

Paradigmas de Linguagens Computacionais

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

Hash Table em Haskell

```
1  type Hash = [(Int, Int)]
2
3  get :: Hash -> Int -> Int
4  get [] key = -1
5  get ((k,v):xs) key
6  | k == key = v
7  | otherwise = get xs key
8
9  put :: Hash -> Int -> Int -> Hash
10 put [] key val = [(key, val)]
11 put ((k,v):xs) key val
12 | k /= key = (k,v) : put xs key val
13 | otherwise = (k,v) : xs
14
15 remove :: Hash -> Int -> Hash
16 remove [] key = []
17 remove ((k,v):xs) key
18 | k == key = xs
19 | otherwise = (k,v) : remove xs key
20
21 hasKey :: Hash -> Int -> Bool
22 hasKey [] key = False
23 hasKey ((k,v):xs) key
24 | k == key = True
25 | otherwise = hasKey xs key
```

Função GET da Hash

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

Função PUT da Hash

```
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

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

Função `hasKey` da `Hash`

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

```
1  contains :: (Eq t) => [t] -> [t] -> Bool
2  contains x y
3      | x == [] = True
4      | otherwise = (exists y (head x) && contains (tail x) y)
5
6  exists :: (Eq t) => [t] -> t -> Bool
7  exists xs e
8      | xs == [] = False
9      | head xs == e = True
10     | otherwise = exists (tail xs) e
11
12  intersect :: (Eq t) => [t] -> [t] -> Bool
13  intersect xs ys
14      | xs == [] = False
15      | exists ys (head xs) = True
16      | otherwise = intersect (tail xs) ys
17
18
19  comparaConjuntos :: (Eq t) => [t] -> [t] -> String
20  comparaConjuntos x y
21      | contains x y && contains y x = "A igual a B"
22      | contains x y = "B contem A"
23      | contains y x = "A contem B"
24      | intersect x y = "A intersecciona B"
25      | otherwise = "Conjuntos Disjuntos"
```

Função contains do set check

```
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

```
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

```
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

```
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 intersecciona B"
  | otherwise = "Conjuntos Disjuntos"
```

Dúvidas ?