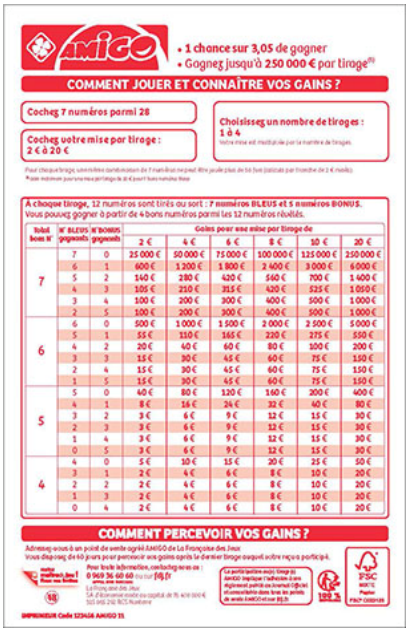
Toute calculatrice autorisée.
Le sujet comporte un total de 3 exercices
que vous pouvez traiter dans l'ordre de votre choix.

Les codes demandés devront être **clairement** commentés !

EXERCICE 1. La Française des Jeux est votre amie. (Tu parles !).

On a reproduit ci-dessous un bulletin de jeu (recto-verso) du jeu « AMIGO » de la Française des Jeux (FDJ dans la suite du sujet).



Le principe de ce jeu est le suivant : le joueur choisit 7 numéros distincts dans l'intervalle $[\![1;28]\!]$ (l'ordre des numéros choisis est sans importance). Dans la suite, un tel choix sera appelé « combinaison ». A chaque partie, la FDJ choisit au hasard un total de 12 numéros, tous distincts les uns des autres, dans l'intervalle $[\![1;28]\!]$: 7 numéros « bleus » et 5 numéros « bonus ». Si le joueur possède, dans sa combinaison, un certain nombre de numéros « bleus » et un certain nombre de numéros « bonus » (cf. le tableau des gains sur la figure de droite ci-dessus) alors il obtient le gain correspondant.

Partie A : un peu de probabilité et un calcul efficace d'un coefficient binomial

On note $C(i,j)$ le nombre de combinaisons gagnantes comportant *i* numéros « bleus » et *j* numéros « bonus ». Par exemple, si le joueur a choisi les numéros 2-7-11-15-19-22-25 et si le tirage de la FDJ est 4-7-9-13-15-22-25 (numéros « bleus ») et 2-5-11-17-20 (numéros « bonus »), il dispose d'une combinaison $(4,2)$ puisque sa combinaison comporte 4 numéros « bleus » (7, 15, 22 et 25) et 2 numéros « bonus » (2 et 11).
On note $p(i,j)$ la probabilité de l'événement « le joueur a choisi une combinaison (i,j) ».

1. Expliquer pourquoi on a : $C(i,j)=\binom{7}{i}\times\binom{5}{j}\times\binom{16}{7-i-j}$ puis en déduire que l'on a :

$$p(i,j)=\frac{\binom{7}{i}\times\binom{5}{j}\times\binom{16}{7-i-j}}{\binom{28}{7}}$$

Souhaitant calculer les probabilités $p(i,j)$ pour les couples (i,j) correspondant à des combinaisons gagnantes, on peut procéder de diverses façons pour évaluer l'expression ci-dessus. Plus précisément, on va s'interroger sur la complexité du calcul du coefficient binomial $\binom{n}{p}=\frac{n!}{p!\times(n-p)!}$. On évalue ici cette complexité comme le nombre de multiplications requises et on la note $N(n,p)$.

2. Dans cette question, on suppose que l'on évalue $\binom{n}{p}$ en calculant chacune des trois factorielles apparaissant dans le coefficient binomial.
   Donner alors $N(n,p)$.

3. On a aussi $\binom{n}{p}=\frac{n\times(n-1)\times(n-2)\times\ldots\times(n-p+1)}{p\times(p-1)\times(p-2)\times\ldots\times2\times1}$.
   Donner $N(n,p)$ lorsque l'on procède de la sorte.

4. On a enfin $\binom{n}{p}=\binom{n}{n-p}$. Donner alors $N(n,n-p)$.

5. Ecrire une fonction Python `Cbinom` recevant en argument deux entiers `n` et `p` et renvoyant un entier correspondant à $\binom{n}{p}$ (on ne cherchera pas à valider, dans la fonction, que les entiers *n* et *p* vérifient : $0\le p\le n$) en effectuant un minimum de multiplications.

