UNIVERSIDADE FEDERAL DO CEARÁ

FERNANDA ALBUQUERQUE DE ABREU

BUS TOTEM:
SISTEMA DE INFORMAÇÃO DE LINHAS DE ÔNIBUS

QUIXADÁ

2023

RESUMO

“Terminal de informações em uma rodoviária onde um passageiro potencial pode se informar, se existem linhas de ônibus que passam por este terminal ou que partem dele, que vão para uma cidade onde ele quer ir.”

Palavras-chave: passagens; ônibus; horários.

SUMÁRIO

1 INTRODUÇÃO

O sistema de informação de linhas de ônibus foi um trabalho do qual entender a finalidade foi tranquilo, já que fazemos muito para viajarmos ao nosso destino de origem (quem não é de Quixadá, obviamente). A minha maior dificuldade, foi realmente entender como todo o sistema ia ser implementado, já que o arquivo não estava compreensível a ponto de ficar claro todos os pontos a serem implementados, o trabalho do semestre passado estava mais claro, até porque a maioria foi visto em sala e isto facilita a compreensão. Portanto, foi difícil separar cada componente, classe, o todo.

A primeira entrega estava bem básica, mas estava no caminho. Nela não estavam todas as partes implementadas e nem todo o necessário como a descrição do trabalho propõe. Já nesta entrega tem mais implementações feitas, e o sistema basicamente funciona. Como falei no relatório anterior, foi desafiador o todo, entender, fazer sozinha, ter que modificar ou encontrar erros e senti muita dificuldade.

Na primeira versão, basicamente, entreguei somente um Nó e uma classe para uma fila duplamente encadeada, onde estavam bem parecidas com os códigos que aprendemos em sala de aula. Inclusive, fui fazendo e acabei utilizando nomes em inglês. Porém, quando o trabalho foi ficando cada vez maior, comecei a confundir tudo. Como não sou fluente em inglês, resolvi modificar todos os nomes para português, a fim de ficar mais claro e parar de me confundir. A partir de agora tudo está em português.

Além disso, o projeto não está completo no que foi pedido para fazer. O meu programa NÃO lê arquivos. E o mais importante, o meu programa não faz a procura de uma linha de ônibus - precisaria para conseguir finalizar mais uma semana no mínimo. Apesar de não fazer essas duas coisas, faz as demais.

Outro ponto a ser dito aqui, é que eu em determinado ponto do trabalho pedi ajuda a um amigo que já fez a disciplina, não sabia como fazer a “lista_encadeada.h” e “main.cpp” funcionar, então pedi ajuda.

A seguir mostrarei tudo que foi implementado.

2 IMPLEMENTAÇÕES DOS NÓs

Vou mostrar e explicar a estrutura do código dos NÓs.

2.1 NOs

2.1.1 NO_DUPLAMENTE_ENCADEAVEL

Características das paradas das linhas de ônibus: Alterei muitas coisas aqui, na primeira versão estava usando template. E na main.cpp, criei uma struct onde recebia os valores como: nome, horários de saída e de chegada. Atualmente essas variáveis estão implementadas no próprio Nó, não utilizando mais o uso de template. E o nó ainda tendo seus ponteiros para próximo e anterior. A classe contém um construtor para a criação de um novo Nó e um destrutor, apenas. Como na primeira versão, contendo apenas a diferença descrita acima.

2.1.2 NO_ENCADEAVEL

Essa parte não estava pronta na primeira entrega.

Essa lista possui as características das linhas de ônibus, que são: número da linha, que é ùnico para todo país e o nome da companhia, além de um ponteiro para o próximo nó. Essa diferentemente da anterior, antes de criar seu construtor e destrutor, é criada uma nova lista duplamente encadeada que conterá as suas respectivas paradas, realizando assim o ciclo. Já que as paradas ficam dentro das linhas de ônibus. Dessa maneira, quando vamos fazer o destrutor desse nó, apagamos as paradas antes de apagar a linha de ônibus. Até porque não seria correto apagar a linha, e deixar as paradas de ônibus como lixo flutuante, logo temos essa nova lista de ônibus criada (explicarei essa parada de ônibus abaixo).

3 IMPLEMENTAÇÃO DAS LISTAS

Vou mostrar e explicar a estrutura do código parte por parte das listas.

3.1 LISTAS

3.1.1 LISTA_DUPLAMENTE_ENCADEADA_CIRCULAR

Apesar desta antes se chamar “BusSistem.h”, não mudou muito.

