UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MIKAELLA FERREIRA DA SILVA
LEONARDO DEORCE LIMA DE OLIVEIRA

1º TRABALHO PRÁTICO:

NETMAP

VITÓRIA
2018

MIKAELLA FERREIRA DA SILVA
LEONARDO DEORCE LIMA DE OLIVEIRA

1º TRABALHO PRÁTICO:

NETMAP

Trabalho apresentado à disciplina Estrutura de Dados I (INF09292) do Curso de Ciência da Computação da Universidade Federal do Espírito Santo no 2º semestre do ano 2018, como requisito para avaliação.
Orientador: Prof.ª Patrícia Dockhorn Costa

VITÓRIA
2018

RESUMO

Apresenta a implementação de uma rede para transferência de dados entre terminais, com o objetivo de aplicar conceitos de tipos abstratos de dados e modularização com a estrutura de dados chamada lista. A rede NetMap, criada neste trabalho, funciona de forma similar à internet, manipulando roteadores e terminais, bem como suas conexões e trabalhando métodos de busca, sendo uma importante e útil aplicação.

Palavras-chave**:** Rede. Dados. Implementação. Lista.

SUMÁRIO

1 INTRODUÇÃO

Este trabalho incentiva a busca por novas formas de resolução de problemas encontrados ao longo de seu desenvolvimento, bem como induz a prática da manipulação de estruturas do tipo Lista e o uso de tipos abstratos de dados.

Com intenção de organizar o uso de funções, o código do programa foi dividido em três bibliotecas com objetivos específicos, cada uma implementada pelo seu arquivo-fonte de mesmo nome. Uma biblioteca é encarregada de leitura do arquivo de entrada e execução de seus comandos, enquanto as outras duas manipulam as listas de terminais e roteadores.

Por questões acadêmicas, as listas de terminais e roteadores são de tipos diferentes: a lista de terminais é sem sentinela e simplesmente encadeada enquanto a lista de roteadores é com sentinela e duplamente encadeada.

2 BIBLIOTECAS

2.1 Leitura e execução de comandos da entrada

2.1.1 Visão geral

A biblioteca que define as funções de manipulação do arquivo de entrada foi denominada netmap.h. O arquivo de entrada é do tipo texto e contém comandos de operação do netmap. A biblioteca possui duas funções definidas, sendo uma responsável pela criação do netmap a partir do arquivo e outra responsável pela escrita do netmap em linguagem DOT.

A sintaxe esperada de cada linha do arquivo de entrada denominado entrada.txt consiste em nome do comando seguido de seus argumentos, sendo um máximo de dois argumentos por comando e mínimo de nenhum argumento e cada item da mesma linha deve ser separado por um espaço. A implementação desenvolvida permite flexibilidade na quantidade de espaços entre itens e após o último item em uma linha.

2.1.2 CriaNetmap

A função é segmentada, sendo esta a parte encarregada de verificar a passagem de argumento e abertura do arquivo de entrada. Ela recebe a quantidade de argumentos passados para o programa (int) e o vetor de strings contendo o nome do arquivo de entrada (char**). Não há retorno. Após execução: comandos de entrada executados e arquivos criados de acordo.

Ao iniciar, a função verifica se o houve algum argumento passado ao programa. Caso não houve, uma mensagem de erro é criada e passada como parâmetro para a função auxiliar EscreveLOG (definida em terminal.h), que imprime mensagens de erro no arquivo log.txt. Caso houve passagem de argumento, este é usado como nome do arquivo a ser aberto e tratado como entrada.

Ao abrir o arquivo com sucesso, a função Leitura é chamada, passando o arquivo aberto como parâmetro. No fim da execução, o arquivo é fechado e CriaNetmap encerra. Entretanto, no evento de falha na abertura do arquivo, a mesma mensagem de erro anterior é criada e escrita em log.txt através da função EscreveLOG.

2.1.3 Leitura

A função lê as linhas do arquivo passado, identificando itens individuais em cada uma. Ela recebe o arquivo de entrada aberto (FILE*). Não há retorno. Após execução: todos os comandos até o comando FIM ou final do arquivo de entrada executados e arquivos criados de acordo.

Ao iniciar, são declaradas diversas variáveis auxiliares. A variável str é encarregada de armazenar a linha resgatada do arquivo, enquanto a variável item recebe os pedaços retirados de str, formando um vetor de strings onde a primeira string é sempre o comando e as demais são seus argumentos, se exisirem.

