Companhia de Táxis

Mestrado Integrado em Engenharia Informática e Computação

Algoritmos e estruturas de dados

2016/2017

Beatriz de Henriques Martins - up201502858@fe.up.pt

Diogo Peixoto Pereira - up201504326@fe.up.pt

Pedro Miguel Ferraz Nogueira da Silva - up201505460@fe.up.pt

18/11/2016

Índice

Descrição

Este projeto realizado em C++ permite gerir uma companhia de táxis.

A companhia contém um grupo de clientes que estão presentes na sua base de dados quer sejam eles particulares ou mesmo empresas que usem os seus serviços regularmente. Pode também aceitar clientes que usem ocasionalmente os seus serviços, mas neste caso os mesmos não ficam guardados na sua base de dados.

A companhia oferece diversos tipos de pagamentos desde monetários (na hora em que o serviço é utilizado), através de multibanco, a crédito ou pagamentos no final de cada mês. Para este último caso os clientes ocasionais não terão acesso.

Todos estes procedimentos são geridos e monitorizados por um grupo de classes.

Solução Implementada

Para este projeto decidiu-se criar um grupo de classes principais que representam a companhia de táxis, os clientes (particulares e empresas), os pagamentos e as viagens. Tem também algumas classes auxiliares com funcionalidades importantes na implementação do trabalho como: data, hora, percursos e táxis.

Por questões de melhor organização de informação foram criadas novas estruturas de dados que se baseiam em árvores binárias de pesquisa, em tabelas de dispersão e em filas de prioridades.

Para além disto o projeto contém uma classe menu que trata de realizar toda a intercomunicação entre as diversas classes bem como a interface com o utilizador.

➢ Classe Companhia Táxis

Um objeto desta classe representa a companhia de táxis em torno da qual se vai desenvolver todo o programa. Este objeto tem um nome, um capital associado, guarda a informação de todos os seus clientes registados e tem o registo de todos os seus táxis.

Membros dado:

1. string nome;
2. float capital;
3. vetor<Cliente *> Clientes;
4. vetor<Taxi *> taxisTotais;
5. tabCli inativos;
6. tabCli ativos;
7. BST<Viagem> viagens;
8. BST<Viagem> viagens_ocasionais;
9. priority_queue<Taxi*> taxis;

Construtores:

1. CompanhiaTaxis();
2. CompanhiaTaxis(string a, float c);

Para além dos membros dados e dos construtores da classe, existem ainda outras funções que permitem retornar alguns membros dados da classe, as funções get.

Funções get:

1. string getNome();
2. float getCapital();
3. vector<Cliente *> getClientes() const;
4. vector<Ocasionais> getOcasionais() const;

5. vector<Taxi> getTaxisTotais() const;

Tem também uma série de funções que lidam com os táxis, adicionando, removendo e procurando táxis disponíveis.

Funções para os táxis

1. void adicionaTaxi(Hora horI, Hora horO);
2. bool removeTaxi(int id);
3. int procuraTaxi(int n) const;
4. void mostrarTaxis();

Esta classe tem o trabalho de gerir todos os seus clientes quer sejam eles particulares ou empresas. Para isso existe um conjunto de funções que o fazem:

Funções que adicionam e retiram clientes:

1. void adicionaClienteParticular(string nome, string morada, string email, int nT, int nif, string tipoPagamento);
2. void adicionaClienteEmpresa(string nome, string morada, string email, int nT, int nif, string tipoPagamento, int numFuncionarios);
3. bool removeCliente(int id);
4. void setClientes(vector<Cliente*> c);
5. void concaClientes(vector<Cliente*> c);

Para além destas existem ainda umas funções que procuram clientes e mostram os mesmos de forma organizada e ordenada por algoritmos de ordenação específicos (mais à frente este método irá ser apresentado).

Funções de procura de clientes:

1. int procuraCliente(int id) const;

2. int ultimoIDcliente();

Funções que apresentam dados sobre clientes:

1. void mostrarClientesPorCapital();
2. void mostrarClientesPorID();

De seguida tem as funções que trabalham com as viagens de cada cliente. Estas funções controlam a criação de viagens e tratam de as adicionar ao conjunto de viagens a que pertencem. Associado a isto temos também de realizar pagamentos sendo estes feitos quando se criam as viagens (se forem pagamentos feitos na hora) ou chamando funções específicas. No caso dos clientes não registados o pagamento é feito quando se criam as viagens uma vez que não se pode pagar no final do mês.

