Lycée Thiers — Informatique
PC*

TD : Feuille d'Exercices 6

Les piles

Consignes

Pour tous les exercices de cette planche on aura préalablement écrit l'implémentation des piles à capacité limitée ou illimitée vue en cours, et l'on ne s'autorisera à n'utiliser sur les piles que leurs seules primitives :

creer_pile(), depiler(), empiler(,), top(,), taille(), est_vide()

On travaillera au choix avec des piles à capacité limitée ou illimitée. (*Indication :* il est légèrement plus simple d'utiliser des piles à capacité illimitée).

Partie I : manipulation de piles

Exercice 1.

(1) Ecrire une fonction `renverse(pile)` prenant en argument une pile et qui retourne la pile obtenue en inversant l'ordre des éléments de `pile`. La pile `pile` pourra être vidée.
(2) Même question, sauf qu'à la fin l'argument `pile` doit être inchangé.

Exercice 2. Ecrire une fonction `supprime(pile,p)` qui supprime dans la pile `pile` son p-ième élément compté en partant du sommet (indication : il faudra utiliser une deuxième pile).
Si p dépasse la taille de la pile la fontion retournera `False`, et sinon retournera `True`.

Exercice 3. Ecrire une fonction `echange(pile,p,q)` qui échange dans la pile `pile` ses p-ième et q-ième éléments, comptés en partant du sommet.
Si p ou q dépasse la taille de la pile la fonction retournera `False`, et sinon retournera `True`.

Partie II : Utilisation d'une pile pour déterminer si une chaîne est bien parenthésée.

Exercice 4. Adapter l'algorithme vu en cours pour déterminer si un mot est bien parenthésé pour englober les cas des 3 types de parenthèses `()`, `[]`, {}.

On se bornera dans la suite aux seules parenthèses `(` et `)`.

Exercice 5. Adapter l'algorithme vu en cours pour tester si une chaîne est bien parenthésée, de sorte que :
– il retourne `False` pour une chaîne non correctement parenthésée,

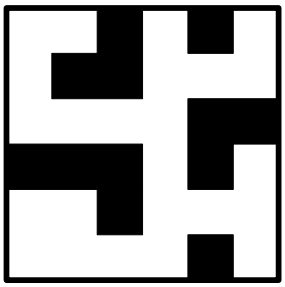
– il retourne la position des parenthèses pour une chaîne bien parenthésée ;
par exemple avec pour argument `'2*(1+(3-1))'` l'algorithme retournera la liste : `[(6,10),(3,11)]`.

Partie III : Recherche d'un chemin dans un labyrinthe (Parcours en profondeur d'un graphe)

Exercice 6. On se donnera un labyrinthe carré par un tableau carré de type `ndarray` (`numpy`). Un 0 symbolisera un passage, un 1 un mur. On prendra par exemple :

```
import numpy as np
T = np.array([
[0,0,1,0,1,0],
[0,1,1,0,0,0],
[0,0,0,0,1,1],
[1,1,1,0,1,0],
[0,0,1,0,0,0],
[0,0,0,0,1,0]
])
```

pour :



De chaque case on peut se déplacer sur une case libre voisine horizontalement ou verticalement. Entrée et sortie du labyrinthe seront des cases données par leurs indices, par exemple $(0, 0)$ et $(5, 5)$.

Le but de l'exercice est d'écrire une fonction qui détermine un chemin de l'entrée à la sortie du labyrinthe par une recherche en profondeur utilisant une pile. Le principe est :

- Chaque case sera symbolisée par le couple `(i,j)` de ses indices dans le tableau.
- Lorsqu'une case aura été déjà visité, sa valeur dans le tableau sera mise à -1.
- Une fonction `voisin()` prendra en paramètre le tableau et les indices d'une case et retournera la liste de ses cases voisines libres et non déjà visitées.
- De chaque case on se déplacera sur une telle case voisine libre et non déjà visitée.
- Une pile à capacité illimitée constituera le 'fil d'ariane' : on empilera les cases visitées successivement. Dès qu'une case n'aura plus aucune case voisine non déjà visitée on la dépilera, pour poursuivre le trajet à partir de la case précédente.

(1) Ecrire la fonction `voisin()` comme décrite ci-dessus.

(2) Ecrire la fonction `trajet()` prenant en paramètre le tableau du labyrinthe, et les couples d'indices de l'entrée et de la sortie, et qui retourne la pile des cases successivement visitées constituant un trajet de l'entrée à la sortie.

(3) La tester sur le tableau ci-dessus avec pour entrée `(0,0)` et sortie `(5,5)`.