BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2024 : Bac blanc

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.
Ce sujet comporte 11 pages numérotées de 1/10 à 11/10.

EXERCICE 1 (4 points)

Cet exercice porte sur les notions de routage et de systèmes sur puces.

Un constructeur automobile utilise des ordinateurs pour la conception de ses véhicules.

Ceux-ci sont munis d'un système d'exploitation ainsi que de nombreuses applications parmi lesquelles on peut citer :

- un logiciel de traitement de texte ;
- un tableur ;
- un logiciel de Conception Assistée par Ordinateur (CAO) ;
- un système de gestion de base de données (SGBD).

Chaque ordinateur est équipé des périphériques classiques : clavier, souris, écran et est relié à une imprimante réseau.

Ce constructeur automobile intègre à ses véhicules des systèmes embarqués, comme par exemple un système de guidage par satellites (GPS), un système de freinage antiblocage (ABS) ...

Ces dispositifs utilisent des systèmes sur puces (SoC : Système on a Chip).

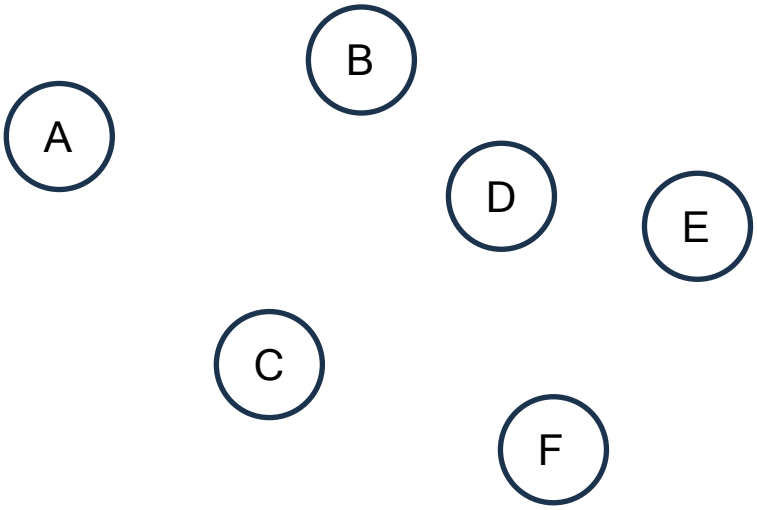
1. Citer deux avantages à utiliser ces systèmes sur puces plutôt qu'une architecture classique

Ce constructeur automobile possède six sites de production qui échangent des documents entre eux. Les sites de production sont reliés entre eux par six routeurs A , B , C, D, E et F

On donne ci-dessous les tables de routage des routeurs A à F obtenus avec le protocole RIP :

Routeur A		Routeur B		Routeur C		Routeur D		Routeur E		Routeur F	
Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass
B	B	A	A	A	A	A	C	A	B	A	D
C	C	C	C	B	B	B	C	B	B	B	E
D	C	D	C	D	D	C	C	C	B	C	D
E	B	E	E	E	B	E	E	D	D	D	D
F	B	F	E	F	D	F	F	F	F	E	E

2. Déterminer à l'aide de ces tables le chemin emprunté par un paquet de données envoyé durouteur A vers le routeur F.
3. On veut représenter schématiquement le réseau de routeurs à partir des tables de routage.Recopier sur la copie le schéma ci-dessous :



En s'appuyant sur les tables de routage, tracer les liaisons entre les routeurs.

