RAPPORT

Projet : les consignes à bagages

UNIVERSITÉ DE NANTES

Mahboubi Saad , El Farissi Belal

05/05/2020

Licence 2 informatique – Groupe 484

INTRODUCTION

Le travail dans cette partie consiste à développer deux structures de données représentant les tickets et les consignes.

La mécanique demander à l'issu de cette partie est de pouvoir déposer un bagage dans un casier libre et récupérer un ticket en retour et avec ce dernier pouvoir récupérer le bagage qui lui correspond.

Les contraintes sont qu'un utilisateur ne sait pas ou se trouve le bagage , et les bagages respecte une stratégie bien spécifique dans la consigne avec un ordre où le bagage est mis dans le casier libre dont la dernière utilisation est la plus ancienne.

PARTIE 1

Ainsi pour la réalisation de ces deux structures de données , nous avons tout d'abord établis le code la structure Ticket avec un code secret généré aléatoirement par une suite d'entier et une SDA comme demandé dans le sujet avec les quatre opérations demandées:

-Un constructeur sans paramètres

-Le calcul d'un code de hachage

-Un test d'égalité et un test de différence de deux tickets

Structure Ticket

SDA : Ticket

Signature	**rôle**	**Precondition**
Ticket(); Ticket -> Ticket	Constructeur d'un ticket et le renvoie avec son numéro vide	// Ø
size_t hash_code() const ø -> size_t	Création d'un code de hashage à partir du numéro de ticket	//ticket en entré
bool operator==(Ticket const& autre)const Ticket X Ticket -> boolean	Opérateur == pour la classe ticket , il renvoie vrai si les deux tickets ont le même code secret	// precondition : 2 tickets à comparer
bool operator!=(Ticket const& autre) const Ticket X Ticket -> boolean	Opérateur != qui renvoie vrai si les deux ticket 2 ont un code secret différent	// precondition : 2 ticket à comparer

SDC: Ticket

int my_rand (void)	my_rand contient le code secret du ticket généré aléatoirement lors de la création du ticket sou forme d'entier

Complexité des opérations:

Méthode	**Ordre de Complexité**
Ticket()	O(n) Dépendra de la taille du code donnée
size_thash_code()	O(1) une affectation de valeur
bool operator==(Ticket const& autre)	O(1) Seulement un return d'une comparaison
bool operator!=(Ticket const& autre)	O(1) Seulement un return d'une comparaison

Par la suite , nous avons effectué le code de la structure consigne qu'on a représenté sous la forme d'une liste de casiers.Notre SDA comme demandé dans la consigne respectera quatre opération:

- un constructeur

- un test permettant de savoir si une consigne est pleine (aucun casier n'est libre)

-une opération pour déposer un bagage

-une opération pour récupérer un bagage

En effet , par la suite ont peut se poser la questions suivantes:

Comment trouver un casier libre lors d'un dépôt, sachant en plus qu'on doit choisir celui dont la dernière utilisation est la plus ancienne ?

Pour cela , nous avons opté pour qu'à chaque fois qu'un casier est vide , on l'ajoute à la fin de la queue, à l'aide de la fonction push_back() pour qu'ainsi le premier élément de la queue soit celui dont la dernière utilisation est la plus ancienne.

Structure Consigne

SDA : Consigne

Signature	rôle	Precondition
Consigne(int size)	liste de casier : nombre prédéfini	// Saisie d'un entier positif
estPlein() Sortie -> boolean	Test si une consigne est pleine	// la consigne existe
tailleRestante() Consigne -> entier	Retourne la taille de la liste vide (getter)	// la consigne existe
ajoutBagage(BAG bag) Consigne X bagage -> Ticket	ajout d'un élément de type bagage dans une liste	//une consigne ou sera inséré un bagage
retraitBagage(Ticket ticket) Ticket -> bagage	retrait d'un élément de type bagage de la liste	// il faut que le ticket soit valable et correspond à un bagage donnée

SDC : Consigne

std::unordered_map<Ticket, Casier> listeCasierPris	conteneurs associatifs d'une valeur de clé et d'une valeur
queue<Casier> casierLibre	liste trié qui facilite l'accès à l'élément le plus ancien
int unsigned tailleListe	taille de la liste restante

Complexité des opérations:

Méthode	**Ordre de complexité**
Consigne(int size)	O(1) Seulement un affichage
estPlein()	O(1) Seulement un return vrai ou faux
tailleRestante()	O(1) seulement un return de la taille de la liste vide
ajoutBagage(BAG bag)	O(n) Ordre de complexité linéaire qui dépend de la taille de la consigne
retraitBagage(Ticket ticket)	O(n) Complexité linéaire , dépend de la taille de la consigne

Partie 2

On s'intéresse maintenant à la partie plus spécifique de ce projet en insérant à chaque casier et bagage des volumes (en litre).

