Twitter

Documentação do Trabalho Prático 1
Universidade Federal de Minas Gerais – Sistemas de Informação

Estrutura de Algoritmos e Dados II
Professores: Dorgival Neto & Adriano Veloso

Tatiana Santos Camelo de Araujo – 2015086298

1. Introdução

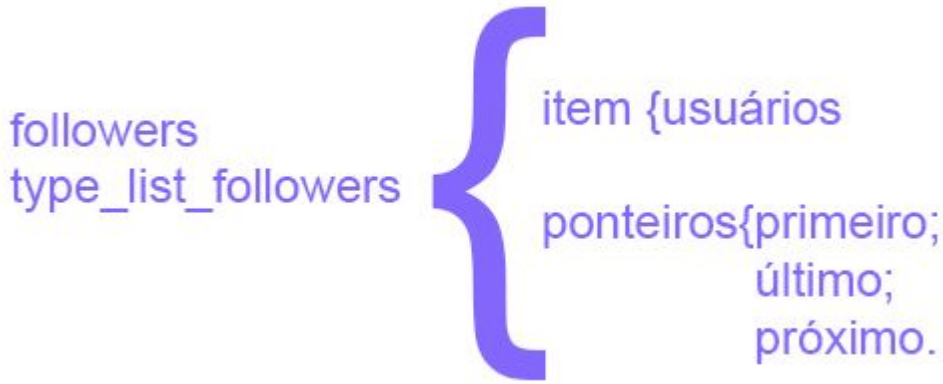
O trabalho aqui descrito foi criado como atividade prática da disciplina de Estrutura de Algoritmos e Dados II. A ideia do trabalho é criar uma rede social simplificada, com uma lógica semelhante a rede social Twitter. Pretende-se, ao longo da implementação do código, fortificar e aplicar o conhecimento adquirido ao longo do semestre. O uso de Tipos Abstratos de Dados (TADs), e estruturas de lista, fila e pilha foram constantes, sendo estes os conceitos que direcionaram o trabalho em si.

A compilação é feita pelo comando "gcc *.c -Wall -o main"gcc *.c -Wall -o main".

2. Modelagem e Funcionamento

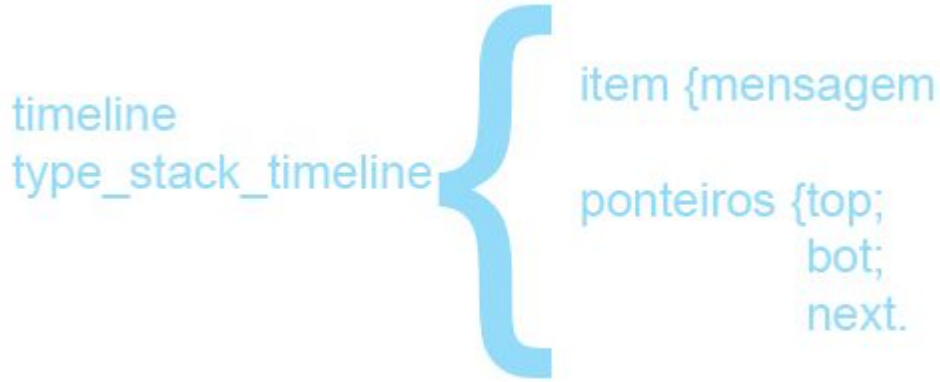
Para facilitar e manter linearidade ao nome das funções e variáveis, usou-se a padronização dos nomes em inglês, descritivos, e palavras separadas pelo caractere "_". Esta escolha foi feita para tentar facilitar ao máximo o entendimento do código até de forma localizada, sem precisar verificar diferentes arquivos ou outras funções para entender o que cada função faz, ou o que cada variável representa.

Em questão de modelagem, foi decidido que o sistema de gerenciamento de amizades seria feito por estrutura de lista, com os ítem de tipo usuário (que será descrito posteriormente). O esquema abaixo mostra de forma simplificada a estrutura da lista de seguidores. A lista não é duplamente encadeada, pois não foi apresentado nenhuma necessidade de acesso reverso aos usuários na lista.

