BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

Mars 2024 – Bac Blanc

N.S.I.
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 13 pages numérotées de 1 à 13

Exercice 1 : *(6 points)*

Cet exercice porte sur les arbres binaires de recherche, la programmation orientée objet et la récursivité.

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 :

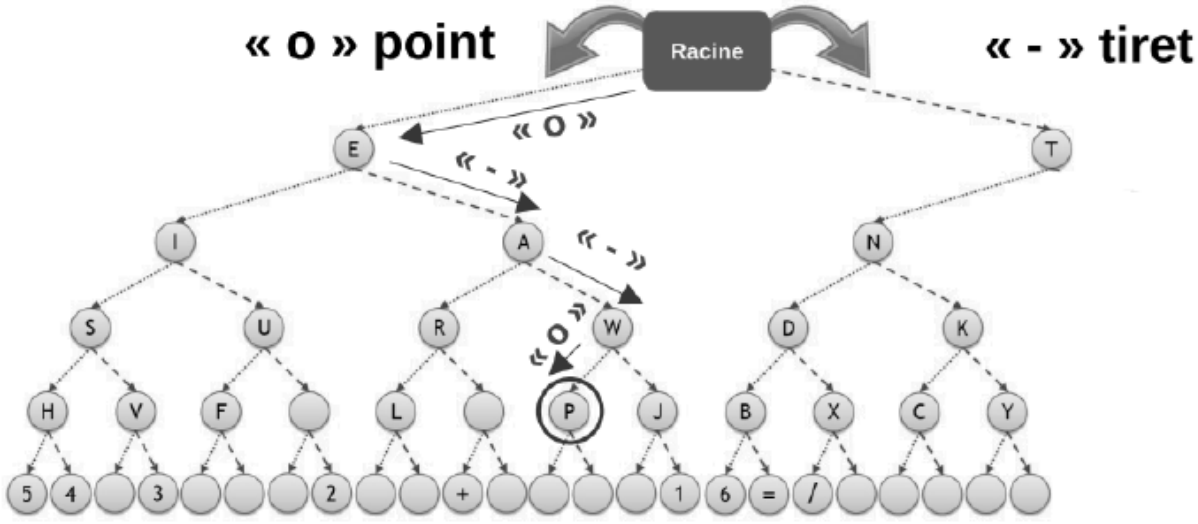


Figure 1 : Extrait de l'arbre binaire du code Morse

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 :** `--oo`

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|
```

Exercice 2 : *(7 points)*

Cet exercice porte sur les bases de données, la représentation des données et les réseaux.

Cet exercice utilise certains des mots-clés du langage SQL suivants : `DELETE, FROM, INSERT, INTO, JOIN, ON, SELECT, SET, UPDATE, VALUES, WHERE.`

Les vacances d'été se rapprochent et le propriétaire d'une pension pour animaux gère les places dont il dispose à l'aide d'une base de données dont voici le schéma relationnel :

`client(num_client, nom_client, prenom_client, mail_client, tel_client)`

`animal(num_animal, nom_animal, categorie_animal, taille_animal, num_client)`

`cage(num_cage, taille_cage, secteur_cage)`

`reservation(num_reservation, date_debut_reservation, date_fin_reservation, num_client, num_animal, num_cage)`

Ci-dessous, on donne des extraits des tables `client`, `animal`, `cage` et `reservation`.

Extrait de la table `client` :

num_client	nom_client	prenom_client	mail_client	tel_client
16	Dupont	Marc	marc.dupont@mail.com	0604050401
345	Morel	Fabien	fabien.morel@mail.com	0700051020

Extrait de la table `animal` :

num_animal	nom_animal	categorie_animal	taille_animal	num_client
22	Yuki	souris	petit	16
112	Balou	chat	moyen	141
320	Api	chien	grand	237
423	Rex	chien	moyen	259
491	Rex	chien	petit	345

Extrait de la table `cage` :

num_cage	taille_cage	secteur_cage
4	grand	chien
12	petit	chien
23	moyen	chien
31	moyen	chien
32	petit	rongeur
33	grand	chat

Extrait de la table `reservation` :

num_reservation	date_debut_reservation	date_fin_reservation	num_client	num_animal	num_cage
44	2022-08-23	2022-08-25	26	12	12
45	2022-07-11	2022-07-22	345	491	23
46	2022-08-11	2022-08-22	345	491	23
47	2022-08-23	2022-09-10	345	491	23
48	2022-10-11	2022-10-22	345	491	23

1. **Étude du schéma relationnel**
   a. Pour chaque attribut de la relation `cage`, spécifier son type, en utilisant le tableau des types suivant :

`CHAR(t)` `VARCHAR(t)`	Texte de longueur fixe de t caractères. Texte de longueur variable de t caractères au maximum.
`INT`	Nombre entier de -2^{31} à $2^{31}-1$ (signé) ou de 0 à $2^{32}-1$ (non signé).
`FLOAT`	Réel à virgule flottante.
`DATE` `DATETIME`	Date format AAAA-MM-JJ. Date et heure format AAAA-MM-JJ HH:MI:SS.

   b. Préciser, pour la relation `reservation`, le nom de la clé primaire pouvant être utilisée.

   c. Indiquer, pour la relation `reservation`, la ou les clés étrangères (ou secondaires) et en indiquer l'utilité.

2. **Requêtes**
   a. Indiquer le résultat de l'exécution de la requête suivante :

