Servidor de mensagens instantaneas

Aluno: Luiz Otávio Resende Vasconcelos
Matrícula: 2015042142

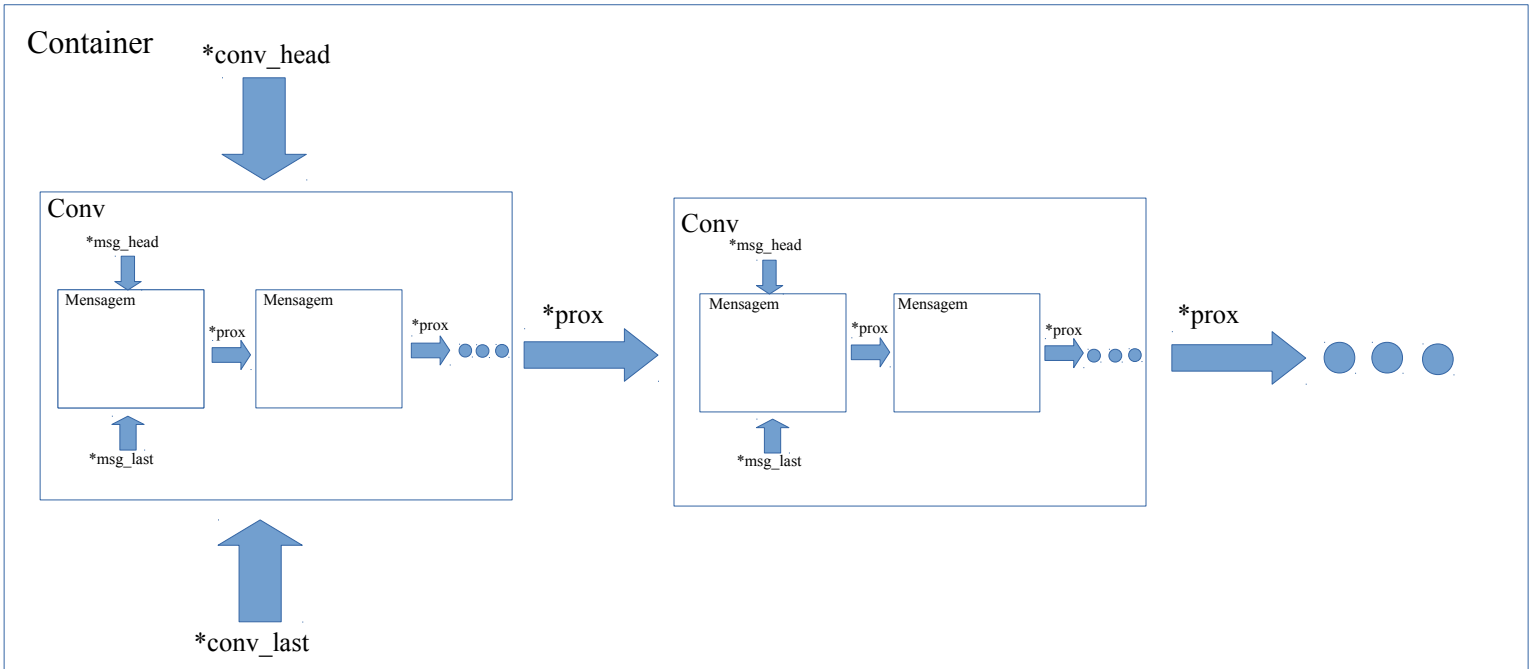
1. Introdução

O presente trabalho visa implementar o conteúdo visto em sala de aula referente às estruturas abstratas de dados ligadas por uma referência, 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 destinatários seguindo uma ordem de prioridades.

2. Implementação

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



As Mensagens são agrupadas pelo seu "pair_id" dentro da Conv com o mesmo "pair_id".
As Conversas são criadas no Container já de forma ordenada crescente (pelo "pair_id") e as Mensagens dentro de cada Conversa também já ordenadas pela sua ordem "ord". Ou seja, muitas vezes se faz

necessária a inserção no meio da lista.

Passo a passo do código:

Primeiro é declarado um ponteiro para um Container chamado “root”.
É então alocada memória para ele e feito as atribuições necessárias para uma lista encadeada.
Começamos a leitura:
Lemos o valor de 'k' e 'lote_number' no *stdin* e então começamos a leitura das mensagens.

