Type abstrait de données
Exemple des structures de données associatives

Objectifs

— Définir un type abstrait de données (TAD)

— Proposer plusieurs implantations d'une même structure de données

— Mettre en pratique tous les concepts vus

On appelle structure de données associative [1] une structure de données qui permet d'enregistrer et rechercher des données en fonction d'une clé identifiant de manière unique une donnée. Un exemple classique d'une telle structure de données est le dictionnaire. La clé est un mot et la donnée sa définition. Un autre exemple est un annuaire où la clé est le nom d'un abonné et la donnée les informations le concernant (numéro de téléphone, adresse, etc.).

Une structure de données associative (SDA) fournit généralement les opérations suivantes.

1. Initialiser une SDA. La SDA ainsi initialisée est vide.
2. Enregistrer une donnée en précisant sa clé.
3. Modifier la donnée associée à une clé.
4. Supprimer la donnée associée à une clé dans une SDA.
5. Obtenir la donnée associée à une clé.
6. Obtenir la taille d'une SDA (le nombre de données enregistrées).
7. Afficher les clés et données enregistrées dans une SDA.
8. Vider une SDA, c'est-à-dire supprimer toutes les informations enregistrées dans cette SDA.

Dans la suite, nous allons envisager plusieurs manières de réaliser une telle structure de données associative (type SDA et opérations associées). Dans les exemples, nous considérerons que la clé est une chaîne de caractères et que la donnée est un entier. Nous choisissons d'adopter une approche de type *programmation défensive*.

Exercice 1 : Liste Chaînées Associative

Nous appelons *Liste Chaînées Associative* (LCA) cette première réalisation d'une structure de données associative car elle s'appuie sur une liste chaînée linéaire simple où chaque cellule est composée d'une clé, d'une donnée et d'un accès à la cellule cellule suivante.

1. Écrire l'interface du module LCA.

1. Cette structure de données correspond à la notion de *dictionnaire* (dict) en Python.

2. Écrire un programme minimal qui insère successivement dans une même LCA la donnée 1 associée à la clé "un" et la donnée 2 associée à la clé "deux".
3. Écrire l'implantation du module LCA. On pourra se demander s'il ne serait pas judicieux de définir de nouveaux sous-programmes.
4. Indiquer les inconvénients/avantages d'une implantation par listes chaînées d'une SDA.

Exercice 2 : Tables de hachage
Pour améliorer le temps d'exécution, on peut s'appuyer sur le principe des *tables de hachage* (TH) pour réaliser notre SDA. L'idée est d'utiliser 1) un tableau (donc un accès direct) pour stocker les données et 2) une fonction, appelée *fonction de hachage*, qui à partir d'une clé calcule la position de la donnée dans le tableau. La valeur calculée par une fonction de hachage est appelée *valeur de hachage*.

La fonction de hachage pourrait être la fonction qui renvoie le nombre de caractères d'une chaîne de caractères, le code ASCII de l'initiale [2] de la chaîne, la somme des codes ASCII de ses caractères, etc.
1. Schématiser la table de hachage de capacité 11 obtenue après enregistrement des données entières 1, 2, 3 et 4 associées aux clés "un", "deux", "trois" et "quatre", la valeur de hachage étant le nombre de caractères de la clé.
2. Identifier deux problèmes qui peuvent survenir et proposer des solutions pour les éviter.
3. En utilisant la même table de hachage que dans la question 1, schématiser la table de hachage pour l'enregistrement des valeurs entières 1, 2, 3, 4, 5, 99 et 21 avec pour valeur de hachage le nombre de caractères de la clé associée ("un", "deux", "trois", "quatre", "cinq", "quatre-vingt-dix-neuf" et "vingt-et-un").
4. Qui devrait définir la capacité de la table et la fonction de hachage ?
5. Écrire l'interface du module TH.
6. Écrire l'implantation du module TH.

Exercice 3 : Arbre binaire de recherche
Une autre implantation possible pour les structures de données associatives consiste à utiliser des arbres binaires de recherche (ABR). La contrainte est de disposer d'une relation d'ordre totale sur les clés. Pour les chaînes de caractères, la relation d'ordre peut être l'ordre lexicographique (celui du dictionnaire).

Dans un arbre binaire de recherche, chaque *nœud* contient :

— une clé,

— une donnée (associée à la clé)

— un *sous-ABR gauche* et un *sous-ABR droit*, éventuellement vides.

Un ABR est *vide* s'il ne contient aucun nœud.

La *racine* d'un arbre binaire est le nœud situé le plus « haut » dans cet arbre. Par extension, chaque nœud d'un ABR peut donc être considéré comme la racine d'un sous-arbre binaire de l'ABR complet.

Une *feuille* est tout nœud d'un arbre binaire dont les deux sous-ABR sont vides.

2. On retrouve alors le principe des répertoires papier.

Un *arbre binaire de recherche* est construit de façon à toujours respecter la contrainte suivante : *si n est un nœud de l'arbre et n_g est un nœud du sous arbre gauche de n, alors la clé de n_g est strictement inférieure à celle de n ; de même si n_d est un nœud du sous arbre droit de n, alors la clé de n_d est strictement supérieure à celle de n.*

Ainsi, pour insérer un nouvel élément (clé + donnée) dans un arbre binaire de recherche, on procède comme suit :

— si l'ABR est vide, on créé un nouveau nœud qui devient la racine et unique feuille de cet ABR ;

— sinon, si la clé du nouvel élément est strictement inférieure à la clé de la racine de l'ABR, on insère le nouvel élément dans le sous-ABR gauche ;

— sinon, on insère le nouvel élément dans le sous-ABR droit.

1. Schématiser l'ABR obtenu suite à l'enregistrement des valeurs 99, 2, 3, 4, 5, 1 et 21 associées aux clés quatre-vingt-dix-neuf", "deux", "trois", "quatre", "cinq", "un" et "vingt-et-un".
2. Écrire l'interface du module ABR.
3. Écrire l'implantation du module ABR. On écrira en priorité les sous-programmes correspondant aux opérations initialiser, taille, insérer et afficher.