6. Ecrire une fonction Python `Combi` recevant en argument deux entiers `i` et `j` et renvoyant un flottant correspondant à $C(i,j)$ (on ne cherchera pas à valider, dans la fonction, que le couple (i,j) correspond à une combinaison gagnante). Votre fonction utilisera la fonction `Cbinom` de la question précédente.

Partie B : une communication efficace mais…

La FDJ annonce « 1 chance sur 3,05 de gagner » (cf. la figure de gauche de la première page).

7. Dans un script Python, on dispose de la ligne suivante :

```
TCG = ((7,0,25000),(6,1,600),(5,2,140),(4,3,105),(3,4,100),
(2,5,100),(6,0,500),(5,1,55),(4,2,20),(3,3,15),(2,4,15),
(1,5,15),(5,0,40),(4,1,8),(3,2,3),(2,3,3),(1,4,3),(0,5,3),
(4,0,5),(3,1,2),(2,2,2),(1,3,2),(0,4,2))
```

`TCG` est un tuple de tuples de la forme $(i,j,GB(i,j))$ où le couple (i,j) correspond à une combinaison gagnante et $GB(i,j)$ au gain brut associé pour une mise de 2€. On rappelle qu'un tuple Python est indexable au même titre qu'une liste. Ainsi, `TCG[1][2]` correspond à la valeur 600.

Ecrire un code Python permettant, à partir de `TCG`, de calculer :

- la probabilité de gagner (i.e. la probabilité que le joueur ait choisi une combinaison gagnante quelle qu'elle soit.).
- l'espérance du gain net du jeu AMIGO pour une mise de 2€.

Pour information : un jeu est équitable lorsque l'espérance du gain net est nulle. Le calcul précédent fourni, pour l'espérance du gain net, une valeur approchée de -0,65€. Le jeu AMIGO n'est donc absolument pas un jeu équitable… On ne s'en étonnera pas mais la FDJ se garde bien de diffuser ce genre d'information…

Partie C : une simulation

On cherche désormais à valider les résultats précédents à l'aide d'une simulation.

Pour ce faire, on vous demande d'écrire un script qui :

- demande à l'utilisateur le nombre de joueurs effectuant un choix de 7 numéros. Ce nombre sera stocké dans la variable `Njoueurs`.
- effectue un tirage aléatoire de 12 numéros deux à deux distincts dans l'intervalle $[\![1;28]\!]$ (on utilisera la fonction `randint` du module `random`). Ces 12 numéros représenteront le tirage de la FDJ.
- effectue `Njoueurs` choix de 7 numéros dans l'intervalle $[\![1;28]\!]$ (ici encore, on utilisera la fonction `randint` du module `random`). Pour chacun de ces choix, on déterminera le nombre de numéros « bleus » et le nombre de numéros « bonus »

obtenus. On en déduira alors le gain brut correspondant (pour une mise de 2€), une combinaison non gagnante conduisant à un gain brut nul.

- calcule et affiche, après les `Njoueurs` choix :
  - pour chaque gain brut non nul apparaissant au recto du bulletin (et dans le tuple `TCG`) : sa fréquence et la probabilité associée.
  - la proportion des choix ayant donné un gain brut non nul et la probabilité de gagner.
  - le gain net moyen (pour une mise de 2€) et l'espérance du gain net.

EXERCICE 2. Le problème du drapeau hollandais (ou : Dijkstra, le retour).

Comme vous le savez, Edsger DIJKSTRA était hollandais. Vous avez pu faire sa connaissance à travers son célèbre algorithme de détermination d'un plus court chemin entre deux sommets d'un graphe pondéré. Le revoici, mais cette fois à l'origine d'un bel algorithme de tri…

C'est le drapeau de son pays (le nôtre aurait tout autant fait l'affaire ☺) qui lui a inspiré le problème suivant :

« *Soit une liste d'objets alignés dans le désordre et uniquement colorés en bleu, blanc ou rouge. Comment trier ces objets de sorte que les objets bleus soient situés à gauche, les objets blancs au centre et les objets rouges à droite ?* »

Par exemple si la situation initiale est la suivante (On utilise les lettres B, W et R pour désigner les couleurs bleu, blanc et rouge respectivement car l'impression noir et blanc constitue un réel obstacle à l'illustration de la situation… ☺) :

R R B W R B W W R B R

alors l'algorithme devra fournir la situation finale suivante :

B B B W W W R R R R R

Une implémentation de l'algorithme recevra en fait en entrée une liste de *n* entiers ne prenant que les valeurs 0 (pour le bleu), 1 (pour le blanc) ou 2 (pour le rouge).
Remarque : un ou deux couleurs peuvent parfaitement être absentes de la liste !

