BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2026

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 1

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 16 pages numérotées de 1/16 à 16/16.

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

EXERCICE 1 (6 points)

Cet exercice porte sur la programmation Python, les types abstraits, les graphes et les réseaux informatiques.

Dans cet exercice on considère le réseau ci-dessous.

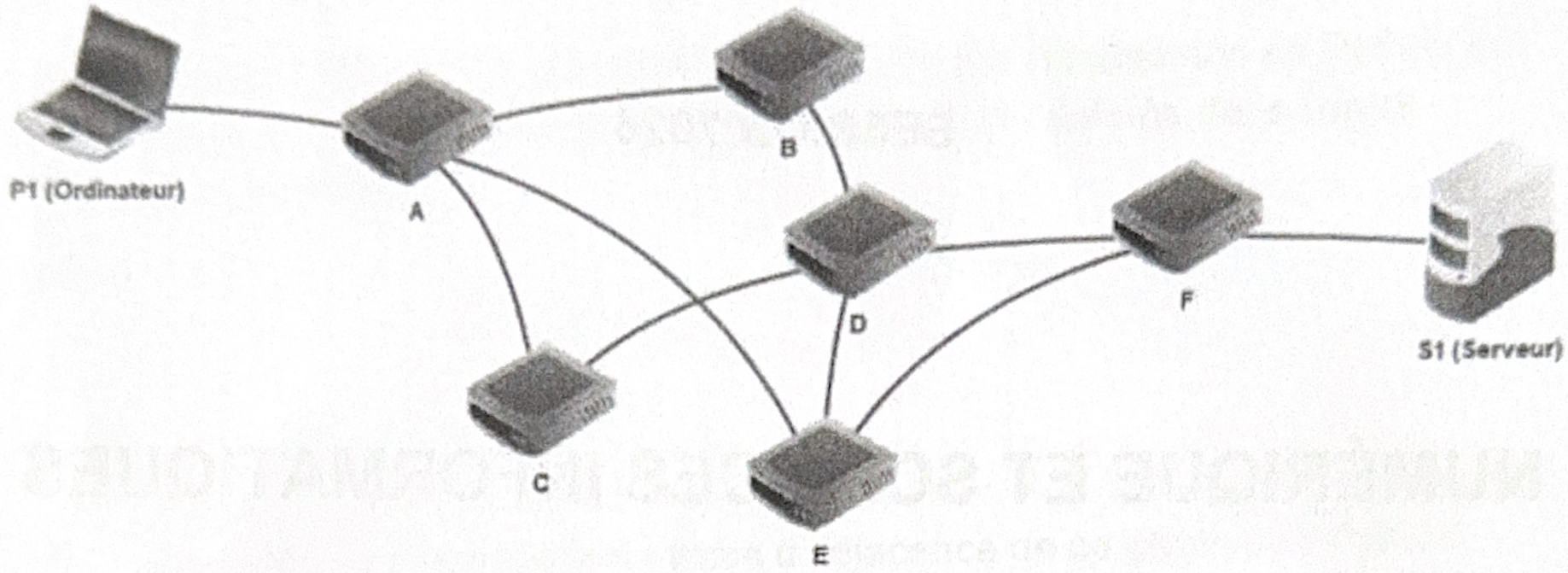


Figure 1. Réseau de routeurs

Partie A

Dans cette première partie, le réseau de routeurs va être modélisé par un graphe (les routeurs constituant les nœuds et les liaisons les arêtes) et le parcours en largeur (BFS) de ce dernier va permettre de trouver le chemin le plus court en termes de routeurs (nombre de sauts) pour aller du routeur A au routeur F. On parle alors de protocole de routage dynamique RIP (Routing Information Protocol) :