EXERCICE 2 (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 :

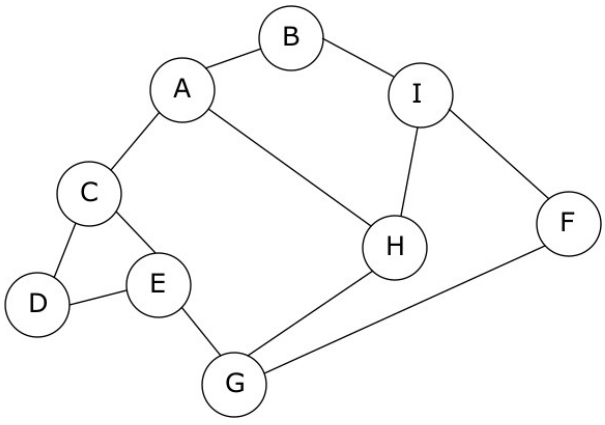


Figure 1. Graphe G2

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

Proposer une implémentation en Python du graphe G2 à l'aide d'un dictionnaire.

4. 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.

5. Expliquer pourquoi la fonction `cherche_itineraires` peut être qualifiée defonction récursive.
6. Expliquer le rôle de la fonction `cherche_itineraires` dans le programme p1.
7. 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']]
```

8. Donner une explication au problème décrit ci-dessus. Vous pourrez vousappuyer 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 dansun fichier texte.

Mais, après réflexion, les développeurs optent pour l'utilisation d'unebase de données relationnelle.

9. 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	125694	67
2	Tours	37	136252	34.4
3	Lyon	69	513275	47.9
4	Chamonix	74	8906	246
5	Rennes	35	215366	50.4
6	Nice	06	342522	72
7	Bordeaux	33	249712	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.

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

```
SELECT nom

FROM ville

WHERE num_dep = 74 AND superficie > 70
```

13. Écrire une requête SQL permettant de lister les noms de l'ensemble despiscines présentes dans la table `sport`.

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

14. Écrire une requête SQL permettant de modifier la note du terrain multisport"Ballon perdus" de 6 à 7.
15. Écrire une requête SQL permettant d'ajouter la ville de Toulouse dans la tableville. Cette ville est située dans le département de la Haute-Garonne (31). Elle a une superficie de 118 km^2. En 2023, Toulouse comptait 471941 habitants. Cette ville aura l'identifiant 8.
16. Écrire une requête SQL permettant de lister les noms des murs d'escaladedisponibles à Annecy.

EXERCICE 3 (8 points)

Cet exercice traite de manipulation de tableaux, de récursivité et du paradigme « diviser pour régner ».

Dans un tableau Python d'entiers tab, on dit que le couple d'indices (i, j) forme une inversion lorsque i < j et tab[i] > tab[j]. On donne ci-dessous quelques exemples.

- Dans le tableau `[1, 5, 3, 7]`, le couple d'indices $(1,2)$ forme une inversion car $5 > 3$.Par contre, le couple $(1,3)$ ne forme pas d'inversion car $5 < 7$.

  Il n'y a qu'une inversion dans ce tableau.

- Il y a trois inversions dans le tableau `[1, 6, 2, 7, 3]`,à savoir les couples d'indices $(1,2)$, $(1, 4)$ et $(3,4)$.
- On peut compter six inversions dans le tableau `[7, 6, 5, 3]` : les couples d'indices $(0,1)$, $(0, 2)$, $(0,3)$, $(1,2)$, $(1,3)$ et $(2,3)$.

On se propose dans cet exercice de déterminer le nombre d'inversions dans un tableau quelconque.

Questions préliminaires

1. Expliquer pourquoi le couple $(1,3)$ est une inversion dans le tableau `[4, 8, 3, 7]`.
2. Justifier que le couple $(2,3)$ n'en est pas une.

Partie A : Méthode itérative

Le but de cette partie est d'écrire une fonction itérative `nombre_inversion` qui renvoie lenombre d'inversions dans un tableau. Pour cela, on commence par écrire une fonction `fonction1` qui sera ensuite utilisée pour écrire la fonction `nombre_inversion`.

1. On donne la fonction suivante.

