BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2024

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Sujet 0.A

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 14 pages numérotées de 1 / 14 à 14 / 14.

Le sujet est composé de trois exercices indépendants.

EXERCICE 1 (6 points)

Cet exercice porte sur l'architecture matérielle, les réseaux, les routeurs et les protocoles de routage.

On considère un réseau local N1 constitué de trois ordinateurs M1, M2, M3 et dont les adresses IP sont les suivantes :

- M1 : 192.168.1.1/24 ;
- M2 : 192.168.1.2/24 ;
- M3 : 192.168.2.3/24.

On rappelle que le "/24" situé à la suite de l'adresse IP de M1 signifie que l'adresse réseau du réseau local N1 est 192.168.1.0.

Depuis l'ordinateur M1, un utilisateur exécute la commande `ping` vers l'ordinateur M3 comme suit :

```
util@M1 ~ % ping 192.168.2.3

PING 192.168.2.3 (192.168.2.3): 56 data bytes
Hôte inaccessible
```

1. Expliquer le résultat obtenu lors de l'utilisation de la commande `ping` (on part du principe que la connexion physique entre les machines est fonctionnelle).

On ajoute un routeur R1 au réseau N1 :

"Un routeur moderne se présente comme un boîtier regroupant carte mère, microprocesseur, ROM, RAM ainsi que les ressources réseaux nécessaires (Wi-Fi, Ethernet…). On peut donc le voir comme un ordinateur minimal dédié, dont le système d'exploitation peut être un Linux allégé. De même, tout ordinateur disposant des interfaces adéquates (au minimum deux, souvent Ethernet) peut faire office de routeur s'il est correctement configuré (certaines distributions Linux minimales spécialisent la machine dans cette fonction)."

Source : Wikipédia, article "Routeur"

2. Définir l'acronyme RAM.
3. Expliquer le terme Linux.
4. Expliquer pourquoi il est nécessaire d'avoir "au minimum deux" interfaces réseau dans un routeur.

Le réseau N1 est maintenant relié à d'autres réseaux locaux (N2, N3, N4) par l'intermédiaire d'une série de routeurs (R1, R2, R3, R4, R5, R6) :

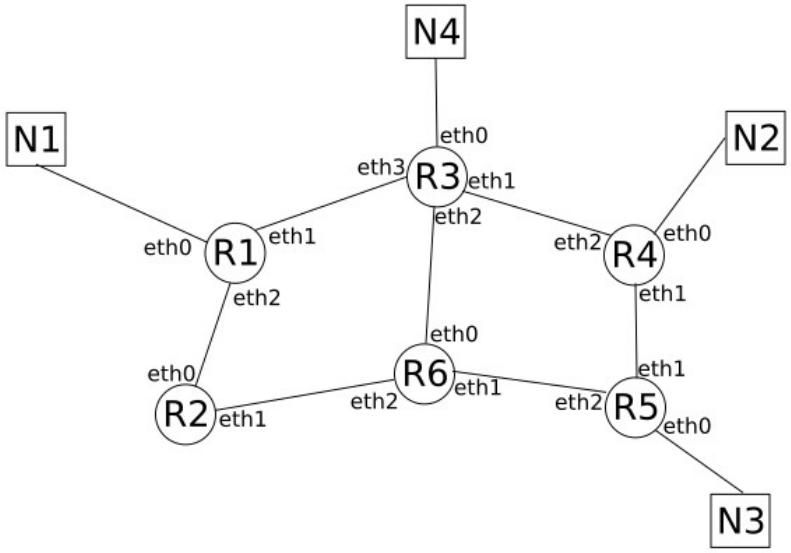

Figure 1. Schéma du réseau

5. Attribuer une adresse IP valide à l'interface eth0 du routeur R1 sachant que l'adresse réseau du réseau N1 est 192.168.1.0.