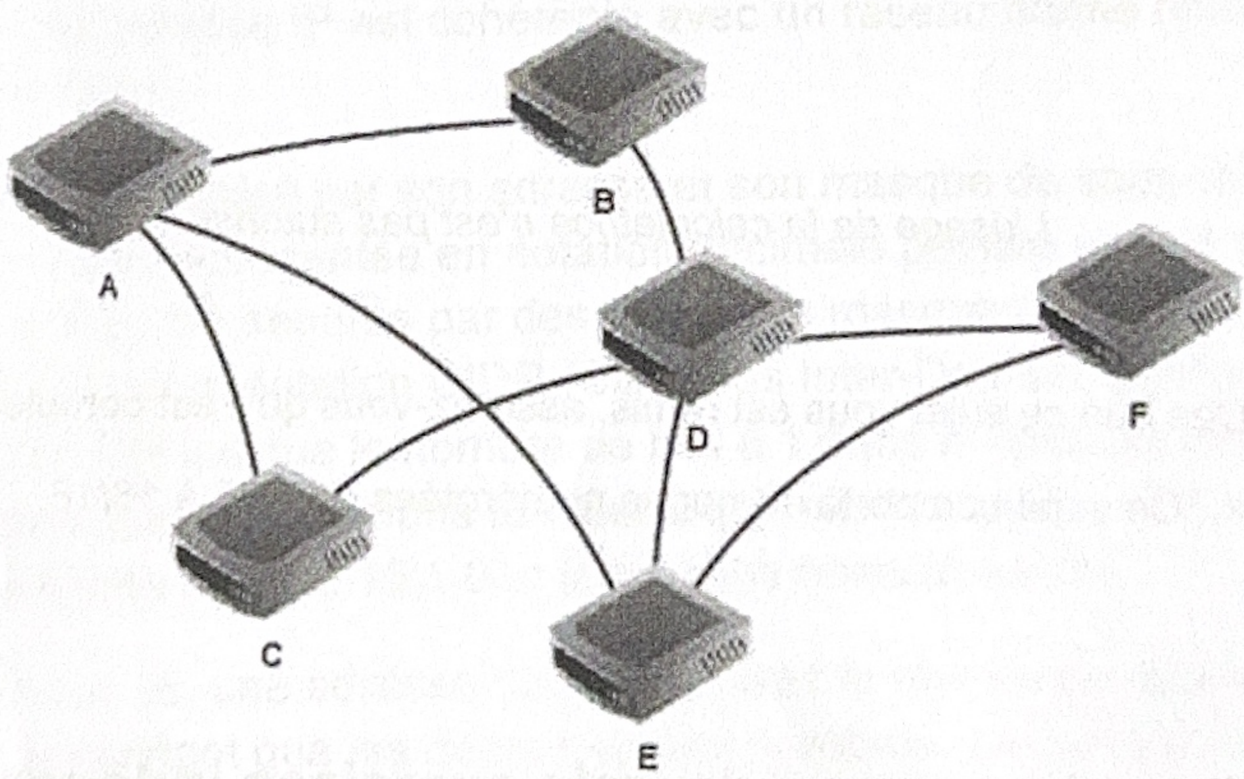


Figure 2. Graphe

1. Déterminer la distance entre les nœuds A et F avec le graphe de la Figure 2.
2. On suppose dans cette question que le routeur E est défaillant. Déterminer la nouvelle distance entre les nœuds A et F.

Pour toutes les autres questions, le routeur E est considéré opérationnel.

3. Déterminer un parcours en largeur (BFS) possible de ce graphe en partant du nœud A.
4. Déterminer un parcours en profondeur (DFS) possible de ce graphe en partant du nœud A.
5. Compléter la matrice d'adjacence de ce graphe représentée en Python par une liste de listes (les nœuds du graphe sont considérés dans l'ordre lexicographique) :

```
graphe_matrice = [[ 0 , 1 , 1 , 0 , 1 , 0 ],
                  [ 1 , 0 , 0 , 1 , 0 , 0 ],
                  [ 1 , 0 , 0 , 1 , 0 , 0 ],
                  [...,...,...,...,...,...],
                  [...,...,...,...,...,...],
                  [...,...,...,...,...,...]]
```

6. Compléter le dictionnaire de listes d'adjacence de ce graphe en Python.

```
graphe_dict = {'A': ['B', 'C', '...'],
               '...': ['...', '...'],
               '...': ['...', '...'],
               '...': ['...', '...', '...', '...'],
               ...
               ...}
```

Partie B

Dans cette seconde partie, on s'intéresse aux adresses IPv4 et plus particulièrement à vérifier qu'une adresse IP est cohérente avec un réseau donné (située dans la plage de ce dernier).

Un réseau est caractérisé par son adresse et son masque de sous-réseau. En IPv4, une adresse IP est représentée en notation décimale pointée avec 4 nombres compris entre 0 et 255 séparés par des points. Le masque peut être représenté de deux manières et ici la notation CIDR (Classless Inter-Domain Routing) sera employée ; le CIDR indique le nombre de bits à 1 dans la notation binaire du masque de sous-réseau. Dans le réseau qui relie l'ordinateur P1 au routeur A, l'adresse du réseau est 172.16.1.0 et le masque associé est /16.

Pour un CIDR de 16, une adresse cohérente avec le réseau se détermine très rapidement en vérifiant que les deux premiers nombres (en partant de la gauche) sont identiques aux deux premiers nombres de l'adresse du réseau.