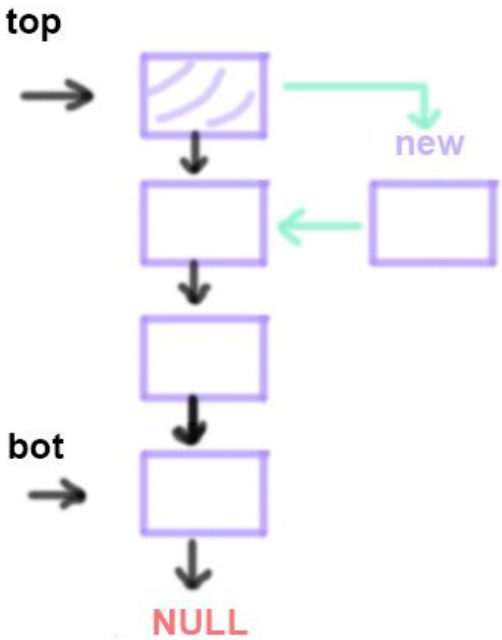


Para gerenciamento das timelines foi decidido que seria a estrutura de pilha. Esta escolha se deu de forma a, ao exibir a timeline, a mensagem mais recente seja a primeira a ser mostrada, ou seja, no topo da pilha. A pilha possui itens do tipo mensagem (que também será descrito posteriormente). Para a timeline, é interessante dizer que criei também a timeline de cada usuário, podendo então ter acesso às mensagens que um especifico usuário mandou ao longo de todo programa. Essa separação é feita pelo nome escolhido da variável do tipo timeline, podendo ser type_stack_timeline_seen – esta é a

timeline que o usuário vê, com as mensagens se seus amigos e suas próprias –, e type_stack_timeline_user – pilha que armazena as mensagens individuais de cada usuário.



A escolha da estrutura de pilha para a timeline é bem interessante para este trabalho, já que pela definição de pilha, a mensagem mais atual será sempre a primeira a ser visualizada.

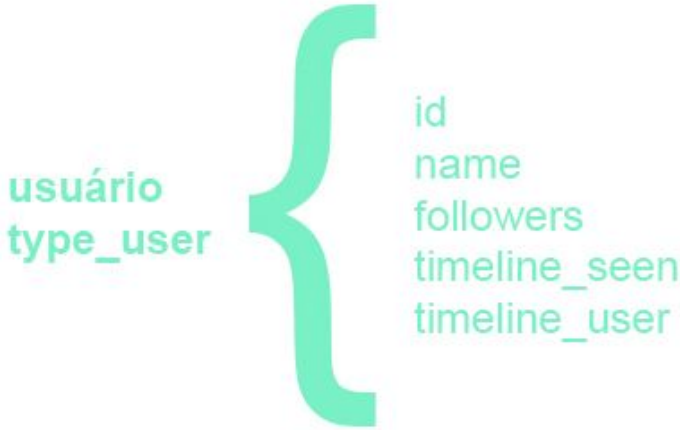


3. Implementação

3.1. TADs

Foram usados dois tipos de TAD para a implementação deste trabalho, sendo estes os tipos usuário e tipo mensagem.

Abaixo está uma representação do TAD usuário, com seus atributos. Todo usuário recebe um número de identificação, nome, lista de amigos e pilha de timeline. Como citado previamente, foi uma escolha da programadora que todo usuário também tenha a timeline própria, que armazena suas mensagens separadamente.



Por fim, o TAD das mensagens, que possuem texto, identificação, hora de postagem e/ou última atualização e curtidas.

3.2. Principais funções

Código 1:
postarMensagem()
Esta função tem quatro ações implementada. Ela vê a lista de todas amizades do usuário que postou a mensagem; coloca a mensagem na timeline dos amigos; coloca a mensagem na timeline do usuário. Nesta função, como ela será chamada no tempo que que a mensagem será postada, é considerado, então, que a mensagem postada é a mais atual, indo para o topo da pilha de timelines.

Código 2:
iniciarAmizade()

Nesta função, foi considerado necessário que ela incluísse de forma mútua os amigos como followers de cada usuário. Assim, inicia-se uma amizade entre usuário1 e usuários2, logo, usuário1 segue o usuário2 e virce versa. Pela escolha de lista para armazenar as amizades, esta ação fará que o amigo mais recente apareça no final da lista.
Como escolha na implementação da função postarMensagem() coloca a mensagem na timeline apenas em quem é amigo do usuário que postou, não há adição das mensagens do momento antes do início da amizade de nenhum dos usuários.

Código 3:
cancelarAmizade()
Esta função retira um usuário da lista de amigo de outro. Foi implementada de forma que faça o mesmo inversamente. Nunca há situações em que a amizade nesta rede não seja mútua. Esta função não deleta mensagens que já apareceram antes do cancelamento da amizade.

Código 4:
curtirMensagem()
Esta função apenas incrementa o número de curtidas de uma mensagem enviada e atualiza seu tempo, colocando-a novamente no topo da pilha da timeline.

