

**UNIVERSIDADE FEDERAL DE SERGIPE**

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Programação Funcional

Docente: Profª Beatriz Trinchao Andrade de Carvalho

Turma: 07 – 2023.1

Projeto Mapa em Haskell

Discentes autores: Enzo Guilherme Dias Marques

Daniel Felipe Dávila Santos Lima

Guilherme de Jesus Cunha

Emilly Maria Santos Luna

**FUNÇÕES**

As funções abaixo ocorrem de forma recursiva, respeitando os critérios de correção da primeira etapa e utilizando estratégias e métodos aprendidos em sala de aula, como por exemplo: condicionais, compreensão, recursão, dentre outras. Vale ressaltar que houve auxílio dos monitores da disciplina para o desenvolvimento de determinadas funções.

Funções sem entradas que retornam um mapa: A função **mapaVazio** retorna um mapa vazio. A função **mapaImperioRomano** retorna o mapa do império romano na época de Augusto (27 a.C até 14 d.C). A função **mapaContinental** retorna um mapa com os continentes de forma geral. A função **mapaPokemon** retorna um mapa do Pokemon Fire Red na época do Game Boy Advanced (GBA) e Nintendo. A função **mapaUfs** retorna o mapa da Universidade Federal de Sergipe (UFS), mais especificamente do campus de São Cristóvão, esse mapa foi inspirado no mapa real até então do ano atual (2023). Dessa forma, é totalizado no projeto 5 mapas para teste.

A função **adicionarCidade** recebe o mapa, o nome da cidade, coordenadas da cidade e as cidades vizinhas (que possuem ligação direta), e retorna o mapa com as cidades e coordenadas adicionadas. Através de Dessa forma, agora tratado os erros a função começa verificando se a cidade que desejamos adicionar é única (ou seja, se ela já existir não será adicionada) e verifica se as coordenadas que desejamos adicionar já existem, sempre retornando uma lista de rotas vazias, já que teremos uma função específica para adicionar rotas.

A função **removerCidade** recebe o mapa bruto (que por sua vez é uma lista de cidades), recebe o nome da cidade que queremos remover e ela nos retorna o mapa sem a cidade selecionada para remoção. Usando casamento de padrão para tratar o caso de o mapa ser uma lista vazia, retorna automaticamente uma lista vazia. Caso não seja uma lista vazia, ela faz uso de casamento de padrões para dividir a lista em uma tupla contendo nomeDacidade, coordenadas, estradas (que são as cidades) e o restante da lista “nomeDoMapa” (que é o mapa que estamos alterando para fazer a remoção). Com guardas, se o nome da cidade que colocamos for igual ao nome da cidade contida no mapa, então a cidade do mapa é removida (chamando recursivamente a função removerCidade), vale ressaltar que é usada a função filter (que está presente ao fim da função usando Where) para remover as estradas que estão ligadas a cidade selecionada para remoção. Caso contrário, a função retorna o mapa sem alterações.

A função **adicionarEstrada** recebe um mapa, o nome da cidade que queremos adicionar a estrada e o nome da estrada, e vai retornar o mapa com a estrada adicionada. Entretando, dessa vez a função adiciona a estrada da cidade A para a cidade B (caminho de ida) e da cidade B para a cidade A (caminho de volta). Se o nome da estrada selecionada para adicionar ao mapa não estiver presente na lista de estradas da cidade escolhida (usando ‘not elem’) e for diferente do nome da cidade de origem (tratando caso de o nome da estrada ser igual ao nome da cidade) então adiciona o nome da nova estrada na cidade e consequentemente no mapa, caso contrário é retornada a lista original sem alterações (esse procedo é feito tanto na ida, quanto na volta).

A função **removerEstrada** recebe o mapa que queremos remover a estrada, o nome da cidade que queremos remover a estrada e o nome da estrada que queremos remover e retorna o mapa “atualizado”. Caso seja uma lista vazia (mapa vazio) é retornado instantaneamente uma lista vazia (mapa vazio). Se não for uma lista vazia, a função primeiro verifica se o nome da cidade inserido é igual ao nome da cidade que precisa ter as estradas removidas, se sim, a função remove as estradas dessa cidade usando filter, se não, a função retorna o mapa sem alterações.

A função **distEuclidiana** recebe duas coordenadas e retorna a distância euclidiana entre as duas coordenadas através da raiz da soma dos quadrados das variações de X e Y.

A função **haEstradas** recebe o mapa e o nome de duas cidades para saber se há uma estrada entre elas (ou seja, verifica se elas estão ligadas diretamente através da lista de estradas das cidades), caso exista uma estrada entre as cidades a função retorna true, caso não exista, retorna false. Para os tratamentos de erros, se o mapa estiver vazio, a função retorna um erro com a mensagem "O mapa está vazio!", caso a cidade1 for uma string vazia e a cidade2 for especificada, a função retorna um erro com a mensagem "Falta a cidade de origem, você colocou somente a cidade de destino!" e por fim, se a cidade2 for uma string vazia e a cidade1 for especificada, a função retorna um erro com a mensagem "Falta a cidade de destino, você colocou somente a cidade de origem!". Entretanto, caso não ocorra nenhuma situação citada anteriormente, a função **haEstradas** aciona a função interna **verificaEstradas** que foi definida no “Where”, que percorre as tuplas presente no mapa escolhido, se o nome da cidade que foi colocado como entrada (cidade1) for igual ao nome da cidade de alguma tupla do mapa (nomeCidade), através do ‘elem’ é verificado se a cidade2 está presente na lista de estradas da cidade1, caso esteja é retornado True e caso contrário ele continua procurando pelo resto do mapa (ou seja, pelo resto de todas as outras tuplas presentes no mapa), caso não ache, é retornado False.

