[fix] one last mistake in instructions
Gerald Oster authored 13 hours ago

c610686e

README_FR.md 12.7 KB

CS'54 2023 - 2024 Examen d'algorithmique et de résolution de problèmes

L'examen comprend 3 exercices indépendants, de difficulté croissante.

Chaque exercice aboutit au développement d'un programme dans l'un des fichiers python fournis dans le répertoire de l'examen.

L'examen sera évalué de manière automatique au travers de tests appliqués à chacune des fonctions. NE CHANGEZ PAS LES SIGNATURES des FONCTIONS FOURNIES. Si vous avez besoin de fonction supplémentaires, donnez leur des noms différents et appelez les depuis les fonctions spécifiées.

NE TENTEZ PAS DE FRAUDER. La fraude aux examens entraîne l'invalidation immédiate de l'intégralité du module CS54 et est suivi d'une saisine automatique du conseil de discipline ouvrant la porte à des sanctions complémentaires.

> Votre mission, si vous l'acceptez, consiste à résoudre les trois exercices. Comme toujours, si vous ou un membre de votre équipe venait à ne pas réussir, le département niera avoir eu connaissance de vos agissements. Ce sujet s'auto-détruira dans les trois heures. Bonne chance!

Exercise 0: Quelques éléments complémentaires

Les conditions de l'examen seront les suivantes :

- L'examen se passera en salle 1.18 et 1.19
- Le seul matériel autorisé à garder sera de quoi écrire. Toutes les autres affaires personnelles, y compris les ordinateurs portables, les téléphones portables, les montres connectées, etc. seront laissées à l'entrée de la salle, contre le mur. Cela signifie notamment que si vous souhaitez avoir l'heure, il faudra penser à prendre un autre appareil qu'une montre connectée.
- Il se fera exclusivement sur les machines de l'école, avec un compte d'examen individuel spécifique, dont les informations de connexion vous seront données au début de l'examen. Les machines seront configurées avec les outils nécessaires, notamment le module Python de Visual Studio Code. Vous ne pourrez pas installer d'autres outils ou d'autres plugins que ceux déjà installés.
- Vous n'aurez ni accès à vos anciens codes, ni à votre compte habituel, ni même à internet. Seuls certains sites, comme la documentation officielle de Python, seront accessibles, le reste sera bloqué au niveau réseau. Vous n'aurez pas non plus accès à ChatGPT, Copilot ou autre IA.
- Les fichiers de code que vous devrez compléter ou créer seront dans un répertoire spécifique, nommé `EXAM_CS54` , à la racine de votre répertoire personnel. Seuls les fichiers présents dans ce répertoire `EXAM_CS54` seront pris en compte pour l'évaluation. Tout fichier en dehors de ce répertoire sera purement et simplement ignoré, sans aucune possibilité de contestation de votre part.
- La sortie de la salle avant l'examen est strictement interdite.

Exercise 1 : Chéri, j'ai rétréci les mots !

Rétrécir un mot consiste à retirer l'un de ses caractères de telle sorte à ce que le mot ainsi rétréci reste valide dans le dictionnaire. Prenons comme exemple le mot anglais `planet` qui est un mot valide du dictionnaire anglais. Rétrécir le mot `planet`, revient à enlever l'une de ses lettres, à savoir: `p`, `l`, `a`, `n`, `e` ou `t`. Si l'on retire `e`, le mot résultant est `plant`, toujours un mot valide dans le dictionnaire anglais. La propriété (rester un mot valide) est également vraie si on retire `t` à `planet`, la résultante étant `plane` (un avion ou un plan), un mot du dictionnaire officiel anglais.

Un mot super cool (Super Cool Word en anglais) est un pot qui peut être rétréci à un seul caractère car il existe un chemin dans lequel chacune des réductions intermédiaires forme un mot valide. Restons sur le mot `planet`, c'est un mot super cool car toutes ses réductions, y compris la dernière le réduisant à un caractère unique, sont des mots valides comme démontré ci-dessous :

- planet
- plant
- pant
- pan
- an
- a

`full` lui n'est pas un mot super cool dans notre dictionnaire, car il n'existe aucune séquence de réductions qui permet d'atteindre une lettre unique présente dans le dictionnaire.

Nous faisons l'hypothèse que les mots ne sont composés que de lettres minuscules. Votre mission est de construire l'algorithme "Super Cool Word" (en anglais dans le texte) ainsi que quelques functions associées pour faciliter la classification.

