Universidade do Porto

Faculdade de Engenharia

FEUP

Relatório Projecto 1: Grafos

T3: Loja de Comércio Eletrónico

Concepção e Análise de Algoritmos 2012/13

Turma 2MIEIC2 - Grupo D

Elementos:
Jorge Reis – 200800544 – ei08053@fe.up.pt
Miao Sun – 200803743 – ei08162@fe.up.pt
Vítor Castro – 200900648 – ei09131@fe.up.pt

26 de Abril de 2013

Introdução

Foi proposto um projecto, no contexto da disciplina de Conceção e Análise de Algoritmos, que consiste na elaboração de uma aplicação para gestão de uma loja de comércio eletrónico. Assim iniciou-se a implementação desta aplicação apresentada neste relatório com base nas premissas de que deveria ser permititida a fácil e eficiente gestão da loja, tendo em conta a necessidade desta de manter registos das lojas tradicionais (não virtuais) e dos clientes das respetivas zonas.

1. Descrição do Problema

No problema que nos foi proposto, era pedido que fosse desenvolvida uma aplicação na linguagem de programação C++ para gerir uma loja eletrónica.

Os clientes das lojas efectuam as encomendas online e a loja Eletrox (loja eletrónica) encaminha essas encomendas para lojas tradicionais (não virtuais) associadas à zona de residência do cliente. Estas lojas devem dar uma resposta tão rápida quanto possível à encomenda do cliente, ficando responsáveis pela entrega do produto na casa do cliente. O mapa que representa a área de atuação da loja ELETROX encontra-se dividido em zonas, definidas à priori, que não se sobrepõem. Em cada zona só existe uma loja tradicional. Cada uma destas é responsável por servir os clientes da zona em que se encontra.

No entanto, se a loja tradicional não tiver disponibilidade para determinada encomenda (não vende o produto, ou não o tem em stock do mesmo), a loja virtual ELETROX contacta as restantes lojas tradicionais até conseguir encontrar uma que consiga satisfazer a encomenda.

Neste processo de contactar uma loja tradicional para satisfazer determinada encomenda, o critério a usar pode ser:

1 - Menor distância da loja que possui esse produto ao cliente;

2 - Menor número de zonas a atravessar, entre a loja que possui esse produto e a residência do cliente.

É ainda permitida a edição do mapa, a definição de zonas, localização de clientes e lojas tradicionais e é possível ver graficamente o mapa recorrendo para à biblioteca de representação de grafos: *GraphViewer*. Permite também observar, passo a passo, a forma como a escolha da loja é realizada, a partir de animações no grafo.

2. Implementação

2.1 Descrição da Solução Implementada

No planeamento da forma de abordagem ao problema, decidiu-se utilizar uma classe principal *LojaElectronica*, que contém um vector de apontadores para Clientes, e outro para encomendas, e um objecto do tipo *Graph<Zona*>* que permite manusear toda a estrutura de dados em grafo. De ressalvar que a implementação da Classe *Graph*, assim como de *Vertex* e *Edge*, foram em parte fornecidas e desenvolvidas durante as aulas teórico-práticas da unidade curricular.

Cada *Vertex* da estrutura *Graph*, possui como tipo uma Zona, que possui um apontador para uma loja associada (ou mesmo nenhuma caso a zona não possua loja). Cada *Edge* da estrutura representa a ligação existente entre *Zonas*, denotada pela distância entre as mesmas.

Por fim, cada loja contém um "vector" de produtos existentes na mesma, e identificados por uma designação, um preço e um stock (que é controlado, ou seja, decrementado, aquando de uma encomenda). Importa referir que cada cliente possui um apontador para sua zona de residência, determinante para a escolha da loja que o irá servir.

