UFMG – DCC – Algoritmos e Estruturas de Dados II – 2º Semestre de 2016
Trabalho Prático 1 – Listas, Pilhas e Filas [10 pontos]
Professores : Dorgival Guedes e Adriano Veloso
Prazo Final para Entrega: 23:55 de 03/11/2016

Rede Social — AEDs II

1 Introdução

O objetivo desse trabalho é explorar e fixar o conteúdo pertinente ao módulo 2 da disciplina, ou seja, estruturas de dados, especialmente filas, pilhas e listas. O importante é compreender e manipular cada uma das estruturas de maneira correta. Para isso, deve ser implementado um sistema que simule uma rede social usando as estruturas aprendidas em sala.

Milhões de pessoas jovens e adultos possuem hoje um perfil em pelo menos alguma rede social. Existem diversos sistemas e funcionalidades entre eles, mas a grande maioria tem em comum o fato de você poder marcar usuários como amigos e fazer postagens na rede. O ***Twitter***, por exemplo, é uma rede social e um servidor para microblogging, que permite aos usuários enviar e receber atualizações pessoais de outros contatos em textos de até 140 caracteres, conhecidos como "*tweets*". As atualizações são exibidas no perfil de um usuário em tempo real e também enviadas aos usuários que são seguidores do mesmo.

Dentre os vários desafios de um sistema para uma rede social podemos citar com destaque a funcionalidade de gerenciar a "*timeline*" dos usuários. Pessoas criam conteúdo o tempo inteiro nessas redes e o trabalho do sistema é encontrar uma solução para armazenar todo esse conteúdo, bem como enviá-lo a todos os seus seguidores. Quando um usuário acessa seu perfil numa rede social, como o *Twitter*, ele visualiza uma lista de mensagens que foram postadas por seus amigos/seguidores, para isso, uma requisição é feita no servidor e as novas mensagens são carregadas e exibidas na tela.

Sua tarefa neste trabalho é simular uma rede social simplificada. No desenvolvimento do trabalho você deve gerenciar as ações dos usuários na rede, e exibir corretamente quais mensagens devem aparecer na timeline de cada usuário. O usuário será capaz de postar uma mensagem na rede, se tornar amigo de novos usuários, além de visualizar e curtir mensagens de outros usuários. Portanto, todos os usuários terão um conjunto de amigos e uma linha do tempo. Além disso, um usuário pode realizar as seguintes ações: postar mensagens, iniciar amizade, cancelar amizade, curtir mensagens e exibir sua timeline. Ao final do trabalho, o sistema deve gerar um arquivo de histórico contendo o que cada um dos usuários visualizou em sua timeline em cada um dos instantes de ações do sistema.

2 Descrição das tarefas

2.1 Ações do Usuário

A rede social implementada possui um conjunto de ações já definidas que um usuário pode realizar, como visto na Tabela 1. Você irá receber como arquivo de entrada um conjunto de usuários previamente definidos. Portanto, não é necessário implementar funções para criar ou excluir usuários da rede social.

Função	**Código**
Postar mensagem	1
Iniciar amizade	2
Cancelar amizade	3
Curtir mensagem	4
Exibir timeline	5

Tabela 1: Códigos das ações do usuário na rede social

Das funções que você deve gerenciar:

- ***Postar mensagem:*** uma nova mensagem será adicionada ao sistema e deverá ficar disponível para visualização na timeline de cada um dos usuários que o seguem.
- ***Iniciar amizade:*** o link decorrente da ação de iniciar amizade é não-direcionado, ou seja, dois usuários $Usuario1$ e $Usuario2$ só poderão ser amigos simultaneamente ($Usuario1$ é amigo de $Usuario2$ e $Usuario2$ é amigo de $Usuario1$) e ambos poderão ver as mensagens um do outro na sua timeline. Somente as

mensagens posteriores à ação de seguir um usuário serão exibidas na timeline dos amigos. Ou seja, a ação *Iniciar Amizade* não irá adicionar mensagens que foram postadas antes da ação de se tornar amigo de um usuário em sua timeline.