Tous les codes de cet exercice doivent être fournis dans le fichier python **`shrinking.py`** qui se trouve dans le répertoire d'examen. Toutes les signatures y sont déjà données.

Question 1.1: charger un dictionnaire (1 pt)

Le répertoire d'examen contient des fichiers de dictionnaires, l'un dénommé `sampleDictionary.txt`, un deuxième de 3000 mots dénommé `commonWords.txt` et un troisième comprenant 10000 mots nommé `mit10000.txt`. Le premier contient quelques mots afin que vous puissiez tester facilement votre code. Le second contient un sous ensemble (3000 mots) des mots qui sont valides dans le dictionnaire anglais, un mot par ligne; le troisième comprend 10000 mots du dictionnaire anglais. Ce sont ces dictionnaires qui nous serviront pour les tests.

Remplissez le code de la fonction ci-dessous afin qu'elle renvoie une liste de mots, chacun composé d'une liste de caractères. Par exemple, étant donné un dictionnaire comprenant les mots : `a`, `an` et `ban`, la fonction renvoie : `[['a'], ['a','n'], ['b','a','n']]`. L'ordre d'apparition des mots dans la liste n'est pas important.

```
def loadDictionary(fileName: str) -> list[list[str]]:
```

NOTE: si vous êtes dans l'incapacité d'écrire cette fonction, vous pouvez continuer l'examen en utilisant la variable `testDictionary` dans l'entrée principale de votre code. Celle-ci contient un dictionnaire minimaliste et est du même type que le dictionnaire renvoyé par la fonction de chargement d'un dictionnaire réel.

Question 1.2: vérifier l'appartenance (0,5 pt)

Ecrivez une fonction qui vérifie la présence d'un mot dans un dictionnaire. Le mot est donné comme une liste de caractères (ex. `['t','a','n']`) et le dictionnaire comme une liste de mots, chaque mot lui-même formé d'une liste de caractères.

```
def isMember(word: list[str], dictionary: list[list[str]]) -> bool:
```

Question 1.3: réduction suivante (1,5 pt)

Ecrivez une fonction qui, étant donné un mot, élabore tous les mots réduits (de longueur `len(word) - 1`) issus du mot paramètre sans vérification de la validité de ces mots vis-à-vis du dictionnaire. Tous les mots ainsi composés sont retournés dans une liste. Si aucune réduction n'est possible, la liste vide est retournée.

```
def nextLevelShrink(word: list[str]) -> list[list[str]]:
```

Question 1.4: mot super cool (3 pts)

Réalisez une fonction qui, étant donné un mot (sous forme d'une liste de caractères) et un dictionnaire (sous forme d'une liste de liste de caractères), renvoie `True` si le mot est super cool, `False` sinon.

```
def superCoolWord(word:list[str], dictionary: list[list[str]]) -> bool:
```

Question 1.5: super cool words (1 pts)

Ecrivez une fonction qui, étant donné un dictionnaire, renvoie tous les mots super cools de longueur `length` qu'il contient.

```
def superCoolDictionary(dictionary: list[list[str]], length: int) -> list[list[str]]:
```

Question 1.6: cool chain (1 pt)

Développez une fonction qui, étant donné un mot (sous la forme d'une liste de caractères) et un dictionnaire, renvoie la séquence de réduction (tous les mots de la chaîne qui le réduit à un caractère lui-même valide dans le dictionnaire) sous la forme d'une liste. Si le mot n'est pas super cool, une liste vide est renvoyée. En cas de succès la liste comprend le mot lui même puis sa réduction d'un caractère, puis la réduction suivante et ainsi de suite jusqu'à un caractère unique.

```
def superCoolChain(word: list[str], dictionary: list[list[str]]) -> list[list[str]]:
```

Exercise 2 : Chérie, j'ai perdu les signes!

Vous disposez d'une liste de `k` entiers positifs distincts (l'ordre d'apparition de ces valeurs ne compte pas) et une liste de `k` caractères qui peuvent être soit `P` soit `N` (ex. `['P', 'P', 'N', 'P']`) et qui décrivent le signe de la progression attendue de l'évaluation de l'expression. `P` signifie positif et `N` négatif. Votre objectif est de trouver une expression valide combinant les valeurs données dans la première liste avec les opérations autorisées : addition (+), soustraction (–) de telle sort que le résultat du calcul partiel de l'expression de longueur `i` est soit négatif (`N`) soit positif (`P`) comme demandé dans la liste des contraintes.