Assim, a parte mais relevante da aplicação, trata-se de quando se realiza uma encomenda. É pedida a identificação do cliente, e obtida automaticamente a sua zona de origem. De seguida, é apresentada uma lista de todos os produtos existentes em todas as lojas associadas, afim de o utilizador escolher a que pretende encomendar. Agora, mediante o algoritmo de pesquisa definido no início do programa, é percorrido o grafo por ordem de proximidade à zona origem (do cliente), verificando para cada zona se a mesma possui o produto pretendido, a fim de encontrar a primeira que satisfaça o pedido.

Utilizando o *GraphViewer*, permitimos observar o mapa de zonas, e ao realizar uma encomenda verificar passo a passo, com base numa animação criada, a escolha feita: inicialmente é identificada a azul a zona origem da encomenda (cliente); depois são pintadas a verde e vermelho as zonas em que a loja tem capacidade para satisfazer o pedido ou não, respectivamente; e por fim é pintado a amarelo o caminho desde a zona inicial até à zona onde se encontra a loja que irá tratar a encomenda, representada por fim com a cor laranja.

2.2 Classes Implementadas:

O nosso projecto está dividido em várias classes. Cada uma destas representa uma das partes importantes do nosso projecto.

Construímos as seguintes classes:

- **Cliente**
- **Encomenda**
- **Loja**
- **LojaEletronica**
- **Produto**
- **Zona**
- **Excepcao**

Recorremos ainda a outras classes já existentes sendo elas:

- **Connection**
- **Graph (incluindo Vertex e Edge)**
- **Edgetype**
- **GraphViewer**

Usou-se também funções auxiliares num ficheiro "*funcoes.h*", de salientar uma de *split* de *strings*, uma função que retorna a data/hora actual e ainda um método de *pesquisa sequencial* adaptado ao nosso problema.

Em cada uma das anteriores classes implementaram-se os métodos que permitem actualizar (métodos set) e ler (métodos get) os campos de cada classe relacionada com o seu nome.

Na **classe Cliente**, implementaram-se os métodos *getCodCliente*, que retorna o código do cliente, tendo em conta que cada cliente tem um código e este código nunca se repete nem pode ser alterado, assim como *getNome*, *getNIF*, *getContacto*, *getMorada*, *getEmail*, que permitem retornar as informações pessoais de cada cliente, *getNumClientes* que retorna o número actual de clientes registados na loja electrónica, *getZona* que retorna um apontador para a zona do cliente, *setNome*, *setMorada*, *setContacto*, *setEmail* que permitem actualizar as informações pessoais do cliente, *imprimeCliente* que permite imprimir no ecrã as informações de cada cliente. Ainda foi implementado o *operator==* que permite ver se dois clientes são iguais, sendo que tal acontece se tiverem o mesmo nome e o mesmo número de identificação fiscal.

Na **Encomenda**, implementaram-se os métodos *getCodEncomenda* que retorna o código da encomenda, *getData* que retorna a data em que a encomenda foi feita, *getCliente* que retorna o cliente que fez a encomenda, *getLoja* que retorna a loja onde a encomenda está a ser tratada, *getProduto* que retorna o produto que o cliente pretende adquirir, *setData, setCliente, setProduto* que permitem actualizar os campos data, cliente e produto para uma

encomenda, *imprimeEncomendas* que permite imprimir no ecrã as informações de cada encomenda e ainda *operator==* que permite ver se duas encomendas são iguais, isto acontece quando duas encomendas têm o mesmo código de encomenda.

Na **classe Loja**, implementou-se *getCodLoja* que permite retornar o código da loja**,** *getNome* e *getMorada* que permitem retornar nome e morada, *getProdutos* que permite retorna um vetor com todos os produtos da loja; *setNome* e *setMorada* para actualizar o nome e morada da loja, *addProduto, listaProdutos* e *removeProduto* que permitem adicionar produtos a uma loja, listar produtos e remover um produto da loja. Ainda *imprimeLoja* permite imprimir no ecrã toda a informação da loja, e o *operator==* permite verificar se duas lojas são iguais, o que acontece quando têm o mesmo código.

