Trabalho Pratico II – AEDS II

Servidor de mensagens instantaneas

Aluno: Luiz Otavio Resende Vasconcelos
Matricula: 2015042142

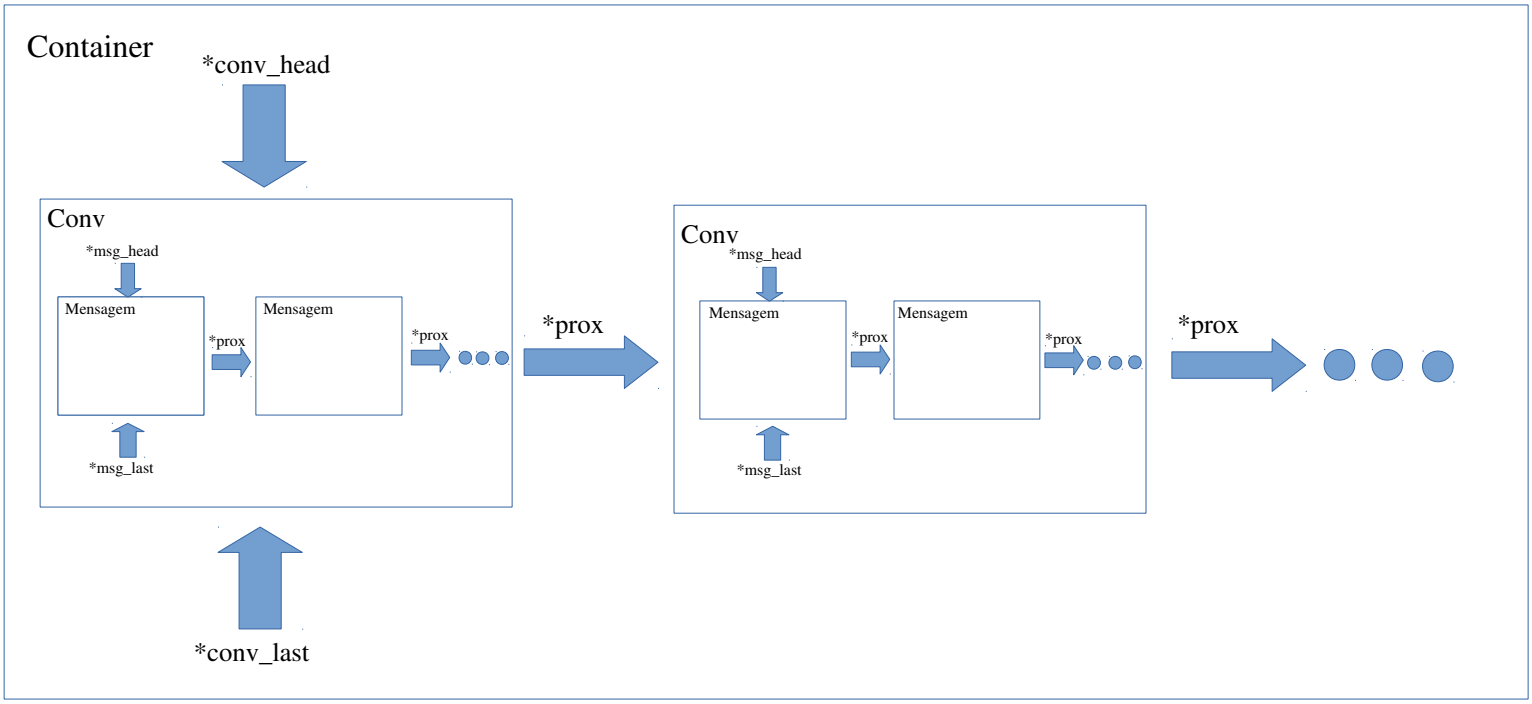
1. Introducao

O presente trabalho visa implementar o conteudo visto em sala de aula referente as estruturas abstratas de dados ligadas por uma referencia, o qual, chamamos de Listas Encadeadas.

Fora solicitado o desenvolvimento de um “servidor de chat”, o qual recebemos lotes de mensagens, armazenamos e enviamos-as para os destinatarios seguindo uma ordem de prioridades.

