Documentação Rede Social
Guilherme Henrique Resende de Andrade

Dados Necessários à Compilação:

```
gcc -w Main.c -o tp List.o Usuario.o Timeline.o
gcc -g -c List/List.c
gcc -g -c Timeline/Timeline.c
gcc -g -c Usuario.c
```

1- Introdução:

Na atualidade é comum o uso de redes sociais, que têm por objetivo conectar pessoas e fazê-las interagir no mundo virtual. Normalmente, esses meios de comunicação são utilizados e atualizados em tempo real, contudo, neste trabalho faremos com que uma rede social execute de forma automática.

O objetivo deste trabalho é fazer com que a partir de uma base de arquivos, o programa leia os dados, interprete as ações à serem feitas e gere um histórico das ações dos usuários, imprimindo esse histórico em um arquivo de saída.

2- Implementação:

Estrutura de Dados:

Para a implementação deste trabalho foram criados os TADs **Usuário, Timeline e List.**

Usuário:

```
typedef struct{
    int id
    char *nome
    t_list *seguidores
    t_timeline *timeline
}t_usuario;
```

A variável **id** é utilizada para salvar o identificador pessoal de cada usuário, já **nome** é utilizado para salvar o nome do usuário, e as variáveis **seguidores** e **timeline** são responsáveis por salvar, respectivamente, a lista de seguidores e de postagens de um dado usuário.

Timeline:

```
typedef struct{
    t_nodo *first
    t_nodo *last
    int list_size
}t_timeline;
```

```
typedef struct nodo{
        struct nodo *previous
        t_msg *msg
        struct nodo *next
}t_nodo;

typedef struct{
        int message_id, num_likes, moment
        char message[140]
}t_msg;
```

Na estrutura **t_msg** a variável **message_id** é responsável para guardar o identificador único da mensagem, **num_likes** é responsável por salvar o número de curtidas de determinada postagem, **moment** guarda o instante em que a mensagem sofreu alterações e **message** salva o conteúdo da postagem.

Na estrutura **t_nodo** a variável **msg** armazena a estrutura de determinada mensagem, enquanto as variáveis **previous** e **next** salvam, respectivamente, quais serão as células antecessoras e sucessoras.

Na estrutura **t_timeline** a variável **list_size** armazena o tamanho da timeline e as variáveis **first** e **last** são responsáveis por salvar a primeira e a última postagem da timeline.

List:

```
typedef struct{
        t_cell *first
        t_cell *next
}t_list;

typedef struct cell{
        int seguidor
        struct cell *next
}t_cell;
```

Na estrutura **t_cell** a variável **seguidor** guarda o ID do seguidor de determinado usuário e a variável **next** armazena o endereço para a próxima célula da lista.

Funções e Procedimentos:

Este trabalho utilizou diversas funções padrões de manipulação de Listas Simplesmente Encadeadas e Listas Duplamente Encadeadas que estão descritas nos arquivos **List.h** e **Timeline.h** e serão omitidas nas especificações abaixo**.**

t_usuario *aloca_usuarios(int numero_de_jogadores): Essa função recebe o número de usuários da rede social como parâmetro, aloca dinamicamente um espaço na memória e, posteriormente, retorna um ponteiro para o tipo **t_usuario.**

void inicia_usuarios(FILE *in, t_usuario *usuario): Esse método recebe um ponteiro para um fluxo de entrada de dados e um ponteiro para o tipo **t_usuario.** Posteriormente, inicia todas as variáveis da estrutura do usuário com os valores lidos no arquivo de entrada.

void cancelar_amizade(t_usuario *vetor_de_usuarios, int numero_de_usuarios, int id_usuario1, int id_usuario2): Esse método recebe um vetor do tipo **t_usuario**, uma variável que representa o número de usuários, e duas variáveis que representam os usuários que não serão mais amigos. Com isso, percorre o vetor de usuários procurando os ID's que devem ser apagados da lista de seguidores dos usuários e os excluem.

void iniciar_amizade(t_usuario *vetor_de_usuarios, int numero_de_usuarios, int id_usuario1, int id_usuario2): Esse método recebe um vetor do tipo **t_usuario**, e três variáveis inteiras numero_de_usuarios, id_usuario1, id_usuario2. Desta forma, percorre o vetor de usuários procurando pelos ID's das pessoas que se tornarão amigas, quando encontrados, a função **add** do TAD List os adicionam como seguidores mútuos.