Primeiro que depois de ter feito corretamente o “no_duplamente_encadeavel”. O ”template <typename Type>” não foi mais necessário, como explicado anteriormente. Estou usando string criadas no próprio Nó. A classe tem um nó sentinela e uma variável que indica o tamanho. Um construtor default, que até então é parecido com a primeira versão. A partir de agora não será, porque será implementada com as características do nó.

Agora vou explicar algumas funções, as mais difíceis de fazer e entender:

3.1.1.1 AdicionaParadaAposIndice

```
29      void AdicionaParadaAposIndice(unsigned indice_anterior, NoDuplamenteEncadeavel *nova_parada) {
30          if (indice_anterior > this->tamanho_) {
31              cout << "Indice de parada fora do intervalo de paradas." << endl;
32              return;
33          }
34
35          NoDuplamenteEncadeavel *no_iterador = this->no_sentinela_;
36          unsigned counter = 1;
37          while (counter < indice_anterior) {
38              no_iterador = no_iterador->proximo_no;
39              counter++;
40          }
41
42          nova_parada->no_anterior = no_iterador;
43          nova_parada->proximo_no = no_iterador->proximo_no;
44          no_iterador->proximo_no->no_anterior = nova_parada;
45          no_iterador->proximo_no = nova_parada;
46          this->tamanho_++;
47      }
```

Começamos verificando se o índice não é maior que o tamanho da lista (linha 30-33), caso aconteça, gera a informação, da qual não conseguiremos criar uma nova parada.

Caso não seja maior, criamos um novo nó que recebe o sentinela. O sentinela é como se fosse o primeiro - no caso ele aguarda o primeiro a chegar. Imagine comigo, suponha que você seja um operador de caixa esperando o primeiro cliente chegar para passar suas compras. Você não será o primeiro cliente, mas a responsabilidade de abrir o caixa é sua. Portanto, você se torna um ponteiro para o primeiro, mas o primeiro, será o cliente que primeiro passar as compras.
Espero que essa analogia possa ajudar a entender.
Após criar um contador começando em 1. Dado que esse contador fosse menor que o índice, o nó que criamos, recebe ele mesmo apontando para o próximo, e incrementamos o nosso contador (linhas 35-39). Nossa nova parada na posição anterior recebe esse novo nó, a nova parada na sua próxima posição receberá o novo nó na próxima posição dele, já que eles estão em uma nova posição agora. Aloca a nova parada para sua posição final de acordo com o índice passado, e incrementamos o tamanho (linhas 42-46).

3.1.1.2 AlterarParadaPeloIndice

```
56      void AlteraParadaPeloIndice(unsigned indice, string nome, string chegada, string partida) {
57          if (indice < 1 || indice > this->tamanho_) {
58              cout << "Indice de parada fora do intervalo de paradas." << endl;
59              return;
60          }
61
62          NoDuplamenteEncadeavel *no_iterador = this->no_sentinela_;
63          unsigned counter = 1;
64          while (counter < indice) {
65              no_iterador = no_iterador->proximo_no;
66              counter++;
67          }
68
69          no_iterador->nome_da_parada = nome;
70          no_iterador->horario_de_chegada = chegada;
71          no_iterador->horario_de_partida = partida;
72          cout << "Parada alterada para " << nome << " ("
73              << chegada << " - " << partida << ")." << endl;
74      }
```

Esta função mudou, porque não havia índices antes.
Fora que, nessa função precisasse de mais variáveis para fazer a mudança de uma parada.
Realizando primeiramente uma verificação de parada, se o índice escolhido for maior que o tamanho ou menor que um. O programa para nesse momento e informa a não possibilidade de prosseguir (linhas 57-60).
Se não cair nesse caso, cria um novo nó e um novo contador que começa em um, como anteriormente. Enquanto o contador for menor que o índice informado ele vai andando para o próximo e próximo nó, até que o contador seja maior que o índice (linhas 62-66).

Quando isto acontece, o nó que aponta para o nome da parada, para o horário de chegada e horário de partida, recebe os novos valores e atualiza a edição dessa parada (linhas 69-73).

