Proposta de Trabalho:

Gestão de Informação e Navegação numa rede de Geocaching

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

Linguagens Programação II	Algoritmos e Estruturas de Dados II
Rui Silva Moreira rmoreira@ufp.edu.pt	José Torres jtorres@ufp.edu.pt
Bruno Gomes bagomes@ufp.edu.pt	Bruno Cunha bmcunha@ufp.edu.pt
Diogo Machado diogomachado@ufp.edu.pt	Miguel Chaves mchaves@ufp.edu.pt
Tiago Costa tscosta@ufp.edu.pt	

Março 2021

Universidade Fernando Pessoa

Faculdade de Ciência e Tecnologia

1. Definição do problema

Neste projeto pretende-se que os alunos **modelem**, **implementem**, **testem** e **documentem** uma aplicação Java para manipular e gerir informação relativa a uma rede de geocaching (e.g. caches e utilizadores), bem como utilizar a informação para navegação entre caches (e.g. rede de pontos representando a distribuição geográfica dos diferentes caches). Mais concretamente, pretende-se que combinem a utilização de estruturas de dados orientadas a objetos (e.g. tabelas de símbolos e grafos) para armazenar e gerir a informação necessária.

Geocaching pode ser visto como uma "caça ao tesouro". Os participantes recebem coordenadas GPS para localizar uma pequena caixa (geocache ou cache) que foi escondida. Para além da componente de aventura, o Geocaching é útil para dar a conhecer pontos de interesse, enquanto possibilita a troca de objetos entre utilizadores. A troca de objetos é feita quando um "aventureiro" deixa um objeto na cache e mais tarde outro "aventureiro" troca este objeto por outro seu. Um caso particular de objeto são os "Travel bugs", esta marcação de objetos permite a quem, inicialmente, colocou o "Travel bug" saber onde este se encontra, proporcionando a criação de missões, por exemplo: "este objeto deve ser levado para o ponto X". Contrariamente aos restantes objetos, sempre que um "Travel bug" é recolhido o "aventureiro" compromete-se a transportar o mesmo até uma outra cache. Para mais informações, pode ser visitado o website: https://www.geocaching.com/play.

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. caches, utilizadores, items, logs). Por exemplo, para cada cache deverá ser registado o seu tipo (e.g basic, premium) , localização, dificuldade, histórico de logs e items. Cada utilizador (basic, premium ou admin) deverá ser capaz de consultar o seu histórico de caches escondidas e caches visitadas. Como proprietário das suas caches escondidas, um utilizador deverá ser capaz de consultar o histórico de localizações do(s) seu(s) travel bug, com feedback, em tempo real, dos caminhos mais curtos que o seu travel bug terá de percorrer, para atingir uma qualquer cache ou região.

Deverão utilizar diferentes estruturas do tipo grafo para armazenar a informação relativa à rede (mapa de pontos) de geocaching. Neste mapa deve ser possível representar os diferentes nós (caches) e a suas respetivas ligações, caracterizadas por diferentes tipos de pesos (e.g. distância, tempo, dificuldade, etc). Para além do grafo global, deverão também

ser considerados subgrafos constituídos pelas redes de diferentes tipos de cache (e.g. rede de caches do tipo basic, rede de caches do tipo premium);

Embora a solução pudesse utilizar uma arquitetura cliente-servidor, 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, árvores, tabelas de símbolos etc.), 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 um *package* de classes com várias funcionalidades úteis à implementação do problema proposto e que satisfaçam os requisitos listados a seguir. Pretende-se, também, que se implementem casos de teste desse *package* para cada um dos requisitos. As funções de teste devem ser responsáveis por chamar todos os métodos necessários que demonstrem a correta funcionalidade de cada um dos requisitos. 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/ou escritos em ficheiro.

Em concreto a aplicação deverá cumprir os seguintes requisitos:

Fase 1

R1. Modelizar num diagrama UML as classes e associações de dados necessárias para representar todas as redes e respetivas ligações, bem como toda a meta-informação associada a caches, utilizadores, items, logs, entre outras. No modelo de dados deverão reutilizar estruturas de dados base (e.g. grafo, árvore, *hashmap*, etc.) e respetivos métodos/operações (cf. propriedades, travessias, pesquisas, etc.).

R2. Deverão utilizar funções e tabelas de *hash* em, pelo menos, uma *Symbol Table* (ST) cuja chave não tenha que ser ordenável. Deverão usar BSTs balanceadas (*redblack*)

nas STs cuja chave é ordenável (e.g., tempos, ordem, etc.). A implementação deverá suportar valores (classe genérica *Java Value*) para os elementos das várias tabelas de símbolos a considerar no projeto.

R3. Devem existir 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 caches que são mencionados noutras STs, existam na ST caches.

R5. Deverão considerar a remoção de informação das várias estruturas de dados, e nesses casos arquivar a informação necessária. Por exemplo, ao remover um utilizador da base de dados deve garantir-se a sua remoção total do sistema e respetivo arquivamento (em ficheiros ou em estruturas auxiliares).

R6. Devem popular as diversas STs da aplicação com o conteúdo de ficheiros de texto de entrada (carregar a informação a partir de ficheiros txt).

R7. Deve garantir-se o *output* (*dump*) de toda a informação para ficheiros de texto, isto é, de caches, utilizadores, items, logs, etc. (gravar a informação para ficheiros txt).

