Tabela Hash
+
comparaConjunto

3º Trabalho PLC

O que é?

Constitue uma abordagem comum para o problema de armazenar e procurar dados de forma eficiente

Implementação

```haskell
type Key = Int
type Objeto = (Key,String)
type HashTable = [Objeto]

baseTable :: HashTable
baseTable = []

initTable:: HashTable->Int-> HashTable
initTable ht sz | sz == 100  = ht
                | otherwise  = initTable ((-1,""):ht) (sz+1)


hash::Int->Int->Int
hash k m = (k) `mod` m

checkNext :: Int->Int->HashTable->Int
checkNext k v ht| k == (fst (ht!!v)) = v
                | (hash k 100) == v = -1
                | otherwise = checkNext k ((v+1) `mod` 100) ht


get:: Key->HashTable->String
get k ht | k == (fst (ht!!v)) = (snd (ht!!v))
         | (checkNext k (v+1) ht) == -1 = "key invalida"
         | otherwise = snd (ht!!(checkNext k (v+1) ht))
           where v = ((hash k 100))
```

Implementação

```
nextAvailabe:: Int->Int->Int->HashTable->Int
nextAvailabe k v e h | -1 == (fst (h!!v)) || k == (fst (h!!v))= v
                    | v == e = -1
                    | otherwise  = (nextAvailabe k ((v+1) `mod` 100) e h)

put:: Key->String->HashTable->HashTable
put k s ht | -1 == (fst (ht!!mk)) || k ==(fst (ht!!mk))  = (take (mk) ht) ++ (k,s):(drop (mk+1) ht)
           | nb == -1 = ht
           | otherwise =  (take (nb) ht) ++ (k,s):(drop (nb+1) ht)
        where  mk = ((hash k 100))
           nb = (nextAvailabe k (mk+1) mk ht)

remove::Key->HashTable->HashTable
remove k ht | k == (fst (ht!!v)) = (take (v) ht) ++ (-1,""):(drop (v+1) ht)
            | (checkNext k (v+1) ht) == -1 = ht
            | otherwise = (take (na) ht) ++ (-1,""):(drop (na+1) ht)
            where v = (hash k 100)
                  na = (checkNext k (v+1) ht)

hasKey:: Key->HashTable->Bool
hasKey k [] = False
hasKey k (x:xs) | k==(fst x) =True
                | otherwise = (hasKey k xs)
```

Implementação

```haskell
haveElement :: (Eq t)=>t->[t]->Bool
haveElement a [] = False;
haveElement a (s:xs) | a==s = True
                     | otherwise = haveElement a xs

contem:: (Eq t) => [t]->[t]->Bool
contem a [] = True
contem a (b:xb) = (haveElement b a) && (contem a xb)

intersecta:: (Eq t) => [t]->[t]->Bool
intersecta a [] = False
intersecta a (b:xb) = (haveElement b a) || (intersecta a xb)

comparaConjuntos ::(Eq t) => [t]->[t] ->String
comparaConjuntos a b     | cab && cba = "A igual B"
                         | cab = "A contem B"
                         | cba = "B contem A"
                         | iab = "A interseciona B"
                         | otherwise = "Conjuntos Disjuntos"
                           where   cab = (contem a b)
                                   cba = (contem b a)
                                   iab = (intersecta a b)
```

Custo

Comparação: m*n

Hash: Em média O(1) e no pior caso O(n)