 'G', 'E']]`
Quand on appelle la fonction `itineraires_court(G2, 'A', 'F')` toujours depuis la console, la liste `tab_itineraires` n'a pas été "vidée" (puisque le programme p1 n'a pas été exécuté de nouveau), elle contient donc toujours, entre autres, l'itinéraire `['A', 'C', 'E']`. La fonction `itineraires_court` permet de choisir l'itinéraire le plus court parmi ceux qui se trouvent dans la liste `tab_itineraires`. Sachant que les itinéraires pour aller de A à F ont au minimum une taille de 4, c'est donc l'itinéraire `['A', 'C', 'E']` qui est renvoyé par la fonction, d'où le problème constaté. L'exécution du programme p1 entre les deux appels fait disparaitre le problème puisque la liste est vidée lors de la 2e exécution.

Exercice 6 - *arbre binaire de recherche, programmation orientée objet et récursivité* 6 points

Q1

- o o o o o o

Q2



Q3

```
arbre.droite.gauche.droite = Noeud("K")
arbre.gauche.gauche.gauche = Noeud("S")
```

Q4.a

```
def est_present(n, car) :
    if n == None :
            return False
    elif n.valeur == car :
            return True
    else :
            return est_present(n.droite, car) or est_present(n.gauche, car)
```

Q4.b

oui, elle s'appelle elle-même en ligne 7

Q4.c

Préfixe

Q5.a

```
def code_morse(n, car):
    if n.valeur == car :
            return ''
    elif est_present(n.droite, car) :
            return "-" + code_morse(n.droite, car)
    else :
            return "o" + code_morse(n.gauche, car)
```

Q5.b

```
def morse_message(n: Noeud, message : str) -> str:
    code = ''
    for lettre in message:
            code += code_morse(n, lettre) + '|'
    return code
```