UNIVERSITÉ DE NANTES

Algorithmique et structures de données

Rapport de projet

Mohammed BA RAGAA

Groupe 485

2019-2020

Introduction

Ce projet est le projet de fin de module Algorithmique et structures de données 2, Dans ce projet nous intéressons dans un système de dépôt de bagage en utilisant la programmation orienté objet et implémenter des différentes structures de données grâce à la connaissance acquis dans ce module.

GitHub

Ce Projet est disponible (code + fichier d'exécution) sur GitHub.

Plan du contenu

Partie 1

Cette partie consiste à construire un système de gestion de dépôt et retrait de bagage, pour cela un 2 classes principales (Ticket et Consigne) sont construits en utilisant des structures de données comme file et tableau de hashage et en plus de quelques alias de types.

Classe Ticket

SDA

Méthode	Rôle
creerTicket(T) Ø →Ticket	Créer un objet tu type ticket.
detruireTicket(T) Ticket → Ø	Détruire l'objet tu type Ticket.
getCode(T) Ticket → String	Accesseur pour le code du ticket.
hash_code(T) Ticket → Entier	Calculer la valeur de hash du code du ticket.

SDC

Chaque ticket a un code aléatoire créé au moment de son construction.

Le code hash est calculé par le fonction **std ::hash**

Type Ticket = Enregistrement

String _code

Opération	Complexité temporelle
construireTicket(T)	Θ (1)
detruireTicket(T)	Θ (1)
getCode(T)	Θ (1)
hash_code(T)	Θ (1)

Classe Consigne

SDA

Méthode	Rôle	Précondition
construireConsigne (C, capa) Entier → Consigne	Créer un objet tu type Consigne avec une capacité donné.	Capa > 0
DetruireConsigne(C) Consigne → Ø	Détruire l'objet tu type Consigne	Aucune
EstPlein (S). Consigne → booléen	Vérifier si la consigne est plein	Aucune
Déposer (C, B). Consigne, Bagage → Ticket	Déposer un bagage dans un casier de la consigne et	La Consigne ne doit pas être plein.

	retourner un ticket.	
Retirer (C, T) Consigne, Ticket → Bagage	Récupérer le bagage dans C avec le ticket T.	La Consigne ne doit pas être vide.

SDC

Afin de permettre les opérations précédentes de s effectuer en temps constant, une table associative a été créé en utilisant la fonction de défaut du hashage fournie par la classe, pourtant aucune collision est gérée dans le code.

Pour garantir l'utilisation équilibré -hypothèque – nous avons utilisé le structure file pour stocker les cases et utiliser la case le moins récemment utilisé et en temps constant.

Un vecteur est créé pour stocker les cases et chaque case sait sa place dans le vecteur par l'attribut _place.

Chaque pair est créé par ordre dans la table associatif et pour cela la variable _nextcase est créé pour le premier remplissage où la variable est plus petite que la capacité, puis on utilise la file pour organiser l'usage des cases déjà utilise au moins une fois.

Type Casier = Enregistrement

- Entier : _place. //l'indice du casier dans le vecteur des casiers.
- Bagage : _package. // le bagage au casier.

Type Consigne = Enregistrement

- Table associative <Ticket, Case> _map (table associative Ticket → Case)
- File<entier> _emptyCases (File des cases vide après premier usage)
- Entier : _capacité (nombre des cases dans la consigne)
- Entier : _usedCases (le nombre des cases actuellement pleine)
- Entier :_nextCase

Méthode	Complexité temporelle
construireConsigne (C, capa)	Θ (1)
DetruireConsigne(C)	Θ (1)
EstPlein (C).	Θ (1)
Déposer (C, B).	Θ (1) Amorti avec O(n) et Ω(1)
Retirer (C, T)	Θ (1) Amorti avec O(n) et Ω(1)

Partie2

Cette partie est un amélioration de la premier partie, nous nous intéressons ici en ajouter un volume pour les bagages et déposer les bagages avec une stratégie optimale dans la quelle on utilise la case minimale en volume dont la utilisation est le plus ancienne.

Classe Bagage (Abstrait)

Ce classe est ajouté afin de améliorer la extensibilité du code et faire plusieurs types de bagage même si ce n'est pas le cas dans ce code.

Ce classe possède des méthodes virtuelles ainsi que des attributs privées _volume et _ID

SDA

Méthode	Rôle
getBagegeID(B) Bagage → String	Accesseur pour le ID du ticket.
getVolume(B) Bagage → réel	Accesseur pour la Volume du ticket.