7. Donner toutes les adresses cohérentes avec un réseau en 172.16.1.0 /16 :

a) 172.16.0.2
b) 172.17.1.1
c) 168.16.1.1
d) 172.1.1.1
e) 173.16.0.1
f) 172.16.16.172

8. Expliquer pourquoi la valeur de chaque nombre d'une adresse ne peut pas dépasser la valeur 255.
9. Déterminer la taille de ce sous-réseau (nombre maximal d'hôtes possibles).

Pour un CIDR de 8, 16 ou 24 il est donc aisé d'identifier la cohérence d'une adresse avec le sous réseau mais pas pour les autres valeurs.

En considérant maintenant que le réseau a un CIDR de 17 bits, la méthode va consister à convertir les deux adresses en binaire (adresse à vérifier et l'adresse du réseau) puis vérifier que les 17 premiers bits sont identiques et si oui, l'adresse à vérifier est bien cohérente avec le sous réseau.

10. Après avoir converti en binaire les adresses 172.16.1.0 et 172.16.128.0, commenter la cohérence de ces deux adresses.

Le programme suivant permet de déterminer la cohérence de deux adresses :

```
1   def ad_en_nb(ad: str) -> int:
2       res = 0
3       liste_nb = ad.split('.')
4       for nb in liste_nb:
5           res = res * 256 + int(nb)
6       return res
7
8   def verifier_coherence(ad1: str, ad2: str, masque: int)->bool:
9      masque_nb = 2**32 - 2**(32 - masque)
10      return masque_nb & ad_en_nb(ad1) == masque_nb & ad_en_nb(ad2)
```

L'instruction de la ligne 3 `liste_valeurs = ad.split('.')` permet d'obtenir la liste des 4 nombres d'une adresse.

Par exemple : `ad = '172.16.1.0' -> ['172', '16', '1', '0']`.

L'instruction de la ligne 10 utilise à deux reprises un `&` qui effectue une opération logique `ET` (`AND`) entre les bits des nombres (nombres convertis en binaire) sur lesquels il est appliqué.

11. Donner, en notation binaire, les valeurs prises successivement par la variable `res`, lors du premier appel de la fonction `ad_en_nb`, après avoir saisi les instructions suivantes :

```
>>> ad_reseau = '172.16.1.0'
>>> masque_reseau = 17
>>> ad_a_verifier = '172.16.131.13'
>>> verifier_coherence(ad_reseau, ad_a_verifier, masque_reseau))
False
```

12. Proposer un exemple d'adresse IP pour la variable `ad_a_verifier` qui permettrait de renvoyer `True` avec la même valeur pour `ad_reseau` et `masque_reseau`.

EXERCICE 2 (6 points)

Cet exercice porte sur les bases de données, les types construits et le traitement de données en table.

Dans cet exercice, on pourra utiliser les clauses du langage SQL pour :

- construire des requêtes d'interrogation à l'aide de `SELECT`, `FROM`, `WHERE` (avec les opérateurs logiques `AND`, `OR`), `JOIN ... ON ;`
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE`, `INSERT`, `DELETE ;`
- réaliser des agrégations à l'aide de `COUNT`.

On soumet à ChatGPT (version gratuite 3.5) la requête de production d'une base de données avec le prompt suivant :

"Peux-tu me donner un schéma de base de données avec 4 tables ? Peux-tu remplir ces tables avec au moins 5 tuples par table ? Les dates doivent être au format ISO 8601. La base doit être sur un thème sportif."

On obtient les deux documents suivants, au format `csv` :

joueurs.csv

```
ID_Joueur  Nom  Prenom Date_Naissance Poste Nationalite ID_Equipe
1   Dupuis Mathieu    1990-05-15 Gardien French 1
2   Martin Pierre 1988-09-22 Attaquant Canadian   2
3   Johnson    Emile  1992-03-10 Defenseur American   3
4   Smith  James  1987-11-28 Milieu English    4
5   Lopez  Sofia  1995-07-03 Attaquant Spanish    5
6   Tremblay   Jean   1991-08-12 Gardien Canadian   1
7   Gagnon Jean-Marie 1993-06-20 Attaquant French 2
8   Rodriguez  Carlos 1994-04-25 Defenseur Spanish    3
9   Schmidt    Lukas  1990-12-01 Milieu German 4
```

equipes.csv

```
ID_Equipe   Nom    Ville  Pays   Annee_Creation
1   Les Aigles Paris  France 1990
2   Les Loups  Montreal   Canada 1985
3   Les Tigres New York   USA    1992
4   Les Lions  Manchester Angleterre 1987
5   Les Taureaux   Madrid Espagne    1995
```