Prenons un exemple. Soient les données suivantes en entrée: `[19, 4, 21]` et `['P', 'N', 'P']` , les expressions suivantes sont valides :

- `4 – 19 + 21` car 4 est bien positif, 4-19 est négatif comme exprimé dans les contraintes, et, 4-19+21 est à nouveau positif comme exigé par les contraintes.
- `4 – 21 + 19` car 4 est positif, 4-21 est négatif comme attendu, et, 4-21+19 est à nouveau positif comme exigé par les contraintes.
- `19 – 21 + 4` car 19 est positif, 19-21 est négatif comme attendu, et, 19-21+4 est à nouveau positif comme exigé par les contraintes.

A titre de contre-exemple, `19 – 4 + 21` n'est pas une expression valide car `19 – 4` est positif alors qu'une expression négative est attendue après combinaison de deux entiers.

Attention: une expression valide peut parfaitement commencer par un nombre négatif. Par exemple, si les données d'entrée de votre fonction sont `[4, 2, 3]` et les contraintes `['N', 'P', 'N']` , des solutions valides sont:

- `– 2 + 3 – 4`
- `– 2 + 4 – 3`
- `– 3 + 4 – 2`

Vous l'avez !

Votre code doit être inséré dans le fichier python **`mixops.py`** qui se trouve dans votre répertoire d'examen.

Question 2.1 (7 pts)

Concevez une fonction qui, étant donnée une liste de valeurs et une liste de progression des signes, retourne une expression valide.

Le résultat est une liste d'entiers signés. Par exemple, sur les données ci-dessus, un résultat possible de cette fonction est `[4, –19, 21]` pour la première série de paramètres et `[–2, 3, –4]` avec les seconds paramètres.

Si aucune solution n'existe pour les paramètres donnés en entrée, la fonction retourne une liste vide.

```
def validExpression(values: list[int], constraints: list[str]) -> list[int] :
```

Exercise 3: Chéri, j'ai rétréci les mugs !

Il y a bien longtemps, dans une galaxie lointaine, très lointaine tous les élèves découvrirent la beauté de la récursion. Leurs compétences croissaient à un vitesse incroyable et ils réussirent à implémenter le fameux algorithme de résolution du problème des récipients. Pour toutes celles et ceux qui ont mené cette quête à son terme (et à tous les autres jeunes padawan), voici le bonus.

Vous avez à votre disposition un ensemble de mugs (identiques aux récipients mais mugs c'est plus cool, comme les mots de l'exercice 1 :-) de capacités différentes et un puit (dans lequel vous pouvez vider un mug à la fois). Il n'y a pas de source (ou de robinet) mais votre configuration commence avec l'un des mugs rempli avec l'extraordinaire cocktail "Flying Solo" adoré par les Jedis. Le mug rempli est donné comme un paramètre du problème par son index.

Question 3.1 (5 pts)

Soit un ensemble de capacités de mugs (une liste d'entiers), un volume cible et l'index du mug initialement plein (tous les mugs qui ne sont pas celui à l'index donné sont vides), écrivez une fonction de recherche en largeur d'abord qui recherche une solution au problème. La signature de la fonction est la suivante:

```
def mugshot(mugsCapacity: list[int], target: int, filledMugIndex: int, maxExplorations: int) -> list[list[int]]:
```

La fonction retourne la plus courte séquence de configurations (chaque configuration est une liste de valeurs entières représentant les volumes contenus dans chaque mugs) qui ont mené à la solution, y compris la solution elle même (en dernière position), une liste vide si aucune solution n'est trouvée. Les chemins explorés qui n'aboutissent pas ne font pas partie de la solution, tout comme les boucles (passer par une configuration déjà rencontrée).

Le premier paramètre indique la capacité maximale de chaque mug. Le deuxième indique la cible (le volume attendu dans l'un des mugs). Le troisième paramètre indique l'index du mug initialement rempli (tous les autres sont considérés vides). Le quatrième paramètre indique la profondeur maximale de l'arbre que vous êtes autorisés à explorer (le niveau de la racine est considéré comme une hauteur de 0).

Votre code doit être intégré dans un fichier dénommé **`mugshot.py`** qui existe déjà dans votre répertoire et qui contient la signature de test.