As listas são inicializadas antes de um loop que só para uma vez que FIM for lido ou quando alcançar o final do arquivo.

São utilizadas funções da biblioteca string.h. A função strtok é usada para dividir str usando caracteres espaço como divisor. Enquanto a função strchr é usada para verificar a presença de \r ou \n no último item lido, a fim de removê-los. A função strlen

verifica se houve leitura indevida após o último argumento lido, o que ocorre caso não haja quebra de linha imediatamente após o último argumento.

A variável i passa então a guardar o número de itens presentes no vetor de strings item. A função ExecutaComando é chamada e item, i, lista de terminais e lista de roteadores são passados como parâmetros. Por fim, ao término do loop, as listas são liberadas.

2.1.4 ExecutaComando

A função executa o comando recebido. Ela recebe o vetor de strings com o comando e seus argumentos (char**), uma variável auxiliar cujo valor é o número de itens no vetor de comandos (int), a lista de terminais (CelTerm*) e a lista de roteadores (LsRot*). Retorna a lista de terminais pois esta é sem sentinela (discutido em 2.2.1). Após execução: comando recebido executado.

Ao iniciar, a função switch é usada para comparar o comando passado apenas com aqueles que condizem com seu número de argumentos. Através da função strcmp, o nome do comando recebido é comparado com nomes conhecidos e executado de acordo.

2.1.5 ImprimeNetMap

A função escreve o netmap em linguagem DOT em um arquivo com a ajuda de funções auxiliares. Ela recebe como argumentos a lista de terminais (void*) e a lista de roteadores (void*). Não há retorno. Após execução: netmap escrito em linguagem DOT em arquivo saida.dot.

Os tipos void* foram usados para eliminar a necessidade de incluir as bibliotecas terminal.h e roteador.h em netmap.h.

Ao iniciar, a função abre ou cria o arquivo saida.dot e verifica se obteve sucesso. Caso sim, inicia a escrita e chama as funções auxiliares enviando como parâmetros o ponteiro para o arquivo aberto e as listas correspondentes a cada função. Caso não, imprime mensagem de erro na tela.

As funções auxiliares são necessárias porque há limitações na manipulação de estruturas implementadas em arquivos diferentes. ImprimeTerm usa variáveis auxiliares para percorrer a lista de terminais escrevendo terminais, tanto aqueles desconectados quanto os conectados a um roteador. ImprimeRot usa variáveis auxiliares para percorrer a lista de roteadores escrevendo roteadores, tanto conectados a outros roteadores quanto desconectados.

Por fim, a função finaliza a escrita e fecha o arquivo saida.dot.

2.2 Lista de terminais

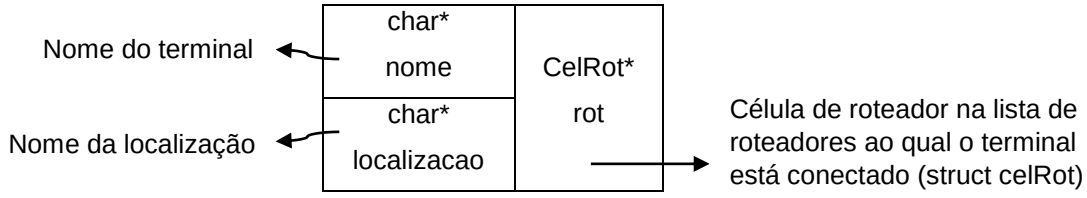
2.2.1 Visão geral

A biblioteca que define as funções de manipulação da lista de terminais do netmap foi denominada terminal.h. Esta é dividida em funções principais, que atendem requisitos de funcionamento do netmap, e funções auxiliares, que facilitam a comunicação com as implementações de outras bibliotecas.

A lista de terminais é do tipo sem sentinela e simplesmente encadeada. A escolha desse tipo de lista parte da sua simplicidade e da variedade que trás ao projeto. Abaixo seguem descrições dos elementos principais presentes na implementação da biblioteca terminal.h.

2.2.2 Estrutura do terminal (struct terminal)

Figura 1: struct terminal

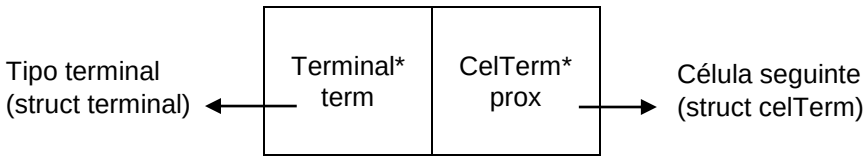


Representação da estrutura do tipo terminal.