Partie A

La fonction `DictReader` prend en paramètre un descripteur de fichier `csv` et renvoie une liste de dictionnaires dont :

- les clés sont définies par la première ligne du fichier ;
- les valeurs correspondent aux différents champs des autres lignes du fichier.

```
1 import csv
2 with open('joueurs.csv', ) as playerFile :
3     jr = csv.DictReader(playerFile, delimiter='\t')
4     for ligne in jr :
5         print(ligne['Prenom'],ligne['Nom'],ligne['Date_Naissance'])
```

1. Donner les trois premières lignes affichées par le programme précédent.

2. Recopier et compléter les lignes 4 et 8 du programme python ci-après qui affiche le nom du joueur le plus âgé.

```
1 import csv
2 with open('joueurs.csv', ) as playerFile:
3     jr = csv.DictReader(playerFile, delimiter='\t')
4     date_max = ...
5     for ligne in jr:
6         if ... :
7             date_max = ...
8             nom = ...
9 print(nom)
```

3. Recopier et compléter à partir de la ligne 9, le code python de la fonction `nom_eq` ci-après qui prend en paramètre un numéro d'équipe, et qui renvoie le nom de l'équipe.

```
1  def nom_eq(no_eq):
2      """Renvoie le nom de l'équipe dont l'identifiant a été passé
3          en paramètre.
4         Paramètre: no_eq: STR, numéro (identifiant) de l'équipe
5          Résultat:STR, le nom correspondant à l'identifiant
6      """
7      with open('equipes.csv', mode='r', encoding='utf8') as
team_file:
8      team_set = csv.DictReader(team_file, delimiter='\t')
9      ...
```

4. En utilisant la fonction précédente, écrire une fonction `nationalite` qui prend en paramètre une liste de nationalités `nat` et qui renvoie la liste des triplets (nom, prénom, nom de l'équipe) pour chacun des joueurs ayant une nationalité appartenant à la liste `nat`.

Partie B

On considère maintenant que les données sont dans une base de données relationnelle. La table `equipe` contient les données du fichier `equipe.csv` et la table `joueur` contient les données du fichier `joueurs.csv` :

- `joueur(id_joueur, nom, prenom, date_naissance, poste, nationalite, #id_equipe)`
- `equipe(id_equipe, nom, ville, pays, annee_creation)`

joueur						
id_joueur	nom	prenom	date_naissance	poste	nationalite	id_equipe
1	Dupuis	Mathieu	1990-05-15	Gardien	French	1
2	Martin	Pierre	1988-09-22	Attaquant	Canadian	2
3	Johnson	Emily	1992-03-10	Defenseur	American	3
4	Smith	James	1987-11-28	Milieu	English	4
5	Lopez	Sofia	1995-07-03	Attaquant	Spanish	5
6	Tremblay	Jean	1991-08-12	Gardien	Canadian	1
7	Gagnon	Marie	1993-06-20	Attaquant	French	2
8	Rodriguez	Carlos	1994-04-25	Defenseur	Spanish	3
9	Schmidt	Lukas	1990-12-01	Milieu	German	4

equipe				
id_equipe	nom	ville	pays	annee_creation
1	Les Aigles	Paris	France	1990
2	Les Loups	Montreal	Canada	1985
3	Les Tigres	New York	USA	1992
4	Les Lions	Manchester	Angleterre	1987
5	Les Taureaux	Madrid	Espagne	1995

5. Donner les clés primaires des tables `equipe` et `joueur`.

6. Préciser le rôle de la requête SQL suivante et proposer son résultat.

```
SELECT COUNT(*)
FROM joueur
WHERE id_equipe = 2 ;
```

On rappelle que la fonction `COUNT` permet de compter le nombre d'enregistrements lors d'une requête.

7. Proposer une requête SQL qui donne le nom des équipes européennes.

8. Proposer une requête SQL qui, pour chaque joueur américain ou canadien, donne son nom, son prénom et le nom de son équipe.