C'est pourquoi un bagage de volume v doit être déposé dans un casier de volume suffisant.

De plus on s'intéresse ici à une stratégie de dépôt optimal , en effet parmi tous les casier libres , on choisira le casier de volumes minimale et supérieure à v (pour nous volume) dont la dernière utilisation est la plus ancienne.

Ainsi un bagage possède donc un volume. nous avons donc réaliser une classe abstraite bagage qui laisse place à des sous classe pour différencier plusieur type de bagages

Structure VConsigne

SDA Vconsigne

Signature	**rôle**	**précondition**
Vconsigne(vector< pair<int, int> > casiers)	constructeur	//ø
estPlein() Sortie -> bool	vérification si la consigne est pleine	//la Vconsigne courante

ajoutBagage(BAG bag) Bagage -> Ticket	Ajoute un Bagage bag dans la consigne avec comme obligation de mettre le casier le plus optimal en fonction du volume et le casier le dont la dernière utilisation est la plus ancienne	//un bagage de type BAG et la Vconsigne courante // le Casier doit être non plein et avoir un casier de volume satisfaisante

retraitBagage(Ticket ticket)
 	
Ticket-> bagage
	retrait d'un bagage de la consigne en demandant son ticket à l'utilisateur	//un ticket de type Ticket et la Vconsigne courante
		
		// le ticket doit exister

testVolumeCasier(int volume) Volume -> boolean	test s'il existe au moins un casier libre dont le volume est supérieur à un volume donné	//la Vconsigne courante et un volume en entier

SDC : Vconsigne

Nous avons utilisé la notion de vector car cela nous paraissait plus simple à implémenter.

De plus , le vector répondait bien au concept de gérer une structure de donnée premier arrivé premier servie et donc répondre à la question posé.

Enfin, la méthode push_back nous permet de répondre à la question posé par le sujet (on doit choisir celui dont la dernière utilisation est la plus ancienne) en l'ajoutant à la fin de la file à chaque fois qu'on vides un casier.

Une question se posera donc , pourquoi nous n'avons pas pris de std::queue ?

Pour notre part , une std::queue n'est pas optimisé pour faire de l'accès direct et du tri.

vector<TypeCasier> casierLibre	//conteneurs de séquences représentant des casiers dont la taille est dynamique
queue<Casier>* casiers	//casiers qui pointe vers une queue de casiers
std::unordered_map<Ticket, Casier> listeCasierPris	//conteneurs associatifs d'une valeur de clé (Ticket) et d'une valeur (Casier)
int unsigned nbCasiers	//nombre de casiers présents

Complexité des opérations:

Méthode	**Ordre de complexité**
estPlein()	O(1) Seulement un return vrai ou faux
ajoutBagage(BAG bag)	O(n) Ordre de complexité linéaire qui dépend de la taille de la consigne
retraitBagage(Ticket ticket)	O(n) Complexité linéaire , dépend de la taille de la consigne
testVolumeCasier(int volume)	O(n) Complexité linéaire qui dépend du volume

CONCLUSION

A l’issue de ce projet, nous avons pu appliquer et approfondir nos connaissances concernant la création et la manipulation des structures de données et de certaine opérations (operator== , operator!= …).

De plus , de nouvelles notions ont été abordées, telles que les piles, files, table associative et la notion de vector et queue pour notre part. La réalisation de certaines fonctions demandées à pu s’avérer parfois complexe à décomposé , notamment dans la dernière partie, en revanche nous sommes généralement satisfaits du rendu finale. L’intégralité des fonctions et procédures ont été implémenté et compilent. L’assemblage de toutes ces fonctions nous à permis de réaliser l’objectif initial du projet, à savoir la réalisation d’une consigne à bagage.