Dans un premier temps, on utilise le protocole de routage RIP (Routing Information Protocol). On rappelle que dans ce protocole, la métrique de la table de routage correspond au nombre de routeurs à traverser pour atteindre la destination.

La table de routage du routeur R1 est donnée dans le tableau suivant :

Table de routage du routeur R1		
destination	interface de sortie	métrique
N1	eth0	0
N2	eth1	2
N2	eth2	4
N3	eth1	3
N3	eth2	3
N4	eth1	1
N4	eth2	3

6. Déterminer le chemin parcouru par un paquet de données pour aller d'une machine appartenant au réseau N1 à une machine appartenant au réseau N2.

Le routeur R3 tombe en panne. Après quelques minutes, la table de routage de R1 est modifiée afin de tenir compte de cette panne.

7. Dresser la table de routage du routeur R1 suite à la panne du routeur R3.

Le routeur R3 est de nouveau fonctionnel. Dans la suite de cet exercice, on utilise le protocole de routage OSPF (Open Shortest Path First). On rappelle que dans ce protocole, la métrique de la table de routage correspond à la somme des coûts :

coût = $\frac{10^8}{d}$ (où *d* est la bande passante d'une liaison en bit/s).

Le réseau est constitué de 3 types de liaison de communication :

- Fibre avec un débit de 1 Gbit/s ;
- Fast-Ethernet avec un débit de 100 Mbit/s ;
- Ethernet avec un débit de 10 Mbit/s.

8. Calculer le coût de chacune de ces liaisons de communication.

La table de routage du routeur R1 est donnée dans le tableau suivant :

Table de routage du routeur R1		
destination	interface de sortie	métrique
N1	eth0	0
N2	eth1	10,1
N2	eth2	1,3
N3	eth1	11,3
N3	eth2	0,3
N4	eth1	10
N4	eth2	1,2

D'autre part, le type des différentes liaisons inter-routeurs sont les suivantes :

- R1 - R2 : Fibre ;
- R1 - R3 : Ethernet ;
- R2 - R6 : INCONNU ;
- R3 - R6 : Fast-Ethernet ;
- R3 - R4 : Fibre ;
- R4 - R5 : Fast-Ethernet ;
- R5 - R6 : Fibre.

9. Déduire de la table de routage de R1 et du schéma du réseau le type de la liaison inter-routeur R2 - R6.

Des travaux d'amélioration ont été réalisés sur ce réseau : la liaison inter-routeur R1-R3 est désormais de type Fibre.

10. Modifier la table de routage de R1 en tenant compte de cette amélioration.

On ajoute un réseau local N5 et un routeur R7 au réseau étudié ci-dessus. Le routeur R7 possède trois interfaces réseaux eth0, eth1 et eth2. eth0 est directement relié au réseau local N5. eth1 et eth2 sont reliés à d'autres routeurs (ces liaisons inter-routeur sont de type Fibre).

Les deux tableaux suivants présentent un extrait des tables de routage des routeurs R1 et R3 :

Extrait table de routage du routeur R1		
destination	interface de sortie	métrique
…	…	…
N5	eth1	1,2
N5	eth2	0,2

Extrait table de routage du routeur R3		
destination	interface de sortie	métrique
…	…	…
N5	eth1	1,3
N5	eth2	1,1
N5	eth3	0,3

11. Recopier et compléter le schéma du réseau (Figure. 1) en ajoutant le routeur R7 et le réseau local N5.

EXERCICE 2 (6 points)

Cet exercice porte sur les listes, les dictionnaires et la programmation de base en Python.

