Trabalho Prático 4

Continuação da implementação de um sistema de gestão de séries televisivas

1. Informação geral

O Trabalho Prático 4 aplica conceitos de Programação Orientada a Objetos e consiste na implementação de classes baseadas em grafos e tabelas de dispersão.

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 **28 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 **`EDA_2024_MTP4.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`, `UserManagementTree`, `HashTable` e `UserManagementGraph`).
- `TVseries.cpp`: implementação dos métodos relativos às classes definidas em `TVseries.hpp`.
- `series_testTP4.cpp`: inclui o programa principal que invoca e realiza testes básicos às funções implementadas.
- `.txt`: ficheiros de texto para teste das funções implementadas.

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 terceiro trabalho.**

O ficheiro contém as classes:

1. `User`,
2. `TVSeries`,
3. `TVSeriesManagement`,
4. `UserManagement`,
5. `TVSeriesManagementList`,
6. `UserManagementList`,
7. `NodeUser`,
8. `UserManagementTree`,
9. `HashTable`,
10. `UserManagementGraph`.

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 classes fazem o mesmo que a segunda e terceira, respetivamente, mas recorrendo a listas. A sétima representa um nó da árvore definida na oitava classe, uma árvore BST com todos os utilizadores registados. A nona classe define uma tabela de dispersão para armazenamento das estatísticas por país e, finalmente, a décima define um grafo 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 séries que pretendemos ver, por ordem de chegada (`wishSeries`)
9) inteiro que armazena a distância a determinado nó (`length`)

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`. `P`ossuem 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, respetivamente.

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.

Classe `HashTable`

Os objetos da classe `HashTable` definem uma tabela de dispersão para representação das estatísticas por país, tendo os seguintes atributos:

1) tamanho da tabela de dispersão (`tableSize`)
2) vetor de apontadores para elementos do tipo `CountryStats`, representativo da tabela de dispersão (`table`)

a. `CountryStats` é uma *struct* com os campos `country`, `nUsers`, `nTVSeries`, `averageTVseries` e `nGenre` que correspondem, respetivamente, ao nome do país, número de utilizadores desse país, número de séries distintas visualizadas nesse país, valor médio do número de séries visualizadas pelos utilizadores nesse país, e vetor do número de séries visualizadas, por género, nesse país.

3) número efetivo de elementos na tabela de dispersão (`totalCountryStats`)

Classe `UserManagementGraph`

Os objetos da classe `UserManagementGraph` representam a rede de interações entre os utilizadores (quem segue quem) e possuem os seguintes atributos:

1) número total de nós do grafo (`totalUsers`)
2) vetor de apontadores para objetos da classe `User` que identifica os utilizadores incluídos (`userNodes`)
3) vetor de listas de apontadores para objetos da classe `User` que representa uma lista de adjacências e identifica as arestas do grafo, ou seja, quem segue quem (`network`)

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

Classe `UserManagementGraph`

1. `vector<User*>` **`mostFollowing`**`();`
   Determina qual(quais) o(s) utilizador(es) que segue(m) mais utilizadores. Retorna um vetor de apontadores para esse(s) utilizador(es), ou seja, o vetor terá mais do que um elemento no caso de mais do que um utilizador seguir o mesmo número de pessoas.

2. `TVSeries*` **`followingMostWatchedSeries`**`(User *userPtr);`
   Determina qual a série mais visualizada pelos utilizadores que determinado utilizador (`userPtr`*) segue, tendo em conta o número de episódios visualizados de cada série por cada um desses utilizadores. No caso de duas ou mais séries terem o mesmo total de episódios vistos, é selecionada a série visualizada pelo maior número de utilizadores; no caso de este critério também resultar em empate, é selecionada a série que se encontra alfabeticamente à frente. Verifica se o nó pertence ao grafo e, em caso de erro, retorna* `NULL`.

   SUGESTÕES
   Para verificar se um nó pertence ao grafo, pode recorrer ao método `userNodePosition()`.

