Trabalho Prático 3

Continuação da Implementação de um sistema de gestão de séries televisivas

1. Informação geral

O Trabalho Prático 3 aplica conceitos de Programação Orientada a Objetos e consiste na implementação de classes baseadas em filas de prioridade e árvores

Este trabalho deverá ser feito de forma autónoma, por cada grupo, até à data-limite estabelecida. A consulta de informação nas diversas fontes disponíveis é aceitável. No entanto, o código submetido deverá ser apenas da autoria dos elementos do grupo e quaisquer cópias detetadas serão devidamente penalizadas. A incapacidade de explicar o código submetido por parte de algum elemento do grupo implicará também uma penalização.

O prazo limite para submissão (através do Moodle) é o dia **14 de abril às 23:59h**.

2. Conceito

Uma plataforma de visualização de séries televisivas pretende obter um sistema que permita gerir as séries visualizadas por diferentes utilizadores.

3. Implementação do trabalho

O arquivo comprimido **`ESDA_2024_MTP3.zip`** contém os ficheiros necessários para a realização deste trabalho, nomeadamente:

- `TVseries.hpp`: definição das classes para representação do sistema (`User`, `TVSeries`, `TVSeriesManagement`, `UserManagement`, `TVSeriesManagementList`, `UserManagementList`, `NodeUser` e `UserManagementTree`.
- `TVseries.cpp`: implementação dos métodos relativos às classes definidas em `TVseries.hpp`.
- `series_testTP3.c`: inclui o programa principal que invoca e realiza testes básicos às funções implementadas.
- `series.txt`: ficheiro de texto com a informação das séries visualizadas pelos utilizadores.
- `user_updateWateched.txt`: ficheiro de texto com a informação de que series cada utilizador viu.

Notas importantes:

1. **Apenas deverá ser alterado o ficheiro `TVseries.cpp`, sendo somente necessário incluir a implementação de cada função na submissão do código em CodeRunner, no Moodle.**
2. **Cada atributo e método das classes definidas apresenta detalhes adicionais junto a cada um deles em `TVseries.hpp`.**
3. **A amarelo estão as alterações verificadas nas classes em relação ao segundo trabalho.**

O ficheiro contém as classes `User`, `TVSeries`, `TVSeriesManagement`, `UserManagement`, `TVSeriesManagementList`, `UserManagementList`, `NodeUser` e `UserManagementTree`. A primeira permite caracterizar cada série televisiva, a segunda caracteriza cada utilizador, a terceira permite a gestão das séries da plataforma e a quarta define os utilizadores inscritos na plataforma, a quinta e sexta fazem o mesmo que a segunda e terceira, mas em listas, a sétima representa um nó da árvore BST com todos os utilizadores registados e, finalmente, a nona representa uma árvore BST com todos os utilizadores registados.

Classe `User`

Os objetos da classe `User` têm os seguintes atributos:

1) nome de identificação utilizador (`username`)
2) nome do utilizador (`name`)
3) *string* com o país do utilizador (`country`)
4) vetor de inteiros com os géneros de séries favoritos do utilizador (`favoriteGenres`)
5) vetor com as séries visualizadas pelo utilizador (`watchedSeries`)
6) vetor de inteiros com as classificações atribuídas pelo utilizador (`ratings`); por exemplo, no índice 0, encontra-se a classificação atribuída à série na primeira posição do vetor `watchedSeries`
7) vetor de inteiros com o número de episódios visualizados de cada série (`episodesWatched`); por exemplo, no índice 0, encontra-se o número de episódios visualizados da série na primeira posição do vetor `watchedSeries`
8) fila de desejo com as series que pretendemos ver, por ordem de chegada(wishSeries)

Classe `TVSeries`

Os objetos da classe `TVSeries` têm os seguintes atributos:

1) título da série (`title`)
2) número atual de temporadas da série (`numberOfSeasons`)
3) vetor de inteiros com o número atual de episódios de cada temporada (`episodesPerSeason`); por exemplo, no índice 0, encontra-se o número de episódios da primeira temporada
4) *string* com a identificação do género da série (`genre`)
5) *float* com a classificação da série (`rating`)
6) booleano que indica se a série se encontra ou não terminada (`finished`)

Classe `TVSeriesManagement`

Os objetos da classe `TVSeriesManagement` possuem um vetor de apontadores para objetos da classe `TVSeries`, representando todas as séries disponíveis na plataforma.