2.2.3 Estrutura de uma lista/célula de terminais/terminal (struct celTerm)

Uma lista sem sentinela, simplesmente encadeada não circular é manipulada com o auxílio de um ponteiro para a primeira posição da lista. Portanto, a lista e suas células compartilham a mesma estrutura.

Figura 2: struct celTerm



Representtação da estrutura da célula/lista de terminal/terminais.

2.2.4 CadastraTerminal

A função adiciona um terminal à lista de terminais do netmap. Ela recebe como parâmetros o nome do terminal (char*), o nome da localização do terminal (char*) e o ponteiro para lista de terminais (CelTerm*). A função retorna a lista de terminais atualizada, pois esta é sem sentinela e o endereço da lista é sujeito à mudança. Após execução: uma estrutura terminal deve ter sido criada e preenchida com os dados fornecidos; uma célula criada e encadeada na lista preenchida com ponteiro para o terminal.

Ao iniciar, duas células (CelTerm*) são declaradas, uma auxiliar apontando para a mesma célula para a qual a lista fornecida aponta e outra alocada dinamicamente que será encadeada na lista. Em seguida, chama-se a função interna criaTerminal que aloca um tipo terminal preenchido com nome e localização fornecidos e que não está conectado a roteador.

Para facilitar a impressão do arquivo saida.dot, novas células são encadeadas ao final da lista de terminais. Portanto, a célula nova recebe um ponteiro para o terminal criado e seu ponteiro para a próxima célula recebe NULL.

A função então verifica se a lista fornecida está vazia. Caso esteja, o ponteiro para a lista passa a apontar para a nova célula, tornando-a o primeiro e único elemento da

lista. Caso não esteja, a célula auxiliar percorre a lista até a última posição, permitindo o encadeamento da nova célula ao fim da lista.

2.2.5 RemoveTerminal

A função remove um terminal existente da lista de terminais do netmap. Ela recebe como argumentos o nome do terminal (char*) e o ponteiro para a lista de terminais (CelTerm*). A função retorna a lista de terminais atualizada, pois esta é sem sentinela e o endereço da lista é sujeito à mudança. Após execução: o tipo terminal cujo nome é idêntico ao nome fornecido será liberado; a célula que aponta para o tipo terminal em questão é desencadeada da lista e liberada.

Ao iniciar, são declaradas três células auxiliares: uma que recebe a célula da lista fornecida que aponta para o terminal cujo nome foi fornecido através da função auxiliar BuscaTerminal; outra que aponta para a primeira posição da lista fornecida; e outra não inicializada encarregada de reter a posição anterior a medida que a lista é percorrida.

A lista de terminais é percorrida pelas células auxiliares. Uma vez que o terminal não é encontrado na lista, uma mensagem de erro é escrita em um vetor que é passado como parâmetro para a função EscreveLOG.

Caso o terminal seja encontrado, a função verifica se sua célula é a primeira da lista. Caso seja, a lista de terminais passa a apontar para a próxima posição (que pode ser NULL). Caso não seja, a posição anterior à célula a ser removida passa a apontar para a próxima posição. Por fim, célula e seu tipo terminal são liberados.

2.2.6 ConectaTerminal

A função conecta um terminal a um roteador, ambos existentes no netmap. Ela recebe como argumentos o nome do terminal (char*), o nome do roteador (char*), o ponteiro para a lista de terminais (CelTerm*) e o ponteiro para a lista de roteadores (void*). Não há retorno pois a lista de terminais permanece inalterada. Após execução: o tipo terminal do terminal cujo nome foi fornecido agora aponta para a célula do roteador cujo nome foi fornecido na lista de roteadores.

Ao iniciar, é declarada uma célula de terminal auxiliar que recebe um ponteiro para a célula do terminal cujo nome foi fornecido através da função BuscaTerminal, que retorna NULL caso não encontre o terminal passado. Se a célula for NULL, o terminal não existe no netmap e uma mensagem de erro é então criada e passada como parâmetro para a função EscreveLOG, que imprime a mensagem em log.txt.

Caso o terminal seja encontrado, a função verifica se o roteador existe no netmap, e resgata sua célula caso exista através da declaração de uma célula auxiliar para roteador e chamada da função BuscaRoteador. Novamente, se a célula receber NULL, uma mensagem de erro é criada e passada como parâmetro para a função EscreveLOG, que imprime a mensagem em log.txt.