3. `int` **`shortestPaths`**`(User *userSrc, User *userDst);`
   Encontra a distância mínima entre dois nós do grafo (`userSrc` → `userDst`*), aplicando o algoritmo de Dijkstra. Verifica se os nós pertencem ao grafo. Retorna o valor da distância encontrada, -1 em caso de erro ou -2 se não existir um caminho entre os dois nós.*

   SUGESTÕES
   Para aplicar o algoritmo de Dijkstra, recorra à utilização de *min-heaps* com a função de comparação `CompareP`. De relembrar o atributo `length` e os métodos `setLength()` e `getLength()` de `User`.

Classe `HashTable`

4. `int` **`insertCountryStats`**`(CountryStats &countryS);`
   Adiciona um novo elemento (`countryS`*) à tabela de dispersão, retornando o índice do elemento inserido ou -1 caso elemento já exista ou em caso de erro. A chave do elemento a inserir corresponde ao campo* `country`*. Atualiza o valor do número de elementos efetivos na tabela de dispersão. Caso ocorra uma colisão, o elemento pode ser inserido numa posição livre ou numa posição da qual foi previamente eliminado um elemento (chave do índice corresponde a "apagado").*

5. `int` **`importFromVector`**`(UserManagement &userManager);`
   Preenche a tabela de dispersão com base no vetor de utilizadores do objeto `userManager`*. Cria um novo elemento do tipo* `CountryStats` *por cada país distinto dos utilizadores, determinando o valor de cada um dos seus campos. Retorna -1 em caso de erro ou 0 em caso de sucesso.*

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_testTP4`. 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_testTP4`, quando executado, deverá apresentar o seguinte resultado:

```
INICIO DOS TESTES


...verifica_mostFollowing: (Grafo vazio) - retorno é um vetor vazio (ok)

...verifica_mostFollowing: (Grafo preenchido) Tamanho do vetor (=4) e' o esperado (ok)

...verifica_mostFollowing: (Grafo preenchido) Os elementos  (=john_doe - mia_davis - rodrigo8 - emily_c) são os esperados (ok)

OK: verifica_mostFollowing passou


...verifica_followingMostWatchedSeries: (User não existe) retorno é Null (ok)

...verifica_followingMostWatchedSeries: (User existe),  A serie (=The Office) e' o esperado (ok)

...verifica_followingMostWatchedSeries: (User existe-empate entre series),  A serie (=Stranger Things) e' o esperado (ok)

OK: verifica_followingMostWatchedSeries passou


...verifica_shortestPaths: (Nó não existe) o retorno é o esperado (=-1)  (ok)

...verifica_shortestPaths: (Nó existe), Distancia (=3) (ok)

...verifica_shortestPaths: (Caminho não existe), Retorno (=-2) (ok)

OK: verifica_shortestPaths passou
```

```
...verifica_insertCountryStats: (1º Insersão 'Portugal' numa tabela de tamanho 11) campo onde foi inserido (=6) (ok)

...verifica_insertCountryStats: (Nó existe), campo onde a Alemanha foi inserido (=10) (ok)

OK: verifica_insertCountryStats passou


...verifica_importFromVector: (Importar o vetor) numeros de elementos inseridos (=7) (ok)

...verifica_importFromVector: (Importar o vetor) Portugal foi encontrado (ok)

...verifica_importFromVector: (Importar o vetor) numero de utilizadores em Portugal (=13) (ok)

...verifica_importFromVector: (Importar o vetor) numero de series vistas em Portugal (=17) (ok)

...verifica_importFromVector: (Importar o vetor) média de series vistas por utilizador em Portugal (=2.23077) (ok)

...verifica_importFromVector: (Importar o vetor) numero de series de Action vistas em Portugal (=2) (ok)

OK: verifica_importFromVector 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 (MTP4) é dada por:

$$MTP4 = Implementação \times avaliação\ oral$$

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.