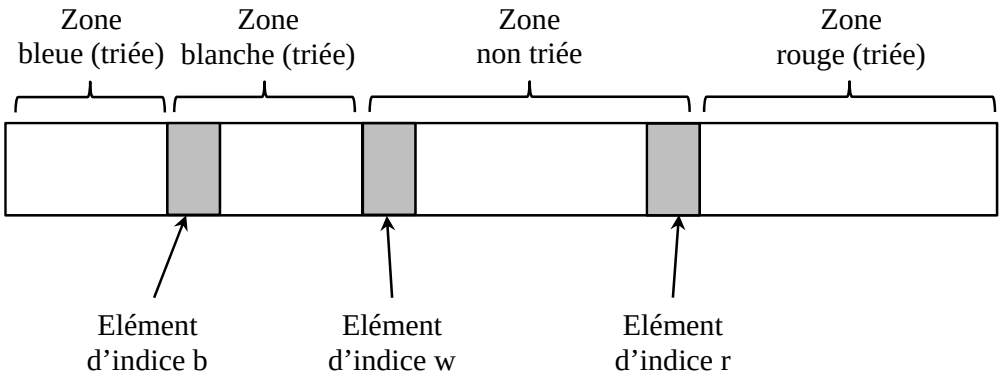
Par exemple, pour la situation initiale ci-dessus, la liste passée en argument sera :

```
L = [2,2,0,1,2,0,1,1,2,0,2]
```

Le tri doit se faire en place et ne pas utiliser de tableau(x) auxiliaire(s). Pendant l'exécution de l'algorithme, la liste comporte 4 zones (de la gauche vers la droite) : une zone dont tous les éléments sont bleus (des 0), une zone dont tous les éléments sont blancs (des 1), une zone dont les éléments sont de couleur quelconque (ce sont les éléments non triés) et, enfin, une zone dont tous les éléments sont rouges (des 2).

On va gérer en permanence trois variables :

- `b`, correspondant à l'indice du premier élément de la liste qui n'est pas bleu.
- `w`, correspondant à l'indice du premier élément de la liste qui n'est pas blanc.
- `r`, correspondant à l'indice du dernier élément de la liste qui n'est pas rouge.



Initialement, les zones triées sont vides (tous les éléments sont considérés comme non triés) et, en tenant compte de l'indexation des listes Python, on aura :

`b=0`, `w=0` et `r=n-1`

On a alors l'algorithme :

```
Tant que w <= r faire :
     Si L[w] = blanc alors :
          w = w + 1
     Sinon :
          si L[w] = bleu alors :
               Echanger L[w] et L[b]
               b = b + 1
               w = w + 1
          sinon :
               Echanger L[w] et L[r]
               r = r – 1
Fin Tant que
```

1. Ecrire une fonction Python `DutchFlag` qui code cet algorithme.
2. Pourquoi l'algorithme se termine-t-il ?

Dans cet algorithme, les principaux éléments de complexité sont les tests et les échanges.

On note n la longueur de la liste à trier et $n(b)$, $n(w)$ et $n(c)$ les nombres d'éléments respectivement bleus, blancs et rouges dans la liste. On a bien sûr : $n = n(b) + n(w) + n(r)$.

On note également $t(n)$ et $e(n)$ le nombre de tests et d'échanges effectués pour trier une telle liste.

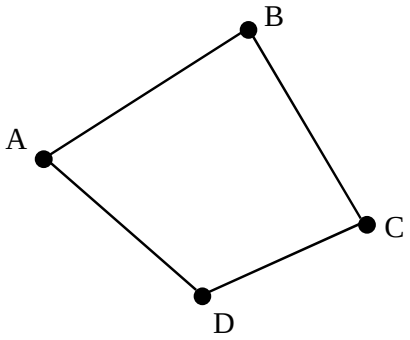
3. Evaluer $t(n)$ et $e(n)$ dans le meilleur et le pire des cas (on donnera à chaque fois un exemple correspondant à la situation).
4. Evaluer la complexité moyenne en termes de tests puis en termes d'échanges.

