Proposta de Trabalho

Voos e Companhia

Aplicação de Estruturas de dados não lineares e Programação Orientada aos Objetos (POO)

Linguagens Programação II

Rui Silva Moreira
rmoreira@ufp.edu.pt
Christophe Soares
csoares@ufp.edu.pt
Beatriz Gomes
argomes@ufp.edu.pt

Algoritmia Estruturas Dados II

José Torres
rmoreira@ufp.edu.pt
André Pinto
apinto@ufp.edu.pt

Março 2017

Universidade Fernando Pessoa
Faculdade de Ciência e Tecnologia

1. Definição do problema

Neste projeto pretende-se que os alunos modelizem, implementem, testem e documentem uma aplicação Java para manipular e gerir informação relativa a aeroportos, ligações aéreas, companhias aéreas, histórico de voos realizados e aviões que realizam as viagens nas condições mais económicas possível. Mais concretamente, pretende-se que combinem a utilização de tabelas de símbolos e grafos para armazenar e gerir a informação necessária.

As estruturas do tipo *Symbol Table* (e.g. *Hashmaps*, *Binary Search Trees*, *Redblack Trees,* etc.) deverão permitir armazenar e gerir a informação relativa às entidades que se pretendem manipular (e.g. aeroportos, ligações aéreas, companhias aéreas, aviões, voos, etc.). Por exemplo, para cada avião, deve registar-se a sua velocidade de cruzeiro, a altitude de cruzeiro mais eficiente e o consumo. Quando um avião voa a uma altitude superior ou inferior à sua altitude de cruzeiro então o seu consumo sobe ou desce em função da diferença de altitude (tipicamente, *n* litros por cada 1000 metros de altitude acima ou *m* litros por cada 1000 metros abaixo – sendo *n* e *m* números inteiros). Deverão ser sempre consideradas altitudes múltiplas de 1000. Por exemplo, para *n=200* considera-se que o consumo sobe 200 litros por cada 1000 metros acima da altitude de cruzeiro, ou seja, neste caso cada 1000 metros de altitude levam o avião a gastar 200 litros de combustível extra. Para simplificar a estimativa de consumo numa viagem, assume-se que qualquer subida ou descida de altitude acontece instantaneamente, i.e., em tempo zero, tanto no início como no fim da cada viagem. Pode ainda considerar-se que cada voo é realizado a uma velocidade e altitude constantes.

As estruturas do tipo grafo permitirão armazenar a informação relativa às ligações aéreas existentes entre os vários aeroportos. Cada aeroporto corresponderá a um nó (vértice) do grafo e poderá ter um conjunto de ligações (arestas) a outros aeroportos. As ligações serão dirigidas e poderão caracterizar-se por vários pesos (e.g., distância, custo, tempo, etc.). A aplicação deverá permitir combinar e gerir a informação das ligações aéreas, dos aeroportos e das estatísticas de dos voos e número de passageiros transportados.

Para cada viagem é dado o aeroporto de início e de fim, bem como a velocidade do vento. Considera-se que uma velocidade positiva (*tailwind*) aumenta a velocidade do avião ao passo que uma velocidade negativa diminui a velocidade do avião numa proporção direta. Deverá considerar-se que a altitude do avião será sempre um múltiplo inteiro de 1000 metros, entre os 20000 e 40000 m. Deverá ser possível calcular a melhor altitude para uma dada viagem de forma a minimizar o consumo de combustível do avião, em função das linhas aéreas existentes e das respectivas altitudes de cruzeiro.

Para facilitar a implementação, irá utilizar-se uma arquitetura *standalone*, ou seja, uma implementação que deverá funcionar num único PC. Os alunos deverão utilizar pacotes de software pré-existentes que oferecem estruturas de dados genéricas (cf. grafos e árvores), que possam ser reutilizadas na implementação do problema proposto. Desta forma não terão que implementar as estruturas de dados básicas, podendo concentrar-se na lógica e requisitos funcionais da aplicação proposta.

1.1. Requisitos funcionais

Pretende-se que os alunos sigam uma abordagem orientada aos objetos na modelização e implementação do problema proposto. Em concreto deverão desenhar os diagramas de classes necessários que permitam modelizar o problema, reutilizando pacotes/classes pré-existentes (cf. grafo, árvores, *hashmap*, etc.) através de herança, composição ou agregação.