Funções sobre as viagens:

1. void fazerViagemOcasional(Data dia, Hora horaIn, Percurso p1);
2. void fazerViagemCliente(int id, Data dia, Hora horaIn, Percurso p1, bool disc, float per);
3. void cobrarPagamentoMensal();
4. void somaCapital(float n);

Para melhor organização do trabalho foram criadas quatro novas estruturas de dados para organizar clientes e táxis.

Funções sobre as tabelas de dispersão:

1. tabCli getInativos() const;
2. tabCli getAtivos() const;
3. void removeClienteInativo(Cliente* c);
4. void removeClienteAtivo(Cliente* c);
5. void criarTabelasClientes();
6. void resetTabelasClientes();

Estas funções servem para criar e gerir tabelas de dispersão que guardam clientes ativos e inativos. Um cliente diz-se inativo se estiver um mês sem realizar uma viagem, mas mal faça uma viagem, o cliente troca da tabela de inativos para a tabela de ativos.

Funções sobre a fila de prioridade:

1. priority_queue<Taxi*> getTaxis() const;
2. void setTaxis(priority_queue<Taxi*> t);

Estas funções servem para criar e gerir a fila de prioridades dos táxis. Quando se marca uma viagem, o táxi que estiver à frente na tabela de prioridade é escolhido se tiver horário para a realizar, senão passamos ao próximo táxi da fila.

Funções sobre a árvore binária de pesquisa:

1. BST<Viagem> getViagens();
2. BST<Viagem> getViagensOcasionais();
3. void addViagemBST(Viagem &v);
4. void addViagemBSTOcasionais(Viagem &v);
5. void mostrarViagensBST();
6. void mostrarViagensBSTOcasionais();

Estas funções servem para criar e gerir a árvore binária de pesquisa de viagens. Cada viagem realizada entra nesta BST por ordem de nome de cliente e de viagens mais antigas. Estas funções também permitem mostrar as viagens no monitor a partir da BST.

➢ Classe Utentes

Esta classe é a classe base para todos os clientes da companhia de táxis, quer sejam registados ou não. Tem dois membros dados que são comuns a todas as suas funções derivadas, o nome (string nomeC) e o valor gasto na companhia de tàxis (Pagamento custo). Para além disso esta classe contém um construtor, uma função que retorna o nome do cliente e o valor gasto, bem como funções que alteram os membros dados.

Funções da classe:

1. Utente(string nome, string tipo_pagamento);
2. string getNomeC() const;
3. void setNomeC(string nome);
4. void changeCustoTotal(float n);
5. void changeCustoTipo(string tipo);
6. Pagamento getCusto();

- Classe Ocasionais

Esta classe é derivada da classe utente e diz respeito aos clientes não registados na companhia, sendo que apenas contém um construtor para facilitar os cálculos de custos de viagens.

- Classe Cliente

Esta também é uma classe derivada da classe utente mas por outro lado serve de classe base aos dois tipos de clientes com os quais a companhia de táxis trabalha, sendo eles particulares e empresas. Os membros dados desta classe complementam a informação fornecida nos membros dados da classe utentes.

Membros dados:

1. static int ultidc;
2. int id;
3. int NIF;
4. string morada;
5. string email;
6. int numeroTelemovel;
7. vector<Viagem> historicoViagens;
8. vector<Viagem> viagensMensais;
9. int cartaoPontos;

Nesta classe estão implementados dois construtores e também um destrutor, sendo eles:

Construtores e destrutor:

1. Cliente(int id, string nome, int nif, string morada, string email, int numeroTelemovel, string tipoPagamento, bool pOuE);
2. Cliente(string nC, string m, string mail, int nT, int nif, string tipoPagamento, bool pOuE);
3. virtual ~Cliente() {}

À semelhança da classe companhia de táxis, existem uma série de funções que permitem obter os valores guardados nos membros dados para posterior uso, bem como funções que permitem alterar esses mesmos valores em cada objeto desta classe.