Se terminal e roteador existem no netmap, o tipo terminal do terminal cujo nome foi fornecido passa a apontar para a célula do roteador cujo nome foi fornecido na lista de roteadores.

2.2.7 DesconectaTerminal

A função desconecta um terminal existente no netmap de seu roteador. Ela recebe como argumentos o nome do terminal (char*) e o ponteiro para a lista de terminais (CelTerm*). Não há retorno pois a lista de terminais permanece inalterada. Após execução: o tipo terminal do terminal cujo nome foi fornecido agora aponta para NULL, indicando que está desconectado.

Ao iniciar, é declarada uma célula de terminal auxiliar que recebe um ponteiro para a célula do terminal cujo nome foi fornecido através da função BuscaTerminal, que retorna NULL caso não encontre o terminal passado. Se a célula for NULL, o terminal não existe no netmap e uma mensagem de erro é então criada e passada como parâmetro para a função EscreveLOG, que imprime a mensagem em log.txt.

Caso a célula do terminal seja encontrada, o tipo terminal para o qual ela aponta passa a apontar para um roteador NULL, indicando que o terminal está desconectado.

2.2.8 EnviarPacotesDados

A função avalia a possibilidade de envio de pacotes de dados de um terminal para outro. Ela recebe como argumentos os nomes de ambos os terminais (char*) e a lista de terminais (CelTerm*). Não há retorno. Após execução: nomes dos terminais e resultado SIM ou NÃO escrito em arquivo saida.txt.

Ao inciar, declara células auxiliares para os terminais passados e, com o auxílio da função BuscaTerminal, verifica se os terminais existem no netmap. Caso algum terminal não exista, uma mensagem de erro é criada e passada como parâmetro para a função EscreveLOG, que imprime a mensagem em log.txt.

A função então verifica se os terminais passados estão conectados a algum roteador, caso algum não esteja, a comunicação entre eles não é possível, portanto uma mensagem de resultado NÃO é criada e passada como parâmetro para a função EscreveSAIDA, que imprime a mensagem em saida.txt.

Caso ambos estejam conectados a um roteador, o nome de cada roteador é resgatado e comparado. Se o nome for o mesmo, ambos estão conectados ao mesmo roteador e o envio de dados é possível, portanto uma mensagem SIM é criada e passada como parâmetro para a função EscreveSAIDA. Caso estejam conectados a roteadores diferentes, é chamada a funcaoBusca, detalhada na descrição de roteador.h, que retorna 1 para SIM e 0 para NÃO. O procedimento de escrita do resultado segue de acordo com o retorno da funcaoBusca.

2.2.9 FrequenciaTerminal

A função conta a quantidade de terminais cadastrados no netmap que são de uma determinada localização. Ela recebe como argumentos o nome da localização (char*) e a lista de terminais. Não há retorno. Após execução: o nome da localização e sua frequência absoluta de ocorrência na lista de terminais são escritos no arquivo de texto saida.txt.

Ao iniciar, é declarada uma variável do tipo int encarregada de armazenar o valor da contagem de ocorrências de terminais pertencentes à localização fornecida

(inicialmente zero). Também é declarada uma célula auxiliar de terminal inicializada apontando para a primeira posição da lista de terminais.

A função anda pela lista com a célula auxiliar e compara a localização de cada terminal apontado por cada célula com a localização desejada, realizando a contagem com o auxílio da variável declarada no início.

Por fim, uma mensagem contendo o nome da localização e sua frequência é passada como argumento para a função EscreveSAIDA, que imprime a mensagem em saida.txt.

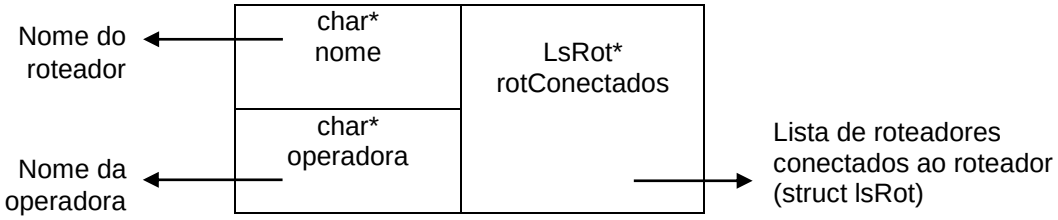
2.3 Lista de roteadores

2.3.1 Visão geral

A biblioteca que define as funções de manipulação da lista de roteadores do netmap e seus enlaces foi denominada roteador.h. Esta é dividida em funções principais, auxiliares internas, que serão usadas somente dentro da própria biblioteca para deixar o código mais legível e para o reuso nas funções, e auxiliares externas, que serão usadas em outras bibliotecas, como na terminal.h.