2. Implementacao

Utilizei a seguinte estrutura de dados para implementar o trabalho pratico:



TADS

Fora utilizado um TAD chamado **Container** para armazenar as **Conv** que armazenam as **Mensagens**.
As Mensagens sao agrupadas pelo seu “pair_id” dentro da Conv com o mesmo “pair_id”.
As Conversas(Conv) sao criadas no Container ja de forma ordenada crescente (pelo “pair_id”) e as Mensagens dentro de cada Conversa tambem ja ordenadas pela sua ordem “ord”.
Ou seja, muitas vezes se faz necessaria a insercao no meio da lista.

// **Container**:
// estrutura responsavel por fornecer a referencia
// para as Conversas

```
typedef struct cont_st {
   Conv *conv_head, *conv_last;          → ponteiros para as conversas
} Container;
```

// **Conv**:
// contem dados agrupados de cada conversa
// e fornece os ponteiros para as mensagens

```
typedef struct conv_st {
   int pair_id;                          → id do par
   int last_msg;                         → quantidade de mensagens trocadas (ou ultima mens +1)
   int k;                                → quantidade de lotes sem trocar mensagens da Conv
   Message *msg_head, *msg_last;         → ponteiros para a lista de mensagens
   struct conv_st *prox;                 → ponteiro para a proxima Conv
} Conv;
```

// **Message**:
// estrutura responsavel por armazenar as mensagens

```
typedef struct message_st {
   char message[MAX];                    → armazena a String da mensagem
   int pair_id;                          → id do par (o mesmo da conversa)
   int order;                            → ordem da mensagem
   struct message_st *prox;              → ponteiro para a proxima mensagem
} Message;
```

Passo a passo do codigo:

Primeiro e declarado um ponteiro para um Container chamado “root”.
e entao alocada memoria para ele e feito as atribuicı es necessarias para uma lista encadeada.
Comecamos a leitura:
Lemos o valor de ’k’ e ’lote_number’ no *stdin* e entao comecamos a leitura das mensagens.

Loop de leitura/insercao de mensagens:

O TAD Conv possui um ’int k’ inicializado em 0 assim que alocada memoria para o TAD.
Antes de ler um lote, chamemos a funcao

```
void addKonConvs(Container* root);
```

que ira adicionar +1 em todos os valores de k de todas as Conv.
Ao final da leitura de cada lote, essa funcao e chamada novamente.

Chamemos entao a funcao

```
void* selectConv(Container* root, int pair_id);
```

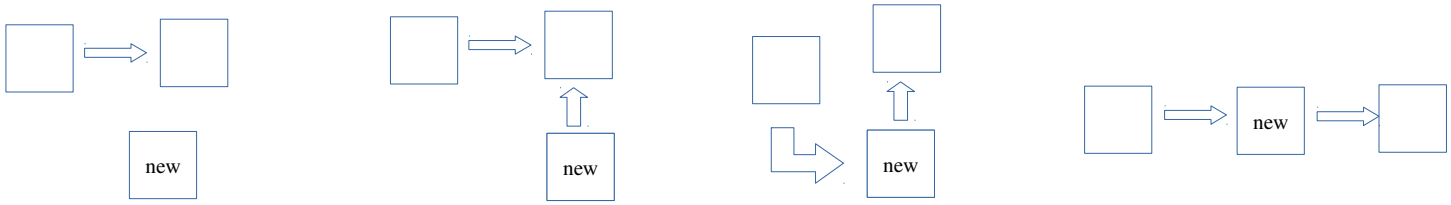
que, a partir do ’pair_id’, me retorna a Conv do par. Essa funcao percorrera o container atras da Conv com o ’pair_id’ da mensagem:
Caso ela encontre a encontre, a retorna.
Caso encontre uma Conv a qual a Conv→prox tem ’pair_id’ maior que o informado, ele chama a funcao

```
Conv* createConvMid(Conv* conv, int pair_id);
```

como sabemos que o aux→prox tem ’pair_id’ maior que o informado e o aux tem o ’pair_id’ menor, devemos inserir nossa nova conversa no meio das duas.

Entao a funcao acima cria uma ’Conv* new’ e entao:

```
new->prox = aux->prox;
aux->prox = new;
```



ou seja, inserimos no meio da lista a nova Conv.
Caso nao encontre uma maior, ela deve ficar no final da lista. Entao e alocado o espaco e atribuido o ponteiro de last para ela.

