UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

ANTHONY DOS ANJOS MARQUES
HEITOR BITTENCOURT DE ALMEIDA
JOSÉ CLÁUDIO BARBOSA DE SANTANA FILHO
SEBASTIÃO OLIVEIRA COSTA NETO

RELATÓRIO
PROJETO MAPAS

Aracaju, SE
2023

ANTHONY DOS ANJOS MARQUES
HEITOR BITTENCOURT DE ALMEIDA
JOSÉ CLÁUDIO BARBOSA DE SANTANA FILHO
SEBASTIÃO OLIVEIRA COSTA NETO

RELATÓRIO
PROJETO MAPAS

Relatório referente ao projeto denominado "Mapa" como parte da disciplina de Programação Funcional, turma 07, ministrada pela Prof.ª Beatriz Trinchão Andrade.

Aracaju, SE
2023

Sumário

1. INTRODUÇÃO

Neste projeto, temos como objetivo a criação de uma mapa de cidades usando a linguagem Haskell. A primeira etapa tem como objetivo a criação de um sistema que permita a representação e manipulação de um mapa e suas cidades. E, para fazê-lo, foram utilizados alguns tópicos aprendidos em sala de aula como: recursividade, tipos de dados e estruturas condicionais.

2. MODELAGEM

2.1. Modelagem de Dados

Para começar, utilizamos a definição dos tipos que está no módulo **Mapa** passado pelos monitores, sendo esses, ***Nome*** para os nomes de uma cidade, ***Localizacao*** que representa as coordenadas de uma cidade, ***Rotas*** representando uma lista de cidades conectadas por estradas, ***Cidade*** para uma tupla que contém o nome, as coordenadas e as cidades conectadas e ***Mapa*** para representar uma lista de cidades. Esses dados além de facilitar o entendimento do problema, auxiliam na hora de programar, pois relacionam as variáveis no projeto a partir da associação com seu significado físico.

```
8   type Nome = String
9   type Localizacao = (Double, Double)
10  type Rotas = [Nome]
11  type Cidade = (Nome, Localizacao, Rotas)
12  type Mapa = [Cidade]
```

2.2. Funções Implementadas

Depois de definidos os tipos, foi realizada a implementação de funções auxiliares que foram úteis para essa primeira parte do projeto e que poderão ser úteis na segunda etapa também. Então, foram criadas funções para obter os atributos do mapa, como obter o nome , a localização e as estradas que conectam essa cidade a outras, a função ***obterCidade*** para obter a cidade a partir do nome dessa cidade, uma função ***existeCidade*** para verificar se a cidade existe no mapa e uma função ***atualizarEstradas*** para quando for necessário a manipulação de adição e remoção de estradas entre cidades. Tais funções foram criadas com o intuito de tratar casos onde pode ocorrer algum erro e tentar não deixar o código muito poluído.

```haskell
obterNome :: Cidade -> Nome
obterNome (nome, _, _) = nome

obterLocal :: Cidade -> Localizacao
obterLocal (_, coord, _) = coord

obterEstradas :: Cidade -> Rotas
obterEstradas (_, _, estradas) = estradas

obterCidade :: Mapa -> Nome -> Cidade
obterCidade [] _ = error "Nao existe nenhuma cidade com esse nome"
obterCidade ((nome,coord,estradas):resto) cidade
    | cidade == nome = (nome,coord,estradas)
    | otherwise = obterCidade ((nome,coord,estradas):resto) cidade

existeCidade :: Nome -> Mapa -> Bool
existeCidade _ [] = False
existeCidade cidade (c:cs)
    | cidade == obterNome c = True
    | otherwise = existeCidade cidade cs

atualizarEstradas:: Nome -> Rotas -> Rotas
atualizarEstradas _ [] = []
atualizarEstradas cidade (x : xs)
  | cidade == x = atualizarEstradas cidade xs
  | otherwise = x : atualizarEstradas cidade xs
```

Depois de implementadas as funções auxiliares, foi dado início a criação do código com a função que inicializa o mapa, ***mapaInit***.

```haskell
--  Funcao que inicia/cria um mapa

mapaInit :: Mapa
mapaInit = []
```

