Programmation dynamique avec mémoïsation

Exercice 1 ($\star$)**.** Soit $(F_n)_n$ la suite de Fibonacci : $F_0 = F_1 = 1$ et pour tout $n \in \mathbb{N}$, $F_{n+2} = F_{n+1} + F_n$.

1. Écrire une fonction récursive `F(n)` qui, à n entier, renvoie F_n. Le calcul de $F(50)$ est-il possible ? Pourquoi ? Esquisser l'arbre d'appels récursifs.
2. Écrire une version de la fonction précédente avec mémoïsation notée `Fb(n)` Le calcul de `Fb(50)` est-il maintenant possible ?
   On vérifiera que `Fb(100)%1000` *vaut 101.*
   Esquisser l'arbre d'appels récursifs de la fonction `Fb`.
3. **À faire chez vous**, (question de maths), retrouver l'expression explicite de la suite $(F_n)_{n\in\mathbb{N}}$.

Télécharger le fichier `TP2grille.txt` qui est sur CDP et mettez-le dans **le même dossier** que votre fichier `TP2.py`. Chargez les bibliothèques et le fichier avec :

```
import matplotlib.pyplot as plt
import pickle

fichier=open("TP2grille.txt", "rb")
G=pickle.load(fichier)
fichier.close()
```

Faire `Shift+Ctrl+E` pour forcer Pyzo à choisir le dossier où est votre script python. La variable `G` est une liste de liste (que l'on peut assimiler à une matrice) dont les coefficients sont des 0 (codant le noir) et des 1 codant le blanc. Pour visualiser l'image :

```
plt.figure()
plt.imshow(G,cmap="gray")
plt.show()
```

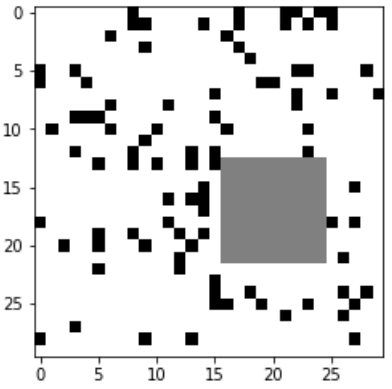
Ainsi, si `G[2][3]` vaut 1, c'est que le pixel qui est à la troisième ligne en partant du haut et quatrième colonne en partant de la gauche est blanc.

Exercice 2 ($\star\star$)**.** On souhaite trouver la taille du plus grand carré blanc présent dans la grille. Pour cela, on va coder une fonction `TPGC(i,j)` qui pour un indice de ligne `i` et un indice de colonne `j` va renvoyer la longueur du plus grand carré blanc dont `(i,j)` est le sommet en bas à droite.

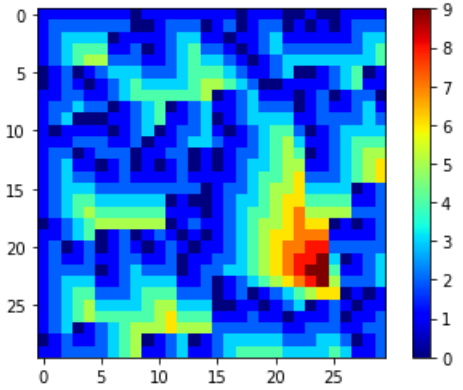
1. Si `TPGC(5,7)=3`, quelles sont les coordonnées des 9 pixels qui forment le carré blanc ?
2. Si `G[i][j]=0` que doit renvoyer `TPGC(i,j)` ?
3. Sinon, si `i=0` ou `j=0`, que doit renvoyer `TPGC(i,j)` ?

On suppose maintenant que `G[i][j]=1` et que `i` et `j` sont différents de 0.