```
def  fonction1(tab,  i):
     nb_elem  =  len(tab)
     cpt = 0

     for j in range(i+1, nb_elem):

          if tab[j] < tab[i]:

               cpt += 1
     return cpt
```

a. Indiquer ce que renvoie la `fonction1(tab, i)` dans les cas suivants.

- Cas n°1 : `tab = [1, 5, 3, 7]` et `i = 0`.
- Cas n°2 : `tab = [1, 5, 3, 7]` et `i = 1`.
- Cas n°3 : `tab = [1, 5, 2, 6, 4]` et `i = 1`.

b. Expliquer ce que permet de déterminer cette fonction.

2. En utilisant la fonction précédente, écrire une fonction `nombre_inversion(tab)` quiprend en argument un tableau et renvoie le nombre d'inversions dans ce tableau.

On donne ci-dessous les résultats attendus pour certains appels.

```
>>> nombre_inversions([1, 5, 7])
0
>>> nombre_inversions([1, 6, 2, 7, 3])
3
>>> nombre_inversions([7, 6, 5, 3])
6
```

3. Quelle est l'ordre de grandeur de la complexité en temps de l'algorithme obtenu ? Aucune justification n'est attendue.

Partie B : Méthode récursive

Le but de cette partie est de concevoir une version récursive de la fonction `nombre_inversions`.

On définit pour cela des fonctions auxiliaires.

1. Donner le nom d'un algorithme de tri ayant une complexité meilleure que quadratique.

Dans la suite de cet exercice, on suppose qu'on dispose d'une fonction `tri(tab)` qui prend en argument un tableau et renvoie un tableau contenant les mêmes éléments rangés dans l'ordre croissant.

2. Écrire une fonction `moitie_gauche(tab)` qui prend en argument un tableau tab et renvoie un nouveau tableau contenant la moitié gauche de `tab`. Si le nombre d'éléments de tab est impair, l'élément du centre se trouve dans cette partie gauche.

On donne ci-dessous les résultats attendus pour certains appels.

```
>>> moitie_gauche([])
[]
>>> moitie_gauche([4, 8, 3])
```

```
[4, 8]
>>> moitie_gauche ([4, 8, 3, 7])
[4, 8]
```

Dans la suite, on suppose qu'on dispose de la fonction `moitie_droite(tab)` qui renvoie la moitié droite sans l'élément du milieu.

3. On suppose qu'une fonction `nb_inv_tab(tab1, tab2)` a été écrite. Cette fonction renvoie le nombre d'inversions du tableau obtenu en mettant bout à bout les tableaux tab1 et `tab2`, à condition que `tab1` et `tab2` soient triés dans l'ordre croissant.

On donne ci-dessous deux exemples d'appel de cette fonction :