Pretende-se que desenvolvam uma API de classes (biblioteca/conjunto de métodos em classes) que satisfaçam os requisitos listados a seguir. Pretende-se, também, que se implementem casos de teste dessa API (funções de teste) para as várias funcionalidades/requisitos implementados. Cada caso de teste deverá ser devidamente documentado numa função *static* que deverá ser caracterizada pelo conjunto de funções a testar, pelos valores de input a utilizar no teste (preferencialmente de ficheiro ou, em alternativa, editando diretamente o código, mas nunca provenientes de valores interactivamente inseridos pelo utilizador), e por valores de output/resultado do teste enviados para a consola e escritos no ficheiro.

Em concreto a aplicação deverá cumprir os seguintes requisitos:

R1. O modelo de dados deverá permitir representar a rede de aeroportos e respectivas ligações aéreas, bem como toda a meta-informação associada a estas entidades; deverá ser ainda integrada a informação sobre os aviões; a ligação entre aeroportos deverá refletir, através de vários pesos (e.g., custos monetários, espaciais, altitude, temporais) da deslocação entre os aeroportos;

R2. Devem suportar modelos de dados para os valores (classe genérica *Java Value* das *Symbol Tables* (ST)) dos elementos das várias tabelas de símbolos consideradas no projeto. Deverão utilizar funções e tabelas de *hash* em, pelo menos, uma ST cuja chave não tenha que ser ordenável. Deverão usar BSTs balanceadas (*redblack*) nas STs cuja chave é ordenável (e.g., tempos, ordem, etc.).

R3. Devem criar funções para inserir, remover, editar e listar toda a informação, para cada uma das várias STs consideradas na base de dados.

R4. Deverão validar a consistência de toda a informação como, por exemplo, que todos os aeroportos que são mencionados noutras STs, existam na ST aeroportos.

R5. Deverão considerar a remoção de informação e nestes casos deve-se arquivar a informação. Por exemplo, ao remover um aeroporto da base de dados deve garantir-se a sua remoção total do sistema (dos voos realizados, etc.) e respectivo arquivamento (em ficheiro ou estruturas auxiliares).

R6. Deve-se popular as diversas STs da aplicação com o conteúdo de ficheiros de texto de entrada (carregar/gravar a informação em ficheiro).

R7. Deve garantir-se o *output* (*dump*) de toda a informação para ficheiros de texto, isto é, de aeroportos, ligações aéreas e de todas as outras STs.

R8. Devem implementar-se diversas pesquisas à base de informação como, por exemplo: quais os voos que chegam ou partem de um dado aeroporto; qual o aeroporto com mais voos; qual o voo que transportou mais passageiros, entre outras pesquisas.

R9. Cada nó ou vértice do grafo representa um aeroporto deverá ter um conjunto de atributos/propriedades principais (e.g., nome, cidade, país, etc.) e outros atributos que possam vir a ser necessários;

R10. O modelo de dados deve prever a utilização de algoritmos genéricos de gestão e verificação de grafos, nomeadamente:
   a) Algoritmos de cálculo: do caminho mais curto entre dois aeroportos selecionados (com base na distância entre nós); do caminho mais barato (com base no custo monetário das viagens); do caminho mais rápido (com base no custo temporal das viagens); do caminho mais rápido com ou sem ligações intermédias (com base na informação das ligações aéreas);
   b) Deverá poder-se verificar se o grafo de ligações entre aeroportos é conexo;
   c) Selecionar um subgrafo e aplicar-lhe os mesmos algoritmos ou funcionalidades descritas anteriormente;
   d) Deverá associar-se a cada nó uma árvore para armazenamento dos dados dos aeroportos e posteriormente permitir a manipulação conjunta de forma a facilitar as pesquisas e as ações de gestão de informação; Poderá usar-se, por exemplo, uma árvore `Red-Black` (RB) para guardar a lista de aeroportos assim como toda a informação associada de forma a facilitar a pesquisa bem como a manipulação da informação acerca do perfil desses aeroportos;