- ***Cancelar amizade:*** essa função faz com que os usuários deixem de ver mensagens postadas pelo antigo amigo após o momento em que a amizade é cancelada. Entretanto, as mensagens que já estavam na timeline e foram postadas enquanto o usuário ainda estava na lista de amigos não são eliminadas.
- ***Curtir mensagem:*** essa função mede a popularidade de uma mensagem pela quantidade de vezes em que esta foi curtida. Quando um usuário executa a ação *Curtir mensagem*, a mensagem é atualizada na timeline, e o atributo referente ao número de vezes que ela foi curtida deve ser atualizado. Entenda por atualizada na timeline, que a mensagem será atualizada em tempo de exibição na timeline, ou seja, seu tempo de ação será atualizado e ela podem mudar de posição na timeline.
- ***Exibir timeline:*** função que permite a um usuário ver as mensagens que ele ainda não visualizou, ou seja, mensagens novas no sistema desde a última ação de *Exibir timeline*. Além disso, as mensagens devem ser exibidas em ordem cronólogica, seguindo o tempo em que elas foram postadas (ou atualizadas com a ação de curtir), sendo as mais recentes primeiro e as mais antigas por último. As postagens devem ser exibidas imprimindo seu texto seguido do número de vezes que ela foi curtida. Essa função inclui a exibição das mensagens que o próprio usuário postou.

Por exemplo, considerando a seguinte lista de ações:

Usuário1 posta **A** no tempo t1

Usuário2 posta **B** no tempo t2

Usuário3 posta **C** no tempo t3

Usuário3 curte **A** no tempo t4

Usuário1 posta **D** no tempo t5

Para algum outro **Usuário 4**, com apenas **Usuários 1 e 2** em sua lista de amigos, ao ver sua timeline em cada instante de tempo, ele veria as mensagens como ilustrado na Figura 1.

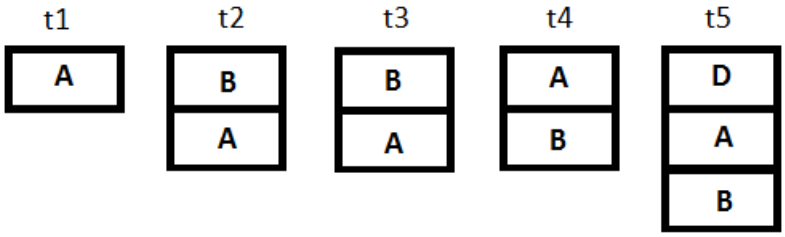

Figura 1: Exemplo da timeline do usuário 4 em cada um dos instantes de tempo

Como ilustrado na Figura 1, apesar de **A** ter sido postada num tempo mais antigo que **B**, como ela foi curtida depois no tempo **t4** ela aparece na frente e portanto foi atualizada. **D** aparece primeiro porque foi a ação mais recente, mesmo sem curtidas. **C** nesse caso não é vista porque **Usuário4** não é amigo de **Usuário3**. **B** aparece por último pois se tornou a mensagem com a ação mais antiga. Ou seja, as ações mais recentes sempre aparecem na frente.

2.2 Tipo Abstrato de Dados

A organização e legibilidade do código é uma parte importante no desenvolvimento da tarefa. A criação de TADs irá ajudar a preparar e modelar o problema. Dois TADs são importantes para este trabalho: usuário e mensagem.

Um usuário deve ser representado por um identificador, um nome e sua lista de seguidores. As mensagens serão representadas por um ID, um corpo formado pelo texto, um tempo **t** que representa o instante em que ela foi postada e o número de curtidas. **As mensagens terão um tamanho máximo de 140 caracteres.**

Já que cada um dos usuários possui uma lista de seguidores e a sua linha do tempo, as funções seguintes devem ser obrigatoriamente implementadas e ter exatamente os nomes aqui citados.

- **iniciarAmizade()**: responsável por incluir amigos na lista de amigos do usuário. **Mensagens da timeline dos usuários postadas antes dessa ação não serão inseridas na timeline, apenas a postadas após essa ação.**
- **cancelarAmizade()**: responsável por remover um usuário da lista de amigos. **A ação de cancelar uma amizade não elimina mensagens anteriores ao tempo em que a amizade entre dois usuários foi cancelada.**
- **postarMensagem()**: inclui a mensagem em cada uma das timelines dos amigos do usuário.
- **curtirMensagem()**: executará a ação que incrementalmente irá gerenciar a quantidade de curtidas recebidas por uma mensagem.
- **exibirTimeline()**: é a função que retornará todas as **novas** mensagens existentes na timeline do usuário. Isso inclui as mensagens postadas pelo próprio usuário bem como as mensagens postadas por seus amigos. Mensagens antigas que já foram exibidas e que não tiveram nenhuma atualização posterior não precisam ser exibidas novamente.
- **verAmigos()**: retorna a lista de amigos, ou seja, retorna todos os usuários que são amigos de um determinado usuário.

