

**Exercice 2 (Algorithmique, listes et programmation dynamique - 6 points)**

Dans un jeu de stratégie, la carte est un carré de  $n \times n$  cases, où  $n$  est un entier strictement positif. Certaines cases sont dites *constructibles*, d'autres ne le sont pas. Toutes les cartes contiennent au moins une case constructible.

Une telle carte est représentée en Python par une liste de listes. Les cases constructibles sont représentées par des cellules contenant la valeur 1, celles qui sont non constructibles par des cellules contenant la valeur 0. On fournit ci-dessous la représentation d'une carte pour laquelle  $n$  est égal à 5 :

```
carte_A = [[0, 0, 1, 1, 1],
           [1, 1, 0, 1, 1],
           [0, 1, 1, 1, 0],
           [0, 1, 1, 1, 0],
           [0, 1, 1, 1, 0]]
```

Une *base* dans la carte est un carré formé de cases constructibles. On cherche à construire la plus grande base possible, ce qui revient donc à trouver un plus grand carré, inclus dans la carte, ne contenant que des valeurs 1. Il peut en exister plusieurs.

Dans la carte précédente, la plus grande base possible est un carré de trois cases de côté, son coin supérieur gauche est la cellule `carte_A[2][1]`, de coordonnées  $(2, 1)$ . On dit que cette base est *issue* de la cellule `carte_A[2][1]` et que sa taille vaut 3.

**Partie A**

On considère la carte\_B donnée ci-dessous :

```
carte_B = [[1, 1, 1, 1],
           [0, 1, 1, 1],
           [0, 1, 1, 1],
           [1, 0, 1, 1]]
```

1. Donner la taille de la plus grande base carrée ainsi que les coordonnées de la cellule dont elle est issue.

Afin de répondre à ce problème, on envisage une recherche exhaustive de solution. Cela signifie que l'on teste tous les carrés inclus dans la carte initiale afin de vérifier s'ils ne contiennent que des cases constructibles. On considère dans un premier temps le carré de taille  $n$ , puis les quatre carrés de taille  $n - 1$  et ainsi de suite jusqu'aux carrés de taille 1. Dès que l'on trouve un carré constructible, on renvoie sa taille et les coordonnées de son coin supérieur gauche.

2. On considère un entier `taille` strictement positif. Déterminer la somme des valeurs de toutes les cellules d'un carré constructible de `taille` cases de côté.

La fonction `est_constructible` prend en paramètres :

- \* la liste de listes `carte` représentant la carte ;
- \* les entiers `i_coin` et `j_coin` indiquant les coordonnées de la cellule dont est issu le carré considéré (`i_coin` est l'indice de la ligne et `j_coin` celui de la colonne) ;
- \* l'entier positif `taille` indiquant la taille de ce carré.

Cette fonction renvoie `True` si le carré décrit par les paramètres est constructible, `False` dans le cas contraire.

Cette fonction n'a pas besoin de vérifier que les coordonnées et la taille passées en paramètres définissent toujours un carré valide dont toutes les cases sont incluses dans la carte. On suppose que c'est toujours le cas.

3. Compléter le code de la fonction `est_constructible`.

```
1 def est_constructible(carte, i_coin, j_coin, taille):
2     s = 0
3     for i in range(i_coin, i_coin + taille):
4         for j in range(..., ...):
5             s = ...
6         ... # Plusieurs lignes possibles
```

La fonction `plus_grande_base_exhaustive` prend en paramètre la liste de listes `carte` représentant une carte et renvoie un triplet  $(taille, i, j)$  dans lequel `taille` est la taille d'un carré constructible de taille maximale et `i` et `j` sont les coordonnées de son coin supérieur gauche. Cette fonction utilise la méthode exhaustive décrite plus haut.

On rappelle qu'une carte contient toujours au moins une case constructible et que cette fonction renverra donc toujours un résultat.

4. Compléter le code de la fonction `plus_grande_base_exhaustive`.

```

1 def plus_grande_base_exhaustive(carte):
2     n = len(carte)
3     # les tailles vont de n à 1, de -1 en -1
4     for taille in range(n, 0, -1):
5         i = 0
6         while i + taille <= n:
7             j = ...
8             while ...:
9                 if est_constructible(...):
10                    return (taille, i, j)
11                j = ...
12            i = i + 1

```

Un décompte du nombre de carrés à étudier dans le pire des cas en fonction de la taille de la carte initiale permet de construire la figure 1 ci-dessous.

