Le but est, outre de réviser le cours, de se prouver qu'on est capable de savoir écrire des algorithmes en décomposant la difficulté.

LE PROGRAMME DE PSI

1. Expliquer l'avantage de la mémoïsation dans un algorithme récursif et sa mise en place.
2. A quoi sert l'algorithme de Floyd-Warshall ? Quel(s) autre(s) algorithme(s) connaissez-vous qui répondent à cette question ? Quelles sont les différences ?
3. Qu'est la distance d'édition (ou de Levenshtein) entre deux mots ?
4. (a) Expliquer sommairement le principe de l'algorithme des k plus proches voisins.
   (b) Qu'est la matrice de confusion ? Que permet-elle de faire ?
5. (a) Expliquer sommairement le principe de l'algorithme des k-moyennes.
   (b) Que dire du résultat renvoyé par cet algorithme ? Comment choisir k ?
6. Quels types de parcours existe-t-il au programme dans les graphes ? Comment les différencier ?
7. Comment modélise-t-on un jeu à deux joueurs, tour à tour ?
8. Qu'est une stratégie ?
9. Qu'est une stratégie gagnante ?
10. Qu'est l'attracteur d'un joueur dans la théorie des jeux ?
11. Citer une stratégie de la théorie des jeux qui n'est pas nécessairement gagnante mais permet de jouer mieux qu'au hasard. De quoi a-t-on besoin pour la mettre en place ?

DES ALGORITHMES POUR SE TESTER

L'emploi de la commande `if x in L` où `L` est une liste est rigoureusement interdit.

12. Soit `texte` une chaine de caractères. Écrire une fonction `Tipex(texte,i,lettre)` qui renvoie une chaîne où le caractère d'indice `i` aura été remplacé par le caractère `lettre` dans `texte`.
    `texte` ne doit pas être modifiée.
13. Soit `L` une liste de flottants. Écrire une fonction `Croissante(L)` qui teste si `L` est une liste triée par ordre croissant ou non (renvoie le booléen adéquat).
14. Écrire une fonction `PasDeDoublons(L)` qui renvoie une liste contenant les mêmes éléments que ceux de la liste `L` mais en un seul exemplaire (complexité attendue en $\mathcal{O}$(`len(L)`)).
    `L` ne doit pas être modifiée.
15. Écrire une fonction `RienAVoir(L1,L2)` qui renvoie `True` si les deux listes `L1` et `L2` n'ont aucun élément en commun et `False` sinon (complexité attendue en $\mathcal{O}$(`max(len(L1),len(L2))`)).
16. Implémenter une fonction `dim(img)` qui prend en entrée une matrice `img` (liste de listes) représentant une image en niveaux de gris (chaque élément est un entier entre 0 et 255) et renvoie le couple (h, w) donnant la dimension de l'image (h pour hauteur et w pour la largeur).
17. Implémenter une fonction `voisins(pixel,img)` qui prend en entrée une matrice `img` et un tuple d'entiers `pixel` (coordonnées (i, j) du pixel) et renvoie la liste des coordonnées des pixels voisins de ce pixel.
    (il y a au plus 4 voisins par pixel et on fera attention aux bords !)

18. Implémenter en Python une fonction `aplatir(L)` qui prend en argument une liste `L` de couples où le premier élément du couple est un objet quelconque et le second une liste (potentiellement vide) d'objets potentiellement quelconques. La fonction doit renvoyer une liste où tous les objets sont mis à la suite.
Un exemple valant plus qu'un long discours :

```
>>> L = [('a', [1, 2, 3]), ('b', ['answer', 42]), (813, [])]
>>> aplatir(L)
['a', 1, 2, 3, 'b', 'answer', 42, 813]
```

19. Implémenter une fonction `unique(L)` qui prend en argument une liste `L` d'objets pouvant servir comme clef d'un dictionnaire et renvoie un dictionnaire dont les clefs sont les éléments de `L` et la valeur correspond à l'indice de la première apparition dans la liste.
À nouveau, un petit exemple permet de clarifier les choses :

```
>>> L = ['zz', 'xy', 'zz', 't', 't', 'xy']
>>> unique(L)
{'xy': 1, 't': 2, 'zz': 0}
```

20. Écrire une fonction `decaler(L:list[int],i0:int) -> NoneType` qui modifie la liste `L` en remplaçant tous les éléments `i` $>$ `i0` par `i-1`. Par exemple avec `L = [5,1,4,9]` et `i0 = 4`, on obtient `[4,1,4,8]`. Votre fonction doit modifier la liste en entrée et renvoyer `None`.