SDC

Type Bagage = Enregistrement

- String : _ID (une chaine associée avec les bagages (peut être dupliquée))
- Réel : _volume

Classe Colis

SDA

Méthode	Rôle
creerColis (C,S,V) String, float →Colis	Créer un objet tu type Colis.
detruireColis(C) Colis→ Ø	Détruire l'objet tu type Colis.
getBagageID(C) Colis→ String	Accesseur pour le ID de Colis.
getVolume(T) Colis→ Réel	Accesseur pour le volume du colis.

SDC

Dans ls SDC le classe abstrait est hérité avec ses méthodes virtuelles pour avoir la possibilité de l'existence d'autres types de bagage même si dans cette partie ce n'est pas très utile.

Type Bagage = Enregistrement

- Réel : _volume
- String : _bagageID

Opération	Complexité temporelle
creerColis (C,S,V)	Θ (1)
detruireColis(C)	Θ (1)
getBagageID(C)	Θ (1)
getVolume(C)	Θ (1)

La classe VConsigne

SDA

Méthode	Rôle	Précondition
construireVConsigne (liste(pair(ni, vi))) liste<pair<entier, float>> → VConsigne	Créer un objet tu type Consigne avec une capacité donné.	ni >0 , vi>0 ,list.size() >0
DetruireVConsigne(C) VConsigne → Ø	Détruire l'objet tu type Consigne	Aucune
EstPlein (S). VConsigne → booléen	Vérifier si la consigne est plein	Aucune
Déposer (C, B). VConsigne, Bagage → Ticket	Déposer un bagage dans un casier de la consigne et retourner un ticket avec un méthode optimale de dépôt	La Consigne ne doit pas être plein.
Retirer (C, T) VConsigne, Ticket → Bagage	Récupérer le bagage dans C avec le ticket T.	La Consigne ne doit pas être vide.
checkVolume(C,V) VConsigne, float → Boolean	Vérifie si il existe une case plus grande ou égale a un volume donné	Aucune

SDC

En prenant en compte que la stratégie de dépôt doit être optimale cette SDC a utilisé un tableau associatif pour organiser et optimiser le coût temporelle des opérations de dépôt et retraite en plus que la condition que la case où le dépôt s'effectuera doit être la plus petit possible dont la dernier utilisation est la plus ancienne.

Afin de vérifier si la volume est valable pour déposer dans la VConsigne un vecteur est créé où tous les volumes sont stockés au moment de la création de la VConsigne ensemble avec un autre vecteur pour les indices des cases libres qui contiennent son indice avec un bagage associé dans la manière utilisé dans le premier partie.

La raison pour laquelle nous ne utilisons plus un file dans la quelle c'est impossible de parcourir tous les éléments donc un vecteur est la décision faite pour stocker les cases vides et les autres cases sont d'ailleurs stockés dans la tableau associatif et ajoutés dans le vecteur et comme cela nous avons réussi de simuler un file mais avec la capacité de le parcourir si nécessaire.

En utilisant la faite que les cases sont ordonnées par la temps de la dernière utilisation il suffit juste de parcourir la première case de chaque volume qui est également la plus ancienne pour le volume donné et choisir la case minimale qui suffit ((minimale trouvé <candidat<=volume à déposer)) pour trouver la case la plus adaptée dans un seul parcours des cases O(n).

Donc nous avons comme SDC

Type VConsigne = Enregistrement

- Table associative <Ticket, Case> _map (table associative Ticket → Case)
- Vecteur<entier> _emptyCases (cases vides initialisés dans la moment de construction) .
- Entier _capacité (nombre des cases dans la consigne (ni x vi))
- Entier _usedCases (le nombre des cases actuellement pleine)
- Vecteur <float> _volumes (le vecteur des volumes des cases de la VConsigne)

Méthode	Complexité temporelle
construireVConsigne (C, capa)	Θ (n) (n est le nombre des cases = (ni x vi))
DetruireVConsigne(C)	Θ (1)
EstPlein (C).	Θ (1)
VérifierVolume(C,V)	Ω(1) si la Consigne est plein et O(n) sinon
Déposer (C, B).	Θ (n)
Retirer (C, T)	Θ (1) Amorti avec O(n) et Ω(1)

Conclusion

Dans ce projet nous avons utilisé la connaissance et la savoir-faire pour créer un système de dépôt de bagage et le rendre plus performant avec une complexité réduite en utilisant les différentes structures de données et plus simple et extensible en utilisant la paradigme orienté objet appris dans ce module.