UNIVERSIDADE FEDERAL DO ESPIRITO SANTO

CARLOS BATTISTI FILHO

KARINNE GOMES VIEIRA

NETMAP

Trabalho 1 de Estrutura de Dados

VITORIA

2018

INTRODUÇÃO

Esse trabalho tem como objetivo de praticar o uso de estruturas de dados na criação de um NetMap entre terminais de acesso e seus roteadores. Um NetMap é uma rede de troca de dados em que seus roteadores possuem enlaces entre outros roteadores ou também a um terminal, este que só pode se conectar com um rotador exclusivo. As manipulações dessas estruturas de dados incluem inicialização, alocação, modificação, como também funções para retirar e liberar tais estruturas. Funciona recebendo comandos de um arquivo de texto com os dados e os comandos que ira executar, criando como resultado um outro arquivo de texto como saída, junto dele se necessário também será gerado um log de erro com os problemas durante a execução do programa.

INPLEMENTAÇÃO E DESENVOLVIMENTO

Funções Terminal

Estruturas:

typedef struct celulaterminal Celulaterminal;

Define a estrutura CelulaTerminal, composta por um endereço de uma subestrutura Terminal e um endereço para a próxima célula;

typedef struct terminal Terminal;

Define a estrutura Terminal, Composta por duas strings, uma para o nome do terminal e outra para seu local, e um valor inteiro para indicar se o terminal possui um enlace com um roteador, que será o único roteador no qual esse roteador estará conectado.

typedef struct listaterminal Listaterminal;

Define a estrutura ListaTerminal que servira de sentinela para a lista encadeada de terminais, composta de endereços para o primeiro e ultimo elemento da lista.

Funções do TAD:

Listaterminal* InicializaListaTerminal();

Função que aloca espaço para o tipo ListaTerminal e define os endereços de primeiro e ultimo lugar na lista como null, retornando o endereço logo depois.

int VerificaListaTerminal(Listaterminal* lt);

Verifica se a lista esta vazia, enviando um log caso esteja e retornando zero, caso contrário, retorna 1.

Terminal* CriaTerminal(char* nome, char* local);

Função que aloca espaço para o tipo Terminal e preenche o nome e local, também define o sinalizador de enlace com roteador em 0, retornando o endereço logo depois.

void InsereTerminal(Terminal* ter, Listaterminal* lt);

A função começa alocando espaço para a célula terminal, após isso verifica a situação da lista e insere a célula de acordo, após a nova célula ser adcionada, o endereço do terminal também é colocado na célula.

Celulaterminal* RetiraTerminal(Listaterminal* lt, Terminal* ter);

Procura a célula do terminal na lista, e a retira se presente.

void RetiraRoteadorDoTerminal(Terminal* ter) ;

Verifica se o terminal possui enlace com roteador, e caso exista muda o sinalizador para zero.

Terminal* ProcuraTerminal(char* nome, Listaterminal* lt);

Procura terminal pelo nome na lista de terminal, o retornando caso ele esteja na lista.

void ConectaRoteadorAoTerminal(Terminal* ter) ;

Muda o sinalizador do terminal para 1 indicando que agora ele possui enlace com um roteador.

void DesconectaTerminalNaListaDeRoteadores(Listaterminal* lt);

Percorre a lista de terminais ate o fim, chamando a função RetiraRoteadorDoTerminal.

int TerminalPorLocal(char* local, Listaterminal* lt) ;

Vasculha a lista e retorna o numero de roteadores presentes em dado local;

void LiberaTerminal(Celulaterminal* cllter);

Libera memoria alocada na célula.

void LiberaListaTerminal(Listaterminal* lt);

Libera a memoria alocada a lista.

char** ImprimeTerminais(Listaterminal* lt, int* ind);

Imprime lista de terminais no arquivo de saída.

int TamanhoListaTerminal(Listaterminal* lt);

Retorna o tamanho da lista de terminais.

Funções roteador:

Estruturas:

typedef struct celularoteador Celularoteador;

Define a estrutura CelulaRoteador que é composta por um endereço para um tipo rotador e outro para a próxima célula.

typedef struct roteador Roteador;

Define a estrutura Roteador composta de duas strings, uma para seu nome e outra para sua operadora,um sinalizador int que será utilizado quando o netmap for percorrido, também possui uma lista de roteadores no qual possui enlaces e outra com terminais com o mesmo objetivo.

typedef struct listaroteador Listaroteador;

Defina a estrutura ListaRoteador que servira de sentinela para a lista de roteadores, com um endereço para as primeiras e ultimas posições.

Funções do TAD:

Listaroteador* InicializaListaRoteador();

Inicializa o tipo ListaRoteador alocando o espaço na memoria e definindo a primeira e última posição como null.

int VerificaListaRoteador(Listaroteador* lr);

Verifica se a lista esta vazia, enviando uma mensagem caso esteja e retornando o valor zero, caso contrário, retorna 1.

Roteador* CriaRoteador(char* nome, char* operadora);

Inicializa o tipo Roteador, aloca seu espaço na memoria, incluído nome e operadora e define a lista de enlaces como null e o sinalizador como 0.

void InsereRoteador(Roteador* rot, Listaroteador* lr);

Inicializa e aloca o tipo célula que será inserido na lista, verifica se a lista esta vazia e preenche de acordo.

void RetiraRoteador(Listaroteador* lr, Roteador* rot);

Procura o roteador na lista, o retirando se presente, liberando a memória após o processo.