4. Si `TPGC(i-1,j-1)`=7, `TPGC(i-1,j)`=4 et `TPGC(i,j-1)`=8, faire un dessin de la situation et trouver la valeur de `TPGC(i,j)`.
5. De façon générale, exprimer `TPGC(i,j)` en fonction de `TPGC(i-1,j-1)`, `TPGC(i-1,j)`, `TPGC(i,j-1)`
6. Coder la fonction `TPGC` de façon récursive avec mémoïsation. On pourra utiliser la fonction `min` de Python.
7. Trouver la plus grande valeur que renvoie la fonction `TPGC` ainsi qu'un couple `(i0,j0)` qui atteint cette plus grande valeur [1].
8. Modifier la valeur de `G[x][y]` à 0.5 pour tout les `x` et `y` qui sont dans ce plus grand carré blanc, puis afficher le résultat.
9. Remplacer l'utilisation de la fonction `min` de Python par une fonction auxiliaire `min3`, à coder, qui, à trois paramètres (trois nombres), renvoie le plus petit.



(a) Le plus grand carré en gris



(b) Plus la couleur d'un pixel est rouge, plus la taille du carré partant de ce pixel sera grande (image en couleur sur CDP)

1. A priori, il n'y a pas unicité du sommet qui atteint le maximum.

Exercice 3 (⋆ ⋆ ⋆ 𝄞)**.** Étant donnés deux mots, on cherche la plus longue sous-séquence commune. C'est-à-dire une chaîne de caractères dont les lettres sont dans les deux mots à la fois tout en conservant l'ordre. Et on cherche une chaîne la plus longue possible. Par exemple, c**ir**c**on**s**tance** et **i**m**p**o**r**t**ance**, la plus grande sous-séquence commune a pour longueur 7 avec **iotance** ou **irtance**[2].

1. On souhaite d'abord construire une fonction `longueur(s,t)` qui a deux chaînes de caractères `s` et `t` va compter la longueur de la plus grande sous-sequence commune.
   (a) Si `s` (ou `t`) est la chaîne vide, que vaut `longueur(s,t)` ?
   (b) Notons `n=len(s)` et `p=len(t)`, si `s[n-1]=t[p-1]`, alors comparer `longueur(s,t)` à `longueur(s[0:n-1],t[0:p-1])`.
   (c) Si `s[n-1]` $\neq$ `t[p-1]`, donner une relation entre `longueur(s,t)`, `longueur(s[0:n-1],t)` et `longueur(s,t[0:p-1])`.
   (d) Coder alors la fonction `longueur` de façon récursive avec mémoïsation.
2. Programmer alors une fonction `PLSC(s,t)` qui trouve une sous-sequence commune de longueur maximal. On distinguera les cas comme pour la fonction `longueur`
3. Télécharger le fichier `TP2ListeMots.txt` qui est sur CDP pour constituer la liste des 1300 mots les plus utilisés de la langue française, trouver deux mots dont la plus longue sous-sequence commune est la plus longue.

2. Mais on pas **irotance** car l'ordre du r et du o ne serait pas respecté. On veut que la séquence soit issue des deux mots, en conservant l'ordre des lettres. On peut en donner une définition plus formelle : si `mot` et `mot2` sont deux mots, on appelle séquence commune une chaine de caractère `c` de longueur p, telle que pour tout $i \in [\![0 ; p-1]\!]$, `c[i]=mot[`a_i`]=mot2[`b_i`]` où $0 \leqslant a_0 < a_1 < \ldots < a_{p-1} < \text{len}(mot)$ et $0 \leqslant b_0 < b_1 < \ldots < b_{p-1} < \text{len}(mot2)$, et parmi toutes les chaînes `c` on souhaite avoir une dont la longueur est la plus grande.

Créer une grille en noir et blanc aléatoire

Exercice 4 (⋆)**.** On souhaite créer des grilles comme à l'exercice 2, pour cela, charger `numpy` avec l'alias `np`. La commande `np.random.binomial(n,p)` simule une variable aléatoire suivant une loi binomiale de paramètre (n, p).

1. Quelle commande renvoie 0 avec probabilité p et 1 avec probabilité $1-p$?
2. Quelle commande permet de créer une liste à N éléments tous aléatoires valant 0 ou 1 avec les même probabilités qu'à la question 1 ?
3. Écrire une fonction `grille(N,p)` qui renvoie une matrice carrée (une liste de listes) d'ordre N dont l'image aura des pixels blancs et noirs, et chaque pixel sera noir avec une probabilité p.