UNIVERSITÉ DE NANTES

Rapport de projet

Algorithmique et structure de donnés 2

Eslam HUMAID et Abrahim Fekri BAMATRAF

Groupe 485

2019-2020

Introduction

Ce travail a été fait dans la cadre de projet de l'UE Algorithmique et structures de données 2, ce projet a pour but la gestion d'un system de dépôt de bagage en créant deux classes (consigne et ticket) et en bien identifiant l'SDA et l'SDC de chaque classe.

Partie 1 :

La classe Ticket :

SDA :

Méthode	Rôle	Précondition
creerTicket(T) Ticket → ∅	Créer un ticket et générer son code aléatoire.	-
detruireTicket(T) Ticket → ∅	Détruire l'objet Ticket	-
getCodeTicket(T) Ticket → chaine	Accesseur pour le code du ticket.	-

SDC :

Type Ticket = enregistrement

- chaine *_codeTicket* (le code unique du ticket)

Opération	Complexité
CreerTicket(T)	Θ(1)
detruireTicket(T)	Θ(1)
getCodeTicket(T)	Θ(1)

La classe Storage :

SDA :

Méthode	Rôle	Précondition
creerStorage(S,n) Storage x entier → ∅	Créer une consigne.	n > 0
detruireStorage(S) Storage → ∅	Détruire l'objet Storage	-
isFull(S) Storage → boolean	Tester si la consigne est pleine.	-
deposit(S,B) Storage x bagage → Ticket	Déposer un bagage dans un casier libre dont la dernière utilisation est la plus ancienne et retourner un ticket.	La consigne n'est pas pleine.
Collect(S,T) Storage x Ticket → Bagage	Récupérer le bagage dans S avec le ticket T.	La consigne n'est pas vide.

SDC :

Type t_case = enregistrement (une structure qui représente une case dans la consigne)

- Entier IndexInVect (l'indice de la case dans le Vector)
- Bagage bag (le bagage stocké dans la case)

Type Storage = enregistrement

- Vector<Ticket> _cases (un vecteur représentant les cases de la consigne)
- Table<Ticket,t_case> _storage (table associative Ticket → t_case)
- File<entier> _emptyCases (File des cases vide après premier usage)
- Entier _nbCases (nombre des cases dans la consigne)
- Entier _filledCases (le nombre des cases actuellement pleine)
- Entier _usingCase (le nombre total des dépôts dans la consigne)

Pour effectuer les opérations de récupération et dépôt en temps constantes, on maintient une table associative pour l'accès rapide aux cases et une file pour y mettre les cases vidées afin de rapidement accéder à la case dont la dernière utilisation est la plus ancienne.

On utilise un vecteur dans lequel on stocke le ticket de la case correspondant dans la table associative (si la case n'est pas vide). Les cases dans le vecteur et les cases dans la table associative sont liées par l'attribut (indexInVect) dans la structure t_case.

Ainsi, pour créer la consigne en temps constantes sans avoir à initialiser toutes les cases dans la queue (qui se fera en temps linéaire) on met les indices des cases vides dans la file uniquement après le premier usage en utilisant l'attribut _usingCase qui s'incrément à chaque dépôt dans la consigne (si _usingCase est inférieur au nombre des cases, on met le bagage dans la case suivant non-utilise dans la consigne et sinon c'est soit que la consigne est pleine soit qu'on a déjà vidé une case donc on prend la premier case dans la file).

Opération	Complexité
creerStorage(S,n)	Θ(1)
detruireStorage(S)	Θ(1)
isFull(S)	Θ(1)
deposit(S,B)	Θ(1)
Collect(S,T)	Θ(1)