Première NSI - Algorithmique

Algorithme Glouton. 3. TP : Problème des étagères

qkzk

2020/03/07

Le problème des étagères

La bibliothèque prévoit de refaire ses étagères. Elle comprend une collection de n livres `b1,b2,... ,bn`. Chaque livre `bi` a une largeur `wi`. Les livres doivent être rangés dans l'ordre fixé (par valeur de i croissante) sur des étagères de largueur `L`.

À faire 1

proposer un algorithme glouton pour le rangement des étagères.

Votre algorithme prend en paramètres une liste des livres, dont on connaît la largeur et retourne la double liste des étagères.

Chaque étagère contient donc différents livres.

Par exemple avec les livres :

livre	0	1	2	3
largeur	4	6	2	3

et la largeur d'étagère `L = 9` :

on obtient :

```
etageres = [
  [0],
  [1, 2],
  [3]
]
```

- En effet, au premier étage, on peut ranger le premier livre mais pas y ajouter le second, il ne reste plus assez de place.
- Au second étage, on peut ranger le second et le troisième livre.
- Au troisième étage, on range le dernier.

À faire 2

1. Télécharger le script Python etageres.py.
2. Compléter la fonction `creer_livres` qui prend en paramètres un nombre entier `n` et retourne une liste de `n` livres dont la taille est un entier aléatoire entre `LARGEUR_MIN` et `LARGEUR_MAX`

   N'oubliez pas de documenter la fonction.
3. Télécharger le script python tester_etageres.py

   Ce script contient des tests de vos fonctions. Ces tests lèvent des exceptions chaque fois qu'une condition imposée n'est pas remplie.
4. Exécuter ce script. Il doit lever une exception car vous n'avez pas complété la deuxième fonction, c'est normal.

   Par contre, le premier test doit fonctionner et vous devez lire le message :

   `test generer_livres : ok`

5. Si le test précédent ne fonctionne pas, revenir en 2.

À faire 3

1. Compléter la fonction fonction `ranger_livres`

   Elle prend en paramètres une liste de livres intitulée `livres` Elle retourne la double liste des étagères décrite plus haut.

2. Exécuter à nouveau le script tester_etageres.py.

   Cette fois vous ne devez plus avoir d'erreur du tout et vous devez lire le message `test ranger_livres: ok`

3. Si vous avez encore des erreurs, revenir en 1.

Complément : rédiger un jeu de tests.

Parmi les compétences importantes en informatique, être capable de générer du code qui ne plante pas est fondamental.

Cela nécessite bien sûr une certaine maîtrise mais aussi de l'organisation. Il faut tester ses fonctions.

Examinons ensemble le code d'un des tests utilisé plus haut.

```
def tester_generer_livres():
    nombre = 10
    livres = generer_livres(nombre)
    assert len(livres) == nombre, "le nombre de livres est incorrect"
    ...
```

Dans cette fonction on choisit un nombre arbitraire de livres (10) et on génère une liste des livres.

On commence par tester avec `assert` qu'il y a le bon nombre de livres. La syntaxe est :

```
assert booleen, string
```

- `booleen` : le test qu'on veut réaliser, ici `len(livres) == nombre`
- `string` est un paramètre optionnel. C'est le message d'erreur : "le nombre de livres est incorrect"

Lorsqu'on exécute cette fonction, on s'assure Python vérifie que le booléen est vrai. S'il est faux, il affiche un message d'erreur accompagné de la `string`.

La suite du code est similaire, on s'assure que tous les livres ont une taille convenable.

Rendu de monnaie

1. Programmer une fonction Python qui prend en paramètre un jeu de pièces et une somme à rendre et retourne les pièces utilisées. Elle utilise l'algorithme glouton.
2. Écrire un jeu de tests pour votre fonction.