Partie C

Les autres tables proposées par ChatGPT sont :

- `match(id_match, #id_equipe_domicile, #id_equipe_exterieur, date_match, score_domicile, score_exterieur)`
- `stat_joueur(id_stat, #id_joueur, #id_match, buts, passes_decisives, tirs_cadres, fautes_commises)`

match					
id_match	id_equipe_domicile	id_equipe_exterieur	date_match	score_domicile	score_exterieur
1	1	2	2023-01-10	2	1
2	3	4	2023-02-15	0	0
3	2	5	2023-03-22	3	2
4	1	3	2023-04-05	1	1
5	4	5	2023-05-1	2	3

stat_joueur						
id_stat	id_joueur	id_match	buts	passes_decisives	tirs_cadres	fautes_commises
1	1	1	0	0	0	0
2	2	1	1	1	3	2
3	3	2	0	0	1	1
4	4	2	0	1	2	0
5	5	3	2	0	4	1
6	6	4	0	0	0	0
7	7	4	1	0	2	1
8	8	5	0	0	1	2
9	9	5	1	1	3	0

9. Indiquez pourquoi l'attribut clé `id_stat` de la table `stat_joueur` est inutile.

Après avoir demandé une correction, ChatGPT renvoie la nouvelle table `stat_joueur` suivante :

`stat_joueur(#id_joueur, #id_match, buts, passes_decisives, tirs_cadres, fautes_commises)`

stat_joueur					
id_joueur	id_match	buts	passes_ decisives	tirs_cadres	fautes_ commises
1	1	0	0	0	0
2	1	1	1	3	2
3	2	0	0	1	1
4	2	0	1	2	0
5	3	2	0	4	1
6	4	0	0	0	0
7	4	1	0	2	1
8	5	0	0	1	2
9	5	1	1	3	0

10. Proposer une requête permettant de connaître le nombre de buts inscrits par des joueurs espagnols lors de matchs joués à domicile, ainsi que leurs noms et prénoms.

EXERCICE 3 (8 points)

Cet exercice porte sur la programmation en général, la programmation orientée objet, les arbres binaires et les algorithmes de parcours d'arbre.

Afin de réduire la taille en mémoire de données, nous utilisons des méthodes de compression de données. Le but de cet exercice est d'étudier la méthode de Huffman.

Partie A

En ASCII chaque symbole textuel, y compris les espaces, est codé sur un octet. Ainsi le texte brut «TA TARTE EST TRES TENDRE» nécessitera 24 octets pour être stocké en mémoire. Avec le codage de Huffman chaque symbole va être codé par un nombre de bits dépendant de sa fréquence d'apparition. Le principe est que plus un symbole est utilisé, moins il nécessite de bits pour être représenté. Voici un exemple de code de Huffman pour le texte «TA TARTE EST TRES TENDRE» (sachant que, pour un même texte, il existe plusieurs codes équivalents) :

Caractère	Nombre d'apparitions	Code de Huffman	Taille du codage en bits
T	6	10	2
E	5	01	2
espace	4	110	3
R	3	001	...
A	2	000	...
S	2	1111	...
D	1	11101	...
N	1	11100	...

1. Décoder le mot codé par `100000011001` avec le code ci-dessus.

2. Vérifier que le nombre de bits nécessaires à la représentation du texte « TA TARTE EST TRES TENDRE » avec ce code de Huffman est 67.

3. Justifier, dans cet exemple, que le taux de compression (pourcentage de gain en bits) est supérieur à 50%.

Partie B

La fonction `list` prend en paramètre une chaîne de caractères et renvoie une liste constituée de chaque caractère de la chaîne.

```
>>> phrase = list('TA TARTE EST TRES TENDRE')
>>> phrase
['T', 'A', ' ', 'T', 'A', 'R', 'T', 'E', ' ', 'E', 'S', 'T', ' ',
'T', 'R', 'E', 'S', ' ', 'T', 'E', 'N', 'D', 'R', 'E']
```

4. Donner la lettre associée à `phrase[4]`.

5. Préciser ce que renvoie de la fonction `mystere` suivante :

```
1   def mystere(phrase):
2       symboles = []
3       for caractere in phrase:
4           if caractere not in symboles:
5               symboles.append(caractere)
6       return symboles
```

6. Écrire le code python de la fonction `nombre_apparitions` ci-après qui prend en paramètres une lettre `lettre` et une chaîne de caractères `phrase`, et qui renvoie le nombre de fois où la lettre `lettre` apparait dans la chaîne de caractères `phrase`.

Exemple :

```
>>> nombre_apparitions('E',['E','X','E','M','P','L','E'])
3
```

7. Recopier et compléter la ligne 5 du code python de la fonction `dico` qui prend en paramètre une chaîne de caractère `phrase` et renvoie un dictionnaire dont les clés sont les symboles présents dans la chaîne de caractère `phrase` et les valeurs associées sont le nombre de fois où les clés apparaissent.

