FCT

Boleias

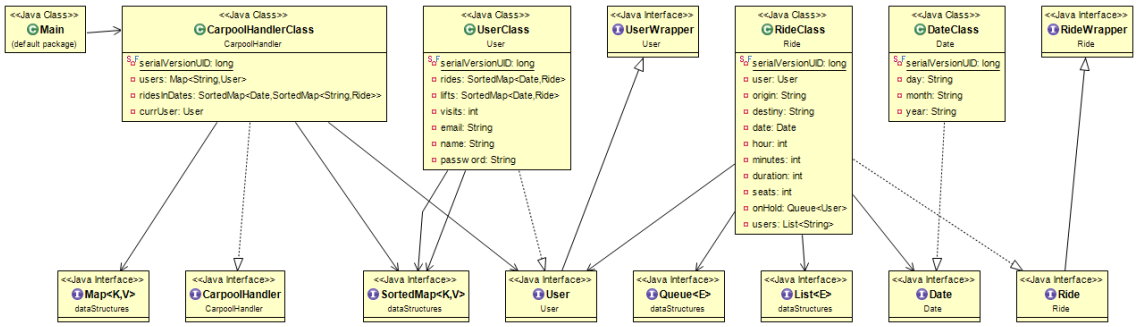
3ª Fase

Algoritmos e Estruturas de Dados

David Antunes nº 55045

Carolina Duarte nº 55645

Ano Lectivo 2019/20



Estruturas de Dados Utilizadas

UserClass

Classe que representa o user e tem guardadas as informações referentes a este, vai ser usados dois mapas ordenados, um para as boleias (Lifts) do user e outro para as deslocações (Rides). Ambos os mapas vão implementados com árvores RB, pois existia a necessidade de estes estarem ordenados e a sua inserção e remoção são mais eficientes do que as árvores AVL.

RideClass

Classe que representa uma deslocação e tem guardadas todas as informções referentes a esta, vai ser usada uma fila(queue) para guardar os users que estão à espera para ter lugar na deslocação, esta fila vai ser implementada com uma lista ligada simples, escolhemos esta implementação pois não sabemos à partida o qual vai ser o tamanho máximo da fila.

Usamos também uma lista de users (contém o seu email) para os users que têm lugar na deslocação, implementamos assim com a lista ligada simples, para uma remoção e inserção mais eficientes.

CarpoolHandlerClass

Classe que serve como classe de topo e gere as deslocações e os users. É usado uma árvore AVL com chave Date e com valor uma árvore vermelha e preta com chave um email e valor uma deslocação. Serve para guardar todas as deslocações do programa. Escolhemos a implementação da AVL para o mapa com chave Date pois a pesquisa é mais rápida face a árvore vermelha e preta e, por sua vez, escolhemos a árvore vermelha e preta para o mapa dentro da AVL, pois as remoções e inserções são mais rápidas do que a AVL.

As classes têm também um mapa de Users que foi implementado com a SepChainHashTable para ter uma inserção e pesquisa rápidas. Podia-se ter usado uma

tabela de dispersão fechada pois não ocorre remoções de utilizadores, usando menos recursos computacionais.

Funcionalidades/Comandos do programa

Variáveis do programa:

U – número de users do programa

R – número de rides do programa

RIU- número de rides dentro do utilizador (<R)

UIR – númer de users dentro de uma Ride(<U)

DIR – número de deslocações numa determinada data

LIU – número de lifts (deslocações onde o user apanha boleia)(<R)

D – número de datas

Comando Lista

O comando Lista vai se desdobrar em quatro subcomandos dependendo da indicação dada, se o utilizador escrever a palavra “minhas” à frente da palavra “Lista” , o comando listaMinhas vai ser executado, se for escrita uma data à frente a listaData vai ser executada, se for a palavra “boleias” a listaBoleias vai ser executada, se for a palavra “todas” a listaTodas vai ser executada e finalmente se for escrito um email de um User a listaEmail vai ser executada.

listaMinhas:

Este subcomando vai listar as deslocações dentro do mapa ordenado de deslocações(Rides) do currUser(variável User dentro da class topo(CarpoolHandlerClass) que guarda o utilizador com a sessão iniciada), para isso o currUser vai devolver um iterator dos valores do mapa e escreve as informações das deslocações.

Complexidade temporal:

Melhor caso: O(log(RIU) + RIU * UIR)

Pior caso: O(log(RIU) + RIU * UIR)

Caso médio: O(log(RIU) + RIU * UIR)

listaData:

Este subcomando vai listar as deslocações os emails dos Users que têm uma deslocação da Date que é pedida, para isso a classe CarpoolHandlerClass vai devolver um iterator que é o valor dentro do mapa de deslocações (RidesInDate) correspondente à data de seguida vai iterar os valores dentro do mapa devolvido.

Complexidade temporal:

Melhor caso: O(log(DIR) + DIR)

Pior caso: O(log(DIR) + DIR)

Caso médio: O(log(DIR) + DIR)

listaBoleias:

Este subcomando vai listar todas as deslocações em que o current user apanha boleia , para isso o currUser devolve à classe topo que por sua vez devolve à Main um iterator com todas as deslocações em que tem boleia e escreve todas as informações da deslocação.

Complexidade temporal:

Melhor caso: O(log(LIU) + LIU)

Pior caso: O(log(LIU) + LIU)

Caso médio: O(log(LIU) + LIU)

listaTodas:

Este subcomando vai listar todas as deslocações existentes no programa, para isso a CarpoolHandlerClass vai devolver um RideIterator que vai iterar sobre o mapa ridesInDates e sobre os seus valores(mapas) para devolver todas as rides no programa.Este subcomando vai escrever a Date e o email do User da deslocação.