Funções get:

1. int getID() const;
2. int getNIF() const;
3. string getMorada() const;
4. string getEmail() const;
5. vector<Viagem> getHistoricoViagens();
6. int getNumeroTelemovel() const;
7. int getPontos();

Funções set:

1. void setNIF(int nif) const;
2. void setMorada(string m);
3. void setEmail(string mail);
4. void setNumeroTelemovel(int nT);

Como se pode observar nos membros dados, existe um sistema de pontos atribuído a cada cliente registado na companhia. Estes pontos são atribuídos consoante a utilização dos serviços da companhia, ou seja, quanto mais viagens realizar, maior o número de pontos atribuídos. Os pontos podem depois ser usados para descontar nos preços de viagens seguintes, sendo os mesmos reduzidos após a utilização dos descontos. Existem duas funções que realizam estas duas funcionalidades.

Funções de pontos:

1. void aumentaPontos();
2. void resetPontos();

As viagens têm um papel bastante importante nesta classe uma vez que são divididas em dois vetores. No vetor do histórico e viagens, entram todas as viagens que são realizadas pelo cliente e no vetor de viagens mensais entram todas as que o cliente escolhe pagar ao fim do mês. Para além disto existem mais duas funções bastante

úteis que realizam os pagamentos do mês e que limpam o vetor de viagens mensais, deixando espaço para se adicionar novas viagens mensais que ainda não foram pagas.

Funções de viagens:

1. void addViagemMensal(Viagem v);
2. void addViagemHistorico(Viagem v);
3. void resetMes();
4. float fimdoMes();

Outras funcionalidades importantes que esta classe oferece são seis funções com impactos diferentes no projeto.

Primeiro tem uma função virtual que calcula e atribui os descontos mensais, sendo a razão por trás da sua atribuição destes descontos diferentes entre clientes particulares e empresas (classes derivadas desta classe). Daí esta função ser uma função virtual.

```
virtual float giveMonthlyPromotion(float p);
```

De seguida a função virtual que mostra informações sobre os clientes. È virtual uma vez que a classe derivada empresa tem um membro dado extra que indica o número de funcionários da empresa. Existem ainda duas funções que mostram as viagens mensais e totais realizadas pelo cliente.

```
virtual string mostrarCliente();
void mostrarViagens();
void mostrarViagensmensais();
```

A somar a isto existe ainda uma função virtual que nos indica o tipo de cliente que o objeto em questão é, se é uma empresa ou um particular.

```
virtual bool isParticular();
```

Finalmente existe um overload do operador “<” que será utilizado pela função de ordenação para ordenar clientes.

bool operator <(Cliente c2);

- Classe Particular

Classe derivada da classe clientes, sem membros dados extra e que contém apenas dois construtores e a implementação das três funções virtuais da classe cliente.

1. Particular(int id, string nome, int nif, string morada, string email, int numeroTelemovel, string tipoPagamento, bool pOuE);
2. Particular(string nC, string m, string mail, int nT, int nif, string tipo_pagamento, bool pOuE);
3. float giveMonthlyPromotion(float p);
4. string mostrarCliente();
5. virtual bool isParticular();

- Class Empresa

Esta classe também é derivada da classe cliente e, à semelhança da classe particular, as funções que a compõem são um construtor e as três implementações de funções virtuais. Esta classe tem no entanto uma diferença em relação à classe particular, que é um membro dado extra que contem o número de empregados que a empresa tem. Esta informação é usada para dar as promoções mensais.

1. Empresa(int id, string nC, string m, string mail, int nT, int nif, string tipo_pagamento, int numeroFuncionarios);
2. float giveMonthlyPromotion(float p);
3. string mostrarCliente();

4. virtual bool isParticular();

Este grupo de classes possui ainda uma classe ClienteInexistente que é classe que trata a exceção para o caso de não haver nenhum cliente com o id que se tentou encontrar.

➢ Classe Pagamento

Esta classe possui dois membros dados que identificam o tipo de pagamento e o montante associado ao cliente que contém o objeto criado por esta classe e dois construtores.

Membros dados:

1. float dinheiro;
2. string tipo;

Construtores:

1. Pagamento();
2. Pagamento(string t);

Por fim tem quatro funções que retornam os valores dos membros dados e que os alteram respetivamente.

Funções get:

1. float getTotal();
2. string getTipo();

Funções change:

1. void changeTotal(float n);
2. void changeTipo(string t);

➢ Classe Percurso

Esta classe possui três membros dados que identificam o tipo de o local de início de viagem e o destino bem como a distância entre os dois locais. Para além disso possui dois construtores.

Membros dados

1. string localPartida;
2. string localDestino;
3. int distancia;

Construtores:

1. Percurso();
2. Percurso(string localPart, string localDest, int distancia);

À semelhança das outras classes existem funções do tipo get e set que são as seguintes.