2.3.2 Estrutura do roteador (struct roteador)

Figura 3: struct roteador



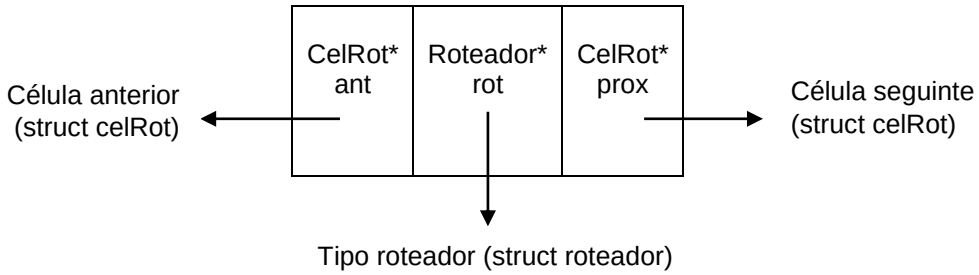
Representação da estrutura do tipo roteador.

2.3.3 Estrutura da célula de uma lista de roteadores (struct celRot)

Em vários problemas é necessário buscar a célula de um roteador em uma lista, por exemplo, ao remover um roteador da lista do netmap e ao desconectar um roteador

do outro, por isso, se faz útil uma função para realizar a busca e com isso é necessário saber qual a célula anterior ao roteador encontrado a fim de fazer o desencadeamento daquela célula, logo, para facilitar, foi pensado em uma lista duplamente encadeada.

Figura 4: struct celRot

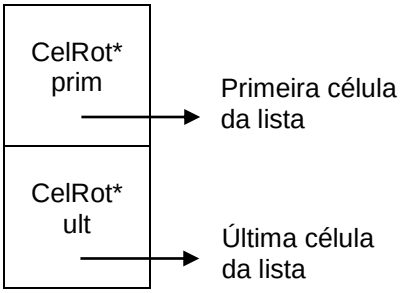


Representação da estrutura da célula de roteador.

2.3.4 Estrutura da lista de roteadores (struct lsRot)

Foi pensada em uma lista com sentinela.

Figura 5: struct lsRot



Representação da estrutura do sentinela da lista de roteadores.

2.3.5 CadastraRoteador

A função irá adicionar um roteador à lista de roteadores do netmap. Os argumentos são um ponteiro para o nome do roteador (char*), um ponteiro para o nome da operadora do roteador(char*) e o ponteiro pra lista de roteadores do netmap (LsRot*). Não há retorno na função. Ao final, o roteador deve ter sido criado, e a célula que aponta para ele encadeada na lista de roteadores do netmap.

Primeiro, cria-se um tipo roteador, o qual armazena o nome e a operadora do roteador e inicializa a lista de roteadores conectados, sendo ela vazia. Utiliza-se uma função auxiliar interna para fazer isso, chamada criaRoteador.

Depois cria uma célula que apontará para o roteador criado. E então encadeia essa na célula na lista de roteadores do netmap. Os roteadores cadastrados no netmap serão adicionados sempre ao final da lista. Utiliza-se uma função auxiliar interna chamada EncadeiaRoteador para fazer isso, que será utilizada em outra função.

2.3.6 RemoveRoteador

A função irá remover um roteador existente da lista de roteadores do netmap. Não há retorno e os argumentos são o nome do roteador (char*), a lista de roteadores do netmap (LsRot*) e a lista de terminais do netmap, que é do tipo void* (mas na verdade é CelTerm*) como uma forma de não ter erros na compilação já que a roteador.h inclui a terminal.h e vice-versa. Ao final, o roteador deve ter sido removido do netmap e desconectado de todos os roteadores e terminais que antes estava.

Primeiro a função busca a célula do roteador na lista de roteadores do netmap. Caso encontre-o, é preciso desconectá-lo de todos os outros roteadores ao qual está conectado, nisso é feito um loop que percorre a lista de roteadores conectados chamando a função DesconectaRoteadores.