```
>>> nb_inv_tab([3, 7, 9], [2, 10])
3
>>> nb_inv_tab([7, 9, 13], [7, 10, 14])
3
```

En utilisant la fonction `nb_inv_tab` et les questions précédentes, écrire une fonction récursive `nb_inversions_rec(tab)` qui permet de calculer le nombre d'inversions dans un tableau. Cette fonction renverra le même nombre que `nombre_inversions(tab)` de la partie A.

On procédera de la façon suivante :

- Séparer le tableau en deux tableaux de tailles égales (à une unité près).
- Appeler récursivement la fonction `nb_inversions_rec` pour compter le nombre d'inversions dans chacun des deux tableaux.
- Trier les deux tableaux (on rappelle qu'une fonction de tri est déjà définie).
- Ajouter au nombre d'inversions précédemment comptées le nombre renvoyé par la fonction `nb_inv_tab` avec pour arguments les deux tableaux triés.

CORRECTION

EXERCICE 1 (4 points)

Cet exercice porte sur les notions de routage et de systèmes sur puces.

Un constructeur automobile utilise des ordinateurs pour la conception de ses véhicules.

Ceux-ci sont munis d'un système d'exploitation ainsi que de nombreuses applications parmi lesquelles on peut citer :

- un logiciel de traitement de texte ;
- un tableur ;
- un logiciel de Conception Assistée par Ordinateur (CAO) ;
- un système de gestion de base de données (SGBD).

Chaque ordinateur est équipé des périphériques classiques : clavier, souris, écran et est relié à une imprimante réseau.

Ce constructeur automobile intègre à ses véhicules des systèmes embarqués, comme par exemple un système de guidage par satellites (GPS), un système de freinage antiblocage (ABS) ...

Ces dispositifs utilisent des systèmes sur puces (SoC : Système on a Chip).

1. Citer deux avantages à utiliser ces systèmes sur puces plutôt qu'une architecture classique.

L'encombrement et la consommation sont plus faibles.

Ce constructeur automobile possède six sites de production qui échangent des documents entre eux. Les sites de production sont reliés entre eux par six routeurs A , B , C, D, E et F

On donne ci-dessous les tables de routage des routeurs A à F obtenus avec le protocole RIP :

Routeur A		Routeur B		Routeur C		Routeur D		Routeur E		Routeur F	
Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass
B	B	A	A	A	A	A	C	A	B	A	D
C	C	C	C	B	B	B	C	B	B	B	E
D	C	D	C	D	D	C	C	C	B	C	D
E	B	E	E	E	B	E	E	D	D	D	D
F	B	F	E	F	D	F	F	F	F	E	E

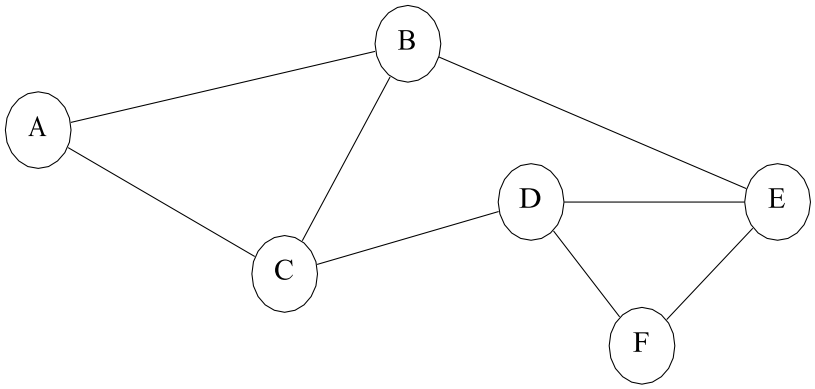
CORRECTION

2. Déterminer à l'aide de ces tables le chemin emprunté par un paquet de données envoyé durouteur A vers le routeur F.

A-B-E-F

3. On veut représenter schématiquement le réseau de routeurs à partir des tables de routage.Recopier sur la copie le schéma ci-dessous :

0En s'appuyant sur les tables de routage, tracer les liaisons entre les routeurs.



CORRECTION

EXERCICE 2 (8 points)

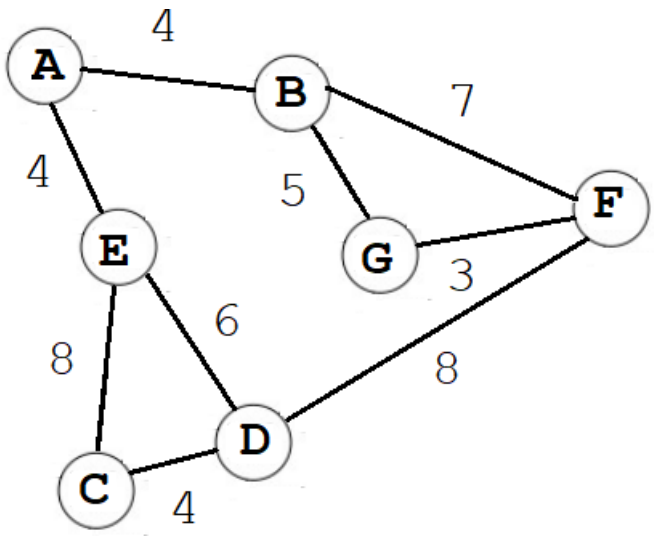
Cet exercice porte sur les graphes, les algorithmes sur les graphes, les bases de données et les requêtes SQL.

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.

A → E → D

CORRECTION

3. Définir la matrice d'adjacence du graphe G1 (en prenant les sommets dans l'ordre alphabétique).

0	4	0	0	4	0	0
4	0	0	0	0	7	5
0	0	0	4	8	0	0
0	0	4	0	6	8	0
4	0	8	6	0	0	0
0	7	0	8	0	0	3
0	5	0	0	0	3	0

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 :

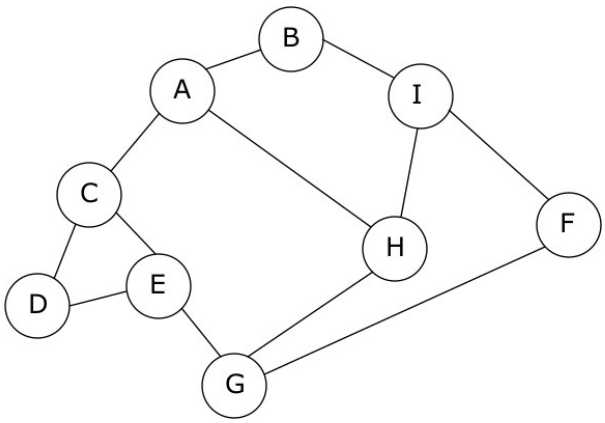


Figure 1. Graphe G2

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

CORRECTION

Proposer une implémentation en Python du graphe G2 à l'aide d'un dictionnaire.