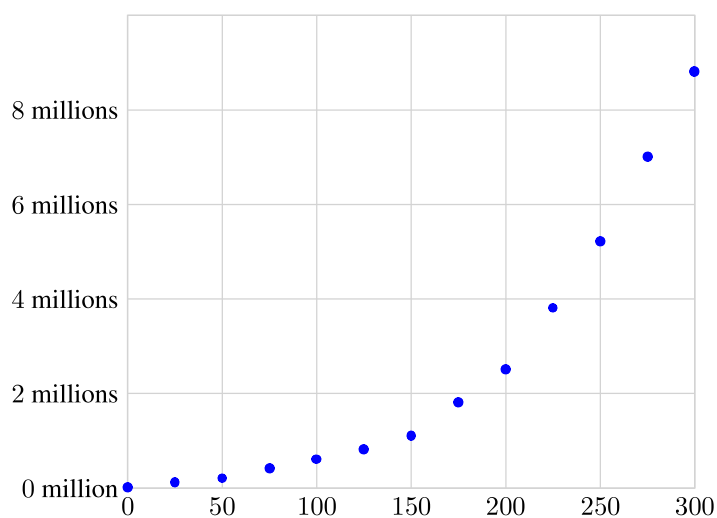


FIGURE 1 – Nombre de carrés à étudier en fonction de la largeur de la carte

5. Expliquer pourquoi cette approche exhaustive est inapplicable dans le cas de grandes cartes.

### Partie B

On souhaite désormais résoudre ce problème en utilisant la *programmation dynamique*. Pour cela, on construit une liste de listes auxiliaire aux de mêmes dimensions que la carte et telle que `aux[i][j]` contienne la taille de la plus grande base (carrée) issue de `carte[i][j]`.

En reprenant l'exemple de `carte_A`, on obtient la liste de listes `aux_A` :

```

carte_A = [[0, 0, 1, 1, 1],
           [1, 1, 0, 1, 1],
           [0, 1, 1, 1, 0],
           [0, 1, 1, 1, 0],
           [0, 1, 1, 1, 0]]

aux_A = [[0, 0, 1, 2, 1],
         [1, 1, 0, 1, 1],
         [0, 3, 2, 1, 0],
         [0, 2, 2, 1, 0],
         [0, 1, 1, 1, 0]]

```

`aux_A[2][1]` contient la valeur 3 car la plus grande base issue de la cellule `carte_A[2][1]` a une taille de 3.

Une fois la liste de listes auxiliaire `aux` remplie, on détermine la taille et les coordonnées de la plus grande base (carrée) de la carte en cherchant la valeur maximale dans `aux`.

6. Déterminer la liste auxiliaire `aux_B` associée à la carte représentée par `carte_B`.

```
carte_B = [[1, 1, 1, 1],
           [0, 1, 1, 1],
           [0, 1, 1, 1],
           [1, 0, 1, 1]]
```

On considère une carte `carte_C` de taille 6 ainsi que la liste auxiliaire `aux_C` associée. Seules certaines valeurs sont données ci-dessous.

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

aux_C = [..., ..., ..., a, 2, ...],
         [b, 0, ..., 1, 1, ...],
         [3, 2, ..., ..., ..., ...],
         [..., ..., ..., c, 2, ...],
         [d, 2, ..., 2, 2, ...],
         [1, 1, ..., ..., ..., ...]
```

7. Déterminer les valeurs des coefficients `a`, `b`, `c` et `d`.

Pour une liste de listes `carte` donnée, lorsqu'on construit la liste auxiliaire `aux` associée, on commence par remplir les cellules de la dernière ligne et de la dernière colonne de `aux` en recopiant celles de `carte`.

On admet de façon générale que, pour toutes les autres cellules, si `carte[i][j]` vaut 0 alors `aux[i][j]` prend aussi la valeur 0, et, si `carte[i][j] = 1`, alors `aux[i][j]` se calcule à l'aide de l'expression suivante :

$$1 + \min(\text{aux}[i + 1][j], \text{aux}[i][j + 1], \text{aux}[i + 1][j + 1])$$

8. Recopier et compléter les lignes 8, 9, 14 et 15 de la fonction `calcule_aux` qui prend en paramètre une liste de listes `carte` et renvoie la liste auxiliaire associée.

```
1 def calcule_aux(carte):
2     n = len(carte)
3     aux = [[0 for j in range(n)] for i in range(n)]
4
5     # Remplissage de la dernière ligne
6     # et de la dernière colonne
7     for k in range(n):
8         aux[n - 1][k] = ...
9         aux[...][...] = ...
10
11    # On complète les lignes de bas en haut
12    for i in range(n - 2, -1, -1):
13        # On complète les colonnes de droite à gauche
14        for j in range(...):
15            if ...:
16                aux[i][j] = 1 + min(aux[i + 1][j], aux[i][j + 1], aux[i + 1][j + 1])
17    return aux
```

La fonction `plus_grande_base` prend en paramètre une liste de listes `carte` et renvoie la taille et les coordonnées du coin supérieur gauche d'une base de taille maximale. Cette fonction utilise la liste auxiliaire `aux` renvoyée par l'appel `calcule_aux(carte)`.

9. Compléter la fonction `plus_grande_base` à partir de la ligne 7. Il est possible d'écrire plusieurs lignes.

```
1 def plus_grande_base(carte) :  
2     n = len(carte)  
3     aux = calcule_aux(carte)  
4     taille_max = aux[0][0]  
5     i_max = 0  
6     j_max = 0  
7     ...
```

`carte_1000` est la liste représentant une carte de  $1000 \times 1000$  cases et `carte_3000` celle représentant une carte de  $3000 \times 3000$  cases. L'appel `plus_grande_base(carte_1000)` s'exécute en 0,4 seconde.

10. Parmi les durées suivantes, indiquer celle qui permet d'estimer le temps d'exécution de l'appel `plus_grande_base(carte_3000)` :

- \* 0,4 seconde ;
- \* 1,2 seconde ;
- \* 3,6 secondes.

Justifier.