Loop de leitura/inserção de mensagens:

O TAD Conv possui um 'int k' inicializado em 0 assim que alocada memória para o TAD.
Antes de ler um lote, chamemos a função

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

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

Chamo então a função

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

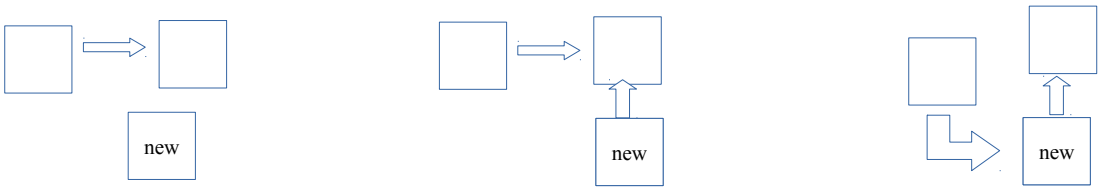
que, a partir do 'pair_id', me retorna a Conv do par. Essa função percorrerá o container atrás da Conv com o 'pair_id' da mensagem:
Caso ela encontre a encontre, a retorna.
Caso encontre uma Conv a qual a Conv→prox tem 'pair_id' maior que o informado, ele chama a função

```
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.

Então a função acima cria uma 'Conv* new' e então:

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



ou seja, inserimos no meio da lista a nova Conv.
Caso não encontre uma maior, ela deve ficar no final da lista. Então é alocado o espaço e atribuído o ponteiro de last para ela.

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

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

Essa função irá inserir a 'msg' na conversa, porém, de forma ordenada também.
Ela também seta o 'k' = 0 da Conv. Pois, fora trocado mensagem na conversa.
Da mesma forma: percorremos a Conv atrás 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 método

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

que fará o análogo ao da Conv.

Após fazermos isso com todas as mensagens recebidas e recebermos um “Fim” estamos prontos para enviar as mensagens.
A primeira função que é chamada é:

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

que irá percorrer todo container atrás de uma conversa com o 'k' igual ao k informado.

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

Então é chamada a função:

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

que basicamente mostra na tela todas as mensagens dentro de cada conversa que estão armazenadas como fora solicitado na descrição do trabalho prático (Listas:)

Agora é chamada a função:

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

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

Basicamente, ela ira percorrer o Container atrás do menor valor de 'last_msg' nas Conv.
Ele representa o número de Messages trocadas entre aquele par.
Ele procurará um novo menor 'last_msg' toda vez que uma mensagem for enviada.
Como as Conv estão 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. Porém se faz necessário também verificar se o par tem Messages a serem enviadas. Deve-se satisfazer a condição:

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

E então chamamos a função:

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

que simplesmente irá 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, senão, termina-o.

3 . Análise de complexidade

Funçao:

main() → temos vários O(1) e O(n²) – dois while's = $O(n^2)$
createConvMid() → $O(1)$ – apenas atribuições
selectConv() → atribuições O(1) e um loop while = $O(n)$
insOnConvMid() → apenas atribuições $O(1)$
insOnConv() → atribuiçoes e um loop while com a função insOnConvMid() dentro que possui apenas atribuições, portanto = $O(1)$
getMinMens() → atribuições e um loop = $O(n)$
sendMessages() → atribuições e um loop com a função getMinMens() dentro com outro loop, logo = $O(n^2)$
printLists() → atribuiçoes e dois loops = $O(n^2)$
printCounts() → apenas um loop = $O(n)$
addKonConvs() → apenas um loop = $O(n)$
desalocateConv() → apenas um loop = $O(n)$

4 . Conclusão

Acredito que a escolha do tema do trabalho fora excelente.
Pude abranger bem o conceito de Listas Encadeadas que é um conceito fundamental para a Ciência da Computação. Ao fazer o trabalho, tirei todas as dúvidas a respeito e posso afirmar que o conteúdo foi bem absorvido.

Apesar de ser uma aplicação prática que talvez possa morrer, pois, linguagens com ponteiros estão cada vez menos populares, é uma lógica/algorítmo extremamente elegante e de suma importância o conhecimento para o cientista da computação.