```
G2 = {
"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", "I", "G"],
"I":["B", "F", "H"]
}
```

4. Proposer un parcours en largeur du graphe G2 en partant de A.

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

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) :

```
12 tab_itineraires=[]
13 def cherche_itineraires(G, start, end, chaine=[]):
14         chaine = chaine + [start]
15         if start == end:
16             return chaine
17         for u in G[start]:
18             if u not in chaine:
19                 nchemin = cherche_itineraires(G, u, end, chaine)
20                 if len(nchemin) != 0:
21                     tab_itineraires.append(nchemin)
22         return []
23
18 def itineraires_court(G,dep,arr):
19     cherche_itineraires(G, dep, arr)
20     tab_court = ...
21     mini = float('inf')
22     for v in tab_itineraires:
18         if len(v) <= ... :
24             mini = ...
25     for v in tab_itineraires:
26         if len(v) == mini:
27             tab_court.append(...)
28     return tab_court
```

CORRECTION

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.

5. Expliquer pourquoi la fonction `cherche_itineraires` peut être qualifiée de fonction récursive.

La fonction `cherche_itineraires est récursive` car elle s'appelle elle-même à la ligne 19.

6. Expliquer le rôle de la fonction `cherche_itineraires` dans le programme p1.

Cette fonction permet de déterminer tous les chemins possibles pour se rendre d'un point à un autre.

7. Compléter la fonction `itineraires_court`.

```
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
```

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

CORRECTION

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']]
```

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

   La variable globale tab_itineraires doit être réinitialisée entre chaque appel.

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 dansun fichier texte.

Mais, après réflexion, les développeurs optent pour l'utilisation d'unebase de données relationnelle.

9. 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.

   Cela évite de dupliquer les données sur les villes pour chaque infrastructure (gain de place + facilité de mise à jour)

CORRECTION

On donne les deux tables suivantes :

Table `ville`				
`id`	`nom`	`num_dep`	`nombre_hab`	`superficie`
1	Annecy	74	125694	67
2	Tours	37	136252	34.4
3	Lyon	69	513275	47.9
4	Chamonix	74	8906	246
5	Rennes	35	215366	50.4
6	Nice	06	342522	72
7	Bordeaux	33	249712	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`).

