Informatique

2
Dictionnaires et programmation dynamique

TD2-3

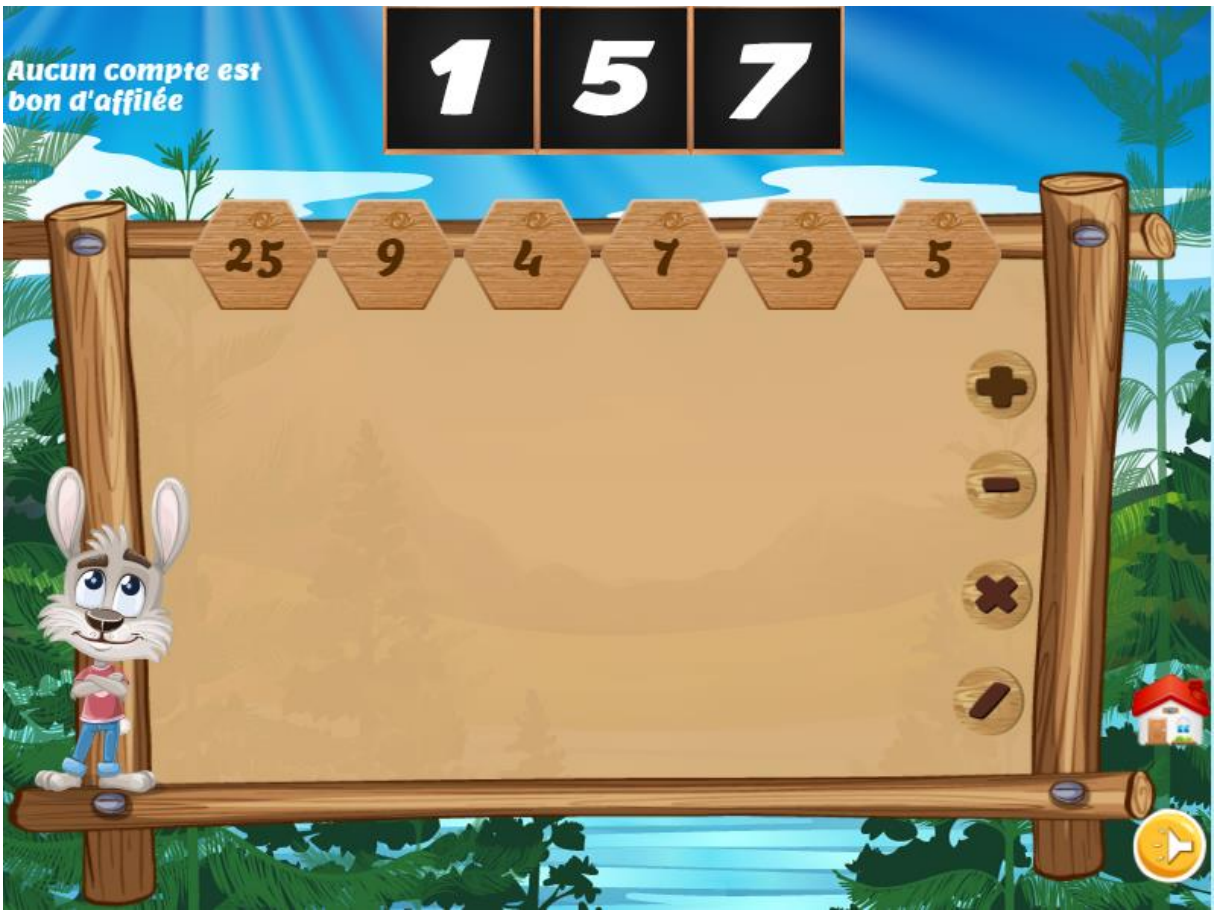
Le compte est bon

Le jeu

Vous connaissez forcément ce jeu qui consiste à trouver un nombre à l'aide d'une liste de nombres prédéfinie, en utilisant les 4 opérateurs classiques +, -, x et /. Par exemple, trouver $V = 928$ avec $L = [50,25,5,3,6,5]$.