Depois disso, é necessário desconectar os terminais desse roteador, utiliza-se a função DesconectaRoteador que está definida na terminal.h. O roteador é desencadeado utilizando a função DesencadeiaRoteador da lista de roteadores do netmap e então liberado usando a função auxiliar LiberaTipoRoteador.

Caso o roteador não exista no netmap, uma mensagem de erro é escrita no log.txt e a função encerrada.

2.3.7 ConectaRoteadores

A função irá conectar dois roteadores existentes no netmap. Não há retorno e os argumentos são o nome de cada roteador (char*) e a lista de roteadores do netmap

(LsRot*). Ao final, cada roteador deve estar na lista de roteadores conectados um do outro.

Primeiro a função busca as células dos roteadores na lista de roteadores do netmap para ter acesso às listas de roteadores conectados e o tipo roteador de cada um. Caso encontre-os, novas células são criadas e apontam para os roteadores, cada uma. Note que não é criado um novo tipo roteador.

Então a célula que aponta para um roteador será encadeada na lista de roteadores do outro roteador e vice-versa usando a função EncadeiaRoteador.

Caso um ou os dois roteadores não existam no netmap, uma mensagem de erro é escrita no log.txt e a função encerrada.

2.3.8 DesconectaRoteadores

A função irá desconectar dois roteadores existentes e conectados no netmap. Não há retorno e os argumentos são o nome de cada roteador (char*) e a lista de roteadores do netmap. Ao final, a lista de roteadores conectados de cada roteador não deve conter uma célula apontando para o roteador um do outro.

Primeiro a função busca as células dos roteadores no netmap para ter acesso à lista de roteadores conectados de cada um.

Depois é preciso buscar o roteador1 na lista de roteadores conectados do roteador2 com a função auxiliar BuscaRoteador para poder chamar a função DesencadeiaRoteador passando como argumento a célula encontrada e a lista de roteadores conectados do roteador2 e vice-versa. Para finalizar, as células são liberadas.

Caso um ou os dois roteadores não existam no netmap ou eles não estejam conectados, uma mensagem de erro é escrita no arquivo log.txt e a função encerrada.

2.3.9 FrequenciaOperadora

A função contará quantos roteadores do netmap são da operadora que é passada como entrada. Não há retorno, já que a quantidade é escrita diretamente no arquivo

saida.txt. Os argumentos de entrada são o nome da operadora (char*) e a lista de roteadores do netmap.

Um ponteiro do tipo CelRot* percorre a lista de roteadores do netmap verificando qual a operadora é cada roteador e se for igual à operadora de entrada, então o contador acrescenta +1.

2.3.10 funcaoBusca

Outra função importante na roteador.h é a funcaoBusca que tem papel principal na função EnviarPacotesDados definida na terminal.h. Ela foi implementada no roteador.c porque é preciso manipular estruturas que são definidas somente na biblioteca roteador.h.

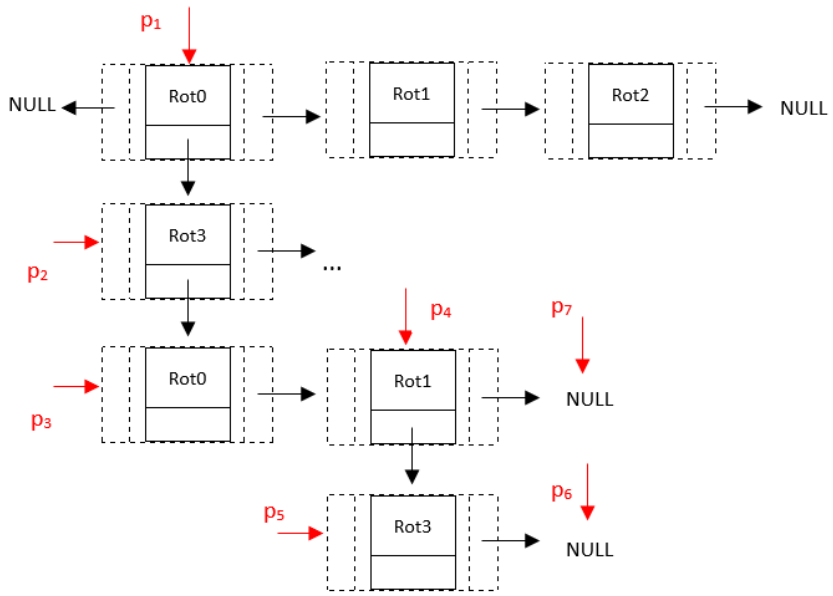
A função tem como argumento uma célula do tipo roteador (CelRot*), o nome dor roteador que está buscando (char*), um vetor vet que irá guardar o nome dos roteadores que já foram analisados e um contador i (int*) que indica a primeira posição livre do vetor. O retorno é 1, caso o roteador seja encontrado e 0, caso contrário.