void ver_amigos(t_usuario usuario): O método recebe uma variável do tipo **t_usuario** e exibe todos os seus seguidores.

void postar_mensagem(t_usuario *vetor_de_usuarios, int numero_de_usuarios, int id_usuario, int id_mensagem, char *mensagem, int instante): O método recebe um vetor de usuários, o número de usuários, o ID da pessoa que postou a mensagem, o ID da mensagem postada, o texto da mensagem e o instante em que a mensagem foi postada. A partir disso, é criada uma nova mensagem que tem seus campos preenchidos com os parâmetros da função e posteriormente é adicionada à timeline dos seguidores da pessoa que postou a mensagem.

void curtir_mensagem(t_usuario *vetor_de_usuarios, int numero_de_usuarios, int id_usuario, int id_mensagem, int instante): Esse método recebe um vetor de usuários, uma variável que representa o número de usuários, o ID de um usuário, o ID de uma mensagem e o instante que a ação é feita. Posteriormente, percorre o vetor de usuários atrás dos

usuários que possuem a mensagem em sua timeline, com isso, a variável **num_likes** é incrementada e a mensagem é movida para a primeira posição da lista.

void exibir_timeline(FILE *out, t_usuario usuario): Esse método recebe um ponteiro para um fluxo de saída de dados e uma variável do tipo **t_usuario**. Feito isso, todas as mensagem não visualizadas da timeline do usuário são transcritas no arquivo de saída e marcadas como visualizadas.

void exe(FILE *in, FILE *out, t_usuario *vetor_de_usuarios, int numero_de_usuarios): Este método é responsável pela execução de todas as ações da rede social, ele recebe dois ponteiros para fluxos de arquivos, um vetor de usuários e o número de usuários da rede. Com isso, inicia um loop de leitura, atribui valor à variável **acao_usuario** e realiza um switch que informa quais são as próximas ações a serem executadas.

t_msg* get_item(t_timeline *list, int item): Essa função recebe um ponteiro para o tipo **t_timeline**, um ID de uma mensagem e posteriormente retorna o endereço da célula de mensagem solicitada.

void set_first(t_timeline *list, int message_id): Esse método recebe um ponteiro para **t_timeline** e um ID de uma determinada mensagem. Assim sendo, ele retira a mensagem da posição atual e a coloca no primeiro lugar da lista.

Programa Principal:

O programa principal declara uma variável do tipo INT que será responsável pelo índice dos arquivos de teste. Posteriormente, são criadas strings formatadas que serão utilizadas na abertura do fluxo de dados, após isso, o código cria um loop que lê o número de jogadores, chama a função **exe()** - Responsável por executar todas as ações - e fecha o fluxo de dados. Por último, o programa libera toda a memória alocada dinamicamente.

3 - Complexidade:

Todas as ordens de complexidade dos procedimentos abaixo foram calculados com relação à um dado número **n** de mensagens.

Função Aloca Usuarios: Essa função não possui relação com o número de mensagens da rede social, portanto, tem ordem de complexidade **O(1).**

Método Iniciar Usuarios: Esse método não possui relação com o número de mensagens da rede social, portanto, tem ordem de complexidade **O(1).**

Método Cancelar Amizade: Esse método não possui relação com o número de mensagens da rede social, portanto, tem ordem de complexidade **O(1).**

Método Iniciar Amizade: Esse método não possui relação com o número de mensagens da rede social, portanto, tem ordem de complexidade **O(1).**

Método Ver Amigos: Esse método não possui relação com o número de mensagens da rede social, portanto, tem ordem de complexidade **O(1).**

Método Postar Mensagem: Esse procedimento é relacionado às mensagens da rede social, contudo, realiza sua ação com um custo também de **O(1).**

Método Curtir Mensagem: Assuma **K** o número de usuários da rede, e **N** o número de mensagens de uma determinada timeline. Esse procedimento percorre todos os membros de uma timeline a **quantidade de usuários** vezes, o que é $\theta(n) = Kn$**.** Contudo, tendo que K é sempre uma constante em determinada execução do programa, podemos dizer a ordem de complexidade é **O(n).**

