N. S. I. Term. - DS05
jeudi 14 décembre 2023
Calculatrice non autorisée
Durée 1 heure

Toutes les réponses sont à donner sur la copie

Exercice 1. Piles

Exercice 2. Files

Exercice 3. Piles (à faire pour lundi)

EXERCICE 2 (4 points)

Cet exercice porte sur les structures de données.

La poussette est un jeu de cartes en solitaire. Cet exercice propose une version simplifiée de ce jeu basée sur des nombres.

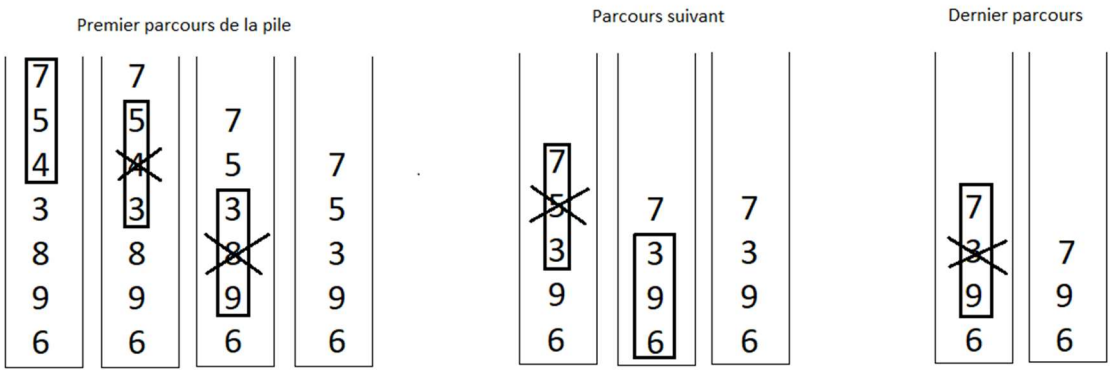
On considère une pile constituée de nombres entiers tirés aléatoirement. Le jeu consiste à réduire la pile suivant la règle suivante : quand la pile contient du haut vers le bas un triplet dont les termes du haut et du bas sont de même parité, on supprime l'élément central.

Par exemple :

- Si la pile contient du haut vers le bas, le triplet 1 0 3, on supprime le 0.
- Si la pile contient du haut vers le bas, le triplet 1 0 8, la pile reste inchangée.

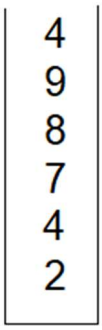
On parcourt la pile ainsi de haut en bas et on procède aux réductions. Arrivé en bas de la pile, on recommence la réduction en repartant du sommet de la pile jusqu'à ce que la pile ne soit plus réductible. Une partie est « gagnante » lorsque la pile finale est réduite à deux éléments exactement.

Voici un exemple détaillé de déroulement d'une partie.

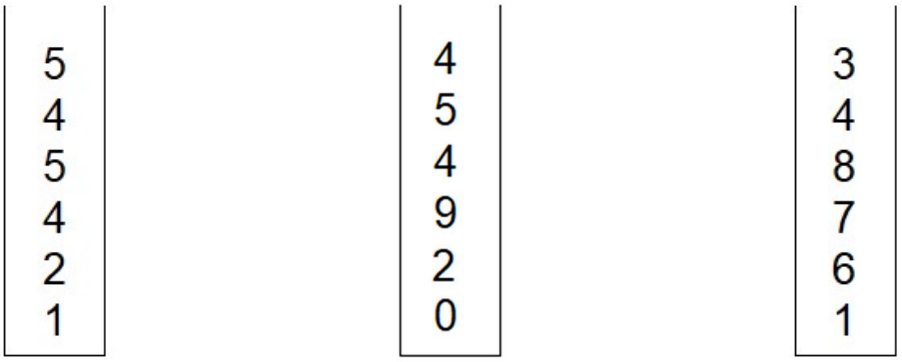


1.

a. Donner les différentes étapes de réduction de la pile suivante :



b. Parmi les piles proposées ci-dessous, donner celle qui est gagnante.