21. Écrire une fonction `Somme(M)` qui prend en paramètre une séquence imbriquée, de profondeur et de structure quelconques, dont tous les composants élémentaires sont des nombres, et calcule la somme de tous ces éléments.

    Par exemple :

```
>>> somme([[[1, 2], [3, 4, 5]], 6, [7, 8], 9])
45
```

    Indication - L'expression booléenne `isinstance(x, numbers.Real)` *permet de tester si* `x` *est un scalaire numérique (dont font partie les flottants et entiers).*

    Par exemple :

```
>>> isinstance(1, numbers.Real)
True
>>> isinstance([1, 2, 3], numbers.Real)
False
```

LE PROGRAMME DE PSI

1. Expliquer l'avantage de la mémoïsation dans un algorithme récursif et sa mise en place.

   La mémoïsation ne présente un intérêt que s'il y a chevauchement des sous-problèmes dans l'algorithme récursif. L'avantage de la mémoïsation est de stocker dans un dictionnaire, initialement vide, les valeurs que va renvoyer la fonction récursive lors de ses appels successifs. Ainsi l'algorithme ne calculera pas deux fois les mêmes valeurs et ira chercher dans le dictionnaire les valeurs déjà calculées. Il est important que le dictionnaire soit partagé entre toutes les instances de la fonction, il ne faut donc pas qu'il soit une variable locale à la fonction récursive.

2. A quoi sert l'algorithme de Floyd-Warshall ? Quel(s) autre(s) algorithme(s) connaissez-vous qui répondent à cette question ? Quelles sont les différences ?

   L'algorithme de Floyd-Warshall permet d'obtenir la matrice des distances (plus court chemin) entre deux sommets quelconques dans un graphe orienté et pondéré.
   L'algorithme de Dijkstra répond également à cette question, mais uniquement en partant d'un unique sommet (vers n'importe quel sommet du graphe).
   L'algorithme $A^\star$ également, mais uniquement depuis un sommet source vers un sommet destination.

3. Qu'est la distance d'édition (ou de Levenshtein) entre deux mots ?

   La distance d'édition (ou de Levenshtein) entre deux mots est le plus petit nombre de modifications (changement, ajout ou suppression de lettre) à effectuer de sorte à transformer le premier mot en le second.

4. (a) Expliquer sommairement le principe de l'algorithme des k plus proches voisins.

   Il s'agit d'identifier une caractéristique d'une donnée, en la comparant à un jeu de données où la même caractéristique de chaque élément du jeu de données est connue.
   Pour ce faire, on définit une distance entre deux données qui caractérise leurs ressemblances (plus la distance est faible plus il y a ressemblance), et on calcule la distance entre la donnée prise en entrée et chaque donnée du jeu de données.
   Ensuite on isole les k plus proches éléments/voisins et on regarde l'étiquette majoritaire parmi ces éléments afin d'identifier la caractéristique cherchée pour la donnée prise en entrée.

   (b) Qu'est la matrice de confusion ? Que permet-elle de faire ?

   La matrice a une taille égale aux nombre d'étiquettes différentes, et le coefficient $M[i][j]$ de la matrice de confusion est égal au nombre de fois où un élément d'étiquette n°i a été prédit par l'algorithme comme un élément d'étiquette n°j.
   Les termes de la diagonale correspondent donc à des prédictions correctes et ceux hors diagonale à des prédictions incorrectes.
   Plus la matrice de confusion ressemble à une matrice diagonale, plus la valeur de k est adaptée à la détection de l'étiquette cherchée.

5. (a) Expliquer sommairement le principe de l'algorithme des k-moyennes.