```
SELECT nom_animal
FROM animal
WHERE categorie_animal = 'chien';
```

   b. Écrire une requête SQL permettant d'afficher les noms de tous les clients dont l'animal a occupé la cage numéro 23.

   c. Un nouvel animal doit être enregistré dans la base de données qui contient actuellement 491 animaux. Il s'appelle Suki, c'est un chat de petite taille dont le propriétaire a déjà été enregistré sous le numéro 342.
   Écrire la requête SQL permettant d'insérer ces nouvelles données dans la base de données.

3. Programmation Python

Suite à une panne, le responsable de la pension n'a plus accès à sa base de données. Heureusement, il avait fait une sauvegarde de ses tables au format csv. Il les a importées à l'aide d'un programme Python, chacune sous la forme d'une liste de dictionnaires.

Pour simplifier, on considérera que la table `reservation` est la liste de dictionnaires suivante :

```
reservation = [
{'num_reservation' : 44, 'date_debut_reservation' : '2022-08-23',
'date_fin_reservation' : '2022-08-25', 'num_client' : 26,
'num_animal' : 12, 'num_cage' : 12},
{'num_reservation' : 45, 'date_debut_reservation' : '2022-07-11',
'date_fin_reservation' : '2022-07-22', 'num_client' : 345,
'num_animal' : 491, 'num_cage' : 23},
{'num_reservation' : 46, 'date_debut_reservation' : '2022-08-11',
'date_fin_reservation' : '2022-08-22', 'num_client' : 345,
'num_animal' : 491, 'num_cage' : 23},
{'num_reservation' : 47, 'date_debut_reservation' : '2022-08-23',
'date_fin_reservation' : '2022-09-10', 'num_client' : 345,
'num_animal' : 491, 'num_cage' : 23},
{'num_reservation' : 48, 'date_debut_reservation' : '2022-10-11',
'date_fin_reservation' : '2022-10-22', 'num_client' : 345,
'num_animal' : 491, 'num_cage' : 23}]
```

a. On donne ci-dessous, le code Python d'une fonction `mystere`.

Numéro de lignes	Fonction `mystere`
1	`def mystere(table, date):`
2	`    liste = []`
3	`    for ligne in table:`
4	`        if ligne['date_debut_reservation'] == date:`
5	`            liste.append(ligne['num_client'])`
6	`    return liste`

On rappelle que l'appel `L.append(x)` ajoute l'élément `x` à la fin de la liste `L`.

Indiquer l'affichage produit par l'exécution de la ligne de code suivante :