3.1.1.3 RemoverParadaPeloIndice

```
108     void RemoveParadaPeloIndice(unsigned indice) {
109         if (indice < 1 || indice > this->tamanho_) {
110             cout << "Indice de parada fora do intervalo de paradas." << endl;
111             return;
112         }
113 
114         NoDuplamenteEncadeavel *no_iterador = this->no_sentinela_;
115         unsigned counter = 1;
116         while (counter < indice) {
117             no_iterador = no_iterador->proximo_no;
118             counter++;
119         }
120 
121         no_iterador->proximo_no->no_anterior = no_iterador->no_anterior;
122         no_iterador->no_anterior->proximo_no = no_iterador->proximo_no;
123         this->no_sentinela_ = no_iterador->proximo_no;
124 
125         if (this->tamanho_ == 1) {
126             this->no_sentinela_ = nullptr;
127         }
128 
129         delete no_iterador;
130         this->tamanho_--;
131     }
```

Esta função mudou, porque não havia índices antes.

Executando aquela mesma verificação para sabermos se existe o índice escolhido (linhas 109-111). Se existir, criamos um novo nó recebendo o nó sentinela, aquele que já expliquei anteriormente, e um contador (linhas 114-118).

E, nesse momento, começa a alocação de nós para deletar o escolhido, depois de feita a alocação, o nó é deletado.

Mas antes verificasse. Porque, se o tamanho for igual a um, só quem estará na lista, será o nó sentinela.

Então atribuímos o nullptr a ele, e depois de ter deletado o nó, decrementa a lista (linhas 121-130).

3.1.2 LISTA_ENCADEADA

No código da primeira entrega estava inutilizada, foi uma das partes mais difíceis e vou explicar algumas funções dela aqui, as mais complicadas:

3.1.2.1 AdicionaNovaLinhaOrdenadamente

```
25      void AdicionaNovaLinhaOrdenadamente(unsigned numero, string companhia) {
26          if (this->EstaVazia()) {
27              cout << "Inserindo " << numero << " numa lista vazia." << endl;
28              NoEncadeavel *nova_linha = new NoEncadeavel(numero, companhia);
29              this->primeiro_no_ = nova_linha;
30              this->ultimo_no_ = nova_linha;
31              this->tamanho_++;
32              return;
33          }
34
35          if (numero < this->primeiro_no_->numero_da_linha) {
36              cout << "Inserindo " << numero << " no inicio da lista." << endl;
37              NoEncadeavel *nova_linha = new NoEncadeavel(numero, companhia, this->primeiro_no_);
38              this->primeiro_no_ = nova_linha;
39              this->tamanho_++;
40              return;
41          }
42
43          if (numero > this->ultimo_no_->numero_da_linha) {
44              cout << "Inserindo " << numero << " no final da lista." << endl;
45              NoEncadeavel *nova_linha = new NoEncadeavel(numero, companhia);
46              this->ultimo_no_->proximo_no = nova_linha;
47              this->ultimo_no_ = nova_linha;
48              this->tamanho_++;
49              return;
50          }
```

```
51
52      NoEncadeavel *no_iterador = this->primeiro_no_;
53      while (numero > no_iterador->proximo_no->numero_da_linha) {
54          cout << numero << " eh maior que " << no_iterador->proximo_no->numero_da_linha << endl;
55          no_iterador = no_iterador->proximo_no;
56      }
57
58      cout << numero << " eh menor ou igual a " << no_iterador->proximo_no->numero_da_linha << endl;
59      cout << "Inserindo " << numero << " entre " << no_iterador->numero_da_linha << " e " << no_iterador->proximo_no->numero_da_linha << endl
60
61      if (no_iterador->numero_da_linha == numero) {
62          cout << "Numero da linha invalido: os numeros das linhas de onibus devem ser unicos em todo o país." << endl;
63          return;
64      }
65
66      NoEncadeavel *nova_linha = new NoEncadeavel(numero, companhia, no_iterador->proximo_no);
67      no_iterador->proximo_no = nova_linha;
68      this->tamanho_++;
69  }
```

Como é para adicionar ordenadamente, precisasse de algumas verificações.

A primeira é se está vazia. Se sim, adicione (linhas 26-32).

A segunda é, que se o número passado é menor que o número do primeiro nó da lista. Se for, insiro agora o novo em primeiro lugar (linhas 35-40).

Para facilitar, verificasse se o número é maior que o último número já inserido. Se for, insiro ele como o último número da lista (linhas 43-49).

Se não cair em nenhum desses casos, vamos à procura de em qual lugar ele será inserido. Criamos um novo nó, que recebe o primeiro nó. E enquanto esse número for maior, que o número que está sendo procurado na lista, continue andando. Até que o número seja menor, efetuando a atribuição (linhas 52-59).

Se o número for igual a outro que está na lista, envio uma mensagem avisando que o número é invalido, pois já tem um igual e eles devem ser únicos (linhas 61-63).