É usado um ponteiro p do tipo CelRot* que irá apontar, inicialmente, para a primeira célula da lista de roteadores conectados do roteador de entrada.

Em um loop (irá parar quando p = NULL) são feitas verificações sobre o roteador que p aponta. Primeiro, se o nome do roteador está contido no vetor, usando uma função auxiliar chamada BuscaNomeVet. Caso esteja, p recebe o endereço da próxima célula, caso contrário, verifica se o nome do vetor é igual ao vetor que está buscando. Se não for o roteador, então guardará o nome desse no vetor e é chamada a funcaoBusca. Após a chamada, verifica se o retorno foi 1, se não, p recebe a próxima célula.

Um exemplo do funcionamento dessa função é dado pelo esquema a seguir. Ele mostra o caso em que o roteador é encontrado. O roteador que se busca é Rot5.

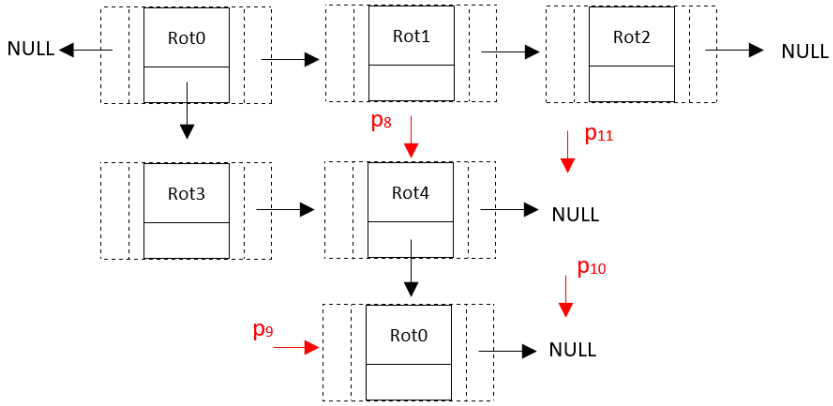
Esquema 1: Exemplo funcaoBusca parte 1



p1:
vet = {}
i = 0
p2:
vet = {“Rot0”}
i = 1
p3, p4:
vet = {“Rot0”, “Rot3”}
i = 2
p5, p6, p7:
vet = {“Rot0”, “Rot3”, “Rot1”}
i = 3

Esquema[1] de funcionamento de funcaoBusca: Primeira parte.