Allez vous amuser ici si vous le souhaitez, mais pas trop longtemps 😊

LIEN



Objectifs

Nous allons dans ce sujet réaliser différents algorithmes permettant de voir si une solution existe, la trouver, et ce avec des parcours en profondeur avec mémoïsation (exercice 2) et en largeur avec stockage des antécédents dans un dictionnaire, puis une remontée du dictionnaire pour obtenir une solution parmi les existantes, si elle existe (exercice 3).

Exercice 1: Préliminaires

Détermination des successeurs d'une situation

On rappelle que python propose la fonction eval(str) prenant en argument une chaine de caractères et évaluant son résultat mathématique.

Question 1: Créer la fonction Operation(L,a,b,op) prenant en argument une liste L, deux entiers a et b, et op un opérateur +-*/ sous forme de chaine de caractères, qui ajoute dans L en place, la chaine de caractères « aopb » si le résultat de cette opération correspond à un entier

Vérifier les exemples suivants :

```
>>> L = []
>>> Operation(L,2,3,"+")
>>> L
['2+3']
```

```
>>> L = []
>>> Operation(L,2,3,"/")
>>> L
[]
```

```
>>> L = []
>>> Operation(L,4,2,"/")
>>> L
['4/2']
```

Remarques :

- Si on ne limite les pas résultats aux entiers dans cette fonction, l'algorithme trouvera par exemple 206=4*(50 + 3/2) là où avec la limitation, il pourra donner aussi 206=2*3+4*50
- J'ai évité les doublons inutiles dans cette fonction en n'insérant pas un résultat dont la chaine de caractères (2*2, ~~2*2~~ ou 2/2, ~~2/2~~) ou le résultat (2+2, ~~2*2~~ ou 2+0, ~~2-0~~) est déjà présent.

Question 2: Créer la fonction Operations(a,b) prenant en argument deux entiers a et b, et renvoyant la liste de toutes les opérations possibles entre a et b sous forme de chaines de caractères

Vérifier les exemples suivants :

Avec doublons	Sans doublons inutiles

`>>> Operations(2,3)`
`['2+3', '2-3', '3-2', '2*3']`
 	
`>>> Operations(2,4)`
`['2+4', '2-4', '4-2', '2*4', '4/2']`
 	
`>>> Operations(2,2)`
`['2+2', '2-2', '2-2', '2*2', '2/2', '2/2']`
 	
`>>> Operations(2,0)`
`['2+0', '2-0', '0-2', '2*0', '0/2']`
	`>>> Operations(2,3)`
	`['2+3', '2-3', '3-2', '2*3']`
	
	`>>> Operations(2,4)`
	`['2+4', '2-4', '4-2', '2*4']`
	
	`>>> Operations(2,2)`
	`['2+2', '2-2', '2/2']`
	
	`>>> Operations(2,0)`
	`['2+0', '0-2', '2*0']`

Remarque : Pour obtenir les mêmes résultats que moi dans la suite, je vous suggère d'organiser les opérations dans le même ordre et de supprimer les doublons inutiles.

On souhaite maintenant créer l'ensemble des possibilités d'action à partir d'une liste d'entiers L donnée. Pour cela, on souhaite qu'à partir de L soit renvoyée une liste de listes de deux termes correspondant à :

- Premier élément : La liste obtenue après l'opération réalisée triée par ordre croissant (utiliser `L.sort()` ou `sorted(L)`) afin de facilement déceler des situations identiques dans la suite
- Second élément : l'opération réalisée sous forme de chaine de caractères

Exemples :

```
>>> Successeurs([2,3])
[[[5], '2+3'], [[-1], '2-3'], [[1], '3-2'], [[6], '2
*3']]

>>> Successeurs([2,5,10])
[[[7, 10], '2+5'], [[-3, 10], '2-5'], [[3, 10], '5-2
'], [[10, 10], '2*5'], [[5, 12], '2+10'], [[-8, 5],
'2-10'], [[5, 8], '10-2'], [[5, 20], '2*10'], [[5, 5
], '10/2'], [[2, 15], '5+10'], [[-5, 2], '5-10'], [[
2, 5], '10-5'], [[2, 50], '5*10'], [[2, 2], '10/5']]
```

Question 3: Créer la fonction Successeurs(L) prenant en argument une liste d'entiers L, et renvoyant la liste de listes attendue

Vérifiez évidemment que vous obtenez les mêmes résultats.

