TP3_BonCompte_python

March 20, 2022

1 TP d'Algorithmique Semestre2

Université Polytechnique des Hauts-De-France / INSA HdF 1ere année

---

1.1 Recherche aléatoire

L'objectif de ce TP est de de créer un programme qui permet de trouver, à partir de six nombres entiers, et des 4 opérations entières, les étapes de calculs permettant d'atteindre un nombre objectif, sachant qu'un nombre ne peut être utilisé qu'une seule fois.

- Exemple de sortie d'un algorithme :

A partir des opérations sur les nombres [10 5 75 4 6 1] le but est d'obtenir 654 .... Fin de recherche, valeur trouvée = 654 Nb d'essais effectues = 2580 Détail des opérations : 75 + 10 = 85 5 + 1 = 6 4 * 6 = 24 24 + 85 = 109 6 * 109 = 654

Ce TP utilise un type particulier de recherche de solution qui est la recherche aléatoire (ou méthode de Monte-Carlo).

A chaque étape : - on prends 2 nombres au hasard (a et b) parmi les restants (2 parmi 6 au début), - on sélectionne aléatoirement une opération op compatible; c'est à dire que a op b doit donner un nombre entier strictement positif. *Par exemple, si a=5 et b=75, les opération - et / ne sont pas valides, seules + ou x peuvent être sélectionnées.* - (a et b) sont extraits du tableau, - le résultat de a op b est ajouté au tableau.

On répète 5 fois ce processus.. Par exemple, cela peut donner :

Etape 1 tab = [10, 5, 75, 4, 6, 1]
tirage de a et b (a=5, b=75) —> tab = [10, 1, 6, 4, 0, 0], tirage de op (+) -> tab = [10 4 6 1 80 0], Etape 2 tab = [10, 1, 6, 4, 80, 0]
tirage de a et b (a=80, b=6) —> tab = [10, 1, 4, 0, 0, 0], tirage de op (-) -> tab = [10, 1, 4, 74, 0, 0], Etape 3 tab = [10, 1, 4, 74, 0, 0]
tirage de a et b (a=10, b=4) —> tab = [74, 1, 0, 0, 0, 0], tirage de op (x) -> tab = [74, 1, 40, 0, 0, 0], Etape 4 tab = = [74, 1, 40, 0, 0, 0]
tirage de a et b (a=40, b=1) —> tab = [74, 0, 0, 0, 0, 0], tirage de op (-) -> tab = [74, 0, 0, 0, 0, 0], Etape 5 tab = [74, 0, 0, 0, 0, 0]
tirage de a et b (a=74, b=39) —> tab = [0, 0, 0, 0, 0, 0], tirage de op (x) -> tab = [2886, 0, 0, 0, 0, 0],

On s'arrête donc au pire au bout de 5 étapes, ou dès que le but est trouvé. Ici, l'objectif (654) n'étant pas atteint, on réitère l'ensemble de l'algorithme précédent; tant que la solution n'est pas trouvée (ce qui suppose qu'il y en a forcément une) et.ou tant que le nombre d'essais max n'est pas atteint.

On utilisera une structure `Calcul` qui contient la valeur obtenue suite à un essai de résolution, ainsi que la suite des étapes effectuées pour obtenir cette valeur. On placera en tête du fichier python :

[1]:
```python
from numpy import  array,zeros
from typing import Iterable
from random import randint

##Les Types
TabEntiers = Iterable[int]
Texte = (str,15)
TabTexte = Iterable[str]

class Calcul:
    etapes:TabTexte=None
    valeur:int=0

##Les constantes
SIGNES:TabTexte=array(["+", "*", "-", "//"])
```

1.2 Travail à faire

1. **Définir la procédure** `copier_tab(src:TabEntiers, dest:TabEntiers,nb:int)` qui copie le contenu des `nb` cases du tableau `src` dans le tableau `dest`

[2]:
```python
#def copier_tab(src:TabEntiers, dest:TabEntiers,nb:int):
    # A FAIRE
```