Na **classe Produto**, implementaram-se os métodos *get* e *set* para os campos: código do produto, preço, stock e designação. Ainda se implementou *info* que permite imprimir no ecrã todas as informações de um produto. Implementámos também o método *operator==* que permite verificar se dois produtos são iguais; isto acontece quando têm o mesmo código.

Na **classe Zona** foram implementados os métodos *get* e *set* para os campos designação, código da zona, loja (visto que uma zona tem sempre uma loja no máximo). Implementámos também o método *operator==* que permite verificar se duas zonas são iguais; isto acontece quando têm a mesma designação.

Na **classe LojaEletronica**, classe que consideramos a classe principal do nosso projecto, implementamos os métodos *listaClientes, addCliente* e *removeCliente* que permitem listar, adicionar e remover clientes, *procuraCliente_Nome* que permite procurar no sistema um cliente pelo seu nome e retorna um apontador para o cliente, *nomesProdutos* que permite procurar todos os produtos de todas as lojas do sistema, *listaZonas, addZona* e *procuraZona***,** que permitem respectivamente listar todas as zonas do sistema, adicionar novas zonas ao sistema e procurar zonas no sistema; a procura pode ser feita por ID da zona e também pelo seu nome. Adicionou-se também o método *addZonaGrafo* que permite adicionar um novo vértice ao grafo visto que o nosso grafo é um grafo de zonas. *addArestaBidirecional* é um método que permite adicionar ao grafo uma aresta bidireccional de forma a torna o nosso grafo num grafo não dirigido. *addLoja***,** *removeLoja* e *listaLojas* que permitem respectivamente adicionar e remover lojas do sistema e listar todas as lojas existentes, *addEncomenda***,** *removeEncomenda*, *listaEncomendas*, *procuraEncomenda* que permitem respectivamente adicionar e remover uma encomenda, listar toda as encomendas do sistema e procurar uma encomenda pelo respectivo id. Implementámos ainda os métodos *load* e *save* que permitem ler e guardar em ficheiros de texto informação de nós e arestas para a construção do grafo e ainda de entidades importantes do sistema como clientes, produtos, lojas, encomendas. Implementou-se ainda o método *Mapa* que permite visualizar graficamente um grafo.

No ficheiro **Funcoes.h,** foram implementados/adaptados alguns métodos auxiliares bastante úteis que usamos com alguma frequência no nosso projecto sendo eles: *isDouble* que permite verificar se um número é decimal ou não**,** *isDigit* que permite verificar se um caracter inserido pelo utilizador é ou não um dígito**,** *intInput* que permite validar se um caracter

introduzido é um inteiro**,** *pesquisaSequencial* que permite verificar se um "vector" contém determinada "string" e *dataActual* que retorna uma "string" com a data actual do sistema.

Por fim, temos a função **Main,** sendo esta responsável pelo arranque da aplicação. Através desta classe conseguimos aceder a uma menu que nos permite utilizar a aplicação de forma mais simples e intuitiva.

2.3 Utilização:

Assim, a aplicação implementada permite uma total gestão de uma loja electrónica, controlando sobretudo as encomendas efectuadas por clientes de diversas zonas e decidindo que loja tradicional irá tratar da encomenda.

Inicialmente, no menu principal do programa, o utilizador pode escolher que método de pesquisa no grafo utilizar: algoritmo para grafos pesados, utilizando o algoritmo *dijkstraShortestPath* (considera a distância entre zonas); ou algoritmo para grafos não pesados, *unweightedShortestPath* (considera o número de zonas a atravessar até à zona final).

Na aplicação é permitida a adição, edição e remoção de Clientes, Zonas, Lojas, e Encomendas (apenas adição e remoção).

É possível verificar, com interface gráfica, a estrutura actual do Mapa de Zonas, e observar animadamente a escolha de uma loja para servir uma encomenda ao realizar uma encomenda.