A função **encontraCidade** é uma função semelhante a função **verificaEstradas**, ela recebe um mapa e o nome da cidade que queremos encontrar e retorna a tupla contendo o nome da cidade caso encontre, caso não encontre, retorna um erro com a mensagem “Cidade não encontrada no mapa”, ou seja, diferente da **verificaEstradas** que retorna um valor booleano caso as cidades forem ligadas diretamente, o **encontraCidade** tem como objetivo encontrar e retornar informações detalhadas sobre uma cidade específica com base no nome da cidade.

A função **haRotas** recebe um mapa e o nome de duas cidades para saber se há uma rota entre elas, diferente da função **haEstradas** que só verifica se as cidades estão ligadas diretamente, **haRotas** começa com dois casos bases: 1- Se a cidade inicial for igual a cidade final, retorna True (foi tentado tratar retornando um erro “A cidade inicial e final são iguais”, mas poderíamos ter o caso “cidade A -> cidade B -> cidade C -> cidade A” e então a cidade inicial e final são iguais e formam uma rota). 2- Se a cidade inicial for diferente da cidade final, a função prossegue pesquisando rotas através da função definida no “Where” **encontraRota** que por sua vez, usa a função **encontraCidade** para obter informações sobre a cidade atual. Na função interna **encontraRota** as rotas da cidade atual são armazenadas na variável “estradas”, então com o uso de “filter” a função filtra as cidades que ainda não foram visitadas (isso só é possível por conta da variável “estradas”), em seguida com uso da função “any” torna-se possível verificar se existe ao menos uma cidade nas cidades não visitadas que satisfaçam os critérios: Se a cidade for igual ao destino retorna True, senão é chamada a função **encontraRota** recursivamente.

A função **estradasDaCidade** recebe um mapa e um nome (nome da cidade do mapa) e retorna uma lista com nomes. Essa função tem como objetivo encontras os nomes das cidades conectadas diretamente a uma cidade específica por rotas ou estradas. A função definida no “Where” chamada de **obterEstradas** recebe a cidade de interesse e o mapa como argumentos e retorna uma lista com nomes. Essa função começa com um tratamento em caso de o mapa estar vazio, se estiver, retorna um erro dizendo: “Cidade não encontrada no mapa.”, se não estiver vazio, então a função percorre o mapa para encontrar a cidade de interesse, se a encontrar no mapa, então a função colhe o nome das estradas da cidade encontrada e retorna uma lista com nome dessas cidades. Caso a cidade não for encontrada no mapa, a função retorna o erro.

A função **cidadesConectadasPorRotas** recebe um mapa e duas cidades, e retorna uma lista com nomes. Essa função tem como objetivo exibir através da lista o nome de todas as cidades que existem entre duas cidades. Apesar da função percorrer caminhos diferentes, ela exibe as cidades de todos os possíveis caminhos em uma lista. Ela aciona a função **encontrarCaminho** que obtém as rotas da cidade origem do mapa através de recursividade e mantém uma lista de cidades visitadas (para evitar um looping infinito), se a segunda cidade escolhida for igual a cidade final, é retornada uma lista reversa, e em seguida cria uma lista através da função **cidadeNaoVisitadas** para encontrar o caminho para a cidade final entre as cidades não visitadas. Se a cidade final não for encontrada, retorna uma lista vazia, que indica que não foi encontrado um caminho válido, se ainda houver cidades não visitadas, a função chama recursivamente **cidadesNaoVisitadas**.

A função **comprimentoDaRota** recebe um mapa, uma cidade inicial e uma cidade final como entrada e calcula o comprimento da rota entre essas duas cidades, através do somatório das distâncias euclidianas entre cada cidade presente na rota. O caso base verifica se a cidade inicial possui o mesmo nome da cidade final, pois, nesse caso o comprimento da rota é 0. No caso geral, se as cidades iniciais e finais forem diferentes a função definida no “Where” **encontrarCaminho** é chamada para encontrar os caminhos entre as cidades de origem e destino, armazenando o resultado na variável “caminho”. Em seguida são usadas e aproveitadas do caso anterior as funções **cidadeNaoVisitada** e **encontraCidade.** A função **calcularComprimento** calcula o comprimento total da rota somando as distâncias euclidianas (com a função já definida **distEuclidiana**) entre as cidades no caminho e a função **coordenadasDaCidade** retorna as coordenadas de uma cidade a partir do mapa.