CORRECTION

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

10. Donner le schéma relationnel de la table `ville`.

Ville(id : entier, nom : texte, num_dep : entier, nombre_hab : entier, superficie : entier)

11. Expliquer le rôle de l'attribut `id_ville` dans la table `sport`.

id_ville est une clef étrangère dans la table sport en relation avec la clef primaire id de la table ville.

12. Donner le résultat de la requête SQL suivante :

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

Chamonix

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

```
SELECT nom
FROM sport
WHERE type = 'piscine'
```

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).

14. Écrire une requête SQL permettant de modifier la note du terrain multisport"Ballon perdus" de 6 à 7.

```
UPDATE sport
SET note = 7
WHERE id = 3
```

15. 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 km^2. En 2023, Toulouse comptait 471941 habitants. Cette ville aura l'identifiant 8.

```
INSERT INTO ville
VALUES (8, "Toulouse", 31, 471941, 118)
```

CORRECTION

16. Écrire une requête SQL permettant de lister les noms des murs d'escalade disponibles à Annecy

```
SELECT sport.nom
FROM sport, ville
WHERE ville.nom = "Annecy"
AND sport.type = "mur d'escalade" AND
sport.id_ville = ville.id0,5
```

EXERCICE 3 (8 points)

Cet exercice traite de manipulation de tableaux, de récursivité et du paradigme « diviser pour régner ».

Dans un tableau Python d'entiers tab, on dit que le couple d'indices (i, j) forme une inversion lorsque i < j et tab[i] > tab[j]. On donne ci-dessous quelques exemples.

- Dans le tableau `[1, 5, 3, 7]`, le couple d'indices $(1,2)$ forme une inversion car $5 > 3$.Par contre, le couple $(1,3)$ ne forme pas d'inversion car $5 < 7$.

  Il n'y a qu'une inversion dans ce tableau.

- Il y a trois inversions dans le tableau `[1, 6, 2, 7, 3]`,à savoir les couples d'indices $(1,2)$, $(1, 4)$ et $(3,4)$.
- On peut compter six inversions dans le tableau `[7, 6, 5, 3]` : les couples d'indices $(0,1)$, $(0, 2)$, $(0,3)$, $(1,2)$, $(1,3)$ et $(2,3)$.

On se propose dans cet exercice de déterminer le nombre d'inversions dans un tableau quelconque.

Questions préliminaires

1. Expliquer pourquoi le couple $(1,3)$ est une inversion dans le tableau `[4, 8, 3, 7]`.

   Car 8 > 7

2. Justifier que le couple $(2,3)$ n'en est pas une.

   Car 3 < 7

CORRECTION

Partie A : Méthode itérative

Le but de cette partie est d'écrire une fonction itérative `nombre_inversion` qui renvoie lenombre d'inversions dans un tableau. Pour cela, on commence par écrire une fonction `fonction1` qui sera ensuite utilisée pour écrire la fonction `nombre_inversion`.

3. On donne la fonction suivante.

```
def  fonction1(tab,  i):
    nb_elem  =  len(tab)
    cpt = 0
    for j in range(i+1, nb_elem):
        if tab[j] < tab[i]:
            cpt += 1
    return cpt
```

a. Indiquer ce que renvoie la `fonction1(tab, i)` dans les cas suivants.

i. Cas n°1 : `tab = [1, 5, 3, 7]` et `i = 0`.

La fonction retourne 0

ii. Cas n°2 : `tab = [1, 5, 3, 7]` et `i = 1`.

La fonction retourne 1

iii. Cas n°3 : `tab = [1, 5, 2, 6, 4]` et `i = 1`.

La fonction retourne 2

b. Expliquer ce que permet de déterminer cette fonction.

La fonction fonction1 calcule le nombre d'inversions (i,j) avec i fixé et j > i.

