Trabalho Prático 1 – Rede Social

Deiziane Natani da Silva – TA1

1. Introdução

O trabalho tem como objetivo praticar os conceitos de lista, pilha e fila, vistos em sala de aula. O TP consiste em criar uma rede social, parecida com o Twitter, em que cada usuário possui amigos e podem interagir postando e lendo mensagens de amigos na timeline.

Os usuários da rede podem efetuar várias ações, cada uma identificada por um código especial, que vai de 1 a 5. As ações são: postar mensagem, iniciar amizade, cancelar amizade, curtir mensagem e exibir timeline. Cada mensagem postada por um usuário pode ser visualizada e curtida por seus amigos. Ao ser curtida, essa mensagem deverá ir para o topo da exibição na timeline de todos os amigos do usuário que a postou, e seu número de curtidas deverá ser incrementado em um. Mensagens mais recentes sempre aparecem no topo.

O programa deve ler uma entrada e gerar a saída de acordo com o que cada usuário visualizou.

2. Implementação

Estrutura de Dados

Para o desenvolvimento do TP, foram utilizadas 3 listas lineares que funcionam como pilhas (“First in, Last out”). Uma armazena os usuários, e tem a seguinte estrutura:

```
typedef struct dados {
    int id; //armazena o id do usuario
    char nome[50]; //armazena o nome
    ListaMigos *migos; //armazena a lista de amigos
    int num_migos; //armazena o numero de amigos
    Pilha *mensagens; //armazena a pilha de mensagens
    int num_msg; //armazena o numero de mensagens
}Dados;

typedef struct node{
    Dados data;
    struct node *prox;
}Node;

typedef struct lista{
    int tamanho;
    Node *cabeca;
}Lista;
```

A outra armazena os amigos de cada usuário e tem a seguinte estrutura:

```c
typedef struct dadosmigos {
    int id_migo; //id do amigo
}DadosMigos;

typedef struct nodemigos{
    DadosMigos data_migo;
    struct nodemigos *prox_migo;
}NodeMigos;

typedef struct listamigos{
    int tamanho;
    NodeMigos *top;
}ListaMigos;
```

E a última armazena as mensagens da timeline de cada usuário:

```c
typedef struct mensagem{
    int id_m; //id mensagem
    char texto[145]; //texto da mensagem
    int tempo; //tempo (useless)
    int likes; // numero de likes da mensagem
    int id_user; //id do usuario que postou a mensagem
}Mensagem;

typedef struct celula {
    Mensagem msg;
    struct celula *prox_c;
}Celula;

typedef struct pilha {
    int size;
    Celula *topo;
}Pilha;
```

Funções e Procedimentos

A lista de Usuários possui as seguintes funções:

- **Lista* criaLista();**
  - Cria uma lista vazia.
- **void criaNode(Lista *l, Dados d);**
  - cria um nó na lista.
- **void imprimeLista(Lista* l, Dados d);**
  - imprime os itens da lista (usada somente para testes)
- **bool vazia(Lista* l)**
  - Verifica se a lista está vazia
- **int atNode(Lista* l, int n)**
  - retorna o índice de um nó com um id (n)

- **Node* atIndex(Lista *l, int indice)**
    - Retorna o nó de um determinado índice (geralmente retornado da função atNode).

- **void iniciarAmizade (Lista* l, int id_1, int id_2) (Obrigatoria)**
    - Inicia a amizade entre dois usuários

- **void cancelarAmizade (Lista* l, int id_1, int id_2); (Obrigatoria)**
    - cancela a amizade entre dois usuários

- **void exibirTimeline(Node *n, char* argv); (Obrigatoria)**
    - imprime a timeline no arquivo de saída. Ao exibir a timeline, apenas novas mensagens apareceram. Para isso uma variável com o número de mensagens novas é criada como auxiliar. Note que, nenhuma mensagem é excluída da pilha de mensagens, porém, ao exibi-las, o programa percorre a pilha apenas até a flag, que é a variável auxiliar criada para este propósito.