Classe `TVSeriesManagementList`

Os objetos da classe `TVSeriesManagementList` possuem uma lista ligada de apontadores para objetos da classe `TVSeries`, representando todas as séries disponíveis na plataforma.

Classe `UserManagement`

Os objetos da classe `UserManagement` possuem um vetor de apontadores para objetos da classe `User`, representando todos os utilizadores registados na plataforma.

Classe `UserManagementList`

Os objetos da classe `UserManagementList` possuem uma lista ligada de apontadores para objetos da classe `User`, representando todos os utilizadores registados na plataforma.

Classe `NodeUser`

Os objetos da classe `NodeUser` são nós da(s) árvore(s) da classe `UserManagementTree`. Possuem um apontador para um objeto da classe `User` representando um utilizador registado na plataforma, e mais dois apontadores para os nós à esquerda e à direita na árvore, respectivamente.

Classe `UserManagementTree`

Os objetos da classe `UserManagementTree` possuem uma árvore BST de apontadores para objetos da classe `User`, representando todos os utilizadores registados na plataforma..

As funções a implementar neste trabalho correspondem a métodos definidos em cada classe.

Classe `UserManagement`

1. `priority_queue<TVSeries> queueTVSeriesCategory (priority_queue <TVSeries>& pq, string cat);`

   A partir de uma tabela classificativa (fila de prioridades) de series por rating pq, cria uma nova tabela (fila de prioridades) só para uma determinada categoria de séries cat.

2. `priority_queue<TVSeries> queueTVSeries(list<TVSeries> listTV, int min);`

   Cria uma tabela classificativa (heap / fila de prioridade) por rating das series em que pelo menos dois episódios foram vistos por um número mínimo de utilizadores min.

> **NOTA**
> Como foi criado uma sobrecarga ao operador < com base no atributo rating da class TVSeries
>
> bool TVSeries::operator <(const TVSeries& tv) const
>
> { return this->rating < tv.rating; }
>
> ao se inserir uma serie na fila de prioridade, essa fila de prioridade fica em função do rating.

Classe `UserManagement`

3. `vector<User*> usersInitialLetter(NodeUser* root, char ch);`
   Cria um vetor com os utilizadores(username) inicializados por uma determinada letra ch (ter em atenção as maiusculas e minusculas. por exemplo: os utlizadores 'Pedro' e 'pcastro' sõa dois utilizadores inicalizados pela letra p.

4. `list<User*> usersNotFan(NodeUser* root);`
   Cria uma lista com os utilizadores que não costumam terminar as séries. (utilizadores que têm mais de duas series em que não viram os episodios todos).

5. `vector<int> usersCategoryStatistics(NodeUser* root, string cat, int perc);`
   Cria um vector com algumas estatisticas:
   1) Na 1ª posição do vetor o numero total de utilizadores que viram determinada série de determinada categoria cat
   2) Na 2ª posição do vetor o numero total de utilizadores que viram um numero minimo de episodios de determinada série de determinada categoria cat. Esse numero minimo é determinado pela percentagem(perc) total de episódios de cada serie.
   3) Na 3ª posição do vetor o numero total de utilizadores que viram um numero minimo de episodios de determinada série de determinada categoria cat. Mas que também tenham como favorito a categoria cat.

Nota: Os ficheiros de entrada e casos de teste em que serão avaliadas as funções submetidas poderão apresentar conteúdo diferente e incluir casos limite (por exemplo, argumentos de funções com gamas não previstas). Como tal, é sua responsabilidade garantir que os argumentos são devidamente testados de forma a aceitá-los apenas quando válidos.

4. Teste da biblioteca de funções

A biblioteca pode ser testada executando o programa `series_testTP3`. Existe um teste por cada função a implementar e que determina se essa função tem o comportamento esperado. Note que os testes não são exaustivos. Por isso, os testes devem ser considerados apenas como um indicador de uma aparente correta implementação das funcionalidades esperadas.