Código 5:
exibirTimeline()
É uma função simples que percorre toda pilha da timeline e imprime na tela todas as mensagens, por meio de uma variável com auto increment da posição de cada caractere, já que a mensagem foi salvo como um vetor para delimitar o número de caracteres. Além de cada mensagem, imprime também as curtidas de cada mensagem.

verAmigos()
Esta função é um loop que retorna cada um dos usuários que estão na lista de amigos de um usuário escolhido.

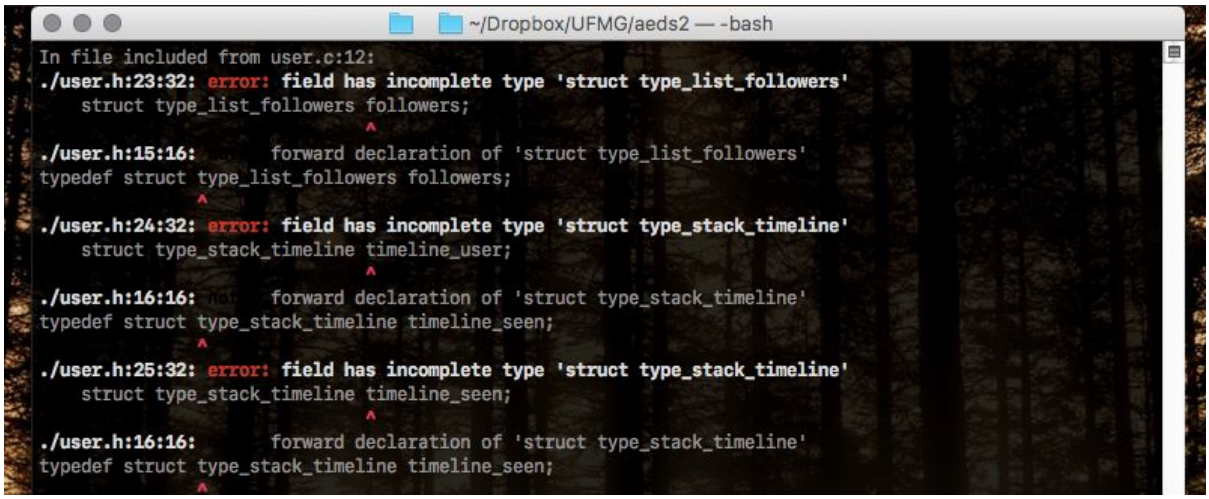
4. Estudo de Complexidade

Função	Complexidade
postarMensagem	O(nˆ2) – [(O(n)*O(n)) + O(1) + O(1)] – Pecorre a lista com dois loops enlaçados e dois usos de stacking
iniciarAmizade	O(1) – Ponteiro para o final da lista impede que seja necessário percorré-la toda
cancelarAmizade	O(nˆ2) – [O(nˆ2)+O(nˆ2)] – Percorre a lista de n amigos de usuário com dois loops enlaçados
curtirMensagem	O(1) – [O(1)+O(1)] – Ação simples de incremento e uso de stacking

exibirTimeline	O(n) – [O(n)*O(141)] – Percorre todas mensagens e seus vetores
verAmigos	O(n) – A função percorre n amigos na lista
make_list/stack	O(1) – A função faz ação simples de criar lista ou pilha
stacking	O(1) – A função faz ação simples de adicionar um elemento
Programa completo	O(nˆ3) – Leitura de arquivo, divisão de string, controle de linhas de arquivo, uso de switch para determinação de cada caso, uso de todas funções estabelecidas acima, impressão em um arquivo de saída

5. Execução

Houve um problema na .O uso de tipos de dado de forma circular não foi solucionado, o que impossibilitou a execução.



6. Conclusão

Este trabalho é essencial para o entendimento de pilhas, filas e listas e suas funcionalidades, funções e implementações. Ao longo da implementação foram usadas todas as formas. As listas foram escolhidas como melhor implementação ao longo do código para organizar as amizades de cada usuário, e pilhas foram escolhidas e usadas para gerenciamento das timelines, tanto para a pessoal do usuário quanto para a timeline que cada usuário vê.

A maior dificuldade encontrada ao longo deste trabalho foi o uso circular de structs. Este problema não foi solucionado – mesmo com muita pesquisa e ajuda –, de forma que não foi possível rodar os testes. Mesmo assim, todas funções foram implementadas e revisadas, acredito que funcionam de forma ideal.

7. Bibliografia

Foram usados os seguintes sites como auxiliar na implementação, além dos materiais da disciplina:

www.tutorialspoint.com
www.cplusplus.com
www.sanfoundry.com/
stackoverflow.com