Projet : solveur pour « Des chiffres et des lettres »

Objectif: L'objectif de ce projet en deux parties est de programmer un solveur pour les deux principales épreuves du célèbre jeu télévisé « Des chiffres et des lettres », appelées respectivement « le mot le plus long » et « le compte est bon ».

1 Travail demandé et modalités d'évaluation

1.1 Organisation du travail

- Le projet est **obligatoire** et sa note vaudra pour un tiers (1/3) de la note finale d'AP2.
- Le projet est à réaliser en **binômes** (deux étudiants au moins, deux étudiants au plus !).
- Le travail fera (probablement) l'objet d'une soutenance. Les dates de rendu et de soutenances seront précisées en temps voulu.
- Les tâches pourront être réparties au sein du binôme, ou l'ensemble du projet réalisé conjointement. Chaque membre du binôme est censé pouvoir expliquer l'ensemble du code.
- Il est encouragé de demander de l'aide à vos chargés de TP, de TD, de cours. Vous avez le droit d'échanger des **idées** avec d'autres étudiants (par exemple sur le forum). Cependant, **tout plagiat détecté entre des binômes différents sera lourdement sanctionné** (on entend par plagiat la recopie de portions non négligeables de code).
- Il est tout à fait possible d'obtenir une bonne note sans traiter la moindre amélioration optionnelle, si la base du projet est bien réalisée et l'ensemble des consignes respectées.
- En cas de difficulté d'organisation **importante** (problème de binôme, problème technique, etc.), contactez en priorité votre chargé de TP, et à défaut l'un des autres enseignants du module.

1.2 Description des fichiers à rendre

Le rendu du projet devra comporter au moins les **trois** fichiers suivants :

- Deux fichiers `lettres.py` et `chiffres.py` contenant les fonctions spécifiques à chacune des épreuves et les programmes principaux associés.

- Un fichier `rapport.pdf` contenant la documentation de votre projet. Pour plus de facilité, vous pouvez, si vous le souhaitez, rédiger ce compte-rendu dans un notebook Python au format Markdown et le convertir au format PDF à l'aide de SageMathCloud ou, si vous préférez, utiliser LibreOffice ou un logiciel du même genre et exporter votre document au format PDF.

Ces trois fichiers (ainsi que tout fichier supplémentaire nécessaire) devront être placés dans le répertoire « Projet » de l'espace personnel de l'un des membres du binôme sur SageMathCloud.

Important : En **aucun cas** le nom des fonctions indiqué dans le sujet, le nombre et le type des paramètres ni le type du résultat ne doivent être modifiés. Nous utiliserons un correcteur automatique pour tester vos fonctions. Si une fonction est mal nommée ou renvoie une valeur d'un type inattendu, le correcteur ne pourra pas la tester correctement et vous risquez d'être pénalisés.

Conseil : N'abordez les paragraphes « améliorations » qu'après avoir **entièrement** terminé toutes les questions obligatoires des **deux** parties du projet.

1.3 Contenu du rapport

Le rapport détaillera de façon **précise** et **succincte** (maximum 5 pages) :

- Le nom et le groupe de TP des deux étudiants ayant participé au projet.
- Un guide utilisateur **bref** indiquant comment se servir de chaque programme, des menus, des options en ligne de commande éventuelles.
- L'état d'avancement de la partie obligatoire du projet (ce qui est terminé et ce qui ne l'est pas), les bugs éventuels.
- Une description des améliorations éventuelles.
- Pour **chaque fonction réalisée**, des commentaires sur son implémentation, sa complexité, etc.
- La répartition du travail entre les membres du binôme, les difficultés rencontrées.

2 Épreuve "le mot le plus long"

2.1 Rappel des règles du jeu

Adapté de Wikipedia.fr.

Dans cette épreuve, les deux candidats décident chacun leur tour s'ils souhaitent qu'une voyelle ou une consonne soit tirée au sort, jusqu'à obtenir 10 lettres. Le but est de trouver le plus long mot possible en utilisant les lettres qui ont été tirées, se rapprochant ainsi du principe de l'anagramme. Les diacritiques (accents, tréma, cédille…) sont négligés.