Roteador* ProcuraRoteador(char* nome, Listaroteador* lr) ;

Procura o roteador na lista, o retornando se existir.

void RetiraTerminalDoRoteador(Terminal* ter, Roteador* rot) ;

Chama a função RetiraTerminal com a lista de terminais no roteador e seu nome.

void VerificaListaTerminalNoRoteador(Roteador* rot);

Verifica se a lista de terminais em dado roteador esta inicializada, e chama a função InicializaTerminal se não.

void InsereTerminalNoRoteador(Roteador* rot, Terminal* ter);

Insere dado terminal na lista de enlaces do roteador usando a função InsereTerminal.

void EnlaceDeRoteador(Roteador* rot1, Roteador* rot2);

Cria um enlace entre os dois roteadores dados.

void DesfazEnlace(Roteador* rot1, Roteador* rot2);

Verifica se ha um enlace entre os dois roteadores e destrói o enlace caso exista.

int RoteadorPorOperadora(char* op, Listaroteador* lr);

Retorna o número de roteadores pertencentes a operadora.

void LiberaListaRoteador(Listaroteador* lr);

Libera a locação de memoria para a lista.

char* RetornaNomeRoteador(Roteador* rot) ;

Retorna o nome do roteador.

Roteador* ProcuraRoteadorPorTerminal(Listaroteador* lr, char* nometer);

Procura por terminal na lista de terminais em cada roteador na lista de roteadores.

void ImprimeTerminaisConectados(char** nomester, Listaroteador* lr, int tam)

Função que imprime no arquivo os terminais que estão conectados a algum roteador.

Funções NetMap:

Funções do TAD:

void CadastraRoteador(char* nome, char* operadora, Listaroteador* lr);

Verifica se a lista de roteadores esta vazia utilizando a função VerificaListaRoteador, saindo se estiver. Caso contrario cria um novo roteador com CriaRoteador e o insere na lista de roteadores com InsereRoteador.

void CadastraTerminal(char* nome, char* local, Listaterminal* lt) ;

Verifica se a lista de terminais esta vazia utilizando a função VerificaListaTerminal, saindo se estiver. Caso contrario cria um novo roteador com CriaTerminal e o insere na lista de terminais com InsereTerminal.

void RemoveRoteador(char*nome, Listaroteador* lr);

Verifica se a lista de roteadores esta vazia utilizando a função VerificaListaRoteador, saindo se estiver. Caso contrario, procura pelo roteador com outra função chamada ProcuraRoteador e caso esse roteador seja encontrado, é retirado com a RetiraRoteador.

void ConectaTerminal(char* nometer, char* nomerot, Listaterminal* lt, Listaroteador* lr);

Verifica as listas de roteadores e terminais com as funções VerificaListaRoteador e VerificaListaTerminais, saindo da função caso uma delas estejam vazias, após isso as listas são varridas procurando pelo terminal e roteador requisitado e então estes são conectados com as funções InsereTerminalNoRoteador e ConectaRoteadorAoTerminal.

void DesconectaTerminal(char* nometer, Listaterminal* lt, Listaroteador* lr);

Verifica a lista de terminais com a função VerificaListaTerminais, saindo da função caso esteja vazia, após isso a lista é varridas procurando pelo terminal requisitado e então este é desconectado do roteador com as funções RetiraTerminalDoRoteador e RetiraRoteadorDoTerminal.

void RemoveTerminal(char* nometer, Listaterminal* lt);

Verifica a lista de terminais, saindo da função caso esteja vazia, após isso a lista é varridas procurando pelo terminal requisitado e então este é desconectado da lista de terminais,

void ConectaRoteadores(char* nomerot1, char* nomerot2, Listaroteador* lr);

Verifica se a lista de roteadores esta vazia e caso esteja aborta a função se não, vasculha a lista procurando pelos dois roteadores e faz o enlace entre eles com a função EnlaceDeRoteador.

void DesconectaRoteadores(char* nomerot1, char* nomerot2, Listaroteador* lr);

Verifica se a lista de roteadores esta vazia e aborta caso esteja, se não, vasculha a lista procurando pelos dois roteadores e desfaz o enlace entre eles com a função DesfazRoteador.

void FrequenciaTerminal(char* local, Listaterminal* lt);

Chama a função TerminalPorLocal e imprime no arquivo de saída quantos Terminais tem no local pedido.

void FrequenciaOperadora(char* operadora, Listaroteador* lr);

Chama a função RoteadorPorOperadora e imprime no arquivo de saída quantos Roteadores pertencem a operadora pedida.

void ImprimeNetMap(Listaterminal* lt, Listaroteador* lr, int contador);

Imprime o NetMap, caso as listas existam, em um arquivo saída.dot.

CONCLUSÃO

Nesse trabalho é mostrado um NetMap utilizando arquivos de texto para criação e edição de seus componentes. Houve algumas dificuldades para a implementação desse trabalho, pois tivemos que considerar e manipular vários tipos de funções e estruturas para a execução do programa. Porem, possivelmente por falta de atenção e nosso domínio sobre a matéria, durante o desenvolvimento deste trabalho, várias funções acabaram não se comportando como o esperado, apresentando erros em sua execução, sendo por alocação de tamanho errado de memoria como também o tipo do arquivo em si. Esse trabalho apresenta uma ideia que no fim acabou melhorando o nosso entendimento sobre o conteúdo dessa matéria.