2.4 Exemplos de Menus:

Titulo: Menu inicial

Descrição: Permite que qualquer utilizador possa fazer a gestão de clientes, gestão de encomendas, gestão de zonas, gestão de grafo, visualizar graficamente o grafo e ainda gravar a informação existente em memória para um ficheiro de texto e sair da aplicação.

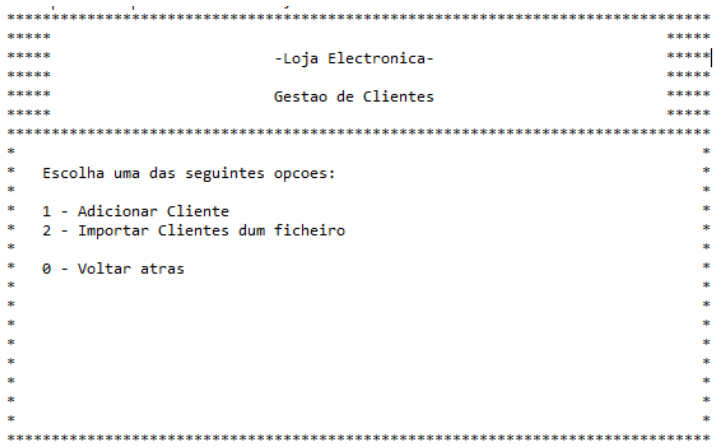
```
*************************************************************************
*****                                                               *****
*****                       -Loja Electronica-                      *****
*****                                                               *****
*****                         Menu Principal                        *****
*****                                                               *****
*************************************************************************
*                                                                       *
*   Escolha uma das seguintes opcoes:                                   *
*                                                                       *
*   1 - Gestao de Clientes                                              *
*   2 - Gestao de Encomendas                                            *
*   3 - Gestao de Zonas                                                 *
*   4 - Gestao de Grafo                                                 *
*   5 - GraphViewer                                                     *
*                                                                       *
*   0 - Gravar e sair                                                   *
*                                                                       *
*                                                                       *
*                                                                       *
*                                                                       *
*                                                                       *
*************************************************************************
```

Figura 1: **Menu Inicial**

Titulo: Gestão de Clientes

Descrição: Permite adicionar clientes ao sistema ou importar clientes do ficheiro de texto. Caso já existam clientes registados na aplicação, o utilizador terá ao seu dispor mais funcionalidades sendo elas: Ver Detalhes de um cliente, Editar um cliente, Remover um cliente, Exportar clientes para um ficheiro

Figura 2: Gestão de clientes.



Titulo: Menu de Gestão de Encomendas

Descrição: Permite que qualquer utilizador possa adicionar encomendas ou importar encomendas de um ficheiro de texto. Caso já existam encomendas no sistema, o utilizador poderá ainda listar uma encomenda, editar uma encomenda, remover uma encomenda ou exportar encomendas para um ficheiro de texto.