   On dispose d'un ensemble de données ayant des caractéristiques, que l'on cherche à regrouper en k groupes, ou clusters, de données ayant le plus de similitudes dans leurs caractéristiques.
   Pour ce faire, on répartit initialement les données aléatoirement dans k clusters, puis on calcule les barycentres de chaque groupe. Ensuite, après avoir défini une distance entre deux données qui caractérise leurs ressemblances, on calcule la distance entre chaque donnée et les barycentres de chaque groupe. On déplace alors chaque donnée pour la mettre dans le cluster dont le barycentre est le plus proche. On réitère ce processus jusqu'à stabilisation des clusters.

   (b) Que dire du résultat renvoyé par cet algorithme ? Comment choisir k ?

   Le résultat de l'algorithme n'est pas optimal, il renvoie une répartition qui réalise une solution qui n'est pas nécessairement la meilleure mais peut être convenable. Ce résultat dépend de la répartition initiale dans les clusters, et est donc différent à chaque fois que l'algorithme est exécuté, même si on l'exécute sur les même données, puisque la répartition initiale est aléatoire.

6. Quels types de parcours existe-t-il au programme dans les graphes ? Comment les différencier ?

   Il existe les parcours en profondeur (avec pile, les sommets fils sont visités avant les sommets frères) et les parcours en largeur (avec file, les sommets frères sont tous visités avant les sommets fils).

7. Comment modélise-t-on un jeu à deux joueurs, tour à tour ?

   On le modélise comme un graphe biparti, où chaque sommet correspond à une position/situation de jeu en précisant le joueur qui joue.

8. Qu'est une stratégie ?

   Une stratégie est une fonction qui associe à chaque situation de jeu un coup à jouer.

9. Qu'est une stratégie gagnante ?

   Une stratégie gagnante pour un joueur J est une stratégie telle que, quels que soient les coups joués par l'adversaire, la victoire est assurée pour le joueur J à la fin de la partie, s'il joue toujours selon la stratégie.

10. Qu'est l'attracteur d'un joueur dans la théorie des jeux ?

    L'attracteur pour un joueur est l'ensemble des sommets (positions de jeu) qui assure l'existence d'une stratégie gagnante partant de ce sommet.

11. Citer une stratégie de la théorie des jeux qui n'est pas nécessairement gagnante mais permet de jouer mieux qu'au hasard. De quoi a-t-on besoin pour la mettre en place ?

    On peut citer la stratégie min-max, qui permet au joueur de jouer le meilleur coup en tenant compte des futurs n coups, en supposant que son adversaire joue au mieux pour le contrer.
    On a besoin au préalable de définir une heuristique, c'est à dire une fonction h basée sur l'expérimentation ou l'intuition et qui permet d'associer à chaque sommet s du graphe représentant le jeu une valeur qui représente au mieux les chances d'arriver à la victoire pour le joueur en question (plus elle est élevée, plus la victoire devrait être probable).

DES ALGORITHMES POUR SE TESTER

L'emploi de la commande `if x in L` où `L` est une liste est rigoureusement interdit.

12. Soit `texte` une chaine de caractères. Écrire une fonction `Tipex(texte,i,lettre)` qui renvoie une chaîne où le caractère d'indice `i` aura été remplacé par le caractère `lettre` dans `texte`.
    `texte` ne doit pas être modifiée.

```
def Tipex(texte,i,lettre):
  return texte[:i]+lettre+texte[i+1:]
```

13. Soit `L` une liste de flottants. Écrire une fonction `Croissante(L)` qui teste si `L` est une liste triée par ordre croissant ou non (renvoie le booléen adéquat).

```
def Croissante(L):
  for i in range(len(L)-1):
    if L[i]>L[i+1]:
      return False
  return True
```

14. Écrire une fonction `PasDeDoublons(L)` qui renvoie une liste contenant les mêmes éléments que ceux de la liste `L` mais en un seul exemplaire (complexité attendue en $\mathcal{O}$(`len(L)`)).
`L` ne doit pas être modifiée.

```
def PasDeDoublons(L):
  D={}
  L2=[]
  for x in L:
    if x not in D:
      L2.append(x)
      D[x]=True
  return L2
```