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) :

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

```
1  def renote_express2(copcorr) :
2      gauche = 0
3      droite = len(copcorr)
4      while droite - gauche > 1 :
5          milieu = (gauche + droite)//2
6          if copcorr[milieu] :
7              ...
8          else :
9              ...
10      if copcorr[gauche] :
11         return ...
12     else :
13         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 3 (8 points)

Cet exercice porte sur les graphes, les algorithmes sur les graphes, les bases de données et les requêtes SQL.

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 :

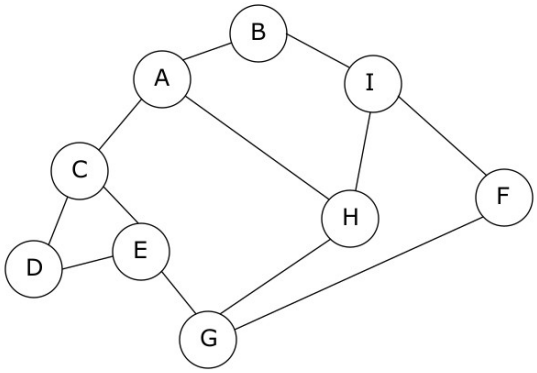

Figure 1. Graphe G2

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.

Dans un premier temps, la société a pour projet de stocker toutes ces données dans un fichier texte. Mais, après réflexion, les développeurs optent pour l'utilisation d'une base de données relationnelle.

10. Expliquer en quoi le choix d'utiliser un système de gestion de base de données (SGBD) est plus pertinent que l'utilisation d'un simple fichier texte.

On donne les deux tables suivantes :

Table `ville`				
`id`	`nom`	`num_dep`	`nombre_hab`	`superficie`
1	Annecy	74	125 694	67
2	Tours	37	136 252	34.4
3	Lyon	69	513 275	47.9
4	Chamonix	74	8 906	246
5	Rennes	35	215 366	50.4
6	Nice	06	342 522	72
7	Bordeaux	33	249 712	49.4

Table `sport`				
`id`	`nom`	`type`	`note`	`id_ville`
1	Richard Bozon	piscine	9	4
2	Bignon	terrain multisport	7	5
3	Ballons perdus	terrain multisport	6	1
4	Mortier	piscine	8	2
5	Block'Out	mur d'escalade	8	2
6	Trabets	mur d'escalade	7	4
7	Centre aquatique du lac	piscine	9	2

Dans la table `ville`, on peut trouver les informations suivantes :

- l'identifiant de la ville (`id`) : chaque ville possède un id unique ;
- le nom de la ville (`nom`) ;
- le numéro du département où se situe la ville (`num_dep`) ;
- le nombre d'habitants (`nombre_hab`) ;
- la superficie de la ville en km² (`superficie`).

Dans la table sport, on peut trouver les informations suivantes :

- l'identifiant de l'infrastructure (`id`) : chaque infrastructure a un id unique ;
- le nom de l'infrastructure (`nom`) ;
- le type d'infrastructure (`type`) ;
- la note sur 10 attribuée à l'infrastructure (`note`) ;
- l'identifiant de la ville où se situe l'infrastructure (`id_ville`).

En lisant ces deux tables, on peut, par exemple, constater qu'il existe une piscine Richard Bozon à Chamonix.

11. Donner le schéma relationnel de la table `ville`.
12. Expliquer le rôle de l'attribut `id_ville` dans la table `sport`.
13. Donner le résultat de la requête SQL suivante :

```
SELECT nom
FROM ville
WHERE num_dep = 74 AND superficie > 70
```

14. Écrire une requête SQL permettant de lister les noms de l'ensemble des piscines présentes dans la table `sport`.

Suite à de bons retours d'utilisateurs, la note du terrain multisport "Ballon perdus" est augmentée d'un point (elle passe de 6 à 7).

15. Écrire une requête SQL permettant de modifier la note du terrain multisport "Ballon perdus" de 6 à 7.
16. Écrire une requête SQL permettant d'ajouter la ville de Toulouse dans la table `ville`. Cette ville est située dans le département de la Haute-Garonne (31). Elle a une superficie de 118 km². En 2023, Toulouse comptait 471941 habitants. Cette ville aura l'identifiant 8.
17. Écrire une requête SQL permettant de lister les noms des murs d'escalade disponibles à Annecy.