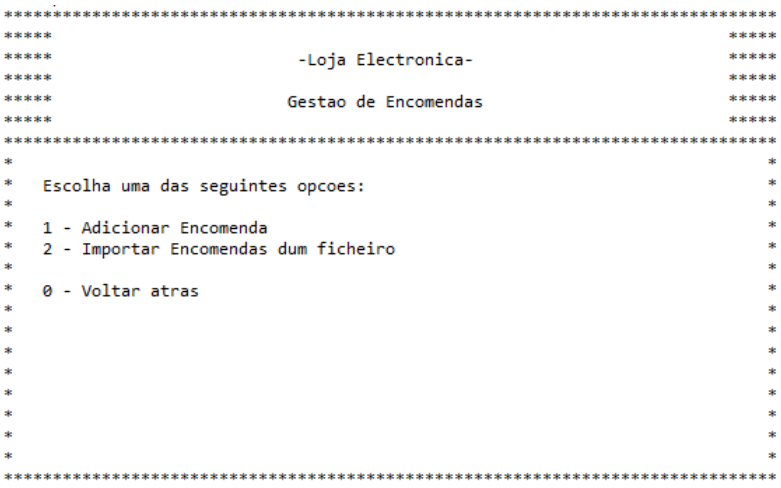
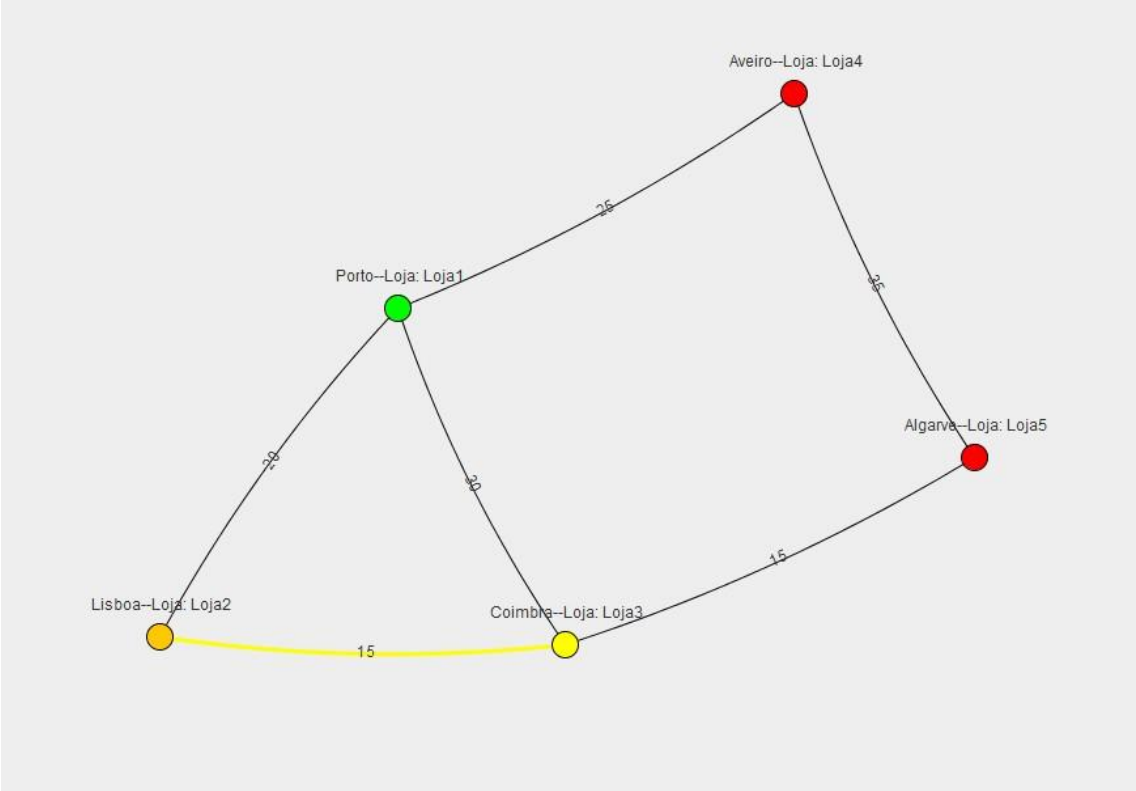


Figura 3: Gestão de Encomendas.

Titulo: Gestão de Zonas

Descrição: Permite que qualquer utilizador possa fazer a gestão de zonas: adicionar, eliminar, editar e remover uma ligação entre zonas sendo esta funcionalidade específica para grafos. Caso não haja zonas no sistema o utilizador poderá apenas adicionar zonas ou importar zonas de um ficheiro de texto

Exemplo do final de uma Apresentação animada da escolha da loja a tratar encomenda:



O nosso projecto contém ainda outras classes que não foram implementadas por nós mas que são essenciais para o funcionamento do nosso projecto sendo elas:

- **Connection**
- **Graph Viewer**
- **Edgetype**

Classes importantes que permitem a visualização gráfica dos grafos;

- **Graph**

Classe que foi desenvolvida ao longo das aulas práticas da disciplina e que é fundamental para o funcionamento da aplicação pois contém as classes Vértice e Aresta e Grafo e todos os métodos que permitem a manipulação de grafos desde remoção/inserção de nós e arestas, saber as arestas adjacentes, saber o caminho entre nós, saber o conjunto de todos vértices de um grafo, saber o caminho mais curto para grafos não pesados, saber o caminho mais curto para grafos pesados, entre outros.