- **NodeMigos* verAmigos(Node *n); (Obrigatoria)**
    - Retorna a lista de amigos

A lista de Amigos possui as seguintes funções:

- **ListaMigos* criaListaMigos();**
    - Cria uma lista vazia.

- **void criaNodeMigos(ListaMigos *l, DadosMigos d);**
    - cria um nó na lista.

- **bool isEmpty(ListaMigos* l);**
    - Verifica se a lista está vazia

- **int atNodeMigos(ListaMigos* l, int id_migo);**
    - retorna o índice de um nó com um id

- **NodeMigos* atIndexMigos(ListaMigos *l, int indice);**
    - Retorna o nó de um determinado índice (geralmente retornado da função atNodeMigos).

- **void excluiMigo(ListaMigos* lm, int indice);**
    - exclui um amigo da lista (usado no cancelar amizade)

- **void pop(ListaMigos* lm);**

- exclui um amigo que está no fim da lista (usado no cancelar amizade)

A lista de Mensagens possui as seguintes funções:

- **Pilha* criaPilha();**
  - Cria pilha vazia

- **bool pilhaVazia(Pilha* p);**
  - verifica se a pilha está vazia

- **void imprimePilha(Pilha* p, int flag, char* argv);**
  - escreve as mensagens no arquivo de saída (usada no exibir timeline)

- **void postarMensagem(Pilha* p, Mensagem m); (Obrigatoria)**
  - posta uma mensagem na pilha dos usuários

- **int atCelula(Pilha* p, int id_m);**
  - retorna o índice de uma célula com um id

- **Celula* atIndexCel(Pilha* p, int indice);**
  - Retorna a celula de um determinado índice (geralmente retornado da função atCelula).

- **void excluiMsg(Pilha* p, int indice);**
  - exclui a mensagem da pilha. Esta função atua como auxiliar no curtir mensagem. Ao ser curtida, a mensagem é retirada da pilha e colocada novamente no topo.

- **void popMsg(Pilha* p);**
  - mesma função de excluiMsg, porém só retira mensagem do topo da pilha.

- **void curtirMensagem(Pilha* p, int id_m, Pilha* pd); (Obrigatoria)**
  - Curte mensagem da timeline.

Programa principal

O programa principal recebe como entrada arquivos txt e lê todas as informações contidas nele, como o número de usuários, nomes etc, e é criada uma lista de usuários para armazenar todas as informações referentes a eles. Cada usuário possui uma lista de amigos própria, e também uma pilha de mensagens individual.

Em seguida, são lidas as ações dos usuários na rede. Para cada ação, são executadas funções referentes a ela.

Não é feita nenhuma leitura de teclado, e é gerado ao final, um arquivo de saída no formato "log.nomedoarquivo.txt".

Organização do Código, Decisões de Implementação e Detalhes Técnicos

O programa está dividido em 7 arquivos: *main.c, usuários.c, usuários.h, mensagens.c, mensagens.h, amigos.c* e *amigos.h*. Cada arquivo (.c e .h) implementa uma lista diferente e o main.c é o programa principal.

Ficou decidido em discussão no forúm da disciplina que em relação à função que realiza as curtidas nas mensagens da timeline, cada aluno deveria optar por uma abordagem. Na implementação que foi utilizada nesse TP, as mensagens curtidas são atualizadas somente na timeline dos usuários que tem o dono da mensagem como amigo. Por exemplo, se o usuário C postou uma mensagem e B a curtiu, então a mensagem será atualizada e irá para o topo da timeline de todos os amigos de C. Usuários que possuem a mensagem, mas não são mais amigos de C (até antes da curtida) não terão a mensagem atualizada no topo da timeline.

O compilador utilizado foi o CodeBlocks 16.01, no sistema operacional Windows 7.

3. Implementação

A análise de complexidade será feita em função da variável num_users e do tamanho da lista de mensagens.