Exercice 2: Première approche récursive

Dans un premier temps, on se propose de réaliser une fonction récursive faisant une exploration des possibilités comme un parcours en profondeur (incomplet), afin de rechercher s'il est possible de trouver une valeur V à partir de tous les éléments d'une liste L.

Question 1: Proposer une fonction récursive lceb1(L,V) renvoyant True si V est atteignable depuis L, False sinon

Vérifier :

```
>>> L = [1,2,3,4,5]

>>> V = 63

>>> lceb1(L,V)
True

>>> V = 1000

>>> lceb1(L,V)
False
```

On souhaite améliorer la fonction lceb1 afin qu'elle renvoie, en plus de True ou False, la chaine de caractères indiquant les opérations à réaliser pour trouver V avec tous les éléments de L.

Question 2: Proposer la fonction améliorée lceb2(L,V)

Vérifier :

```
>>> L = [1,2,3,4,5]

>>> V = 63

>>> lceb2(L,V)
(True, '1+2=3 ; 3*4=12 ; 5*12=60 ; 3+60=63 ; ')
```

Question 3: Etudier le temps mis par cette fonction pour trouver un nombre impossible (np.inf) dans une liste de 1 à 6 termes

On souhaite encore améliorer la fonction en utilisant la technique de mémoïsation à l'aide d'un dictionnaire. Pour cela, vous créerez une fonction récursive rec(L,V) dans la fonction lceb3 qui utilisera un dictionnaire local mémorisant les situations déjà étudiée et leurs résultats, et ne faisant que renvoyer le résultat d'une situation L si déjà trouvée.

Question 4: Créer la fonction lceb3 avec mémoïsation et comparer les temps d'exécution de lceb3 avec lceb2 sur les mêmes exemples

On souhaite finalement améliorer lceb3 afin de renvoyer le résultat s'il est trouvé avant d'avoir utilisé tous les éléments de L.

Question 5: Mettre en place la fonction lceb4(L,V) permettant d'obtenir un calcul n'utilisant pas forcément tous les termes de L

Essayez :

```
>>> L = [1,2,3,4,5]

>>> V = -1

>>> lceb4(L,V)
(True, '1+2=3 ; 3+3=6 ; 4-5=-1 ; ')
```

Vous remarquerez que des calculs inutiles peuvent apparaître. En effet, -1 aurait pu être atteint avec un seul premier calcul 1-2, 2-3, 3-4 ou 4-5. Notre algorithme a réalisé le calcul 4-5 avant les autres et s'est arrêté.
Malheureusement, il n'est pas possible d'obtenir ici le plus court chemin (nombre de calculs) vers le résultat, le parcours en largeur de la suite le permettra.

Exercice 3: Graphe et parcours en largeur

Il est possible de parcourir le graphe en largeur et en profondeur pour créer un dictionnaire le représentant. Ce que nous avons fait dans l'exercice 1 est semblable au parcours en profondeur. Nous allons donc ici réaliser un parcours complet en largeur. L'intérêt de réaliser

- Un parcours complet : Pouvoir obtenir les différentes valeurs atteignables depuis une liste L
- Un parcours en largeur : Les solutions qui seront mémorisées pour atteindre V à partir de L seront les premières solutions atteintes, soit les plus courtes. Si cela ne change rien quand on cherche les valeurs V atteintes avec tous les termes de L, cela devient intéressant lorsque l'on obtient V avant d'avoir utilisé tous les éléments de L.

Réalisation du graphe du jeu

On souhaite réaliser un graphe G dont :

- Les **clés** sont les situations du jeu (liste triée d'entiers L) transformées en tuple afin d'être hachables
- Les **valeurs** sont des listes à deux termes correspondant à :
  - Terme 1 : Situation ayant mené à la clé sous forme de liste (liste triée d'entiers)
  - Terme 2 : Opération réalisée ayant mené à la clé sous forme de chaine de caractères

On veillera à initialiser le graphe avec la liste `[[],'RAS']` pour la clé associée à L0 (tuple trié).