Esquema 2: Exemplo funcaoBusca parte 2

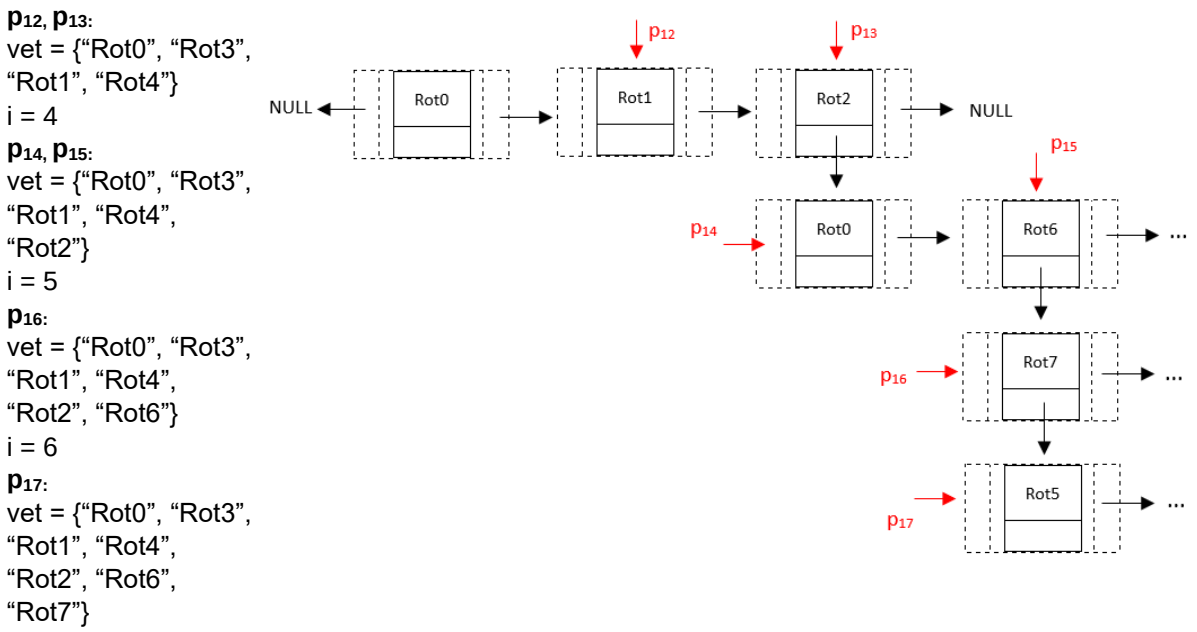


p8:
vet = {“Rot0”, “Rot3”, “Rot1”}
i = 3
p9, p10, p11:
vet = {“Rot0”, “Rot3”, “Rot1”, “Rot4”}
i = 4

Esquema de funcionamento de funcaoBusca: Segunda parte.

[1] Ocultou-se as setas dos ponteiros ant e a sentinela e, além disso, o tipo roteador foi colocado “dentro” da célula para facilitar a representação.

Esquema 3: Exemplo funcaoBusca parte 3



Esquema de funcionamento de funcaoBusca: Terceira parte.

Em p17, o roteador é encontrado, então a função encerra retornando 1. Observe que, mesmo se não encontrasse o roteador, a função pararia retornando 0. Foi pensado da forma a “varrer” todas as listas possíveis com o while e a chamada recursiva da função, mas evitando um loop infinito, utilizando o vetor.

3 CONCLUSÃO

Uma rede pode ser definida como lista de listas, que são as conexões entre roteadores, os quais realizam a passagem de dados entre terminais, o que permite o compartilhamento de informações, sendo muito importante atualmente.

Além das funcionalidades básicas, por exemplo, de inserção de um roteador e um terminal à rede NetMap ou conexão de um terminal a um roteador, nesse trabalho, também fez-se necessário lidar com situações mais profundas, como as conexões implícitas, já que um terminal pode estar conectado indiretamente com outro roteador e também, operar uma lista mais complexa que é composta por uma estrutura que também contém uma lista de elementos do seu mesmo tipo. Então, com os conceitos de manipulação de lista aprendidos em sala de aula, foi possível ampliá-los para

resolver os problemas que exigiam uma análise maior, como a remoção de um roteador do NetMap ou a verificação se é possível enviar dados de um terminal ao outro, com dificuldades que foram superadas com esquematizações e análise de casos.

Outro ponto significativo, foi lidar com arquivos. Em problemas reais, como os que acontecem em empresas ou em projetos pessoais, não seria eficiente inserir os dados de entrada ou imprimir a saída da maneira feita em sala de aula, diretamente na main.c e no terminal, respectivamente. A melhor maneira é pela leitura e escrita de arquivos, porque, assim, as informações permanecem disponíveis mesmo se o programa for encerrado e podem ser reutilizadas em outros programas, além de facilitar a entrada de uma grande quantidade de dados. De fato, a leitura de cada linha para obter os dados de entrada para as funções implementadas foi complicada, em razão de, por exemplo, espaços em branco poderem atrapalhar a execução. Com testes e análise de casos, a manipulação de arquivos foi feita com sucesso, preparada para problemas como o citado.

BIBLIOGRAFIA

C Programming Tutorial. **C - File I/O**. <https://www.tutorialspoint.com/cprogramming/c_file_io.htm>. Acesso em: 20 set. 2018.

C Programming for Beginners. **Pointers and 2-D arrays**. Disponível em: <https://overiq.com/c-programming/101/pointers-and-2-d-arrays/>. Acesso em: 21 set. 2018.

C Programming Language. **Functions: Static functions in C**. Disponível em: <https://www.geeksforgeeks.org/what-are-static-functions-in-c/>. Acesso em: 17 set. 2018.

Stack Overflow. **What is the difference between \r and \n?**. Disponível em: <https://stackoverflow.com/questions/1279779/what-is-the-difference-between-r-and-n/9549183>. Acesso em: 21 set. 2018.

The C Standard Library. **C Library - <string.h>**. Disponível em: <https://www.tutorialspoint.com/c_standard_library/string_h.htm>. Acesso em: 13 set. 2018.