15. Écrire une fonction `RienAVoir(L1,L2)` qui renvoie `True` si les deux listes `L1` et `L2` n'ont aucun élément en commun et `False` sinon (complexité attendue en $\mathcal{O}$(`max(len(L1),len(L2))`)).

```
def RienAVoir(L1,L2):
  D={}
  for x in L1:
    if x not in D:
      D[x]=True
  for x in L2:
    if x in D:
      return False
  return True
```

16. Implémenter une fonction `dim(img)` qui prend en entrée une matrice `img` (liste de listes) représentant une image en niveaux de gris (chaque élément est un entier entre 0 et 255) et renvoie le couple (h, w) donnant la dimension de l'image (h pour hauteur et w pour la largeur).

```
def dim(img):
  return (len(img),len(img[0]))
```

17. Implémenter une fonction `voisins(pixel,img)` qui prend en entrée une matrice `img` et un tuple d'entiers `pixel` (coordonnées (i, j) du pixel) et renvoie la liste des coordonnées des pixels voisins de ce pixel.
(il y a au plus 4 voisins par pixel et on fera attention aux bords !)

```
def voisins(pixel,img):
  h,w=dim(img)
  i,j=pixel
    V = []
    if i>0:
        V.append([i-1,j])
    if j>0:
        V.append([i,j-1])
    if i<h-1:
        V.append([i+1,j])
    if j<w-1:
        V.append([i,j+1])
    return r
```

18. Implémenter en Python une fonction `aplatir(L)` qui prend en argument une liste `L` de couples où le premier élément du couple est un objet quelconque et le second une liste (potentiellement vide) d'objets potentiellement quelconques. La fonction doit renvoyer une liste où tous les objets sont mis à la suite.
Un exemple valant plus qu'un long discours :

```
>>> L = [('a', [1, 2, 3]), ('b', ['answer', 42]), (813, [])]
>>> aplatir(L)
['a', 1, 2, 3, 'b', 'answer', 42, 813]
```

```
def aplatir(L):
  sortie = []
  for couple in L:
    x,liste = couple
    sortie.append(x)
    for y in liste:
      sortie.append(y)
  return sortie
```

19. Implémenter une fonction `unique(L)` qui prend en argument une liste `L` d'objets pouvant servir comme clef d'un dictionnaire et renvoie un dictionnaire dont les clefs sont les éléments de `L` et la valeur correspond à l'indice de la première apparition dans la liste.
À nouveau, un petit exemple permet de clarifier les choses :

```
>>> L = ['zz', 'xy', 'zz', 't', 't', 'xy']
>>> unique(L)
{'xy': 1, 't': 2, 'zz': 0}
```

```
def unique(L):
  c = 0
  d = {}
  for x in L:
    if not(x in d):
      d[x] = c
    c = c + 1
  return d
```

20. Écrire une fonction `decaler(L:list[int],k0:int) -> NoneType` qui modifie la liste `L` en remplaçant tous les éléments `k` $>$ `k0` par `k-1`. Par exemple avec `L = [5,1,4,9]` et `k0 = 4`, on obtient `[4,1,4,8]`.
Votre fonction doit procéder par effet de bord (modifier la liste en entrée) et ne rien renvoyer.

```
def decaler (L, k0):
  for i in range(len(L)):
    if L[i] > k0:
      L[i] = L[i] - 1
```

21. Écrire une fonction `Somme(M)` qui prend en paramètre une séquence imbriquée, de profondeur et de structure quelconques, dont tous les composants élémentaires sont des nombres, et calcule la somme de tous ces éléments.

    Par exemple :

```
>>> somme([[[1, 2], [3, 4, 5]], 6, [7, 8], 9])
45
```

    Indication - L'expression booléenne `isinstance(x, numbers.Real)` *permet de tester si* `x` *est un scalaire numérique (dont font partie les flottants et entiers).*

    Par exemple :

```
>>> isinstance(1, numbers.Real)
True
>>> isinstance([1, 2, 3], numbers.Real)
False
```

```
def somme(M) :
  res = 0
  for x in M :
      if isinstance(x,numbers.Real) :
        res += x
      else :
        res += somme(x)
  return res
```