Funções get:

1. int getDistancia() const;
2. string getLocalPartida() const;
3. string getLocalDestino() const;

Funções set:

1. void setDistancia(int distancia);
2. void setLocalPartida(string localPart, int dist);
3. void setLocalDestino(string localDest, int dist);

Está implementado ainda um overload do operador “<<” para mostrar a informação dos percursos.

friend ostream & operator <<(ostream os, Percurso p);

➢ Classe Taxi

Esta classe serve para guardar informação sobre cada táxi pertencente à companhia. Um dos membros dados desta função é um static, isto para ser possível ir atribuindo números de táxis sucessivamente maiores e sem repetição.

Membros dados:

1. static int ultinumeroTaxi;
2. int numeroTaxi;
3. float rentabilidade;
4. Hora horaIn;
5. Hora horaOff;
6. Float dispo;

Esta classe contém dois construtores e um destrutor para o caso de eliminarmos táxis.

Construtores:

Taxi(Hora horI, Hora horO);

Taxi(int n, float r, Hora horI, Hora horO);

Destrutor:

~Taxi();

Como na classe percurso existe um conjunto de funções get e set e também um overload do operador “<<” com a mesma funcionalidade que a dessa classe.

Funções get:

1. int getDisponivel(Hora hi, Hora hf);
2. int getNumeroTaxi() const;
3. float getRentabilidade();

4. Hora getHoraIn();
5. Hora getHoraOff();

Função set:

1. void setRentabilidade(float n);

Overload do operador “<<”:

friend ostream & operator <<(ostream & os, Taxi t);

Devido à criação de uma fila de prioridade de táxis foi necessário criar mais três funções que permitem manipular a disponibilidade dos táxis e comparar dois táxis baseando-nos nessa disponibilidade, para poderem ser inseridos na fila.

1. Float getDispo()const;
2. Void changeDispo(Float n);
3. Bool operator <(const Taxi t);

Devido à necessidade de usar apontadores para os táxis dentro da fila de prioridade, foi necessário criar uma classe extra que transforma os apontadores em táxis.

Como na classe de clientes, esta classe também possui uma classe de exceções que são lançadas sempre que o táxi que se pretende usar não está disponível.

➢ Classe Viagem

Esta classe tem como propósito criar as viagens e realizar o pagamento das mesmas.

Membros dados:

1. Data data;
2. Percurso deslocaçao;
3. bool pago;
4. float custo;
5. Hora horaIn;
6. Hora horaOut;
7. String cliente;

A classe possui ainda dois construtores e um destrutor, bem como um conjunto de funções get e set que, à semelhança das outras classes, retornam os valores dos membros dados e alteram os mesmos.

Construtores:
Viagem(Data dia, Hora horaIn, Percurso p1);
Viagem(Data dia, Hora horaIn, Hora horaOut, Percurso p1, float c);

Destrutor:
~Viagem();

Esta classe contém também uma série de funções do tipo get e set.

Funções get:

1. Data getData() const;
2. Hora getHoraIn() const;
3. Hora getHoraOut() const;

4. string getPartida() const;
5. string getDestino() const;
6. string getCliente()const;
7. float getCustoViagem() const;
8. Percurso getDeslocacao();

Funções set:

1. string setDestino() const;
2. void setHoraIn(Hora hora);
3. void setHoraOut(Hora hora);
4. void setPartida(string localP, int dist1);
5. void setDestino(string localD, int dist2);

Tal como o nome da classe indica, o propósito desta classe é tratar das viagens, calculando o tempo de viagem e consequentemente a hora a que a viagem terminará, calculando o preço da viagem e realizar o pagamento.

Funções sobre as viagens:

1. float horaFinal();
2. float pagarViagem();
3. void modificaCusto(float per);

Como noutras classes temos uma função overload do operador “<<” para mostrar informações das viagens.

friend ostream & operator <<(ostream & os, Viagem &v);

Por fim existe uma função que passa as informações dos clientes para string que poderão ser mostradas no ecrã.

string toString();

Para além destas funções foi ainda criada uma overload do operador < para ser possível introduzir as viagens na BST criada.

bool operator <(const Viagem v);

Classes e funções extra

➢ Sequencial Search

Esta função é usada para procurar elementos num vetor e é utilizado por classes como a de clientes e de viagens.