- Premier exemple : avec le tirage « B A M E R I L A N E » il est possible d'obtenir le mot REMANIABLE (en 10 lettres).

- Second exemple : avec le tirage « A N E D O S U N E I » il est possible d'obtenir les mots SAOUDIENNE (en 10 lettres) ; ADENOSINE, AUDONIENS et SOUDANIEN (en 9 lettres).

2.2 Objectif

Le solveur pour cette partie du jeu consiste à déterminer, pour un tirage donné de lettres, quels sont les plus longs mots qu'il est possible de constituer. Pour cela, nous vous fournissons un fichier `dico.txt` contenant une liste de mots du français que l'on considérera comme valables (en réalité, il faudrait retirer toutes les formes conjuguées des verbes…). Pour plus de simplicité, nous avons déjà retiré tous les caractères accentués et les traits d'union.

2.3 Recherche de solutions (partie obligatoire)

La première étape de résolution consiste à déterminer si un mot donné est réalisable avec les lettres d'un tirage. Pour reprendre l'exemple ci-dessus, le tirage « B A M E R I L A N E » permet de composer le mot « RAME » mais pas le mot « RAMER ».

L'idée de l'algorithme de résolution est d'essayer de composer chaque mot du dictionnaire avec les lettres du tirage, et de renvoyer la liste de tous les mots possibles, puis de rechercher parmi tous ces mots ceux qui ont le plus de lettres. On décomposera le travail ainsi :

- **Anagrammes :** Écrivez une fonction `mot_possible(tirage, mot)` renvoyant `True` si `mot` peut être composé avec les lettres de `tirage`, et `False` sinon. On pourra par exemple comparer le nombre d'occurrences de chacune des lettres apparaissant dans `mot` au nombre d'occurrences de cette lettre dans `tirage`.

  Par exemple, `mot_possible(['a', 'm', 'e', 'r', 'e'], 'marre')` doit renvoyer `False`, et `mot_possible(['a', 'm', 'e', 'r', 'e'], 'ramee')` doit renvoyer `True`.

  Attention, une fonction de complexité quadratique ou plus sera considérée comme non optimale.

- **Recherche exhaustive :** Écrivez une fonction `tous_mots_possibles(tirage, liste_mots)`, où `tirage` est un tirage de lettres comme précédemment, et `liste_mots` est une liste de mots autorisés. Cette fonction doit renvoyer la liste complète de tous les mots de `liste_mots` que l'on peut composer avec le tirage donné.

  Par exemple, `tous_mots_possibles(['e', 's', 't'], mots_francais)`, où `mots_francais` est la liste des mots lue dans le fichier `dico.txt`, doit renvoyer la liste `['es', 'est', 'et', 'se', 'set', 'te', 'tes']` (l'ordre des éléments n'a pas d'importance ici).

  En considérant la taille du tirage comme fixée, cette fonction doit être linéaire en la longueur de `liste_mots`.

- **Programme principal :** Modifiez le fichier `lettres.py` afin que la ligne de commande `python3 lettres.py` provoque la lecture sur l'entrée standard d'une suite de lettres constituant un tirage, et affiche sur la sortie standard la liste des solutions (un mot par ligne).

  Attention ! Votre programme doit uniquement lire sur l'entrée standard le tirage souhaité (sans espace, et sans message d'invite), et uniquement afficher la liste des solutions (sans message !). Par exemple :

```
dead-pony:code ameyer$ python3 lettres.py
est
['es', 'est', 'et', 'se', 'set', 'te', 'tes']
dead-pony:code ameyer$
```

- **Meilleures solutions :** Modifiez votre programme afin que les solutions soient affichées par longueur décroissante, et que les mots de même longueur soient affichés par ordre alphabétique. Par exemple :

```
dead-pony:code ameyer$ python3 lettres.py
est
['est', 'set', 'tes', 'es', 'et', 'se', 'te']
dead-pony:code ameyer$
```

2.4 Améliorations (partie optionnelle)

Si vous avez fini tout le reste, de nombreuses améliorations sont possibles. La liste (non exhaustive) suivante donne quelques suggestions. Essayez d'en implémenter le plus possible !