Método Exibir Timeline: Esse método recebe um usuário arbitrário e percorre todas as mensagens de sua respectiva timeline. Dessa forma, a ordem de complexidade do procedimento é **O(n).**

Método Exe: Esse procedimento realiza a leitura de todos os dados do arquivo de entrada e, com isso, toma a decisão - a partir de um **Switch** - de quais serão as próximas ações a serem tomadas. Dessa forma, tendo que todas as mensagens são lidas do arquivo, e que a escolha das próximas ações à serem executadas é arbitrária, pode-se dizer que a função de complexidade desse procedimento é $n + \sum_{i=0}^{a} \theta(k)$ onde **n** é o número de mensagens somado ao somatório das funções executadas posteriormente, com $1 \leq K \leq n$. Contudo, como só existem funções na ordem de **O(n)**, pode-se dizer que a ordem de complexidade deste método é também **O(n).**

Função Get Item: Essa função percorre todas as mensagens de uma dada timeline, portanto, possui ordem de complexidade **O(n).**

Método Set First: Esse método faz a chamada de outros dois procedimentos, **set_first()** e **add_begin()**. Desta forma, realizamos a soma

da ordem de complexidade dos dois para atribuirmos a complexidade deste método. Assim sendo, $O(n) + O(1) = O(n)$.

4 - Testes:

Alguns testes foram realizados de forma a comprovar o funcionamento do programa. Os testes foram feitos em uma máquina com estrutura Intel Core I5, 2GB de memória RAM e configurada com sistema operacional Linux.

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

```

(Saída do primeiro arquivo de teste)

```
100 AnaMaria
6002 Eu lembro. Repito p que vc ai se LEMBRE SEMPRE TAMBÉM,sabe como é…logo teremos eleições 0
6000 Freixo se saindo muito bem no debate! Lembrando que pesquisas não decidem eleições!  1
6001 O PT não saiu derrotado nessas eleições, o NINGUÉM venceu nas urnas. 0
7001 Gente, quero ver um filme com meu namorado. Vocês poderiam me indicar um namorado? 0
400 Anderson
6003 Depois das eleições, prefeito manda recolher material de campanha. 0
8003 Acho que vou colocar nome em um Pokémon bem forte e grande "papel de trouxa" e dizer! Papel de trouxa eu escolho você! #PokemonGO 1
6002 Eu lembro. Repito p que vc ai se LEMBRE SEMPRE TAMBÉM,sabe como é…logo teremos eleições 0
6000 Freixo se saindo muito bem no debate! Lembrando que pesquisas não decidem eleições!  1
5000 Neymar chegou a marca de 300 gols na carreira. Apenas 24 anos! 0
6001 O PT não saiu derrotado nessas eleições, o NINGUÉM venceu nas urnas. 0
500 Simone
5001 Uma descontraída antes de começar a final! Nossos craques levam jeito no futebol? 0
6003 Depois das eleições, prefeito manda recolher material de campanha. 0
8003 Acho que vou colocar nome em um Pokémon bem forte e grande "papel de trouxa" e dizer! Papel de trouxa eu escolho você! #PokemonGO 1
6002 Eu lembro. Repito p que vc ai se LEMBRE SEMPRE TAMBÉM,sabe como é…logo teremos eleições 0
6000 Freixo se saindo muito bem no debate! Lembrando que pesquisas não decidem eleições!  1
5000 Neymar chegou a marca de 300 gols na carreira. Apenas 24 anos! 0
6001 O PT não saiu derrotado nessas eleições, o NINGUÉM venceu nas urnas. 0
200 HannahM
7001 Gente, quero ver um filme com meu namorado. Vocês poderiam me indicar um namorado? 1
7002 Aquele momento q vc começa a ver um filme e fica mexendo no celular e esquece de assistir o filme e a tv fica ligada sem ngm
assistindo 0
300 Guedes
```

(Saída do segundo arquivo de teste. A imagem foi cortada para melhorar a visualização do fluxo de saída)

5- Conclusões:

A implementação do trabalho transcorreu sem maiores problemas e os resultados ficaram dentro do esperado.

A principal dificuldade encontrada foi dispor as mensagens de uma forma dinâmica que não fosse alterada quando uma mensagem fosse modificada em uma timeline específica.

Referências:

http://stackoverflow.com/

https://www.tutorialspoint.com/

http://www.cprogressivo.net/