3. Algoritmos Implementados e Estratégia Seguida

Para o desenvolvimento da aplicação, o grupo tinha duas possibilidades:

1. Menor distância da loja que possui esse produto ao cliente

2. Menor número de zonas a atravessar, entre a loja que possui esse produto e a residência do cliente.

O grupo optou por implementar as duas opções, cabendo ao utilizador escolher a pretendida durante a execução da aplicação.

Para o primeiro caso, foi necessário utilizar grafos pesados pois cada aresta teria de ter um peso a si associado para saber as distâncias e com base nestes pesos o algoritmo denominado por Algoritmo de *Dijkstra*, partindo do nó onde se encontra o cliente, obtém a distância a todos os outros nós do grafo somando o peso das arestas. Mais concretamente, o programa verifica se a loja da zona do cliente contém ou não o produto pretendido, e em caso negativo, é determinada a distância da zona do cliente a todas as outras zonas do grafo. Por fim, verifica-se se as lojas mais próximas do cliente até se encontrar uma loja que contenha o produto pretendido.

É um algoritmo ganancioso pois em cada passo permite maximizar o ganho imediato (neste caso minimizar a distância).

O tempo de execução do algoritmo de *Dijkstra*, como foi visto nas aulas teóricas, é O(|V| + |E| + |V|*log|V| + |E|*log |V|) ou simplesmente O(|E|*log |V|), se |E|>|V|