- Modifiez votre programme pour que **seulement** les solutions les plus longues soient recherchées et affichées. Attention, il ne s'agit pas ici de chercher toutes les solutions puis de les trier par longueur comme dans la solution naïve précédente. Une piste de solution est de pré-traiter la liste des mots du dictionnaire.

- Pour déterminer s'il est possible ou non de composer un certain mot avec le tirage considéré, on a suggéré de compter combien il contient d'occurrences de chaque lettre. Il serait plus efficace de ne faire ce calcul qu'**une seule fois** par tirage, et qu'une seule fois pour chaque mot du dictionnaire ! Effectuez un pré-traitement du dictionnaire pour qu'aucun calcul inutile ne soit fait.

- Le programme principal (pour la partie « lettres ») est un peu pauvre… Écrivez une fonction permettant d'afficher un menu interactif (en mode texte) permettant :

  - de spécifier directement le tirage initial sur la ligne de commande ;
  - de spécifier le nom du fichier de dictionnaire ;
  - de spécifier le type de solution attendu (toutes, seulement les plus longues, les solutions ayant un certain nombre minimum de lettres, etc.) ;
  - de choisir aléatoirement un tirage ;
  - de modifier la fréquence initiale des lettres, etc.

  Ensuite, modifiez votre programme principal pour qu'il affiche ce menu si on le lance avec la commande `python3 lettres.py --menu`.

  Attention ! Lorsque l'option `--menu` n'est pas spécifiée, le contenu du programme doit rester le même que précédemment (lecture d'une ligne sans invite, affichage des solutions sans message).

- Ajoutez d'autres options en ligne de commande à votre programme, permettant de spécifier directement ces comportements. Vous pourrez utiliser par exemple le module Python `argparse` (voir la documentation).

- Ajoutez au programme une option permettant de chronométrer la recherche de solutions pour un tirage donné. Vous pourrez ainsi comparer vos résultats à ceux des autres binômes (sur le même tirage évidemment) et atteindre le rang de solveur de mot le plus long ultime.

3 Épreuve “le compte est bon”

3.1 Rappel des règles du jeu

Adapté de Wikipedia.fr.

Le but de cette épreuve est d'obtenir un nombre (de 101 à 999) à l'aide d'opérations élémentaires (addition, soustraction, multiplication et division entière) sur les entiers naturels, en partant de nombres tirés au hasard (de 1 à 10, 25, 50, 75 et 100). Lorsque l'émission n'était pas informatisée, le jeu comportait vingt-quatre plaques : les nombres de 1 à 10 présents en double exemplaire et les nombres 25, 50, 75 et 100 présents en un seul exemplaire.

Sont tirées alors 6 valeurs (pas forcément toutes différentes). Chaque valeur ne peut être utilisée qu'une seule fois au maximum. À défaut de trouver le compte exact, il faut tenter de s'en approcher le plus près possible.

- Premier exemple : nombres tirés 3, 100, 8, 8, 10, 6, résultat demandé 683.

```
6 × 100 = 600
8 × 10 = 80
600 + 80 = 680
680 + 3 = 683
```

- Second exemple : nombres tirés 3, 75, 2, 4, 1, 1, résultat demandé 888.

```
75 - 1 = 74
3 × 4 = 12
74 × 12 = 888
```

3.2 Objectif

Le but de cette partie du projet est de déterminer, pour un tirage donné de nombres et un résultat demandé donné, toutes les suites d'opérations qui permettent d'obtenir ce résultat. Par exemple, si le tirage initial est `1 2 3 4` et le résultat demandé est `24`, le programme affichera (une solution par ligne) :

```
3*2=6 6*4=24
4*2=8 8*3=24
4*3=12 12*2=24
2+1=3 3+3=6 6*4=24
3+1=4 4+2=6 6*4=24
3+1=4 4+2=6 6*4=24
3+2=5 5+1=6 6*4=24
4+2=6 3+1=4 6*4=24
```

3.3 Recherche des solutions (partie obligatoire)

- **Opérations :** La première étape dans la recherche de solution consiste à calculer, étant donné un tirage, la liste des opérations mathématiques que l'on peut effectuer avec ce tirage.

