CSC 301- Structure de Données avec Java II

Ingénieur Logiciel / Consultant IT

Folly AMAH

99 77 82 90

Chapitre 1: Tables de Symboles

Introduction

Les **Symbol Tables** (Tables de Symboles) sont des structures de données fondamentales utilisées pour stocker des paires clé-valeur. Elles jouent un rôle clé dans divers domaines tels que la compilation, la gestion des dictionnaires, les bases de données et l'indexation.

Dans ce chapitre, nous allons explorer leur définition, leurs opérations fondamentales, et étudier différentes implémentations avec leurs avantages et inconvénients.

1. Définition et Opérations de Base

Une **Table de Symboles** est une structure qui permet d'associer une **clé** unique à une **valeur**.

Les opérations fondamentales incluent :

- **Put(Key, Value)** : Insère une paire clé-valeur ou met à jour la valeur si la clé existe déjà.
- **Get(Key)** : Retourne la valeur associée à une clé donnée.
- **Delete(Key)** : Supprime une clé et sa valeur associée.
- **Contains(Key)** : Vérifie si une clé est présente.
- **IsEmpty()** : Vérifie si la table est vide.
- **Size()** : Retourne le nombre d'éléments présents dans la table.

2. Implémentations des Tables de Symboles

Il existe plusieurs façons d'implémenter une table de symboles. Nous allons explorer les plus courantes :

2.1 Implémentation avec des Tableaux non triés

2.2 Implémentations avec des Tableaux triés (Recherche Dichotomique)

2.3 Implémentation avec une liste Chaînée

2.4 Implémentation avec une Table de Hachage

2.1 Implémentation avec des Tableaux non triés

- Cette implémentation repose sur deux listes : une liste pour stocker les clés et une autre pour stocker les valeurs.
- L'ajout d'une nouvelle paire (clé, valeur) se fait en l'ajoutant à la fin des listes.
- La recherche d'une valeur nécessite de parcourir toute la liste jusqu'à trouver la clé correspondante.
- La mise à jour d'une valeur existante remplace simplement la valeur associée à la clé.

- **Avantages** :
- Facilité d'implémentation.
- Bonne performance pour des petites tailles.

- **Inconvénients** :
- Recherche en **O(n)** dans le pire cas (parcours séquentiel nécessaire).
- Inefficace pour un grand nombre d'éléments.

2.1 Implémentation avec des Tableaux non triés (suite)

```java
package com.faadis;
import java.util.ArrayList;
import java.util.Objects;

class UnorderedArrayST<K, V> {  no usages
    private ArrayList<K> keys;  6 usages
    private ArrayList<V> values;  4 usages

    public UnorderedArrayST() {  no usages
        keys = new ArrayList<>();
        values = new ArrayList<>();
    }

    public void put(K key, V value) {  no usages
        for (int i = 0; i < keys.size(); i++) {
            if (Objects.equals(keys.get(i), key)) {
                values.set(i, value);
                return;
            }
        }
        keys.add(key);
        values.add(value);
    }

    public V get(K key) {  no usages
        for (int i = 0; i < keys.size(); i++) {
            if (Objects.equals(keys.get(i), key)) {
                return values.get(i);
            }
        }
        return null;
    }
```

2.1 Implémentation avec des Tableaux non triés (suite)

Cas concrets d'utilisation

Exemple 1 : Gestion d'un carnet d'adresses

Imaginons un programme simple qui stocke des noms et leurs numéros de téléphone.

```java
package com.faadis;

public class Main {
    public static void main(String[] args) {
        UnorderedArrayST<String, String> phoneBook = new UnorderedArrayST<>();

        phoneBook.put("Alice", "0123456789");
        phoneBook.put("Bob", "0987654321");
        phoneBook.put("Charlie", "0567890123");

        System.out.println("Numéro d'Alice: " + phoneBook.get("Alice"));
        System.out.println("Numéro de Bob: " + phoneBook.get("Bob"));

        phoneBook.put("Alice", "0112233445");
        System.out.println("Nouveau numéro d'Alice: " + phoneBook.get("Alice"));
    }
}
```

2.1 Implémentation avec des Tableaux non triés (suite)

Cas concrets d'utilisation

Exemple 2 : Gestion d'un inventaire de produits

Nous pouvons utiliser cette table de symboles pour stocker des produits et leurs prix.

```java
public class Main {
    public static void main(String[] args) {

        UnorderedArrayST<String, Double> inventory = new UnorderedArrayST<>();

        inventory.put("Laptop", 1200.99);
        inventory.put("Smartphone", 799.49);
        inventory.put("Tablet", 499.99);

        System.out.println("Prix du Laptop: " + inventory.get("Laptop"));
        System.out.println("Prix du Smartphone: " + inventory.get("Smartphone"));
    }

```

2.1 Implémentation avec des Tableaux non triés (suite)

Avantages et Inconvénients

Avantages	Inconvénients
Implémentation simple et facile à comprendre	Recherche lente (O(n) dans le pire cas)
Pas de restriction sur le type de clé	L'insertion peut devenir inefficace pour de grands ensembles de données
Convient aux petits ensembles de données	Manque d'optimisation pour les recherches

2.1 Implémentation avec des Tableaux non triés (suite)

Exercice

1.Ajoutez une méthode delete à UnorderedArrayST pour supprimer une clé et sa valeur.

2.Mesurez les performances de UnorderedArrayST pour différentes tailles de données.

3.Implémentez une version améliorée qui utilise une structure plus efficace pour la recherche.