Pile A Pile B Pile C

L'interface d'une pile est proposée ci-dessous. On utilisera uniquement les fonctions figurant dans le tableau suivant :

Structure de données abstraite : Pile
• `creer_pile_vide()` renvoie une pile vide
• `est_vide(p)` renvoie `True` si `p` est vide, `False` sinon
• `empiler(p, element)` ajoute `element` au sommet de `p`
• `depiler(p)` retire l'élément au sommet de `p` et le renvoie
• `sommet(p)` renvoie l'élément au sommet de `p` sans le retirer de `p`
• `taille(p)` : renvoie le nombre d'éléments de `p`

2. La fonction `reduire_triplet_au_sommet` permet de supprimer l'élément central des trois premiers éléments en partant du haut de la pile, si l'élément du bas et du haut sont de même parité. Les éléments dépilés et non supprimés sont replacés dans le bon ordre dans la pile.
Recopier et compléter sur la copie le code de la fonction `reduire_triplet_au_sommet` prenant une pile `p` en paramètre et qui la modifie en place. Cette fonction ne renvoie donc rien.

```
1  def reduire_triplet_au_sommet(p):
2      a = depiler(p)
3      b = depiler(p)
4      c = sommet(p)
5      if a % 2 != .... :
6          empiler(p, ...)
7      empiler(p, ...)
```

3. On se propose maintenant d'écrire une fonction `parcourir_pile_en_reduisant` qui parcourt la pile du haut vers le bas en procédant aux réductions pour chaque triplet rencontré quand cela est possible.

a. Donner la taille minimale que doit avoir une pile pour être réductible.

b. Recopier et compléter sur la copie :

```
1  def parcourir_pile_en_reduisant(p):
2      q = creer_pile_vide()
3      while taille(p) >= ....:
4          reduire_triplet_au_sommet(p)
5          e = depiler(p)
6          empiler(q, e)
7      while not est_vide(q):
8          ............
9          ............
10      return p
```

4. Partant d'une pile d'entiers `p`, on propose ici d'implémenter une fonction récursive `jouer` renvoyant la pile `p` entièrement simplifiée. Une fois la pile parcourue de haut en bas et réduite, on procède à nouveau à sa réduction à condition que cela soit possible. Ainsi :

- Si la pile `p` n'a pas subi de réduction, on la renvoie.
- Sinon on appelle à nouveau la fonction `jouer` , prenant en paramètre la pile réduite.

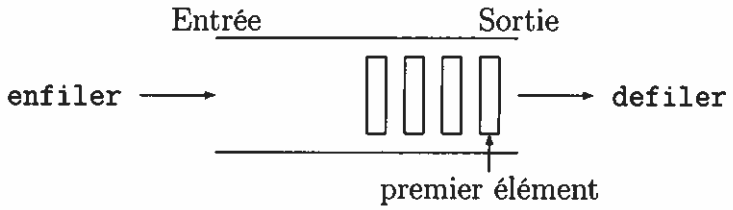
Recopier et compléter sur la copie le code ci-dessous :

```
1  def jouer(p):
2      q = parcourir_pile_en_reduisant(p)
3      if ............. :
4          return p
5      else:
6          return jouer(...)
```

Exercice 5 (4 points).

Cet exercice porte sur les files, les tableaux et les algorithmes associés.

L'objectif de cet exercice est de travailler sur les températures relevées par une station météorologique. Les données sont enregistrées une fois par jour, à la même heure, et traitées dans l'ordre dans lequel elles arrivent.
On choisit d'utiliser une file : une file est une structure de données abstraite fondée sur le principe « premier arrivé, premier sorti ».



On munit la structure de données `File` des quatre fonctions primitives définies ci-dessous :

Structure de données abstraite : File

Utilise : Élément, Booléen

Opérations :

- `creer_file_vide` : $\emptyset \longrightarrow$ `File`
  `creer_file_vide()` renvoie une file vide
- `est_vide` : `File` $\longrightarrow$ `Booléen`
  `est_vide(F)` renvoie `True` si la file `F` est vide, `False` sinon
- `enfiler` : `File, Élément` $\longrightarrow \emptyset$
  `enfiler(F, element)` ajoute `element` en entrée de la file `F`
- `defiler` : `File` $\longrightarrow$ `Élément`
  `defiler(F)` renvoie l'élément en sortie de la file `F` (premier élément) en le retirant de la file `F`

1. Les températures relevées ont été 15, puis 17, puis 14.

   (a) Parmi les quatre propositions suivantes, indiquer celle qui représente correctement cette file :

   Proposition 1 : Entrée 15 17 14 Sortie — Le premier élément est 14

   Proposition 2 : Entrée 14 17 15 Sortie — Le premier élément est 15

   Proposition 3 : Entrée 15 17 14 Sortie — Le premier élément est 15

   Proposition 4 : Entrée 14 17 15 Sortie — Le premier élément est 14

   (b) En utilisant les fonctions primitives précédentes, donner les instructions permettant de créer cette file.

2. On appelle longueur d'une file le nombre d'éléments qu'elle contient.

   La fonction `longueur_file` prend en paramètre une file `F` et renvoie sa longueur `n`.

   Après appel de cette fonction, la file `F` doit avoir retrouvé son état d'origine.