  Écrivez une fonction `operations_possibles(tirage)` renvoyant une liste de tuples `(x, op, y, res)` où `x` et `y` sont des valeurs du tirage, `op` est le caractère correspondant à une opération (`'+'`, `'-'`, `'*'` ou `'/'`) et `res` est le résultat de l'opération `x op y`.

  Attention, Il existe de nombreux cas dans lesquels une opération valide n'ajoute pas d'information au tirage. Par exemple, si `x * y` est une opération possible, il n'est pas nécessaire de calculer `y * x`. D'autre part, il est interdit de calculer avec des nombres négatifs. Faites une étude des différents cas d'opérations possibles (et utiles) et écrivez ensuite la fonction `operations_possibles` en tenant compte de vos observations.

  Exemple : `operations_possibles([1, 2, 3, 4]` doit renvoyer une liste de la forme `[(2, '+', 1, 3), (2, '-', 1, 1), ...]` contenant par exemple l'opération `(4, '/', 2, 2)`, mais pas `(2, '/', 4, 0.5)`.

- **Recherche exhaustive :** Écrivez une fonction `suites_possibles(tirage)`, où `tirage` est une liste de nombres. Cette fonction doit renvoyer la liste de tous les couples `(suite_d_operations, nombres)` tels que `suite_d_operation` est une liste d'opérations possibles depuis `tirage`, et `nombres` est la liste de nombres obtenue après avoir effectué cette suite d'opérations.

  Par exemple, `suites_possibles([1, 2, 3, 4]` pourrait renvoyer une liste commençant par :

  ```
  [([], [1, 2, 3, 4]),
   ([(2, '+', 1, 3)], [3, 4, 3]),
   ([(2, '+', 1, 3), (4, '+', 3, 7)], [3, 7]),
   ([(2, '+', 1, 3), (4, '+', 3, 7), (7, '+', 3, 10)], [10]),
   ...]
  ```

  En effet, depuis ce tirage initial, si l'on applique la suite vide d'opérations on obtient la liste de nombres inchangée `[1, 2, 3, 4]`, si l'on commence par additionner `2` et `1` on obtient `[3, 4, 3]`, si l'on additionne ensuite `3 et 4` on obtient `[3, 7]`, etc.

  Conseil : Il est recommandé d'écrire cette fonction récursivement en utilisant l'algorithme suivant :

  - Si le tirage contient moins de deux nombres, s'arrêter.
  - Sinon, pour chaque opération possible depuis le tirage courant :
    * Calculer la liste de nombres obtenue en effectuant l'opération ;
    * Chercher toutes les suites possibles à partir de cette nouvelle liste de nombres ;
    * En déduire la liste des suites possibles depuis le tirage courant (sans oublier d'inclure la suite déjà obtenue).

- **Solutions exactes :** Écrivez une fonction `solutions(tirage, cible)` qui renvoie la liste des suites d'opérations permettant d'atteindre exactement le résultat `cible`. Cette liste doit être triée par longueur (nombre d'opérations) croissante. Par exemple, `solutions([1, 2, 3, 4], 24)` pourrait renvoyer :

```
[[(3, '*', 2, 6), (6, '*', 4, 24)],
 [(4, '*', 2, 8), (8, '*', 3, 24)],
 [(4, '*', 3, 12), (12, '*', 2, 24)],
 [(2, '+', 1, 3), (3, '+', 3, 6), (6, '*', 4, 24)],
 [(3, '+', 1, 4), (4, '+', 2, 6), (6, '*', 4, 24)],
 [(3, '+', 1, 4), (4, '+', 2, 6), (6, '*', 4, 24)],
 [(3, '+', 2, 5), (5, '+', 1, 6), (6, '*', 4, 24)],
 [(4, '+', 2, 6), (3, '+', 1, 4), (6, '*', 4, 24)]]
```

