Paradigmas de Linguagens Computacionais

Leonardo Espíndola (lre) e Filipe Nogueira Jordão (fnj2)

Hash Table em Haskell

```haskell
type Hash = [(Int, Int)]

get :: Hash -> Int -> Int
get [] key = -1
get ((k,v):xs) key
    | k == key = v
    | otherwise = get xs key

put :: Hash -> Int -> Int -> Hash
put [] key val = [(key, val)]
put ((k,v):xs) key val
    | k /= key = (k,v) : put xs key val
    | otherwise = (k,v) : xs

remove :: Hash -> Int -> Hash
remove [] key = []
remove ((k,v):xs) key
    | k == key = xs
    | otherwise = (k,v) : remove as key

hasKey:: Hash -> Int -> Bool
hasKey [] key = False
hasKey ((k,v):xs) key
    | k == key = True
    | otherwise hasKey xs key
```

Função GET da Hash

```haskell
get :: Hash -> Int -> Int
get [] key = -1
get ((k,v):xs) key
    | k == key = v
    | otherwise = get xs key
```

Funçaõ PUT da Hash

```haskell
put :: Hash -> Int -> Int -> Hash
put [] key val = [(key, val)]
put ((k,v):xs) key val
    | k /= key = (k,v) : put xs key val
    | otherwise = (k,v) : xs
```

Função REMOVE da Hash

```haskell
remove :: Hash -> Int -> Hash
remove [] key = []
remove ((k,v):xs) key
    | k == key = xs
    | otherwise = (k,v) : remove as key
```

Função hasKey da Hash

```haskell
hasKey:: Hash -> Int -> Bool
hasKey [] key = False
hasKey ((k,v):xs) key
    | k == key = True
    | otherwise hasKey xs key
```

```haskell
contains :: (Eq t) => [t] -> [t] -> Bool
contains x y
    | x == [] = True
    | otherwise = ( exists y (head x) && contains (tail x) y)

exists :: (Eq t) => [t] -> t -> Bool
exists xs e
    | xs == [] = False
    | head xs == e = True
    | otherwise = exists (tail xs) e

intersect :: (Eq t) => [t] -> [t] -> Bool
intersect xs ys
    | xs == [] = False
    | exists ys (head xs) = True
    | otherwise = intersect (tail xs) ys


comparaConjuntos :: (Eq t) => [t] -> [t] -> String
comparaConjuntos x y
    | contains x y && contains y x = "A igual a B"
    | contains x y = "B contem A"
    | contains y x = "A contem B"
    | intersect x y = "A interseciona B"
    | otherwise = "Conjuntos Disjuntos"
```

Função contains do set check

```haskell
contains :: (Eq t) => [t] -> [t] -> Bool
contains x y
    | x == [] = True
    | otherwise = ( exists y (head x) && contains (tail x) y)
```

Função exists do set check

```haskell
exists :: (Eq t) => [t] -> t -> Bool
exists xs e
    | xs == [] = False
    | head xs == e = True
    | otherwise = exists (tail xs) e
```

Função intersect do set check

```haskell
intersect :: (Eq t) => [t] -> [t] -> Bool
intersect xs ys
    | xs == [] = False
    | exists ys (head xs) = True
    | otherwise = intersect (tail xs) ys
```

Função compare do set check

```haskell
comparaConjuntos :: (Eq t) => [t] -> [t] -> String
comparaConjuntos x y
    | contains x y && contains y x = "A igual a B"
    | contains x y = "B contem A"
    | contains y x = "A contem B"
    | intersect x y = "A interseciona B"
    | otherwise = "Conjuntos Disjuntos"
```

Dúvidas ?