   Exemple :

   Si F = 10 10 12 12 alors `longueur_file(F)` vaut 4.

   Recopier et compléter le programme Python suivant, implémentant la fonction `longueur_file`.

   Dans le code de la fonction, les trois points (...) peuvent correspondre à une ou plusieurs lignes de programme.

```
def longueur_file(F):
    """File -> Int"""
    G = creer_file_vide() # file temporaire
    n = 0 # initialisation du nombre d'elements
    while not(est_vide(F)):
        ...
    while not(est_vide(G)): # reconstruction de la file initiale
        ...
    return ...
```

3. On s'intéresse à la variation de température d'un jour sur l'autre.

   Par exemple, lorsque les températures relevées sont dans l'ordre d'arrivée 15, 17 et 14, les variations sont 2 et -3.

   Recopier et compléter le programme Python implémentant la fonction `variations` qui prend en paramètre une file non vide `F` et qui renvoie le tableau `tab` contenant les variations successives, ou un tableau vide si la file `F` ne contient qu'une seule température. Il n'est pas demandé ici que la file `F` retrouve son état d'origine après appel de la fonction `variations`.

   Exemple : si `F` est la file qui contient dans l'ordre des relevés les valeurs 15, 17 et 14, `variations(F)` vaut `[2, -3]`.

   Dans le code de la fonction, les trois points (...) peuvent correspondre à une ou plusieurs lignes de programme.

```
def variations(F):
    """File -> Tableau"""
    taille = longueur_file(F)
    if taille == 1:
        ...
    else:
        tab = [0 for k in range(taille - 1)]
        element1 = defiler(F)
        for i in range (taille - 1):
            element2 = defiler(F)
            ...
    return ...
```

4. Écrire une fonction `nombre_baisses` qui prend en paramètre un tableau `tab`, non vide, des variations des températures et qui renvoie un p-uplet contenant le nombre de jours où la

température a baissé par rapport au jour précédent (soit le nombre de valeurs strictement négatives de `tab`), ainsi que la baisse journalière la plus importante (soit la valeur minimale de `tab`).

S'il n'y a aucune baisse (toutes les valeurs de `tab` sont positives), la fonction renvoie le p-uplet `(0,0)` .

Exemple 1 : `nombre_baisses([1, -4, 2, -1, 3])` vaut `(2,-4)`.

Exemple 2 : `nombre_baisses([1, 5, 3, 1])` vaut `(0,0)`.

[exercice 3]

EXERCICE 1 (4 points)

Cet exercice composé de deux parties A et B, porte sur les structures de données.

Partie A : Expression correctement parenthésée

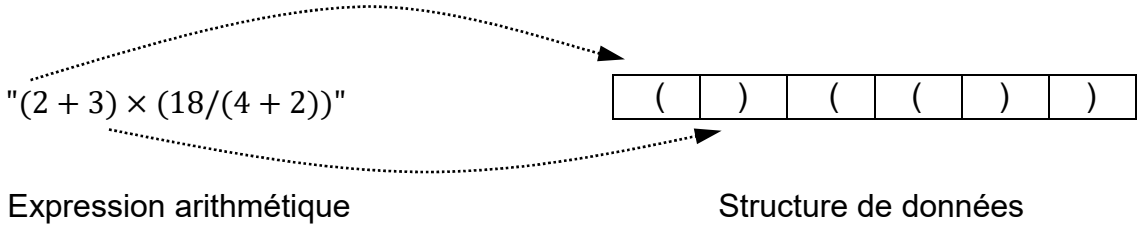
On veut déterminer si une expression arithmétique est correctement parenthésée.

Pour chaque parenthèse fermante ")" correspond une parenthèse précédemment ouverte "(".

Exemples :

- L'expression arithmétique "$(2+3)\times(18/(4+2))$" est correctement parenthésée.
- L'expression arithmétique "$(2+3)\times(18/(4+2$" est non correctement parenthésée.

Pour simplifier les expressions arithmétiques, on enregistre, dans une structure de données, uniquement les parenthèses dans leur ordre d'apparition. On appelle expression simplifiée cette structure.



1. Indiquer si la phrase « les éléments sont maintenant retirés (pour être lus) de cette structure de données dans le même ordre qu'ils y ont été ajoutés lors de l'enregistrement » décrit le comportement d'une file ou d'une pile. Justifier.

Pour vérifier le parenthésage, on peut utiliser une variable `controleur` qui :

- est un nombre entier égal à 0 en début d'analyse de l'expression simplifiée ;
- augmente de 1 si l'on rencontre une parenthèse ouvrante "(" ;
- diminue de 1 si l'on rencontre une parenthèse fermante ")".