Retornamos entao a Conv selecionada.
Atribuo entao o retorno da mesma a um auxiliar 'conv' e chamo outra funcao:

```
void insOnConv(Conv *conv, int ord, char* msg);
```

Essa funcao ira inserir a 'msg' na conversa, de forma ordenada.
Ela tambem seta o 'k' = 0 da Conv. Pois, fora trocado mensagem na conversa.
Da mesma forma: percorremos a Conv atras de uma Message->prox com a ordem 'ord' maior que a nova.
Assim sabemos que devemos inserir a nova Message entre a atual e a Message→prox.
Chamamos o metodo

```
void insOnConvMid(Message* msg, int ord, char* message, int pair_id) ;
```

que fara o analogo ao da Conv.

Apos fazermos isso com todas as mensagens recebidas e recebermos um "Fim" estamos prontos para enviar as mensagens.

Envio das mensagens:

A primeira funcao que e chamada e:

```
void desalocateConv(Container* root, int k);
```

que ira percorrer todo container atras de uma conversa com o 'k' igual ao k informado.

Caso encontre, faz as manipulacı es de ponteiros e desaloca a conversa.

Entao e chamada a funcao:

```
void printLists(Container* root);
```

que basicamente mostra na tela todas as mensagens dentro de cada conversa que estao armazenadas como fora solicitado na descricao do trabalho pratico (Listas:)

Agora e chamada a funcao:

```
void sendMessages(Container *root);
```

que sera responsavel por enviar cada mensagem (Envios:) na devida ordem de prioridade e desalocar as que forem sendo enviadas.

Basicamente, ela ira percorrer o Container atras do menor valor de 'last_msg' nas Conv.
Ele representa o numero de Messages trocadas entre aquele par.
Ele procurara um novo menor 'last_msg' toda vez que uma mensagem for enviada.
Como as Conv estao ordenadas por 'pair_id', agora com o menor 'last_msg', basta percorrer as conv a procura da primeira que tem o mesmo 'last_msg' do encontrado. Porem se faz necessario tambem verificar se o par tem Messages a serem enviadas. Deve-se satisfazer a condicao:

Conv->msg_head->prox->**order** == Conv->**last_msg** + **1**

E entao chamamos a funcao:

```
void printCounts(Container* root);
```

que simplesmente ira imprimir a quantidade de mens trocadas por cada par,
no caso, cada Conv→last_msg

Lemos agora novamente o *stdin*.
Caso um novo lote seja inserido, repete-se o processo, senao, termina-o.

3 . Analise de complexidade

Funcao:

main() → temos varios O(1) e O(n†) – dois while's = O(n†)
createConvMid() → O(1) – apenas atribuicı es
selectConv() → atribuicı es O(1) e um loop while = O(n)
insOnConvMid() → apenas atribuicı es O(1)
insOnConv() → atribuicoes e um loop while com a funcao insOnConvMid() dentro que possui apenas atribuicı es, portanto = O(1)

getMinMens() → atribuicı es e um loop = O(n)
sendMessages() → atribuicı es e um loop com a funcao getMinMens() dentro com outro loop, logo = O(n†)
printLists() → atribuicoes e dois loops = O(n†)
printCounts() → apenas um loop = O(n)
addKonConvs() → apenas um loop = O(n)
desalocateConv() → apenas um loop = O(n)

4 . Conclusao

Acredito que a escolha do tema do trabalho fora excelente.
Pude abranger bem o conceito de Listas Encadeadas que e um conceito fundamental para a Ciencia da Computacao. Ao fazer o trabalho, tirei todas as duvidas a respeito e posso afirmar que o conteudo foi bem absorvido.
Apesar de ser uma aplicacao pratica que talvez possa morrer, pois, linguagens com ponteiros estao cada vez menos populares, e uma logica/algoritmo extremamente elegante e de suma import ncia o conhecimento para o cientista da computacao.

5 . Referencias

→ **Projetos de Algoritmos** – com implementacı es em Java e C++
autor: Nivio Ziviani

→ **Listas Ligadas (wikipedia)** - https://pt.wikipedia.org/wiki/Lista_ligada

→ **Slides** - fornecidos pelo portal *minhaUFMG*

Falta de acentuacao em toda documentacao devido a problemas de encode