Exemple :

```
>>> dico(['E','X','E','M','P','L','E'])
{'E': 3, 'X': 1, 'M': 1, 'P': 1, 'L': 1}
```

```
1   def dico(phrase):
2       liste_cles = mystere(phrase)
3       dico_frequences = {}
4       for symbole in liste_cles:
5           dico_frequences[symbole] = ...
6       return dico_frequences
```

Partie C

Voici un extrait de la classe `ArbreEV` servant à représenter des arbres binaires auxquels on associe une `etiquette` et une `valeur` :

```
1   class ArbreEV():
2
3       def __init__(self, etiquette, valeur):
4           self.etiquette = etiquette
5           self.valeur = valeur
6           self.fils_gauche = None
7           self.fils_droit = None
8
9       def ajouter_gauche(self, fils_gauche):
10          self.fils_gauche = fils_gauche
11
12      def ajouter_droite(self, fils_droit):
13          self.fils_droit = fils_droit
14
15      def est_racine(self):
16          return self.fils_gauche == None and self.fils_droit == None
```

8. Citer un attribut de la classe `ArbreEV`.

9. Citer une méthode de la classe `ArbreEV`.

10. Dessiner l'arbre obtenu après les instructions suivantes :

```
>>> arbre = ArbreEV('E',5)
>>> arbre.ajouter_droite(ArbreEV('A',3))
```

11. Écrire une fonction `fusion` qui prend en paramètres deux objets de la classe `arbreEV` et qui renvoie un nouvel objet de la classe `arbreEV` construit ainsi :

- il n'a pas d'étiquette (`None`) ;
- sa valeur est la somme des valeurs des deux arbres passés en paramètres ;
- son fils gauche est l'arbre passé en paramètre de plus grande valeur ;
- son fils droit est l'arbre passé en paramètre de plus petite valeur.

Exemple :

```
arbre1=arbreEV('E',5)
arbre2=arbreEV('Z',1)
```

la fusion de `arbre1` et `arbre2` donne l'arbre suivant :

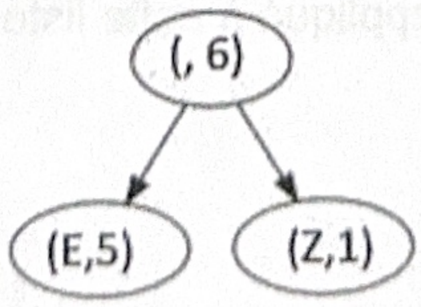


Figure 1. Représentation de la fusion de `arbe1` et `arbre2`

Partie D

Pour obtenir l'arbre de Huffman, on va créer pour chaque élément de notre dictionnaire (couple symbole, nombre d'apparition) un objet de la classe `arbreEV` réduit à une racine (dont les deux fils sont `None`), contenant le symbole (`etiquette`) et son nombre d'apparitions (`valeur`). Les 8 arbres racines correspondants à notre exemple sont donnés en Figure 3.

On applique alors l'algorithme suivant :

- S'il n'y a qu'un seul arbre alors c'est l'arbre de Huffman ;
- Sinon, fusionner 2 des arbres de plus petites valeurs pour obtenir une liste avec un arbre de moins, et renvoyer le résultat de l'application de l'algorithme sur la liste ainsi obtenue.

12. Expliquer pourquoi il s'agit d'un algorithme récursif.

Dans la suite, on appelle **arbre racine** un arbre sans fils. Par ailleurs, pour ces arbres, on ne représente pas l'absence de fils. Ainsi l'arbre :

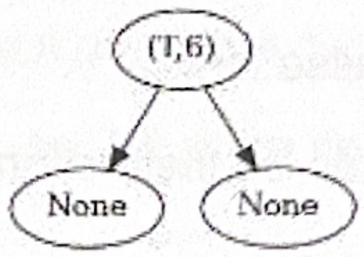


Figure 2. Représentation d'un arbre racine

est plus simplement représenté par :

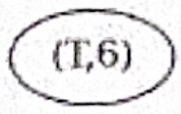


Figure 3. Nouvelle représentation d'un arbre racine

Dans le cas de la phrase « TA TARTE EST TRES TENDRE », l'algorithme est appelé avec la liste d'arbres racines suivante :

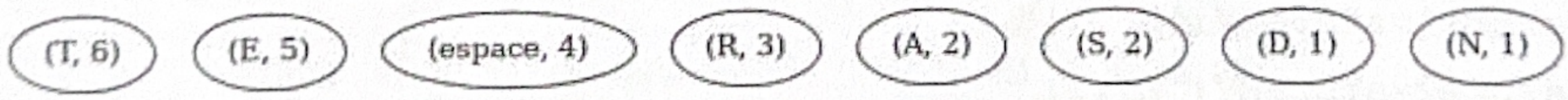


Figure 4. Représentation de la liste des arbres racines

L'arbre de l'algorithme ci-avant appliqué à cette liste passe alors par les étapes données ci-après.