Depois de todas as verificações, termino implementando a alocação da nova linha de ônibus na lista e aumentando o tamanho dela (linhas 66-68).

3.1.2.2 RemoverLinhaPeloNumero

```
96      void RemoveLinhaPeloNumero(unsigned numero_da_linha_para_remover) {
97          if (this->EstaVazia()) {
98              cout << "A lista está vazia. Não há nós para remover." << endl;
99              return;
100         }
101
102         if (this->primeiro_no_->numero_da_linha == numero_da_linha_para_remover) {
103             NoEncadeavel *no_para_remover = this->primeiro_no_;
104             this->primeiro_no_ = primeiro_no_->proximo_no;
105             delete no_para_remover;
106             this->tamanho_--;
107
108             if (this->EstaVazia()){
109                 this->ultimo_no_ = nullptr;
110             }
111             return;
112         }
113
114         NoEncadeavel *no_iterador = this->primeiro_no_;
115         while (no_iterador->proximo_no != nullptr &&
116         no_iterador->proximo_no->numero_da_linha != numero_da_linha_para_remover) {
117             no_iterador = no_iterador->proximo_no;
118         }
119
120         if (no_iterador->proximo_no == nullptr) {
121             cout << "A linha " << numero_da_linha_para_remover
122             << " não existe. Nenhuma linha foi removida." << endl;
123             return;
124         }
```

```
125
126         if (this->ultimo_no_->numero_da_linha == numero_da_linha_para_remover) {
127             this->ultimo_no_ = no_iterador;
128         }
129
130         NoEncadeavel *linha_a_ser_removida = no_iterador->proximo_no;
131         no_iterador->proximo_no = linha_a_ser_removida->proximo_no;
132         delete linha_a_ser_removida;
133         this->tamanho_--;
134     }
```

Aqui vamos precisar do número da linha para removê-lá.

Tendo esse número, vamos às verificações.

Primeiro se ela está vazia. Se sim, avisa e sai do programa. Se não, faço outra verificação.

Se o número a ser removido é o primeiro, se for, removo ele, decremento e o tamanho.

Se vazia, o último nó recebe nullptr (linhas 97-111).

Se não for nenhum desses casos, crio um novo nó, e enquanto meu próximo nó for diferente de nullptr e meu próximo nó que aponta para o número da linha, for diferente do número que preciso remover, eu continuo percorrendo. Quando essa condição não for mais verdadeira, eu saio do while, verifico se esse meu próximo nó é igual a nullptr, se for, eu retorno uma mensagem. Mas, se não for, eu verifico se o último nó que aponta pro número da linha é igual ao meu número a ser removido. Se for, meu último nó, recebe o nó criado anteriormente, que no caso é um nó auxiliar, que vai nos ajudar na manipulação para remover esse número. Faço a interação com a criação de um novo nó, para auxiliar na manipulação, e removo a linha que precisava ser removida, e decremento o tamanho da lista (linhas 114-133).

3.1.3 MAIN

A main está bem intuitiva. Ao executar, no próprio início dela tem a senha de manutenção, que é “senhafraca”, para a manutenção das linhas de ônibus e suas paradas, que no caso é a opção 2. A opção 1, referente a procurar uma linha, não consegui fazer, então, se a selecionar, o sistema te informará exatamente, que ela não está funcionando.

Foi a parte mais demorada, complicada, e chato. Não que o resto tenha sido bom, mas a main deu trabalho.

```
8   const char *SENHA_PARA_MANUTENCAO = "senhafraca";
```

```
25      int escolha = -1;
26      do
27      {
28          cout << "O que voce deseja fazer?" << endl;
29          cout << "  0. Encerrar o programa e sair" << endl;
30          cout << "  1. Procurar uma linha de onibus" << endl;
31          cout << "  2. Realizar manutencao do programa" << endl;
32
33          cin >> escolha;
```

4 CONCLUSÃO

Já havia me sentido bem feliz por ter entregue a primeira parte, então continuo. Apesar de não ter feito tudo, fiz grande parte, então, não é de se zerar também. Foi um trabalho muito difícil, trabalhoso e grande. Ainda mais para a dificuldade que tenho em entender e codificar algumas coisas. Mas foi bom, e querendo ou não, é uma boa experiência.

REFERÊNCIAS

Slides fornecidos pelo professor.

Códigos feitos em sala de aula.

Moodle.

Youtube e stack Overflow.

Um amigo.