Lista de Exercícios 12 (Cap. 28) – INF05008

- Siga as instruções sobre a elaboração de exercícios de INF05008.
- Para todas as funções recursivas definidas por você, mostre o modelo de solução (a seguir) e argumente sobre a terminação da função. Lembre que o problema pode ter vários casos triviais e vários casos mais complexos.

```
;; MODELO DE SOLUÇÃO:
;; Se <problema trivial> então <solução do problema trivial>
;; Se <problema complexo> então
;;           <combinar soluções>
;;                 <solucionar subproblema 1> <gerar subproblema 1>
;;                            ...                        ...
;;                 <solucionar subproblema n> <gerar subproblema n>
```

Considere a definição de dados a seguir para os exercícios 1 a 6 desta lista (defina o tipo `ListaDeString`).

```
(define-struct cidade (nome vizinhas))
;; Um elemento do conjunto Cidade é um par
;;                 (make-cidade n v), onde
;;       n : String, representa o nome da cidade
;;       v : ListaDeString, representa os (nomes das) cidades vizinhas

;; Um Mapa é
;; 1. empty, ou
;; 2. (cons n g), onde
;;       n : Cidade
;;       g : Mapa
```

1. Definar uma constante para representar o grafo de cidades (mapa) do slide 9 do arquivo com os slides da aula (desconsidere o tamanho dos arcos e o fato de existirem mais de um arco entre dois nodos em alguns casos). Seu grafo deve ser uma lista com 16 nodos, um para cada cidade deste mapa. Dê o nome de `MAPA` para esta constante. Os nomes das cidades devem ser colocados somente com a primeira letra maiúscula e as outras minúsculas (no caso de nomes com mais de uma palavra, colocar sem espaço e com a primeira letra da segunda palavra também maiúscula). Exemplos: `"Chicago"`, `"KansasCity"`

2. Construa as funções `está-na-lista?`, que dados um nome (string) e uma lista de strings, nesta ordem, diz se o nome dado está na lista, e `subtrai-lista` que, dadas duas listas de strings, devolve todos os elementos da primeira lista que não estão na segunda (ou seja, subtrai a segunda lista da primeira).

3. Analise a função `vizinhos` mostrada no slide 10 e construa uma função chamada `vizinhos` da seguinte forma: a entrada deve ser o nome de uma cidade, um mapa e uma lista de nomes de cidades (já visitadas), nesta ordem, e a saída deve ser a lista dos nomes das cidades vizinhas deste cidade que não constam da lista de (nomes de cidades de) entrada. Assuma que a cidade passada como argumento pertence ao mapa.

4. Usando a função `vizinhos` construída na questão anterior, desenvolva as funções `encontra-caminho` e `encontra-caminho-vizinhos` que encontram um caminho em um grafo, se existir, sem entrar em loop infinito quando forem encontrados ciclos (ver nos slides 24 e 25 os esboços destes programas).

5. Defina um tipo de dados, chamado `ListaDeListaDeString` para representar um conjunto no qual os elementos são listas de listas de strings (ou seja, uma lista na qual cada elemento é uma lista). Usando a função `monta-caminhos` (slide 27), construa a função `encontra-todos-caminhos` para gerar todos os caminhos possíveis entre 2 cidades de um mapa (sendo que uma mesma cidade pode aparecer em caminhos diferentes, mas não pode aparecer mais de uma vez em um mesmo caminho). Esta função recebe o nome da cidade origem, o nome da cidade destino e um mapa, nesta ordem, e gera uma lista de caminhos, onde cada caminho é representado por uma lista de strings. *Sugestão: Faça essa função com base nas funções `encontra-caminho` e `encontra-caminho-vizinhos`.*

6. Construa a função `mostra-caminhos` que, dada uma cidade origem, uma destino e um mapa de cidades, nesta ordem, gera uma string contendo na primeira linha o número de caminhos existentes da cidade origem até a destino e depois a lista de todos os caminhos (invente uma visualização - como string - para cada caminho). Os caminhos devem estar numerados, um em cada linha. Para fazer a quebra de linha em uma string use `"\n"`. Como o resultado da aplicação desta função será uma string com caracteres especiais (quebras de linha), para visualizar o resultado de seu programa use o comando `display` - para usar esse comando você deve selecionar a linguagem *Advanced Student* . Por exemplo, digite `(display (mostra-caminho "Chicago" "Toronto" MAPA))`.Você pode escolher a formatação que quiser para cada caminho.

7. *(Desafio – Vale 2 pontos extras)* Imagine agora que decidimos que não queremos somente encontrar caminhos no mapa, mas queremos também encontrar o caminho mais curto entre 2 cidades. Para isso, o primeiro passo é colocar a informação sobre as distâncias entre as cidades na representação do mapa, ou seja, precisamos mudar a representação da informação. Considere a definição de mapa revisada a seguir:

```
(define-struct estrada (destino distancia))
;; Um elemento do conjunto Estrada é um par
;;                (make-estrada n v), onde
;;       n : String, representa o nome da cidade destino da estrada
;;       v : Número, representa o comprimento desta estrada (entre origem e este destino)

;; Uma ListaDeEstradas é
;; 1. empty, ou
;; 2. (cons e le), onde e:Estrada e le: ListaDeEstradas

(define-struct cidade-rev (nome vizinhas))
;; Um elemento do conjunto  Cidade é um par
;;                (make-cidade-rev n v), onde
;;       n : String, representa o nome da cidade
;;       v : ListaDeEstradas, representa as estradas para cidades vizinhas

;; Um Mapa-rev é
;; 1. empty, ou
;; 2. (cons n g), onde
;;       n : Cidade
;;       g : Mapa
```

Construa a função `menor-caminho` que, dados os nomes de uma cidade origem e uma cidade destino e um mapa (com estradas), nesta ordem, devolve o caminho mais curto entre essas duas cidades (se houver mais de um caminho com o mesmo menor tamanho, lista todas opções de menores caminhos). O resultado é uma string com todos os menores caminhos (da mesma forma que o exercício anterior), mostrando na primeira linha o tamanho do menor caminho. Se não houver caminho entre as cidades, a string resultante será uma mensagem dizendo isso.

Dica: Decomponha o problema em problemas menores. Como esta questão usa uma representação do mapa diferente das anteriores, você precisará refazer as funções de encontrar caminhos para adapta-las para as novas definições de dados. Isso é proposital nesta lista, e o objetivo é enfatizar a importância de se pensar muito bem nas definições da dados e em todas as informações que se quer obter com eles ANTES de construir os programas, pois quando as representações de dados mudam, isso normalmente gera muitas mudanças no código.