Função iniciarAmizade() : Nesta função, dois usuários iniciam uma amizade entre si e a partir daí, cada um receberá as mensagens postadas pelo outro em sua timeline, além de poder curti-las. Como para essa função, é necessário encontrar a posição de cada usuário na lista de tamanho num_users, então a função pode ser classificada como **O(n).**

Função cancelarAmizade(): Nesta função, dois usuários deixam de ser amigos. A partir daí, nenhum receberá mais atualizações sobre o outro. Nesta função, também é necessário percorrer a lista de usuários e, além disso, é usada outra função chamada excluirMigo() , que percorre a lista de amigos dos usuários em questão. Por esse motivo, a função pode ser classificada como **$O(n^2)$.**

Função postarMensagem(): Nesta função, um usuário posta uma mensagem e esta aparece em sua timeline e na dos seus amigos. Além de procurar pela célula do usuário na lista, depois é necessário adicionar a mensagem na lista de mensagens de todos os amigos dele. Dessa forma, a função pode ser classificada como **$O(n^2)$.**

Função curtirMensagem(): Por ter uma implementação que inicialmente visava a curtida de mensagens apenas por amigos, e posteriormente tendo que mudar para que qualquer usuário possa curtir qualquer mensagem, mesmo não sendo amigo do usuário que a postou, foi necessário criar um loop adicional que percorre a lista de usuários e e para cada usuário, percorre sua pilha de mensagens até achar a mensagem curtida. Esse

procedimento pode ser classificado como $O(n^2)$ pois utiliza dois loops aninhados. Em seguida, a mensagem é atualizada na timeline do usuário que a postou e na dos seus amigos, sendo necessário também um loop. Logo, a função pode ser classificada como $O(n^2)+O(n) =$ **$O(n^2)$.**

Função exibirTimeline(): A função primeiro acha a célula do usuário que quer ver a timeline na lista, esse procedimento é O(n). Em seguida, ela cria um arquivo de saída e escreve as mensagens nele. O procedimento de imprimir mensagens, chamado imprimePilha(), percorre toda a pilha de mensagem do usuário, sendo O(n). Logo, a função pode ser classificada como **$O(n^2)$.**

Função verAmigos(): Esta função retorna a lista de amigos de um usuário. Esse procedimento pode ser classificado como **O(n),** já que primeiro é necessário identificar o usuário na lista de usuários.

4. Testes

Foram feitos testes utilizando os casos de teste disponibilizados no moodle.

Entrada 01:

```
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;Jogao do Atletico e Cruzeiro esse domingo
6;1;300;5003;Terminando o trabalho de AEDS2!!
7;1;300;5004;Viva compilou!!
8;4;100;5002
9;1;300;5005;To de folga agora!!
10;5;400
```

Saída 01:

```
300 Manu
5000 Assistindo O exorcista!! 0
400 Lucas
5005 To de folga agora!! 0
5002 Jogao do Atletico e Cruzeiro esse domingo 1
5004 Viva compilou!! 0
5003 Terminando o trabalho de AEDS2!! 0
```

O mesmo teste foi executado para as entradas 02 e 03.

5. Conclusão

O TP conseguiu ser finalizado e executa todas as funções colocadas na especificação do trabalho. O maior problema encontrado por mim foi escolher uma implementação

adequada para o tipo de problema. Percebi que o TP poderia ser desenvolvido de várias formas diferentes, cada uma com alguns pós e contras.

Algumas funções inicialmente feitas por mim precisaram ser mudadas várias vezes de acordo com as dúvidas que iam surgindo durante a codificação.

6. Referências

Na implementação das listas, utilizei como base a implementação vista nesse vídeo:

Felipe Alves. Estrutura de Dados. Disponivel em: <https://www.youtube.com/watch?v=Fc83QKzs9xU&list=PL1EkVGo1AQ0HgYpSwWwt_7JnavxRmCwt0&index=1>. Acesso em: 15 out 2016

7. Anexos

Listagem dos programas:

- ✓ *main.c*
- ✓ *usuários.h*
- ✓ *usuários.c*
- ✓ *amigos.h*
- ✓ *amigos.c*
- ✓ *mensagens.h*
- ✓ *mensagens.c*