➢ Classe Data

Esta classe é utilizada nas classes que requerem dias e para isso foi necessário criá-la. A classe contém membros dados relativos a dia, mês e ano, tem funções de get e set por questões de alteração de valores e de comparação entre datas bem como uma função para podermos mostrar a informação dos objetos desta classe no monitor.

Membros dados:

1. int dia;
2. int mês;
3. int ano;

Construtores:

1. Data();
2. Data(int d, int m, int a);

Funções get:

1. int getDia() const;
2. int getMes() const;
3. int getAno() const;

Funções set:

1. void setDia(int d);
2. void setMes(int m);
3. void setAno(int a);

Funções extra:

1. string toString();
2. friend ostream & operator <<(ostream os, Data d);
3. bool operator <(const Data d);

Para além das funções da classe, esta está preparada para lançar uma exceção sempre que a data pretendida pelo utilizador não for aceitável.

➢ Classe Hora

Esta classe é utilizada nas classes que requerem horas (viagens) e para isso foi necessário criá-la. A classe contém membros dados relativos a hora, minutos e segundos, tem funções de get, uma função que altera os membros dados, bem como uma função para podermos mostrar a informação dos objetos desta classe no monitor.

Para além disso existem dois overload de operadores de “<” e “<=” para fazer as comparações faladas anteriormente.

Membros dados:

1. int hora;
2. int minutos;
3. int segundos;

Construtores:

1. Hora();
2. Hora(int h, int m, int s);

Funções get:

1. int getHora() const;
2. int getMinutos() const;
3. int getSegundos() const;

Funções soma:

1. Hora somaHoras(int min);

Funções extra:

1. string toString();
2. friend ostream & operator <<(ostream & os, Hora h);

Overload de operadores:

1. bool operator <(Hora h2);
2. bool operator <=(Hora h2);

Para além das funções da classe, esta está preparada para lançar uma exceção sempre que a hora seja inválida.

➢ Menu

A classe menu serve para fazer a ligação entre as várias componentes do trabalho e está também responsável por toda a interface do programa. No início do programa os menus permitem a leitura dos ficheiros de texto que irão criar a companhia com os seus clientes, táxis e viagens realizadas. No final também tem a função de guardar novamente nos ficheiros as novas alterações que podem ter ocorrido durante o programa.

- Construtor Menu

Este construtor lê do ficheiro da companhia de táxis criando a mesma. De seguida entra no menu de início que tem apenas a função de inicializar o programa ou terminá-lo após a leitura feita pelo construtor.

- Menu Entrar

Neste menu acede-se ao resto dos ficheiros de dados e carregam-se essas informações para as devidas estruturas de dados. A funcionalidade deste menu termina com a chamada do menu companhia.

- Menu Companhia

Este é o menu que acede a todas os menus restantes, sendo que permite trabalhar com os clientes, táxis e com a companhia.

Dependendo da opção que se escolhe pode ser mostrado no monitor as informações de clientes, dos táxis e da própria companhia ou mesmo chamar o menu que irá acrescentar viagens e remover clientes.

Neste menu podemos ainda manipular as BST para acrescentar viagens e mostrar as mesmas no ecrã.

- Menu Clientes

Permite realizar viagens e remover clientes. Para além disso contém todas as ligações que tornam possível mostrar as informações dos clientes particularmente ou mesmo todos.

Diagrama UML

Conclusão

Fazer este trabalho foi bastante gratificante uma vez que nos fez perceber as nossas dificuldades como programadores mas também nos serviu para melhorarmos as nossas capacidades de trabalho de equipa, divisão de tarefas e interajuda. No fim conseguimos realizar todas as tarefas impostas no trabalho bem como todas as implementações extra que achámos necessárias para o bom funcionamento do mesmo

As maiores dificuldades encontradas centraram-se nos seguintes pontos:

- Escrita e leitura nos ficheiros de texto;
- Tratamento dos inputs dos utilizadores.

Após a realização de alterações recorrentes da parte 2 do trabalho surgiu uma nova dificuldade que se centra na utilização de *pointers* na fila de prioridade de táxis.

Na realização deste trabalho cada um dos elementos do grupo contribuiu significativamente para a sua realização e posterior conclusão. Dito isto, todos desempenhámos o mesmo esforço e empenho para que o trabalho atingisse o seu final, sem ser deixado nada por fazer.