R8. Devem implementar-se diversas pesquisas sobre a base de informação como, por exemplo, determinar:

a) Todas as caches visitadas por um utilizador. Global e por região;
b) Todas as caches não visitadas por um utilizador. Global e por região;
c) Todos os utilizadores que já visitaram uma dada cache;
d) Todas as caches premium que têm pelo menos um objecto;
e) Top-5 de utilizadores que visitaram maior nº de caches num dado período temporal;
f) travel bugs com maior número de localizações percorridas no seu histórico

R9. Deverão considerar definir um perfil para cada travel bug registado (e.g. histórico de localizações, utilizadores que o transportaram, período temporal associado a cada momento, etc.). Recorrendo a todos os perfis, o sistema deverá construir uma visão global dos travel bugs registados. Por exemplo, o método now() deverá apresentar o estado de todos os travel bugs naquele instante: localização actual, último utilizador que o transportou, período em que foi transportado, caminhos percorridos até ao momento, etc.

R10. Deverão criar unidades de teste, recorrendo a funções *static*, para testar todas as estruturas e funcionalidades pretendidas.

R11. Deverão gerar a documentação de todos os métodos através de Javadoc.

Fase 2

R12. Representar o grafo global da rede de geocaching. Cada nó/vértice do grafo corresponde a uma cache introduzida no sistema. Cada nó/vértice deverá ter associado um conjunto de atributos (e.g. coordenadas relativas a um dado referencial, tipo de cache, etc); Os nós/vértices do grafo são ligados por arestas/ramos, correspondentes aos caminhos pedonais que ligam as diferentes caches representados por esses nós/vértices; As arestas/ramos podem ter vários pesos, representando a ligação física entre os nós/vértices (e.g. distância em metros da ligação, tempo médio em segundos do percurso, diferença de elevação etc.);

R13. Deve ser possível criar um conjunto (*set*) de vértices manual ou baseado em atributos (e.g. zona, dificuldade, número de visitantes, etc); Os conjuntos criados terão como objetivo definir vértices/nós a evitar/preferir para o cálculo de caminhos.

R14. O modelo de dados deve prever a utilização de algoritmos genéricos de gestão e verificação de grafos, nomeadamente:

a) Algoritmos de cálculo do caminho de menor custo entre duas caches; O custo a considerar deverá ter como base um ou mais dos pesos considerados em R12 (e.g. caminho mais curto, caminho mais rápido, caminho mais simples, etc); Deverá também ser considerada a opção para efetuar as mesmas pesquisas evitando um determinado conjunto de pontos (como definido em R13);
b) Selecionar um subgrafo (e.g. caches premium) e aplicar-lhe os mesmos algoritmos ou funcionalidades descritas anteriormente;

R15. Deverá ser criada uma interface gráfica para:

a) Visualizar o grafo da rede de geocaching. Poderão tirar partido das coordenadas definidas para cada vértice/cache para a representação gráfica de toda a rede, distinguindo visualmente os diferentes tipos de cache (e.g. basic e premium);
b) Gerir o grafo através da adição e remoção de nós/vértices e arestas/ramos, bem como permitir a edição dos seus atributos;
c) A manipulação e gestão de toda a informação (e.g. utilizadores, items, logs, etc) e das respetivas pesquisas deverá ser também suportada pela interface gráfica;

R16. Todos os dados referentes à aplicação e respetivas pesquisas deverão poder ser gravados em ficheiros de texto para consulta posterior dos utilizadores;

R17. Deverá ser ainda possível importar e exportar os dados dos modelos de dados (cf. Grafo e STs relacionadas) para ficheiros binários e de texto.

R18. Problema do caixeiro-viajante. Deverá ser implementada uma funcionalidade que permita à aplicação calcular o maior número de caches que podem ser visitadas partindo de uma determinada cache e considerando um limite máximo de tempo.

Cotação dos Requisitos

Fase 1 (40%):

Req	1	2	3	4	5	6	7	8	9	10	11
AED2	0	2	1	2	2.5	1.5	1.5	4	2.5	1	2
LP2	2	1	2	2	2.5	1.5	1.5	2	2.5	1	2

Fase 2 (60%):

Req	12	13	14a	14b	15a	15b	15c	16	17	18
AED2	2.5	2.5	3	2.5	0	2	0	2	2	3.5
LP2	2	1.5	2	1.5	2.5	2	1.5	2	2	3

(NB: cotação 0-20)

2. 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 respetiva 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 os requisitos R1 a R11.
- Na fase 2 serão considerados todos os requisitos mencionados na fase 1, bem como os posteriormente enunciados para a fase 2.

NB: para AED2 alguns dos requisitos são opcionais:

- R1 (não é obrigatório o modelo de dados em UML)
- R15 (não é obrigatória a criação de interface gráfica)

NB: AED2 e LP2

- Na primeira submissão, a implementação de R6 e R7 poderá ser baseada nas classes In e Out das packages fornecidas.

As entregas devem incluir os seguintes componentes complementares:

i) **Modelos de classes** definidos para o projeto (ficheiro **zargo**);
ii) **Código fonte** (diretório **src**) das classes implementadas;
iii) **Ficheiros de teste** e respectivos 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”, no final do semestre, em datas a anunciar pelos docentes. Projetos entregues fora do prazo ou não apresentados presencialmente, não serão considerados para classificação.