TP 6 - Le type abstrait PILES

L. JOANNIC - CC NC BY SA 4.0/FR

Octobre / Novembre 2024 - Terminales NSI

Le but de ce TP est d'appréhender plusieurs implémentations d'un nouveau TAD séquentiel de référence : les Piles.

Dans tout le TP, sauf sujet 0, il est demandé de se placer dans le paradigme de la programmation fonctionnelle.

En particulier, tout effet de bord est proscrit.

Chaque implémentation donnera lieu à un module :

- `Pile_Liste.py` pour la première ;
- `Pile_Tuple.py` pour la seconde ;
- `Pile_LSC.py` pour la dernière.

Celui-ci sera importé sous le nom générique `lifo` dans le module `test.py` comme dans les programmes des deux exemples d'utilisation.

1 Différentes implémentations.

Dans chacun des cas suivants, construire les primitives des Piles.

Attention au contrôle des spécifications.

1.1 En utilisant l'objet `list` de `Python`.

1. Proposer une implémentation de `Pile` basée sur les `list`
2. Tester cette implémentation à partir du module `test.py`

```
import Pile_List as lifo
from random import shuffle

def afficher(pile):
    print("|----\n|")
```

```
    while not(lifo.est_pile_vide(pile)):
        print("| {}\n|".format(lifo.sommet(pile)))
        pile = lifo.depiler(pile)
    print("|----")
    return None

if __name__ == "__main__":
    ma_liste = [i for i in range(10)]
    shuffle(ma_liste)

    print("Liste de départ :", ma_liste)
```

APPELER LE PROFESSEUR POUR VALIDATION DE CETTE IMPLEMENTATION.

1.2 En utilisant des tuples.

APPELER LE PROFESSEUR POUR VALIDATION DE CETTE IMPLEMENTATION.

1.3 En utilisant les listes simplement chaînées.

Il s'agit, ici, d'exploiter les implémentations du TP précédent.
APPELER LE PROFESSEUR POUR VALIDATION DE CETTE IMPLEMENTATION.

2 Exemples d'utilisation.

2.1 Extrait du Sujet 0 de Bac, Épreuve de NSI.

1. (a) On appelle hauteur d'une pile le nombre d'éléments qu'elle contient.
   La fonction `hauteur_pile` prend en paramètre une pile et renvoie sa hauteur.
   Après appel de cette fonction, la pile passée en paramètre doit être dans son état d'origine.
   Exemple : Si `P` est la pile ci-dessous, sa hauteur est 4.

Recopier et compléter le programme `Python` suivant implémentant la fonction `hauteur_pile` en remplaçant les `""" ??? """`.

```
def hauteur_pile(P):
    n = 0
    while not(est_pile_vide(P)):
        # """ ??? """
        P = depiler(P)
    return """ ??? """
```

(b) Créer une fonction `max_pile` ayant pour paramètre une pile `P` et un entier `i`. Cette fonction renvoie la position `j` de l'élément maximum parmi les `i` derniers éléments empilés de `P`.

La position du sommet de la pile est 1.

Après appel de cette fonction, la pile devra avoir retrouvé son état d'origine.

Exemple : Si P est la pile ci-dessus, `max_pile(P, 2) = 1`.

2. Créer une fonction `retourner` ayant pour paramètres une pile `P` et un entier `j`. Cette fonction inverse l'ordre des `j` derniers éléments empilés dans `P` et ne renvoie rien.

On pourra utiliser deux piles auxiliaires.

Exemple : Si `P` est la pile ci-dessus, après appel de `retourner(P, 3)`, l'état de `P` sera

3. L'objectif de cette question est de trier une pile de crêpes.

   On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chacune d'entre elles.

   On souhaite réordonner les crêpes de la plus grande (placée en bas de la pile) à la plus petite (située à son sommet)

   On dispose uniquement d'une spatule que l'on peut insérer dans la pile de crêpes de façon à retourner l'ensemble des crêpes qui sont au-dessus.

   Le principe est donc le suivant :

   - On cherche la plus grande crêpe
   - On retourne la pile à partir de celle-ci de façon à mettre la plus grande tout en haut de la pile
   - On retourne l'ensemble de la pile de façon à ce que cette plus grande crêpe se retrouve tout en bas
   - La plus grande crêpe étant en place, on recommence le principe avec le reste de la pile.

   Créer la fonction `tri_crepes` ayant pour paramètre une pile `P`.

   Cette fonction trie la pile selon la méthode du tri des crêpes et ne renvoie rien.

   Après appel de la fonction sur ma pile `P`, celle-ci est triée.

   On utilisera les fonctions créées dans les questions précédentes.

2.2 Notation Polonaise Inverse.

L'écriture polonaise inverse des expressions arithmétiques place l'opérateur après ses opérandes.

Cette notation ne nécessite ni parenthèses ni règles de priorités.

Ainsi, l'expression "1 2 3 × + 4 ×" désigne l'expression classique $(1 + 2 \times 3) \times 4$.

La valeur d'une telle expression peut être calculée facilement en utilisant une pile pour stocker les résultats intermédiaires.

Pour cela, on observe un à un les éléments de l'expression et on effectue les actions suivantes.

- si l'on voit apparaître un nombre, on le place dans la pile
- si l'on voit un opérateur binaire, on récupère les deux nombres au sommet de la pile, on leur applique l'opérande et on replace le résultat au sommet de la pile.

Si l'expression est bien écrite, il y a toujours deux nombres sur la pile lorsque l'on voit un opérateur.

À la fin du processus, il reste exactement un nombre dans la pile, qui est le résultat.

Écrire une fonction prenant en paramètre une chaîne de caractères représentant une expression en NPI composée d'additions et de multiplications de nombres entiers et renvoyant la valeur de cette expression.

On supposera que les éléments de l'expression sont séparés par des espaces.

Attention, dans le cas d'une expression mal formée, la fonction ne doit pas renvoyer de résultat.