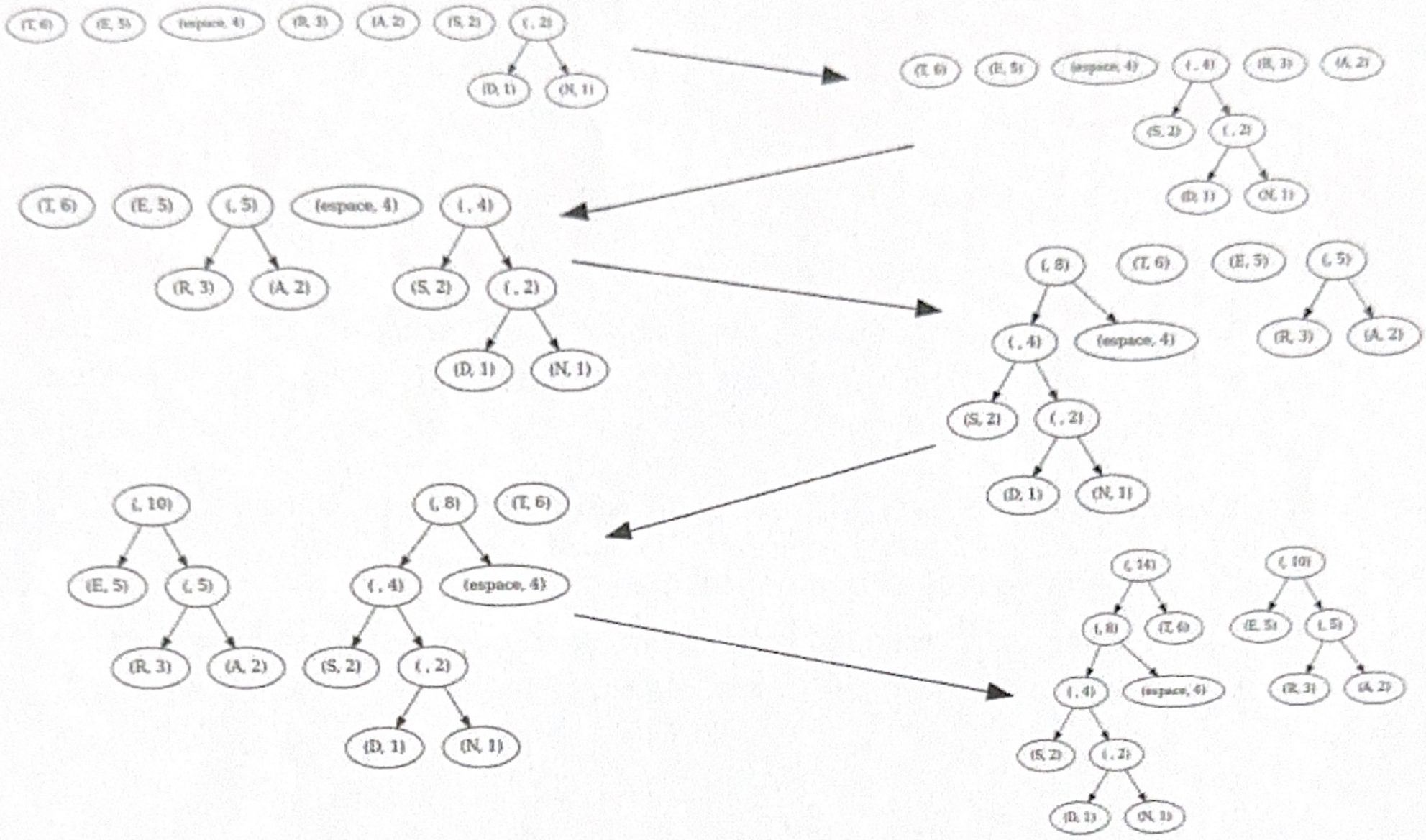


Figure 5. Étape de la construction de l'arbre de l'algorithme associé à la liste

Et donne au final le résultat suivant, appelé arbre de Huffman.