Exemple : On considère l' expression simplifiée A : "( )(( ))"
Lors de l'analyse de l'expression A, `controleur` (initialement égal à 0) prend successivement pour valeur 1, 0, 1, 2, 1, 0. Le parenthésage est correct.

2. Écrire, pour chacune des 2 expressions simplifiées B et C suivantes, les valeurs successives prises par la variable `controleur` lors de leur analyse.
Expression simplifiée B : " (((( )( )"
Expression simplifiée C : "(( ))("

3. L'expression simplifiée B précédente est mal parenthésée (parenthèses fermantes manquantes) car le `controleur` est différent de zéro en fin d'analyse. L'expression simplifiée C précédente est également mal parenthésée (parenthèse fermante sans parenthèse ouvrante) car le `controleur` prend une valeur négative pendant l'analyse.
Recopier et compléter uniquement les lignes 13 et 16 du code ci-dessous pour que la fonction `parenthesage_correct` réponde à sa description.

```
1 def parenthesage_correct(expression):
2    ''' fonction retournant True si l'expression arithmétique
3    simplifiée (str) est correctement parenthésée, False
4    sinon.
5    Condition: expression ne contient que des parenthèses
6    ouvrantes et fermantes '''

7   controleur = 0
8   for parenthese in expression: #pour chaque parenthèse
9      if parenthese == '(':
10        controleur = controleur + 1
11     else:# parenthese == ')'
12        controleur = controleur - 1
13        if controleur ...  : # test 1 (à recopier et compléter)
14           #parenthèse fermante sans parenthèse ouvrante
15           return False
16   if controleur ... : # test 2  (à recopier et compléter)
17      return True #le parenthésage est correct
18   else:
19      return False #parenthèse(s) fermante(s) manquante(s)
```

Partie B : Texte correctement balisé

On peut faire l'analogie entre le texte simplifié des fichiers HTML (uniquement constitué de balises ouvrantes <nom> et fermantes </nom>) et les expressions parenthésées :

Par exemple, l'expression HTML simplifiée :
" " est correctement balisée.

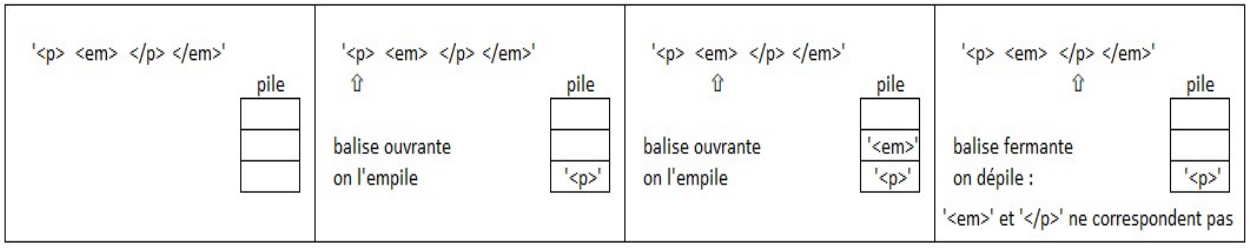
On ne tiendra pas compte dans cette partie des balises ne comportant pas de fermeture comme
 ou <img …>.

Afin de vérifier qu'une expression HTML simplifiée est correctement balisée, on peut utiliser une pile (initialement vide) selon l'algorithme suivant :

On parcourt successivement chaque balise de l'expression :

- lorsque l'on rencontre une balise ouvrante, on l'empile ;
- lorsque l'on rencontre une balise fermante :
  - si la pile est vide, alors l'analyse s'arrête : le balisage est incorrect ,
  - sinon, on dépile et on vérifie que les deux balises (la balise fermante rencontrée et la balise ouvrante dépilée) correspondent (c'est-à-dire ont le même nom) si ce n'est pas le cas, l'analyse s'arrête (balisage incorrect).

Exemple : État de la pile lors du déroulement de cet algorithme pour l'expression simplifiée `"<p><em></p></em>"` qui n'est pas correctement balisée.



État de la pile lors du déroulement de l'algorithme

4. Cette question traite de l'état de la pile lors du déroulement de l'algorithme.

a. Représenter la pile à chaque étape du déroulement de cet algorithme pour l'expression `"<p><em></em></p>"` (balisage correct).

b. Indiquer quelle condition simple (sur le contenu de la pile) permet alors de dire que le balisage est correct lorsque toute l'expression HTML simplifiée a été entièrement parcourue, sans que l'analyse ne s'arrête.

5. Une expression HTML correctement balisée contient 12 balises.
Indiquer le nombre d'éléments que pourrait contenir au maximum la pile lors de son analyse.