```
print(mystere(reservation, '2022-08-23'))
```

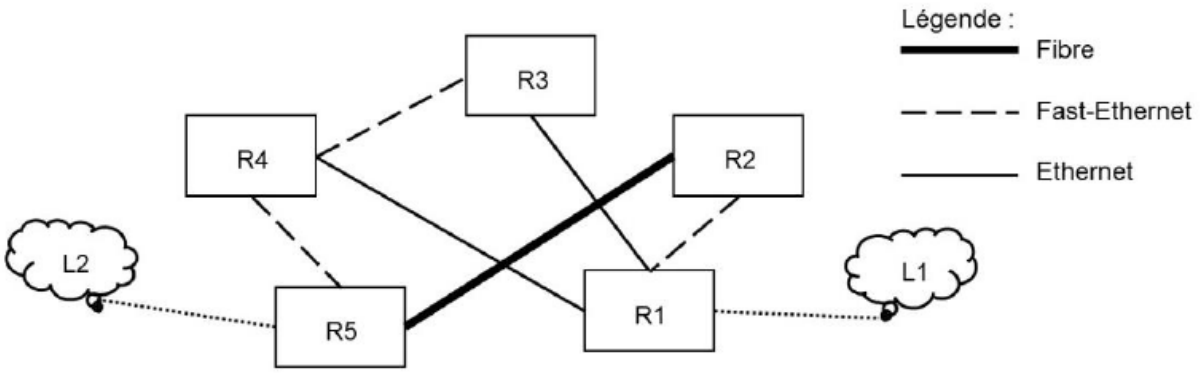
b. Le responsable de la pension veut obtenir le nombre de réservations qui ont été effectuées pour un numéro de client donné.

Écrire les lignes de code après la ligne 7 de la fonction `nombre_reservation` afin de respecter la spécification donnée ci-dessous.

Numéro de lignes	Fonction `nombre_reservation`
1	`def nombre_reservation(table, numero_client):`
2	`    """Paramètres :`
3	`    table : liste de dictionnaires, représentant les réservations`
4	`    numero_client : un entier, représentant le numéro du client`
5	`    concerné`
6	`    Valeur renvoyée : un entier donnant le nombre d'occurrences`
7	`    du numéro du client concerné. """`
…	`    à compléter`

4. Protocole OSPF

La sauvegarde de la base de données est stockée sur le réseau local L1, relié au routeur R1 du réseau suivant.



L'ordinateur de bureau du responsable de la pension fait partie du réseau local L2.

Les réseaux locaux L1 et L2 font partie d'un réseau constitué de 5 routeurs (R1, R2, R3, R4, R5), de liaisons de communication dont les bandes passantes sont de 1 Gbit/s pour la Fibre, 100 Mbit/s pour Fast-Ethernet et 10 Mbit/s pour Ethernet.

On s'intéresse ici au protocole de routage OSPF. Le protocole OSPF cherche à minimiser la somme des coûts des liaisons entre les routeurs empruntés par un paquet de données. Le coût C d'une liaison est donné par :

$$C = \frac{10^8}{d}$$, où d est la bande passante en bit/s de la liaison.

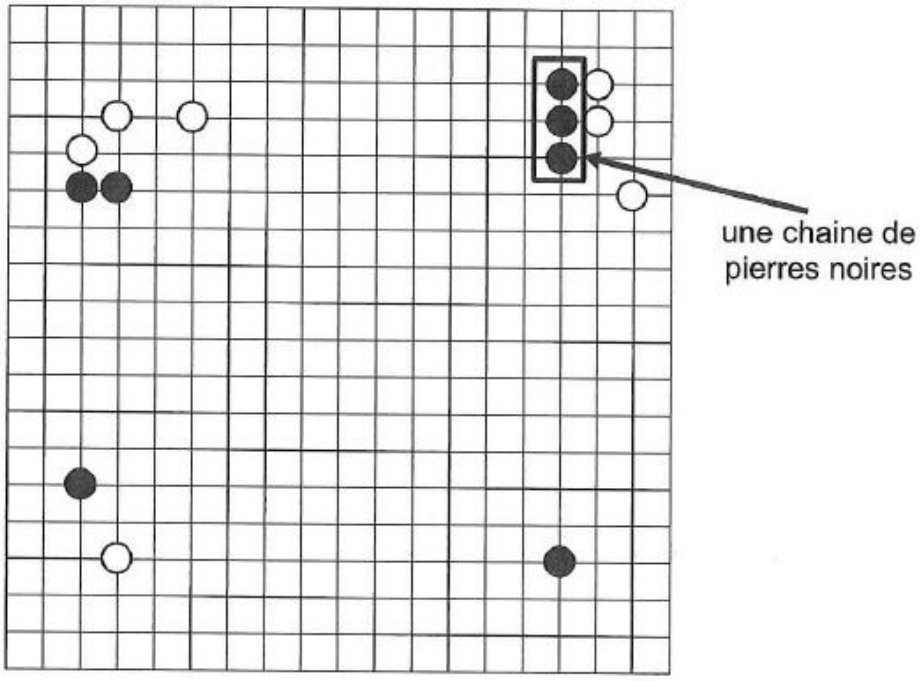
a. Pour passer de L1 à L2, le chemin R1 - R2 - R5 utilisant la fibre a le coût le plus faible. Calculer ce coût.

b. La liaison entre R2 et R5 a été coupée en raison de travaux. Déterminer la route permettant de relier le réseau L1 au réseau L2 selon le protocole OSPF. Justifier.

Exercice 3 : *(7 points)*

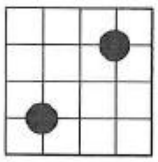
Au jeu de go, les deux adversaires placent à tour de rôle des pierres, respectivement noires et blanches, sur les intersections d'un plateau quadrillé appelé goban. Une partie se joue généralement sur un goban de 19×19 intersections, mais des gobans de 13×13 et 9×9 peuvent être utilisés pour s'entraîner.