Se as funções passarem nos testes unitários incluídos, o programa `series_testTP3`, quando executado, deverá apresentar o seguinte resultado:

```
INICIO DOS TESTES


...verifica_queueTVSeriesCategory: (Categoria incorreta) - retorno é uma fila de prioridade vazia (ok)

...verifica_queueTVSeriesCategory: (Series da categoria comedia) Tamanho da fila de prioridade (=3) e' o esperado (ok)

...verifica_queueTVSeriesCategory: (Series da categoria comedia) 1º elemento da fila de prioridade (=Friends) e' o esperado (ok)

OK: verifica_queueTVSeriesCategory passou


...verifica_queueTVSeries: (lista de series vazia) retorno é uma lista vazia (ok)
```

```
...verifica_queueTVSeries: (Lista de Series existe),  Tamanho da fila de prioridade (=5) e' o esperado (ok)

...verifica_queueTVSeries: (Lista de Series existe) 1º elemento da fila de prioridade  (=Stranger Things) e' o esperado (ok)

OK: verifica_queueTVSeries passou


...verifica_usersInitialLetter: (Nó não existe) o vetor de retorno é vazio  (ok)

...verifica_usersInitialLetter: (Nó existe), retorno(=2) (ok)

...verifica_usersInitialLetter: (Nó existe) Vetor de utlizadores como a carater inicial de m (=mia_davis - mike124) e' o esperado (ok)

OK: verifica_usersInitialLetter passou


...verifica_usersNotFan: (Nó não existe) a lista de retorno é vazia  (ok)

...verifica_usersNotFan: (Nó existe), Tamanho da lista (=4) (ok)

...verifica_usersNotFan: (Nó existe) Lista de utilizadores que não são fans (=emily_c - carlitos - rodrigo8 - teresa_santos) e' o esperado (ok)

OK: verifica_usersNotFan passou


...verifica_usersCategoryStatistics: (Nó não existe) o vetor de estatistica tem todas as posicoes a zero (ok)

...verifica_usersCategoryStatistics: (Percentagem fora dos limites) o vetor de estatistica tem todas as posicoes a zero (ok)

...verifica_usersCategoryStatistics: (Parametros ok), número de utlizadores que assistiram a pelo menos uma série de uma categoria  (=6) (ok)

...verifica_usersCategoryStatistics: (Parametros ok), número de utilizadores que assistiram a uma determinada numero de episodios de pelo menos uma série de uma categoria (=4) (ok)

...verifica_usersCategoryStatistics: (Parametros ok), número de utilizadores que assistiram a uma determinada numero de episodios de pelo menos uma série de uma categoria que gostam (=1) (ok)

OK: verifica_usersCategoryStatistics passou


FIM DOS TESTES: Todos os testes passaram
```

5. Ferramenta de desenvolvimento

A utilização de um IDE ou do Visual Studio Code é aconselhável no desenvolvimento deste trabalho, uma vez que permite fazer depuração de uma forma mais eficaz. Poderá encontrar informações sobre a utilização do Visual Studio Code num breve tutorial disponibilizado no Moodle.

É possível implementar as funções solicitadas diretamente no CodeRunner, sendo aconselhável consultar os ficheiros fornecidos, de modo a compreender todo o contexto do trabalho a ser realizado.

6. Avaliação

A classificação do trabalho é dada pela avaliação feita à implementação submetida pelos estudantes, sendo automaticamente atribuída no Moodle, e à capacidade de os estudantes explicarem o código submetido. A classificação final do trabalho (MTP3) é dada por:

MTP3 = Implementação x avaliação oral

A classificação da implementação é essencialmente determinada por testes automáticos adicionais (por exemplo, recorrendo a ficheiros de teste de maiores dimensões). No caso de a implementação submetida não compilar, esta componente será 0%.

A avaliação oral será dividida em 4 patamares: 100% domina o código ; 75% –algumas falhas 40 % - várias falhas detetas na explicação; 0% – demonstrou graves lacunas.

7. Submissão da resolução

A submissão é apenas possível através do Moodle e até à data indicada no início do documento. A submissão da implementação das funções deverá ser realizada através do CodeRunner, nos espaços preparados no Moodle.