3 Arquivo de Entrada

No início do arquivo de entrada serão indicados o número de usuários da rede e, em seguida, os detalhes de cada usuário, assim como a lista de seus amigos (caso exista). Após isso, seguirão as ações recebidas dos usuários: em cada tempo [t] do sistema, será recebida uma ação de algum usuário no formato padrão das ações descritas. Os dados de cada linha sempre estarão separados por um **ponto e vírgula**.

A entrada do trabalho será um arquivo de texto e seguirá o seguinte formato:

Número de usuários
ID do usuario;Nome;Amigos
ID do usuario;Nome;Amigos
...
Tempo;Ação;Parametros
Tempo;Ação;Parametros
Tempo;Ação;Parametros
Tempo;Ação;Parametros
...

Cada ação no sistema possui seu padrão e seus parametros conforme a Tabela 2. Sempre que cada função aparecer no sistema ela seguirá esse formato, possibilitando então saber quais são os parâmetros de cada uma.

Função	**Código**	**Formatação da ação**
Postar mensagem	1	Tempo;Codigo;Usuario;IdMensagem;Texto
Iniciar amizade	2	Tempo;Codigo;Usuario1;Usuario2
Cancelar amizade	3	Tempo;Codigo;Usuario1;Usuario2
Curtir mensagem	4	Tempo;Codigo;Usuario;IdMensagem
Ver timeline	5	Tempo;Codigo;Usuario

Tabela 2: Códigos das funções da rede social

Por exemplo, um arquivo de entrada com 4 usuários o arquivo poderia ser:

```
4
100;Clara;200;300
200;Matheus;100;400
300;Manu;100
400;Lucas;200
1;2;300;400
2;1;100;5000;Assistindo O exorcista!!
3;5;300
4;1;100;5001;Adoro filme de terror
5;1;200;5002;Jogão do Atlético e Cruzeiro esse domingo
```

6;5;300

Neste exemplo temos 4 usuários. Um deles é o usuário 100 que possui o nome Clara, e tem como amigos os usuários 200 e o 300. Da mesma forma, sabemos que o usuário 200 possui nome Matheus e amigos 100 e 400; e assim por diante. Ainda no exemplo, temos que a primeira ação no tempo 1 foi o início de amizade entre os usuários 300 e 400. No tempo 2 o usuário 100 postou uma mensagem; no tempo 3 novamente o usuário 300 visualizou sua timeline; no tempo 4 o usuário 100 postou uma mensagem; no tempo 5 o usuário 200 postou uma mensagem; no tempo 6 o usuário 300 visualizou sua timeline.

4 Arquivo de Saída

No arquivo de saída é esperado que você descreva o que cada um dos usuários leu no momento em que cada ação de exibir timeline foi realizada. Seguindo o exemplo de entrada especificado acima, na seção de arquivo de entrada, teríamos o seguinte:

300 Manu
5000 Assistindo O exorcista!! 0
300 Manu
5001 Adoro filme de terror 0

O arquivo de saída conterá as visualizações de timeline e deve seguir sempre o padrão: cada visualização de timeline deverá iniciar com o ID do usuário e nome, seguido por uma linha para cada mensagem que ele viu durante aquela ação específica de ver timeline e o número de curtidas daquela mensagem naquele momento. Toda ação de ver a timeline será então registrada na saída do seu programa.

Sua saída deve ser entregue em um arquivo texto com o seguinte nome: log.entrada0x.**txt**, no qual x deve ser substituído pelo caso de teste executado. **Ex:** entrada01.txt irá gerar log.entrada01.txt.