On a reproduit ci-dessous un début de partie sur un goban de 19×19.

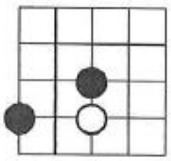


On s'intéresse dans cet exercice à l'une des règles du jeu de go, dite d'encerclement, pour laquelle on doit compter ce qui s'appelle les libertés d'une pierre ou d'une chaine de pierres.

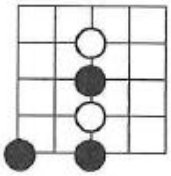
Les libertés d'une pierre sont les intersections libres autour d'elle selon les quatre directions cardinales (nord, sud, est et ouest). Dans chacun des exemples suivants, on donne le nombre de libertés de chacune des pierres noires.



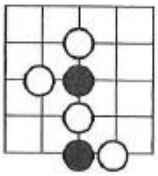
4 libertés



3 libertés



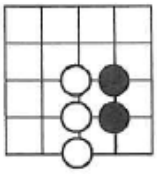
2 libertés



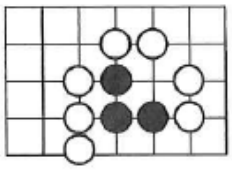
1 liberté

On appelle chaine de pierres une suite de pierres liées entre elles selon les directions cardinales. Le nombre de libertés d'une chaine de pierres s'obtient alors en comptant, sans répétition, les libertés de chacune des pierres qui la composent.

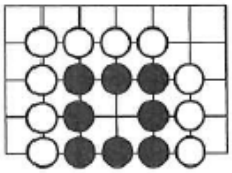
Les exemples ci-dessous mettent en lumière cette règle sur une chaine de pierres.



La chaine de pierres noires a 4 libertés.



La chaine de pierres noires a 3 libertés.



La chaine de pierres noires a une seule liberté.

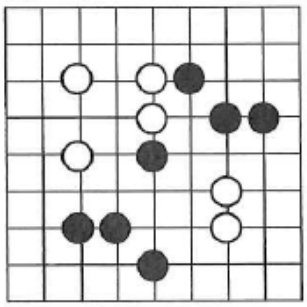
On se propose de représenter un état du jeu sur un goban $n \times n$ (où n peut valoir uniquement 9, 13 ou 19) par une liste de n listes toutes de longueur n. Chaque intersection sera définie par un numéro de ligne `i` et un numéro de colonne `j`.

L'intersection en haut à gauche (nord - ouest) est repérée par les indices `i = 0` et `j = 0` tandis que celle en bas à gauche (sud – ouest) est repérée par les indices `i = n - 1` et `j = 0`.

Pour chaque intersection,

- la présence d'une pierre noire est codée par le nombre 1,
- celle d'une pierre blanche par le nombre -1,
- l'absence de pierre par le nombre 0.

Par exemple, voici un goban de 9×9 et sa représentation par la liste `goban`.