4. En utilisant la fonction précédente, écrire une fonction `nombre_inversions(tab)` qui prend en argument un tableau et renvoie le nombre d'inversions dans ce tableau.

On donne ci-dessous les résultats attendus pour certains appels.

```
>>>  nombre_inversions([1, 5, 7])
0
```

CORRECTION

```
>>>  nombre_inversions([1, 6, 2, 7, 3])
3
>>>  nombre_inversions([7, 6, 5, 3])
6
```

```
def nombre_inversions(tab):
    nb_inversions = 0
    for i in range(len(tab)):
        nb_inversions += fonction1(tab,i)
    return nb_inversions
```

5. Quelle est l'ordre de grandeur de la complexité en temps de l'algorithme obtenu ? Aucune justification n'est attendue.

O(n²) avec n = taille du tableau

Partie B : Méthode récursive

Le but de cette partie est de concevoir une version récursive de la fonction `nombre_inversions`.

On définit pour cela des fonctions auxiliaires.

2. Donner le nom d'un algorithme de tri ayant une complexité meilleure que quadratique.

L'algorithme de Tri-fusion a une complexité en n x log(n) < n².

Dans la suite de cet exercice, on suppose qu'on dispose d'une fonction `tri(tab)` qui prend en argument un tableau et renvoie un tableau contenant les mêmes éléments rangés dans l'ordre croissant.

3. Écrire une fonction `moitie_gauche(tab)` qui prend en argument un tableau tab et renvoie un nouveau tableau contenant la moitié gauche de `tab`. Si le nombre d'éléments de tab est impair, l'élément du centre se trouve dans cette partie gauche.

On donne ci-dessous les résultats attendus pour certains appels.

```
>>> moitie_gauche([])
[]
>>> moitie_gauche([4, 8, 3])
[4, 8]
```

CORRECTION

```
>>> moitie_gauche ([4, 8, 3, 7])
[4, 8]
```

```
def moitie_gauche(tab):
    return tab[:len(tab)//2]
```

Dans la suite, on suppose qu'on dispose de la fonction `moitie_droite(tab)` qui renvoie la moitié droite sans l'élément du milieu.

4. On suppose qu'une fonction `nb_inv_tab(tab1, tab2)` a été écrite. Cette fonction renvoie le nombre d'inversions du tableau obtenu en mettant bout à bout les tableaux tab1 et `tab2`, à condition que `tab1` et `tab2` soient triés dans l'ordre croissant.

   On donne ci-dessous deux exemples d'appel de cette fonction :

```
>>> nb_inv_tab([3, 7, 9], [2, 10])
3
>>> nb_inv_tab([7, 9, 13], [7, 10, 14])
3
```

   En utilisant la fonction `nb_inv_tab` et les questions précédentes, écrire une fonction récursive `nb_inversions_rec(tab)` qui permet de calculer le nombre d'inversions dans un tableau. Cette fonction renverra le même nombre que `nombre_inversions(tab)` de la partie A.

   On procédera de la façon suivante :

   - Séparer le tableau en deux tableaux de tailles égales (à une unité près).
   - Appeler récursivement la fonction `nb_inversions_rec` pour compter le nombre d'inversions dans chacun des deux tableaux.
   - Trier les deux tableaux (on rappelle qu'une fonction de tri est déjà définie).
   - Ajouter au nombre d'inversions précédemment comptées le nombre renvoyé par la fonction `nb_inv_tab` avec pour arguments les deux tableaux triés.

CORRECTION

```
def nb_inversions_rec(tab):
    if len(tab) <= 1:
        return 0
    else:
        tab_g = moitie_gauche(tab)
        tab_d = moitie_droite(tab)
        nb = nb_inversions_rec(tab_g) + nb_inversions_rec(tab_d)
        tab_g_trie = sorted(tab_g)
        tab_d_trie = sorted(tab_d)
        nb = nb + nb_inv_tab(tab_g_trie, tab_d_trie)
    return nb
```