R11. Deverá ser possível efetuar e combinar vários tipos de pesquisas sobre os aeroportos, ligações aéreas e aviões como, por exemplo:
   a) Listar as ligações aéreas que passam num determinado aeroporto;
   b) Listar informação detalhada do tráfego do aeroporto num determinando período de tempo;
   c) Procurar e listar aeroportos num determinado país ou continente;
   d) Efetuar pesquisas com vários critérios, combinados por operadores booleanos (cf. *and*, *or*); Por exemplo, pesquisar aeroportos na europa *and* com mais de 5 ligações aéreas disponíveis;
   e) Para uma dada cidade de origem e destino, determinar quais os aeroportos e respectivas ligações aéreas mais adequadas em função dos critérios de seleção (e.g. custo, distância, tempo, etc.); poderão aplicar-se diversos algoritmos de cálculo de caminhos mais curtos em função dos pesos selecionados;
   f) Procurar num determinado dia e hora todas os voos existentes para uma determinada ligação;

R12. Deverá ser criada uma interface gráfica para:
   a) Visualizar o grafo de aeroportos e respectivas ligações aéreas bem como da árvore de aeroportos, distinguindo de alguma forma os tipos de nós;
   b) Gerir o grafo através da adição e remoção de nós/vértices e arcos/ramos bem como edição dos seus atributos;
   c) A manipulação e gestão de toda a informação e das respectivas pesquisas deverá ser também suportada pela interface gráfica;

R13. Todos os dados referentes à aplicação e respectivas pesquisas deverão poder ser gravados em ficheiros de texto para consulta posterior dos utilizadores;

R14. Deverá ser ainda possível importar e exportar os dados dos modelos de dados (cf. Grafo e STs relacionadas) para ficheiros binários e de texto.

2. Objectivos

Pretende-se que os alunos modelizem e implementem a aplicação descrita, cumprindo todos os objectivos propostos. Deverão nomeadamente:

- Modelizar o problema através de diagramas de classes (UML), reutilizando estruturas de dados base (e.g. grafo, árvore, hasmap, etc.) e respectivos métodos/operações (cf. propriedades, travessias, pesquisas, etc.) que permitam representar e manipular os dados necessários;
- Implementar os algoritmos principais para a pesquisa e processamento da informação de acordo com as funcionalidades solicitadas;
- Implementar o modelo de dados OO utilizando a linguagem Java; em particular os requisitos funcionais enumerados e outros que se revelem úteis ou necessários;
- Implementar um conjunto de casos de teste que recorram a dados de *input* devidamente selecionados;
- Implementar uma interface gráfica que suporte a visualização e gestão de toda a informação (e.g., visualização da rede, gestão e edição dos nós, execução de pesquisas, etc.);
- Implementar a leitura e escrita de informação baseada em ficheiros de texto e binários.

3. Ficheiros e documentos a entregar

O projeto proposto deve ter uma implementação orientada aos objetos. Recomenda-se que todo o código (algoritmos e estruturas de dados) seja complementado com os comentários apropriados, que facilitem a compreensão do mesmo e a respectiva geração automática de documentação. Deve ser incluída uma explicação dos algoritmos implementados e uma menção ao desempenho dos mesmos assim como dos testes efetuados/implementados. Devem ainda ser realizados testes unitários que demonstrem o bom funcionamento das classes desenvolvidas.

Irão existir duas fases/momentos de entrega e avaliação presencial:

- Na fase 1 de entrega, serão considerados obrigatórios os seguintes requisitos:
  - R1, R2, R3, R4, R5, R6, R7, R8.
- Na fase 2 serão considerados todos os requisitos mencionados.

NB: para AED2 alguns dos requisitos são opcionais:

- R1 (não é obrigatório o modelo de dados em UML)
- R12 (não é obrigatório a implementação de uma GUI em Java)
- R14 (não é obrigatório a utilização de ficheiros binários, apenas de texto).

As entregas devem incluir os seguintes componentes complementares:

i) Modelos de classes definidos para o projeto (ficheiro **zargo**);
ii) Código fonte do projeto (diretório **src** das classes implementadas e testadas);
iii) Ficheiros de teste e de dados *input/output* utilizados;
iv) Documentação do código (páginas de HTML geradas com *JavaDoc*).

Estes componentes do projeto devem ser entregues num único ficheiro zip através da plataforma de *elearning* até ao dia registado nos *assignments* da fase 1 e 2.

As duas fases do projeto deverão ser apresentadas e defendidas de "viva voz", a meio do semestre (fase 1) e no final do semestre (fase 2), em datas a anunciar pelos docentes. Projetos entregues fora do prazo ou não apresentados presencialmente, não serão considerados para classificação.