Exemple de graphe :

```
>>> Graphe([1,2])
{(1, 2): [[], 'RAS'], (3,): [[1, 2], '1+2'], (-1,): [[1, 2]
, '1-2'], (1,): [[1, 2], '2-1'], (2,): [[1, 2], '1*2']}
```

Pour réaliser ce graphe, vous réaliserez un parcours en largeur des situations possibles du jeu à partir d'une liste L0. Vous penserez à utiliser une collection deque afin d'améliorer la complexité du parcours.

Question 1: Proposer la fonction Graphe(L0) réalisant le travail demandé

Vérifiez évidemment que vous obtenez les mêmes résultats.
Vous pourrez regarder la taille du graphe réalisé en écrivant `len(G)`.

Résolution

Prenons dans un premier temps une situation rapide :

```
L0 , V = [2,3,4,50] , 206
```

On se limitera dans un premier temps aux solutions qui utilisent l'intégralité des nombres proposés dans L.

Question 2: Proposer un code qui permet de résoudre le jeu en créant puis remontant le dictionnaire tout en affichant les calculs à réaliser dans la console

Vérifiez :

```
Liste disponible:  [1, 50, 100]
Total attendu:  3

Solution:
100/50 = 2
1+2 = 3
```

```
Liste disponible:  [50, 25, 5, 3, 6, 5]
Total attendu:  928

Solution:
50+3 = 53
5*6 = 30
5+30 = 35
25*35 = 875
53+875 = 928
```

Remarque : Plusieurs chemins peuvent mener à la solution. Notre programmation n'en renvoie qu'un.

Vous pourrez transformer votre code en une fonction Resolution(Gr,L0,V) prenant en argument le graphe Gr, la liste L0 et la valeur recherchée V. Cela permettra de l'appeler avec le prochain graphe.

Question 3: Créer et utiliser la fonction Possibles(L) renvoyant la liste de toutes les possibilités de résultats réalisables avec une liste L de départ en utilisant tous les nombres proposés

Vérifier :

```
Liste: [2, 5, 10]
Possibilités [17, -3, 3, 70, 7, -13, 13, -30, -7, 30, 20, 0,
100, 1, 60, -40, 40, 25, -15, 15, 4, 10, -10, 52, -48, 48]
```

On souhaite maintenant considérer aussi les solutions atteintes lorsque tous les nombres de L0 n'ont pas été utilisés.

Question 4: Proposer une nouvelle fonction Graphe_Bis permettant de renvoyer le résultat si le nombre recherché dans le jeu est atteint avant d'avoir utilisé tous les nombres disponibles dans L0

L'initialisation de tous les termes de L0 avec `[[],'RAS']` est primordiale ici pour renvoyer un résultat correct quand on demande l'un des nombres de la liste initiale (même si cela n'a pas beaucoup d'intérêt).

En utilisant cette nouvelle fonction, vérifier :

```
Liste disponible:  [1, 50, 100]
Total attendu:  51

Solution:
1+50 = 51
```

```
Liste disponible:  [1, 50, 100]
Total attendu:  1

Solution:
```

Question 5: Utiliser la fonction Possibles(L) pour afficher la liste de toutes les possibilités de résultats réalisables avec une liste L de départ en utilisant un ou plusieurs éléments de L

Vérifier :

```
Liste: [2, 5, 10]
Possibilités [2, 5, 10, 7, -3, 3, 12, -8, 8, 20, 15, -5, 50,
17, 70, -13, 13, -30, -7, 30, 0, 100, 1, 60, -40, 40, 25, -1
5, 4, -10, 52, -48, 48]
```

Allez maintenant vous amuser à exploser les scores du jeu en ligne ici :

1.	wawa	59	Aujourd'hui à 10h43
2.	Bronleur	25	Aujourd'hui à 07h52
3.	eric	24	Hier à 16h39
4.	mk	20	Hier à 22h27
5.	Petit dauphin 633	17	Hier à 16h47
6.	Mizoumazen	16	Hier à 16h54
7.	Petit dauphin 435	16	Aujourd'hui à 15h26
8.	Petit dauphin 776	14	Hier à 18h07
9.	Petit dauphin 42	14	Aujourd'hui à 09h49
10.	Petit dauphin 480	12	Hier à 21h15