- **Programme principal :** Modifiez le fichier `chiffres.py` afin que la ligne de commande `python3 chiffres.py` effectue les actions suivantes :
  - lecture sur l'entrée standard d'une suite de nombres séparés par des espaces, par exemple : `1 2 3 4`
  - lecture sur l'entrée standard d'un résultat à atteindre (par exemple `30`)
  - affichage des solutions au format donné dans l'exemple introductif (une solution par ligne, chacune sous la forme d'une suite d'opérations séparées par des virgules, par exemple `4+1=5, 3*2=6, 6*5=30`).

  Le programme ne devra **rien afficher d'autre** (pas d'invite de saisie, pas d'annonce des résultats). Par exemple :

```
dead-pony:code ameyer$ python3 chiffres.py
1 2 3 4
24
3*2=6 6*4=24
4*2=8 8*3=24
4*3=12 12*2=24
2+1=3 3+3=6 6*4=24
3+1=4 4+2=6 6*4=24
3+1=4 4+2=6 6*4=24
3+2=5 5+1=6 6*4=24
4+2=6 3+1=4 6*4=24
dead-pony:code ameyer$
```

- **Meilleures solutions :** Modifiez votre programme afin que les solutions soient affichées par longueur croissante (c'est à dire par nombre d'opérations croissant). Il se trouve que c'était déjà le cas dans l'exemple précédent, mais c'est un coup de chance.

3.4 Améliorations (partie optionnelle)

Si vous avez fini tout le reste, de nombreuses améliorations sont possibles. La liste (non exhaustive) suivante donne quelques suggestions. Essayez d'en implémenter le plus possible !

- Dans certains cas, il est impossible d'atteindre exactement le résultat demandé. Dans ce cas, la règle du jeu stipule que c'est le joueur qui s'en approche le plus qui remporte la manche. Modifiez votre programme pour qu'il affiche la ou les meilleures solutions, même si elles ne sont pas exactes.

- Ajoutez au programme une option permettant de chronométrer la recherche de solutions pour un tirage donné de nombres et de cible. Vous pourrez ainsi comparer vos résultats à ceux des autres binômes (sur le même tirage évidemment) et remporter le titre de solveur de compte est bon cosmique.
- En vous inspirant de la partie précédente, améliorez votre programme en lui ajoutant un menu interactif en mode texte et/ou des options en ligne de commande.
- Dans le jeu classique, les opérations autorisées sont l'addition, la soustraction, la multiplication et la division, et seuls les calculs sur les entiers positifs sont autorisés. Une amélioration possible consiste à autoriser de nouvelles opérations et à autoriser les nombres négatifs en cours de calcul. Quel impact remarquez-vous sur le temps de calcul ?
- ★ Si l'on s'intéresse aux solutions de longueur minimale, il peut être intéressant de modifier l'ordre dans lequel on énumère les suites d'opérations. On a vu au dernier cours une technique permettant, dans le problème de coloriage de zone, d'explorer la zone en traitant les pixels du plus proche au plus lointain du pixel d'origine.

  Adaptez cette technique au problème qui nous intéresse.

  Indication : vous pourrez utiliser une des classes de conteneurs vues en cours, ou le module python `collections.deque`.
- ★★ La résolution du problème en utilisant l'algorithme donné ci-dessus est très coûteuse en temps de calcul. Ceci est dû en partie au fait que de nombreuses suites d'opérations produisent le même résultat. Par exemple, depuis le tirage `1 2 3 4` donné plus haut, on peut atteindre le nombre `24` de 8 façons différentes, mais seules 6 d'entre elles sont réellement distinctes.

  Pour éviter un nombre important de calculs inutiles, une amélioration possible est d'intégrer un historique à la fonction `suites_possibles`, permettant de ne pas relancer des calculs si le tirage courant a déjà été traité auparavant.