2. **Définir la fonction** `extraire(tab:TabEntiers, i:int, pos_fin:int)->int` qui retourne la ième valeur de `tab`, et la supprime du tableau, `pos_fin` étant l'indice de la derniere valeure non nulle. Au plus simple, elle remplace cette ième valeur par celle en position finale, et place un zéro en position finale. *Exemple si* `tab=[6,5,7,3,2,0]`, `extraire(tab, 2, 4)` *retourne* `7` *et modifie tab qui devient* `tab=[6,5,2,3,0,0]`

[3]:
```python
#def extraire(tab:TabEntiers, i:int, pos_fin:int)->int:
    # A FAIRE"""retourne la ieme valeur de tab,
```

[4]:
```python
# testez :
# tab:TabEntiers = [1,2,3,4,5,6,7]
# v:int = extraire(tab, 2, 6)
# print(v, tab)
```

```
3 [1, 2, 7, 4, 5, 6, 0]
```

3. **Définir la fonction `verifier(op:str, a:int, b:int)->bool`** qui vérifie que l'opérateur op sur a et b est faisable ou utile en respectant les contraintes suivantes :

- Si op="//" : si division par 1, ou résultat non entier, retourner `False`
- Si op="*" : si multiplication par 1, retourner `False`
- Si op="-", si résultat négatif, retourner `False`
- Dans tous les autres cas, la procédure retourne `True`

[5]:
```
# def verifier(signe:str, a:int, b:int)->bool:
    # A FAIRE
```

[6]:
```
# Tester
# a:int = 1
# b:int = 2
# print(verifier("+", a, b),  verifier("-", a, b), verifier("*", a, b),␣
 ↪verifier("/", a, b))
# a:int = 5
# b:int = 4
# print(verifier("+", a, b),  verifier("-", a, b), verifier("*", a, b),␣
 ↪verifier("/", a, b))
```

```
True False False False
True True True False
```

4. **Définir la fonction `calculer(op:str, a:int, b:int)->int`** qui retourne le résultat entier de l'opération a op b. *Ex. calculer("+", 4, 5) retourne 9, ...,calculer("/", 984, 8) retourne 123.*

[7]:
```
# def calculer(signe:str, a:int, b:int)->int:
    # A FAIRE
```

[8]:
```
#Tester
# print(calculer("+",4,5))
# print(calculer("/",984,24))
```

```
9
41
```

5. **Définir la fonction `choisir_operateur(a:int, b:int)->str`** qui pioche un opérateur applicable sur a et b (passé par l'étape 'vérifier') et le retourne.

- La fonction python `randint(low,top)` retourne un entier pris au hasard entre low et top inclus.
- Il s'agit donc de piocher un nombre x en 0 et 3, de récupérer le signe correspondant (SIGNES[x]), et de vérifier que le signe est compatible avec a et b avant de le retourner.

[9]:
```
from random import randint

# def choisir_operateur(a:int, b:int)->str:
    # A FAIRE
```

[10]:
```
# Tester
# print(choisir_operateur(4,4))
# print(choisir_operateur(4,4))
# print(choisir_operateur(4,4))
```

+
*
*

Définir la procédure `saisir_produits(nb:int, tabPrixNotes:MatReels)` qui permet de saisir 'nb' produits en faisant appel à la procédure précédente

---

On dispose maintenant d'une fonction permettant d'extraire des valeurs d'un tableau, d'une fonction permettant de choisir au hasard une opération compatible avec les valeurs, d'une fonction permettant de calculer le résultat de l'opération sur les valeurs extraites.

Il « ne reste plus qu'à » rédiger en python l'algorithme qui permet un essai de résolution.