   Dans un premier temps, on pourra considérer que l'on a : $n(b)=n(w)=n(r)=\frac{n}{3}$.

   Dans un second temps, on considèrera que l'on a : $n=N(b)+N(w)+N(r)$ où $N(b)$, $N(w)$ et $N(r)$ sont des variables aléatoires d'espérances égales à $\frac{n}{3}$.

EXERCICE 3. Des quadrilatères dans des graphes

Dans cet exercice, on cherche à déterminer le nombre de « quadrilatères vrais » présents dans un graphe simple non orienté, c'est-à-dire le nombre de configurations du type :



où A, B, C et D sont quatre sommets du graphe, deux à deux distincts.

Notons, classiquement :

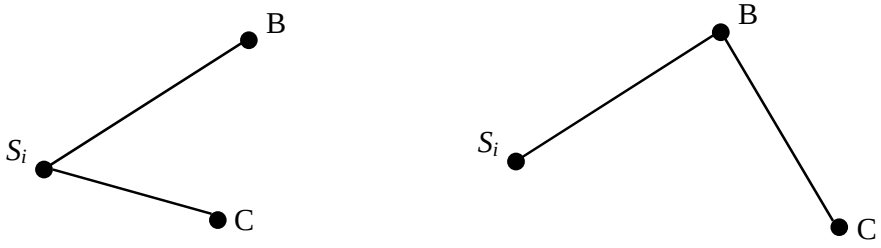
- n l'ordre du graphe considéré.
- Pour tout entier naturel i dans $[\![0\,;n-1]\!]$, d_i le degré du sommet S_i.
- D la somme des degrés des sommets : $D=\sum_{i=0}^{n-1} d_i$.
- $A=\left(a_{ij}^{(0)}\right)$ sa matrice d'adjacence.
- Pour tout entier naturel p, $A^p=\left(a_{ij}^{(p)}\right)$ la puissance p-ième de la matrice A.

On note alors $Q(n)$ le nombre de « quadrilatères vrais » d'un tel graphe.

Pour évaluer $Q(n)$, nous allons nous intéresser, légitimement (☺), à $\mathrm{Tr}\left(A^4\right)=\sum_{i=0}^{n-1} a_{ii}^{(4)}$, la trace de la matrice A^4.

1. Expliquer pourquoi la contribution de $Q(n)$ à $\mathrm{Tr}\left(A^4\right)$ est égale à $8\times Q(n)$.

2. Les autres configurations contribuant à $\mathrm{Tr}\left(A^4\right)$ sont :



Avec les chaînes :

- Configuration de gauche :
  S_i-B-S_i-C-S_i et S_i-C-S_i-B-S_i, sachant que dans ce type de configuration, on peut avoir B = C.
- Configuration de droite :
  S_i-B-C-B-S_i, sachant que dans ce type de configurations, on a $\mathrm{B}\neq\mathrm{C}$.

a. Montrer que la contribution des configurations de gauche à $\mathrm{Tr}\left(A^4\right)$ vaut :

$$\sum_{i=1}^{n} d_i^2$$

b. Montrer que la contribution des configurations de droite à $\mathrm{Tr}\left(A^4\right)$ vaut :

$$\sum_{\substack{i,j \\ i\neq j}} a_{ij}^{(2)}$$

3. Déduire des questions précédentes :

$$\begin{aligned} Q(n) &= \frac{1}{8}\times\left(\mathrm{Tr}\left(A^4\right)-\sum_{i=1}^{n} d_i^2-\sum_{\substack{i,j \\ i\neq j}} a_{ij}^{(2)}\right) \\ &= \frac{1}{8}\times\left(\mathrm{Tr}\left(A^4\right)-\sum_{i=1}^{n} d_i^2-\sum_{i,j} a_{ij}^{(2)}+D\right) \end{aligned}$$

4. Ecrire une fonction Python `NumOfQ` qui reçoit en argument la matrice d'adjacence A d'un graphe simple non orienté et en renvoie le nombre de « quadrilatères vrais ».

   On rappelle que :

   - La fonction `dot` de la bibliothèque `numpy` permet de calculer le produit de deux matrices (tableaux `numpy`).
   - La fonction `shape` (équivalente à la méthode `shape`) de la bibliothèque `numpy` permet d'obtenir les dimensions d'une matrice (tableaux `numpy`). Cette fonction renvoie un tuple.