5 Desafio

Supondo que você construiu a sua rede social e queira incrementá-la, vamos propor um desafio. Imagine que você queira começar a sugerir mensagens interessantes para os usuários. Então você deve propor e implementar uma estratégia para encontrar 3 mensagens que podem ser mais interessantes para cada um dos usuários no futuro. A ideia é pensar em uma estratégia que fará sugestões tendo em vista as mensagens que o usuário curtiu no passado. Para esse tipo de tarefa você pode pensar em estratégias que fazem a utilização de índices de similaridade. Um exemplo de índice de similaridade é o de *Jaccard*[1], o mais antigo dentre esses índices, que é um índice qualitativo, isto é, que não considera as quantidades nas populações que compõem o cálculo.

O seu arquivo de saída, para o desafio, deve se chamar **desafioLog.txt** e respeitar o seguinte formato:

IDUsuário1
IDMensagem1 TextoMensagem1
IDMensagem2 TextoMensagem2
IDMensagem3 TextoMensagem3
IDUsuário2
IDMensagem1 TextoMensagem1
IDMensagem2 TextoMensagem2
IDMensagem3 TextoMensagem3
IDUsuário3
IDMensagem1 TextoMensagem1
IDMensagem2 TextoMensagem2
IDMensagem3 TextoMensagem3

...

Uma seção da sua documentação deve ser dedicada a uma explicação sucinta sobre quais as decisões foram necessárias na modelagem e implementação do desafio.

[1] https://en.wikipedia.org/wiki/Jaccard_index

6 Comentários Importantes

- Devem ser enviados no minha.ufmg os códigos implementados no TP, o arquivo Makefile, quaisquer scripts para compilação/execução e um relatório em formato *.pdf contendo a descrição dos passos realizados para solução do TP. Todos esses arquivos devem ser comprimidos e enviados como apenas um arquivo *.zip.
- **O programa criado não deve conter "menus interativos" ou "paradas para entrada de comandos" (como o system("PAUSE"), por exemplo, que costuma ser usado no ambiente Code Blocks). Ele deve apenas ler os arquivos de entrada, processá-los e gerar os arquivos de saída. Os TPs serão corrigidos em um ambiente Linux. Portanto, o uso de bibliotecas do Windows está PROIBIDO.**
- Quanto ao uso de bibliotecas, está liberado o uso da biblioteca padrão C, que inclui as interfaces definidas pelos arquivos unistd.h, stdio.h e stdlib.h. Qualquer outra biblioteca que você precise de usar, pergunte previamente via moodle.
- O relatório deve ser objetivo e apresentar, preferencialmente, os seguintes tópicos:
  1. **Introdução:** descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa.
  2. **Modelagem e Funcionamento:** é importante discutir quais as estratégias, e quais passos você chegaram até a solução final do seu problema.
  3. **Implementação:** descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
  4. **Estudo de Complexidade:** análise da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O).
  5. **A compilação e execução do seu trabalho devem ser feitas por um arquivo Makefile. Você deve colocar, brevemente, na documentação como o seu programa é executado.**
  6. **Conclusão:** comentários gerais sobre o trabalho e as principais dificuldades encontradas em seu desenvolvimento.
  7. **Bibliografia:** Referências utilizadas para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso.
  8. É importante dar atenção as boas práticas de programação ao organizar seu código.
- A documentação não deve exceder **10 páginas**.
- **Obvio!** Não é permitido o compartilhamento de relatório ou código entre os estudantes de AEDs II. Além disso, os estudantes envolvidos em plágio (copiando ou permitindo a cópia) serão punidos. Busque empregar *suas próprias palavras* ao escrever esse trabalho.
- Faça uso do fórum do minha.ufmg para compartilhar/responder/conversar sobre dúvidas relativas a esse trabalho, tendo como orientação **não fornecer soluções parciais ou completas**. Dúvidas que envolvam trechos específicos de código devem ser enviadas diretamente aos monitores e ao professor por e-mail, não devem ser colocadas no moodle!
- **Data de Entrega:** até 03 de novembro de 2016, às 23:55 horas, ou antes. Após essa data haverá uma penalização por atraso: $(2^d - 1)$ pontos, em que d é o número de dias de atraso.
- **Valor: 10 pontos.**
- Bom trabalho! Comece logo a fazê-lo; afinal, você nunca terá tanto tempo para resolvê-lo quanto agora!