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 2:

Projeto Mapas

Aracaju, SE
2023

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

RELATÓRIO 2:
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 teve como objetivo a criação de um sistema que permita a representação e manipulação de um mapa e suas cidades. E, na segunda etapa, é tido como objetivo a criação de algumas funções para obter cálculos das rotas e estradas que o mapa possui.

2. CÁLCULO DE ROTAS

2.1. Funções de auxílio implementadas

Após a modelagem de dados e a criação das funções iniciais do projeto, na primeira etapa, na segunda etapa foram criadas algumas funções adicionais que serviriam como auxiliares para essa etapa como: ***obterEstradasNome***, que foi utilizada em casos que precisaria obter a estrada a partir do nome da cidade fornecido, é uma das funções obrigatórias etapa 2; ***existeEstrada***, que irá verificar se existe uma estrada que conecta duas cidades, também é uma das funções que foi pedida na etapa 2; ***calculoEuclidiano***, que será necessário para as funções onde precisa calcular distância entre cidades; é uma função que

```haskell
existeEstrada :: Mapa -> Nome -> Nome -> Bool
existeEstrada [] _ _ = error "O mapa esta vazio"
existeEstrada mapa nome1 nome2
    | not (existemCidades nome1 nome2 mapa) = error "Uma das cidades nao existe no mapa"
    | otherwise = percorrerEstrada
    where
        percorrerEstrada = elem nome1 (obterEstradasNome mapa nome2)
```

realiza um cálculo euclidiano.

2.2. Funções para o cálculo de rotas

```haskell
obterEstradasNome :: Mapa -> Nome -> Rotas
obterEstradasNome [] _ = error "O mapa esta vazio"
obterEstradasNome mapa nome
    | not (existeCidade nome mapa) = error "A cidade nao existe no mapa"
    | otherwise = obterEstradas (encontraCidade mapa nome)
```

```haskell
calculoEuclidiano :: Localizacao -> Localizacao -> Double
calculoEuclidiano (p1, p2) (q1, q2) = sqrt ((p1 - q1)^2 + (p2 - q2)^2)
```

A primeira função a ser criada foi a função, ***existeEstrada***, a qual verifica se existe uma estrada que conecta duas cidades, essa função apenas verifica se o nome de uma das cidades está na estrada da outra.

```
existeEstrada :: Mapa -> Nome -> Nome -> Bool
existeEstrada [] _ _ = error "O mapa esta vazio"
existeEstrada mapa nome1 nome2
    | not (existemCidades nome1 nome2 mapa) = error "Uma das cidades nao existe no mapa"
    | otherwise = percorrerEstrada
    where
        percorrerEstrada = elem nome1 (obterEstradasNome mapa nome2)
```

Para a criação das funções que interagem diretamente com as estradas foi criada uma função que será essencial para o funcionamento de todas as outras, é a função ***encontraRota***, que irá retornar uma rota entre as cidades passadas como parâmetro, para que isso acontecesse foi utilizada uma lista de cidades que já teriam sido visitadas, para que não ocorresse uma recursão infinita e para que não houvesse repetição de cidades na rota, essa função faz uma busca de possíveis rotas entre as cidades e retorna a primeira que está na lista dessas rotas.

```
encontraRota :: Mapa -> Nome -> Nome -> Rotas -> Rotas
encontraRota mapa origem destino visitadas
    | origem == destino = [origem] ++ visitadas
    | elem origem visitadas = []
    | otherwise      = if rotasFinais == [] then [] else (head rotasFinais)
    where
        estradas     = obterEstradasNome mapa origem
        rotas        = [encontraRota mapa proxima destino (origem : visitadas) | proxima <- estradas]
        rotasFinais  = [rota | rota <- rotas, rota /= []]
```

Em seguida foi criada a função que verifica se existe uma rota entre duas cidades, ***existeRota***, essa função ficou bem simples uma vez que ela apenas depende da função anterior, caso ***encontraRota*** retorne uma lista vazia, então não existirá nenhuma rota, e se a lista não for vazia, então existirá uma rota.

```
existeRota :: Mapa -> Nome -> Nome -> Bool
existeRota [] _ _ = error "O mapa esta vazio"
existeRota mapa nome1 nome2
    | not (existemCidades nome1 nome2 mapa) = error "Uma das cidades nao existe no mapa"
    | rota == [] = False
    | otherwise  = True
    where
        rota = encontraRota mapa nome1 nome2 []
```

Como a função ***encontraRota***, já está retornando uma possível rota das cidades, nos demos ao trabalho apenas de deixar com que o retorno seja ordenado, e com essa ideia

complementamos com a função ***rotasEntreCidades***, ainda há uma verificação para ter certeza de que a cidade origem não é a cidade destino também, e como a função auxiliar está retornando a rota de forma inversa foi necessário apenas aplicar um reverse na lista.

```haskell
rotaEntreCidades :: Mapa -> Nome -> Nome -> Rotas
rotaEntreCidades mapa origem destino
    | origem == destino = [origem]
    | otherwise = reverse $ encontraRota mapa origem destino []
```

E por último, a função que calcula a distância de uma rota, ***calcularDistanciaRota***, essa função percorre a lista de rotas fazendo um cálculo euclidiano sempre em 2 cidades por vez, dessa forma não há confusão na lógica implementada e fica simples de entender, essa função também utiliza como base a função auxiliar ***encontraRota***.

```haskell
calcularDistanciaRota :: Mapa -> Nome -> Nome -> Double
calcularDistanciaRota [] _ _ = error "O mapa esta vazio"
calcularDistanciaRota mapa origem destino
    | not (existemCidades origem destino mapa) = error "Uma das cidades nao existe no mapa"
    | origem == destino  = 0
    | elem origem (obterEstradasNome mapa destino) = calcularDistancia mapa origem destino
    | otherwise = soma2 mapa (rotaEntreCidades mapa origem destino)
    where
        soma2 :: Mapa -> Rotas -> Double
        soma2 _ [] = error "Nao existe rota entre as cidades"
        soma2 mapa (x:xs)
            | length xs == 1 = calcularDistancia mapa x (head xs)
            | otherwise = (calcularDistancia mapa x (head xs)) + soma2 mapa xs
```

3. CONCLUSÃO

Neste trabalho foram usados conhecimentos de funções da linguagem Haskell para o desenvolvimento do projeto de criação de um mapa de cidades. As funções que foram criadas nesta etapa do projeto servem para calcular a distância e o comprimento da rota entre duas cidades.