• O(|V|*log |V|) - extração e inserção na fila de prioridades

- O nº de extrações e inserções é |V|

- Cada operação destas pode ser feita em tempo logarítmico no tamanho

da fila, que no máximo é |V|

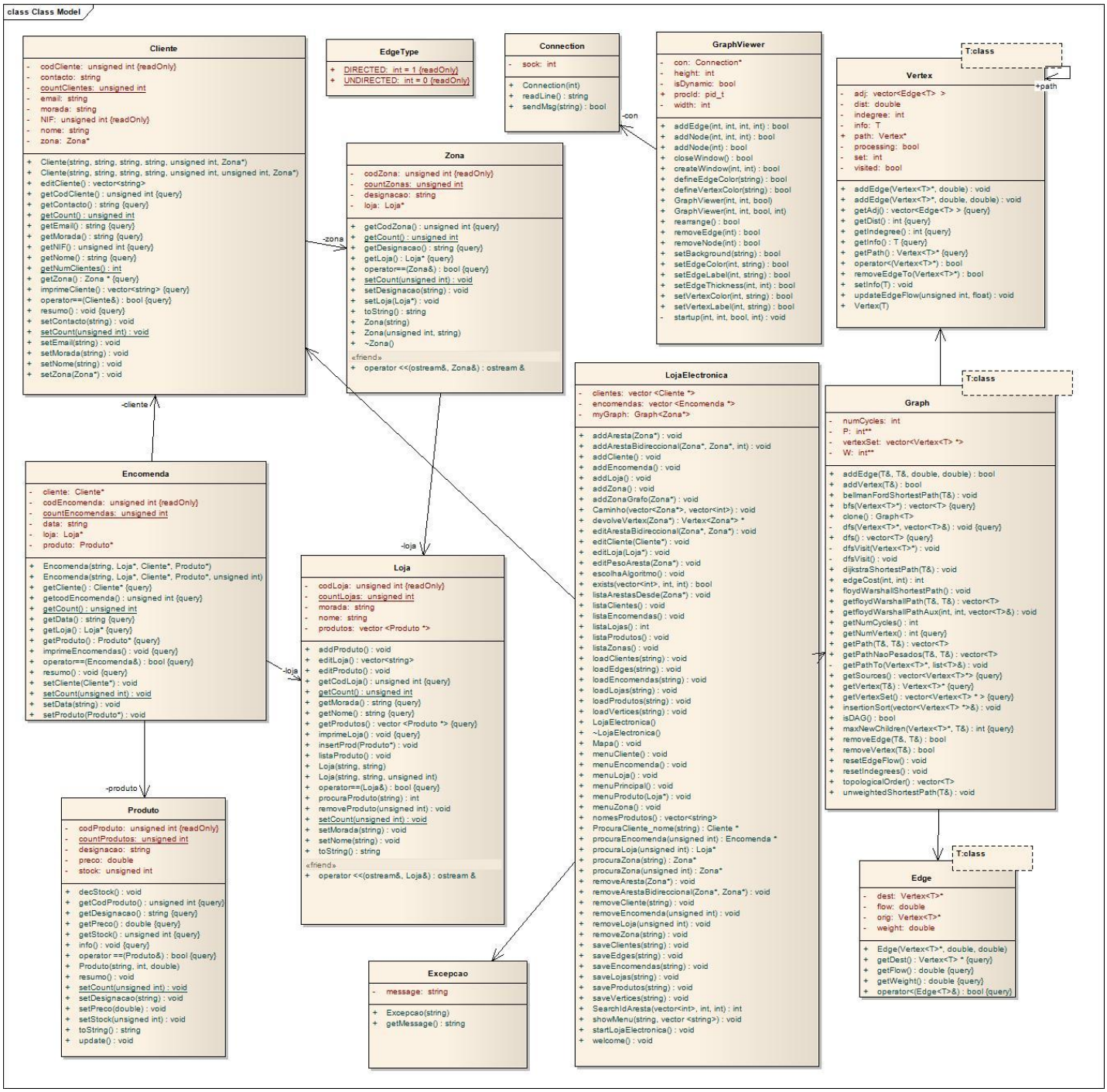
• O(|E|*log |V|) – decreaseKey

- Feito no máximo |E| vezes (uma vez por cada aresta)

- Cada operação destas pode ser feita em tempo logarítmico no tamanho da fila, que no máximo é |V|

Para o segundo caso, foi necessário utilizar grafos não pesados, ou seja, não se considera o peso das arestas. Partindo do nó onde se encontra o cliente, o algoritmo denominado por *shortestPathUnweighted*, determina o número de zonas a todos os restantes nós do grafo. Mais concretamente, o programa verifica se a loja da zona do cliente contém ou não o produto pretendido, e em caso negativo, é determinado o número de zonas a atravessar até chegar a todos os outros nós do grafo. Por fim, verifica-se se o produto existe nas lojas mais próximas, ou seja, as lojas em que é necessário percorrer menos zonas. Este algoritmo tem uma complexidade temporal O(|E| + |V|) e espacial O(|V|).

4. UML - Diagrama de Classes



Devido à dificuldade em apresentar aqui o referido diagrama, o mesmo encontra-se também em anexo em formato de imagem e de ficheiro de Enterprise Architect.

5. Conclusão

A implementação do código decorreu sem grandes problemas, alguns entraves surgiram no decorrer da escrita do código, mas com o tempo e a dedicação do grupo estes foram sempre ultrapassados.

O grupo de trabalho funcionou bem, as tarefas foram distribuídas ficando cada elemento responsável por partes igualmente importantes no projeto final. Quando foi necessário reunimos para discutir ideias e procurar soluções, sendo que tudo decorreu da melhor forma, sem atritos e com todos os elementos do grupo a trabalharem bem e harmoniosamente.

A principal dificuldade ao longo do desenvolvimento do projeto foi o facto de os elementos do grupo não estarem ainda muito habituados à manipulação de grafos contudo os elementos do grupo entreajudaram-se e com o esforço necessário conseguiu adaptar-se bem e conseguiu realizar um bom trabalho.