Complexidade temporal:

Melhor caso: O(R)

Pior caso: O(R)

Caso médio: O(R)

listaEmail:

Este subcomando vai listar todas as deslocações do User cujo email foi dado, para isso a CarpoolHandlerClass vai devolver um iterator RideInMain com as Rides do User dado assim, vai ao mapa de Users procurar o User dado que por sua vez vai devolver um iterator com as deslocações. Escreve todas as informações das deslocações na consola.

Complexidade temporal:

Melhor caso: O(RIU)

Pior caso: O(RIU)

Caso médio: O (RIU)

Comando Regista

Este comando vai registar um novo User. Vai ser dado um email, um nome e uma password. Vai ser verificado se o novo user ainda não existe no mapa e se a password dada é válida, só assim um novo utilizador pode ser criado. O CarpoolHandlerClass vai inserir o novo user no mapa de Users.

Complexidade temporal:

Melhor caso: O(1)

Pior caso: O(U)

Caso médio: O (1 + fator de ocupação)

Comando Sai

O User que está com a sessão iniciada vai sair da sessão e o programa volta ao modo inicial. Para isso o CarpoolHandlerClass vai igualar o currUser a null e retornar o nome do User que vai sair da sessão. Vai escrever o nome do User na consola. Este comando só funciona em modo sessão.

Complexidade temporal:

Melhor caso: O(1)

Pior caso: O(1)

Caso médio: O (1)

Comando Entrada

Um User dá entrada no programa com o email e a password. Vai ser verificado se o user existe no mapa de Users e se a password corresponde ao User, só assim o comando entrada tem sucesso. Se tiver sucesso o currUser passa a ser igual ao user dado.

Complexidade temporal:

Melhor caso: O(1)

Pior caso: O(U)

Caso médio: O (1+ fator de ocupação)

Comando Termina

Este comando termina o programa, escrevendo a respetiva mensagem de fim de execução.

Complexidade temporal:

Melhor caso: O(1)

Pior caso: O(1)

Caso médio: O (1)

Comando Nova

Este comando, quando o utilizador fornece os dados necessários, vai verificar se o currUser já tem alguma deslocação nesse dia, para isso vai procurar nos mapas do currUser e vai verificar se todos os dados são válidos, só assim pode ser acrescentada uma deslocação. A nova deslocação vai ser adicionada ao mapa de rides do currUser e vai ser também adicionada ao mapa RidesInDate que contêm todas as rides do programa.

Complexidade temporal:

Melhor caso: O(1)

Pior caso: O(log(U) + log(R) + log(LU))

Caso médio: O (log(U) + log(R))

Comando Remove

Este comando, quando o utilizado fornece uma data remove a deslocação. Vai ser verificado no mapa de deslocacações do currUser se a esta já tem users lá dentro, se a data está bem formatada e se a deslocação existe nesse dia, só assim é possível faver a remoção. A remoção passa por remover a deslocação do mapa de deslocações do utilizador e do mapa de todas as deslocações.

Complexidade temporal:

Melhor caso: O(1)

Pior caso: O(log(R) + log(DR) + log(RIU))

Caso médio: O (log(R) + log(DR) + log(RIU))

Comando Ajuda

Este comando imprime o menu de ajuda dependendo se o programa está em modo inicial ou modo sessão. Para ver isto, é verificado se a variável correspondente ao utilizador corrente está a null(significa que ninguém tem a sessão iniciada) ou se tem algum user lá dentro.

Complexidade temporal:

Melhor caso: O(1)

Pior caso: O(1)

Caso médio: O (1)

Comando Boleia

Este comando, dado um email e uma data vai tentar adicionar um User a uma Ride, vai ser verificado se o user existe no mapa de users e se existir vai ser verificado se essa ride existe dentro do mapa de deslocações dele, vai ser verificado se o user é o não é o mesmo que o currUser, se o currUser ainda não tem nada nessa data e se a data está no formato válido, só assim o comando tem sussesso. O currUser vai ser adicionado à deslocação se ainda houver espaço na na lista de users o currUser é adicionada, se não é metido na lista de espera.

Complexidade temporal:

Melhor caso: O(1)

Pior caso: O((log(RIU) + log(LU)) + log(U))

Caso médio: O (log(RIU) + log(LU))

Comando Retira

Este comando dado uma data vai retirar do mapa de boleias do user currente a boleia (caso a tenha) e vai retirar da deslocação em si, o user da lista de users. Se a data não tiver bem formatada e a ride não exitir no currUser então o comando não tem sussesso.

Complexidade temporal:

Melhor caso: O(1)

Pior caso: O(log(LU)) + RIU)

Caso médio: O(log(LU)) + RIU)

Comando Consulta

Este comando dado um email e uma data vai procurar no mapa de users o email dado e vai procurar a deslocação correspondente à data dada no map de rides do user e imprime todas as informações referentes à deslocação. Vai ser verificada se o user existe, se a data está bem formatada e se a ride existe dentro do user, só assim o comando têm sucesso.

Complexidade temporal:

Melhor caso: O(1)

Pior caso: O(Log(U))

Caso médio: O (Log(U))

Complexidade Espacial

(U * dimU) + (R * dimR) + (1.1 * U + 2U) + (4*D * 4*DR) + (4* RIU) +(4*LU) + (2* UIR) + (2*RIU)

dimU – Espaço ocupado pela objeto User

dimR – Espaco ocupado pela objeto Ride

1.1 * U – espaço ocupado pelo mapa users da classe Carpoolhandler

(4*D * 4*DR) – espaço ocupado pelo mapa ridesInDate da classe carpoolhandler

(4* RIU) – mapa de deslocações do user

(4*LU) – mapa de boleias do user

(2* dim(String)) – lista de emails dos utilizadores na ride

(2*UIR) – fila de espera de utilizadores na ride