```
goban = [[0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0,-1, 0,-1, 1, 0, 0, 0],
         [0, 0, 0, 0,-1, 0, 1, 1, 0],
         [0, 0,-1, 0, 1, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0,-1, 0, 0],
         [0, 0, 1, 1, 0, 0,-1, 0, 0],
         [0, 0, 0, 0, 1, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0]
        ]
```

1.

a. La valeur de `goban[2][5]` indique la présence d'une pierre : donner sa couleur sans justifier.

b. Sur votre copie, recopier et compléter la fonction ci-dessous afin qu'elle renvoie la représentation d'un goban de $n \times n$ vide.

```
def creer_goban_vide(n):
    assert n==9 or n==13 or n==19, "valeur de n non permise"
    ...
```

c. Que se passe-t-il lors de l'appel `creer_goban_vide(11)` ? Expliquer.

Pour la suite de l'exercice, on aura besoin de la fonction `est_valide(i, j, n)` définie par le code suivant.

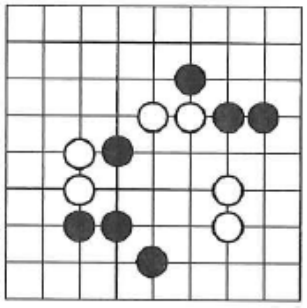
```
def est_valide(i, j, n):
    return 0<=i<n and 0<=j<n
```

Cette fonction renvoie la valeur `True` si les indices `i` et `j` passés en paramètres représentent une position valide sur un goban $n \times n$, et `False` sinon.

Cette fonction pourra être utilisée dans la suite de cet exercice.

2. La fonction `libertes_pierre(go, pi, pj)` a pour paramètres une liste `go` représentant un goban et des indices `pi` et `pj` qui repèrent l'intersection où se situe une pierre. Cette fonction renvoie les positions des intersections libres autour de cette pierre sous la forme d'une liste de tuples où chaque tuple est l'une de ces positions.

   Par exemple, si `goban` représente le goban ci-dessous, l'appel `libertes_pierre(goban, 4, 2)` renvoie une liste qui contient, dans un ordre quelconque, les tuples `(4, 1)` et `(3, 2)`.



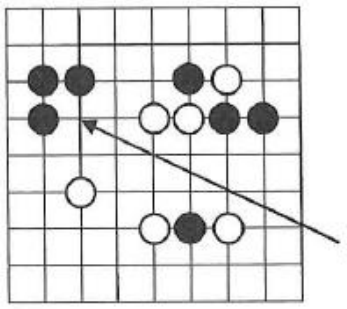
Sur votre copie, recopier et compléter le code ci-dessous.

```
def libertes_pierre(go, pi, pj):
    libertes = []
    n = len(go)

    ... # plusieurs lignes

    return libertes
```

3. Une chaine de pierres sera représentée par une liste de tuples qui repèrent leurs positions. Afin de calculer le nombre de libertés d'une chaine de pierres, on examine les libertés, sans répétition, de chacune des pierres qui la composent.

   On utilise pour cela une liste de listes `marquage` qui contient initialement des booléens tous égaux à `False`. Cette liste `marquage`, qui a les mêmes dimensions que le goban, permet de conserver une trace des libertés qui ne doivent pas être comptées plusieurs fois.

Pour illustrer le rôle de la liste `marquage`, on considère, sur le goban ci-dessous, la chaine formée des pierres dont les positions sont les tuples `(3, 1)`, `(2, 1)` et `(2, 2)`.



En supposant que le premier tuple examiné est `(3, 1)`, la liste `marquage` contiendra alors les valeurs ci-dessous.