Para que seja possível adicionar uma cidade ao mapa, foi criada a função ***adicionarCidade*** que recebe como entrada o mapa, o nome da cidade e as coordenadas e caso essa cidade já exista, então é retornado o mapa que foi passado, do contrário, a cidade é adicionada ao mapa.

```haskell
--  Funcao que adiciona cidade ao mapa
{-  A funcao verifica se a cidade dada como entrada existe no mapa,
    se existir ela retorna o mapa como estava,
    senao ela adiciona a cidade no inicio do mapa
-}

adicionarCidade :: Mapa -> Nome -> Localizacao -> Mapa
adicionarCidade mapa cidade coord =
    if (existeCidade cidade mapa)
        then mapa
        else (cidade, coord, []): mapa
```

Depois foi implementada a ***removerCidade*** que recebe o mapa e o nome da cidade e caso essa cidade exista no mapa, retorna um novo mapa sem a cidade e suas estradas associadas.

```haskell
{-  Funcao para remover uma cidade do mapa
    Caso o mapa esteja vazio ou a cidade nao exista no mapa, nada e feito
    Caso a cidade passada para a remocao for encontrada,
    entao percorre a lista recursivamente para remove-la das outras listas de estradas

    Para atualizar a lista de conexoes foi usada a funcao auxiliar atualizarEstradas
-}

removerCidade :: Nome -> Mapa -> Mapa
removerCidade _ [] = []
removerCidade cidade ((nome, coord, estradas) : resto)
  | cidade == nome = removerCidade cidade resto
  | otherwise = (nome, coord, estradasAtualizadas) : removerCidade cidade resto
  where
    estradasAtualizadas = atualizarEstradas cidade estradas
```

Em seguida, também foi implementada a ***adicionarEstrada*** que tem o objetivo de adicionar uma estrada entre duas cidades se ambas estiverem no mapa; ela recebe o mapa, uma cidade de origem e uma de destino como entrada, percorre toda a lista e quando a cidade origem for encontrada, é adicionada a estrada da cidade destino e vice-versa.

```haskell
{-  Funcao pra adiconar uma estrada entre duas cidades
    Caso o mapa esteja vazio, nada e feito
    Caso contrario, percorre o mapa e verifica se as cidades estao no mapa,
    se estiverem ela vai atualizar a lista de conexoes das cidades
    incluindo as cidades na lista de estradas
-}

adicionarEstrada :: Mapa -> Nome -> Nome -> Mapa
adicionarEstrada mapa origem destino =
    if (existeCidade origem mapa && existeCidade destino mapa)
        then inserir mapa origem destino
        else mapa
    where
        inserir :: Mapa -> Nome -> Nome -> Mapa
        inserir [] _ _ = []
        inserir ((nome, coord, estradas) : resto) origem destino
            | origem == nome = (nome, coord, destino:estradas) : inserir resto origem destino
            | destino == nome = (nome, coord, origem:estradas) : inserir resto origem destino
            | otherwise = (nome, coord, estradas) : inserir resto origem destino
```

Também, implementou-se a função ***removerEstrada***, que recebe um mapa e duas cidades como parâmetro, e utiliza a recursividade para percorrer a lista de cidades e remover as estradas das cidades passadas como parâmetro através da função auxiliar ***atualizarEstradas***.

```haskell
{- Para remover uma estrada de uma cidade pra outra, a funcao verifica se as cidades existem no mapa,
    se existirem usamos uma funcao auxiliar para atualizar lista de conexoes,
    removendo as cidades atraves de uma funcao auxiliar
-}

removerEstrada :: Mapa -> Nome -> Nome -> Mapa
removerEstrada mapa origem destino =
    if (existeCidade origem mapa && existeCidade destino mapa)
        then remover mapa origem destino
        else mapa
    where
        remover :: Mapa -> Nome -> Nome -> Mapa
        remover [] _ _ = []
        remover ((nome, coord, estradas):resto) origem destino
            | origem == nome = (nome, coord, atualizarEstradas destino estradas) : remover resto origem destino
            | destino == nome = (nome, coord, atualizarEstradas origem estradas) : remover resto origem destino
            | otherwise = (nome, coord, estradas) : remover resto origem destino
```

3. CONCLUSÃO

Neste trabalho foram usados conhecimentos de funções da linguagem Haskell para o desenvolvimento da primeira etapa do projeto de criação de um mapa de cidades. As funções que foram criadas formam a estrutura fundamental do programa e servirão para a implementação das funções da próxima etapa do projeto que envolve o cálculo de rotas entre as cidades.