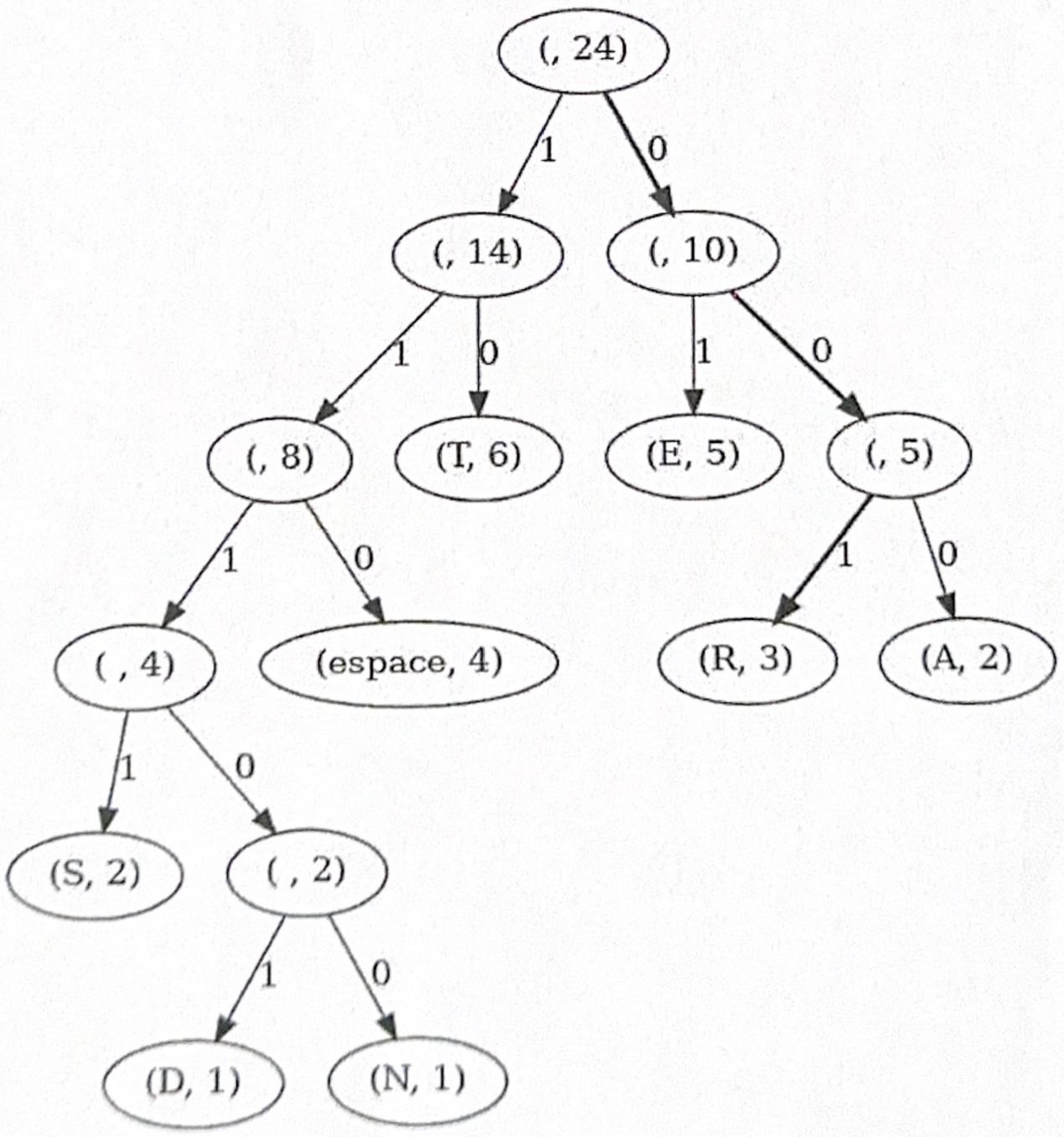


Figure 6. Arbre de Huffman associé

Pour obtenir le code associé à chaque symbole, il ne reste plus qu'à lire les valeurs (0 ou 1) associées aux branches de l'arbre (1 sur les fils gauche, 0 sur les fils droits) menant de la racine de l'arbre au symbole recherché.

Par exemple, pour le symbole R on lit respectivement 0-0-1 soit le code `001`.

13. Donner la taille de l'arbre de Huffman de la Figure 6.
14. Indiquer le type de parcours sur les étiquettes de l'arbre de Huffman qui permet de récupérer les symboles du plus fréquent au moins fréquent dans le texte (en omettant les `None`).

Pour la question suivante, on considère qu'un arbre racine vide est de hauteur 0.

15. Donner la hauteur de l'arbre de Huffman de la Figure 6.
16. Expliquer à quoi correspond la hauteur d'un arbre de Huffman dans le cas d'un texte à coder avec la méthode de Huffman.