Cet algorithme est le suivant (`tab_num` est le tableau d'entiers contenant les nombres de base, `but` est le nombre entier à atteindre) :

[26]:
```
def essayer_calcul(tab_num:TabEntiers, but:int)->Calcul:
    """tente des opérations au hasard sur les nombres prix au hasard
    jusqu'à ce qu'il n'en reste qu'un.
    retourne la chaine menant au calcul s'il arrive au but"""
    calcul = Calcul()
    calcul.etapes = zeros(5, Texte)
    #travailler sur une copie du tableau des nombres
    copie_num:TabEntiers = zeros(6,int)
    copier_tab(tab_num, copie_num, 6)
    j:int=0
    for nb in range(5,0,-1):
        # prendre une valeur entre 0 et nb
        i=randint(0, nb)
        a:int=extraire(copie_num, i, nb)
        i:int=randint(0, nb-1)
        # prendre une valeur entre 0 et nb-1
        b:int=extraire(copie_num, i, nb-1)
        # choisir une operation applicable sur a et b
        signe = choisir_operateur(a,b)
        # lancer le calcul de l'operation
        resultat = calculer(signe,a,b)
        # classer le resultat en fin de tableau
        copie_num[nb-1] = resultat
        # noter le détail de l'operation dans le tableau du calcul
        calcul.etapes[j] = str(a) + " "+ signe + " " + str(b) + " = " +␣
↪str(resultat)
        j = j+1
```

```
        # mettre a jour le resultat actuel dans le calcul
        calcul.valeur = resultat
        # sortir si le but est trouve
        if(resultat==but): return calcul
    return calcul
```

[27]:
```
#Tester
# c:Calcul = essayer_calcul([12,23,34,45,56,65], 77)
# print(c.valeur, " trouvé par ", c.etapes)

# c  = essayer_calcul([12,23,34,45,56,65], 77)
# print(c.valeur, " trouvé par ", c.etapes)
```

```
173316  trouvé par  ['56 - 23 = 33' '34 * 65 = 2210' '2210 + 12 = 222' '45 + 33
= 78'
 '2222 * 78 = 173']
112039200  trouvé par  ['23 + 34 = 57' '45 * 57 = 2565' '2565 * 56 = 143' '12 *
65 = 780'
 '143640 * 780 = ']
```

Bien sûr il est très peu probable que la solution soit trouvée en un seul essai. Il faut donc lancer plusieurs fois la fonction essayer_calcul(tab_num:TabInt, but:int)->Calcul tant que le but n'est pas atteint ou tant que le nombre d'essais n'atteint pas un seuil défini.

6. **Définissez la procédure `lancer_essais()`** qui définit le but à atteindre, les nombres à utiliser et lance plusieurs fois `essayer_calcul` tant que l'objet calcul retourné ne possède pas la même valeur que celle du but ou tant que le nombre d'essais maximum (100000 par exemple) n'est pas atteint.

Bonus ! (+ 3 points) Lorsque le nombre d'essais maximum est atteint sans que la solution soit trouvée, afficher la meilleure des solutions (celle dont la valeur trouvée était la plus proche du but recherché).

[28]:
```
# def lancer_essais(but:int, tab_num:TabEntiers):
    # A FAIRE
```

Testez avec différents tirages : - Exemple : [1,10,4,5,3,50] avec but = 668 - Exemple : [1,2,4,8,10,25] avec but = 789

[29]:
```
lancer_essais(668, [1,10,4,5,3,50] )
```

```
A partir des operations sur les nombres  [1, 10, 4, 5, 3, 50]
le but est d'obtenir  668
--
Fin de recherche, valeur trouvee =  668
Nb d'essais effectues =  18272
Détail des operations :
10 + 3 = 13
1 + 50 = 51
```

```
51 * 13 = 663
5 + 663 = 668
```

[30]:
```
lancer_essais(789, [1,2,4,8,10,25] )
```

```
A partir des operations sur les nombres  [1, 2, 4, 8, 10, 25]
le but est d'obtenir  789
--
Fin de recherche, valeur trouvee =  789
Nb d'essais effectues =  3346
Détail des operations :
10 + 1 = 11
25 * 8 = 200
200 * 4 = 800
800 - 11 = 789
```

[ ]:
```

```