```
[[False, False, False, False, False, False, False, False, False],
 [False, False, False, False, False, False, False, False, False],
 [False, False, False, False, False, False, False, False, False],
 [True , False, True , False, False, False, False, False, False],
 [False, True , False, False, False, False, False, False, False],
 [False, False, False, False, False, False, False, False, False],
 [False, False, False, False, False, False, False, False, False],
 [False, False, False, False, False, False, False, False, False],
 [False, False, False, False, False, False, False, False, False]]
```

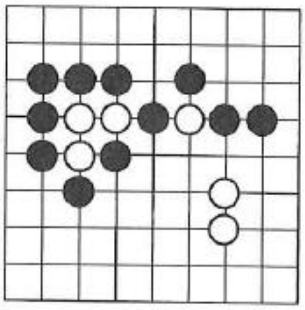
Ainsi, la liberté correspondant au tuple `(3, 2)` ne sera pas comptée une nouvelle fois lors de l'étude du tuple `(2, 2)` car `marquage[3][2]` vaut déjà `True`.

La fonction `nb_liberte_chaine(go, chaine)` prend en paramètres une liste `go` représentant un goban et une liste `chaine` représentant une chaine de pierres, et renvoie le nombre de libertés de la chaine.
Sur votre copie, écrire la séquence d'instructions qui sera exécutée dans la boucle `for i, j in libertes_pierre(go, pi, pj)`.

```
def nb_liberte_chaine(go, chaine):
    n = len(go)
    marquage = [[False for j in range(n)]
                for i in range(n)]
    nb_libertes = 0
    for pos in chaine:
        pi = pos[0]
        pj = pos[1]
        for i, j in libertes_pierre(go, pi, pj):

            ... # plusieurs lignes

    return nb_libertes
```

4. Lorsqu'une chaine ne possède aucune liberté, on dit que les pierres qui la constituent sont prisonnières, et celles-ci sont retirées du goban.

Dans l'exemple ci-dessous, la chaine formée des trois pierres situées aux intersections repérées par les tuples `(4, 2)`, `(3, 2)` et `(3, 3)` n'ont plus de libertés : elles sont donc prisonnières et sont retirées du goban.



On souhaite écrire une fonction qui, si le nombre de libertés d'une chaine est nul, renvoie le nombre de pierres prisonnières et supprime ces pierres du goban.
Écrire une telle fonction `supprime_prisonniers(go, chaine)`. Elle a pour paramètres une liste `go` représentant un goban et une liste `chaine` représentant une chaine de pierres.

5. On souhaite maintenant écrire une fonction `cherche_chaine` qui construit, étant donnée la position `(pi, pj)` d'une pierre, la chaine de pierres qui contient cette pierre. Pour cela, on examine récursivement les pierres voisines de même couleur et on ajoute leurs positions à une liste `chaine` initialement vide.

Sur votre copie, recopier et compléter les lignes 48 et 49 de cette fonction.

```
42  def cherche_chaine(go, pi, pj, chaine):
43      n = len(go)
44      chaine.append((pi, pj))
45      couleur = go[pi][pj]
46      for i, j in [(pi+1, pj), (pi-1, pj),
47                   (pi, pj+1), (pi, pj-1)]:
48          if est_valide(i, j, n) and ... and ... :
49              cherche_chaine( ... )
50      return chaine
```

Par exemple, pour le goban ci-contre, l'appel `cherche_chaine(goban, 3, 1, [])` renvoie la liste `[(3, 1), (4, 1), (2, 1), (2, 2), (2, 3)]`.